Matris Yöntemleri — MIT 18.065 Strang'den

ML Builder için Türkçe Notlar

Phase 2

2026-06-07

İçindekiler

1 Önsöz

2 Bu kitap nedir?

Bu, **Gilbert Strang — MIT 18.065: Matrix Methods in Data Analysis, Signal Processing, and Machine Learning** ders serisinin Türkçe ders notlarıdır. Hedef, videoları izlerken paralel okunabilecek; sonradan tek başına da yeterli olabilecek bir referans seti üretmek.

18.065'in tek bir iddiası var ve her ders onu biraz daha somutlaştırır: **modern makine öğrenmesi, lineer cebirin üstünde durur.** Strang'in 18.06'da kurduğu klasik lineer cebiri (kolon uzayı, özdeğerler, SVD) alıp, doğrudan veriden öğrenmeye — düşük-rank yaklaşıma, gradyan inişine, sinir ağlarına — bağlar. Strang'in sözüyle: *"Linear algebra is the secret to everything."*

Her bölüm bir **Builder Notu** katmanı taşır: kavramın hem **geriye** (18.06 klasik lineer cebir, calculus, olasılık) hem de **ileriye** (PCA, LoRA, backprop, CNN, spektral kümeleme) köprüsü. Bu seriyi "soyut matris teorisi" olarak değil, **veri-odaklı düşünme** disipliniyle okuyoruz: her teorem bir ML aracına iner.

> **i** Kaynak
>
> - **Seri:** MIT 18.065 — Video Lectures (OCW) — Gilbert Strang, Bahar 2018
> - **Yazar:** Gilbert Strang — MIT matematik profesörü, *Linear Algebra and Learning from Data* (2019) kitabının yazarı
> - **Önkoşul:** Klasik lineer cebir (18.06 / Phase 1 *Lineer Cebir*) — bu seri onun ML-uygulamalı devamıdır
> - **Çeviri ve genişletme:** Phase 2 (TR + ML Builder köprüleri)

3 Nasıl Okumalı

Sıralı oku. Seri kümülatiftir — her ders bir öncekinin kurduğu faktorizasyon dilini kullanır. İlk blok (Ders 1–7) lineer cebirin omurgasını ML gözüyle yeniden çerçeveler: kolon uzayı, beş faktorizasyon, özdeğerler, pozitif tanımlılık, SVD. Orta blok (Ders 8–19) sayısal lineer cebir ve matris analizidir. Son blok (Ders 20–36) optimizasyon, derin öğrenme ve sinyal işlemeye iner.

Strang'in önerdiği akış: önce videoyu izle, sonra ilgili dersi oku, en sonunda **kendi elinle hesapla** — küçük matrislerle faktorizasyonları, `numpy` ile doğrulamaları. Bu set videoyu **destekler**, ikame etmez.

> Pratik bir tavsiye
>
> Her bölüm sonundaki **egzersizleri** atlama. Strang'in felsefesi tek cümle: küçük bir matrisle eline al. Bir 3×3 matrisin rankını, A = CR'sini, SVD'sini elle bul; sonra `numpy` ile kontrol et. Aynı faktorizasyonu ileride PCA, LoRA, en küçük kareler kılığında yıllarca farklı yerlerde göreceksin; bir kez elle yapmak onu kara kutu olmaktan çıkarır.

4 34 Ders

Seri kolon uzayından başlayıp SVD, optimizasyon ve sinir ağlarına doğru tırmanır. (OCW kaydında Ders 28–29 laboratuvar oturumları kaydedilmemiştir; numaralandırma OCW ile birebir korunur.)

#	Ders	Ana Fikir
1	Kolon Uzayı — A = CR	Ax = kolonların kombinasyonu; rank; kolon rank = satır rank
2	Beş Faktorizasyon	LU, QR, $Q\Lambda Q^T$, SVD, CR — her biri bir soruya cevap
3	Ortonormal Kolonlar — $Q^TQ = I$	Ortogonallik, dönme/yansıma, uzunluk koruyan matrisler
4	Özdeğerler ve Özvektörler	$Ax = \lambda x$; $Q\Lambda Q^{-1}$ ile köşegenleştirme
5	Pozitif Tanımlı Matrisler	Enerji $x^TAx > 0$; tüm $\lambda > 0$; optimizasyonun temeli
6	Tekil Değer Ayrışımı (SVD)	$A = U\Sigma V^T$ — her matris için var olan ayrışım
7	Eckart-Young / PCA	En iyi düşük-rank yaklaşım = en büyük σ'lar
8	Vektör ve Matris Normları	$\|\cdot\|_1$, $\|\cdot\|_2$, $\|\cdot\|\infty$, Frobenius, nükleer norm
9	En Küçük Kareler — Dört Yol	$A^TA\,\hat{x} = A^Tb$; projeksiyon, QR, SVD, pseudoinverse
10	Ax = b Çözmenin Zorlukları	Kondisyon sayısı, kötü koşulluluk, Krylov
11	Min ‖x‖ ve Gram-Schmidt	Ortonormalleştirme; Arnoldi, Lanczos
12	Özdeğer / Tekil Değer Hesaplamak	QR algoritması, kaydırma, sayısal yöntemler
13	Rastlantısal Matris Çarpımı	Sütun-satır örnekleme; randomized lineer cebir
14	Düşük-Rank Değişim (SMW)	Sherman-Morrison-Woodbury; güncellemeler
15	A(t)'nin Türevi — dA/dt	Matris değişiminin türevi; özdeğer hassasiyeti
16	Ters ve Tekil Değer Türevleri	Pertürbasyon; interlacing, Weyl eşitsizlikleri
17	Hızla Azalan Tekil Değerler	Düşük-rank yaklaşılabilirlik (konuk: A. Townsend)
18	Parametre Sayımı ve Eyer Noktaları	Serbestlik dereceleri; Rayleigh oranı
19	Maxmin Karakterizasyonu	Courant-Fischer; eyer noktası prensibi
20	Ortalama, Varyans, Kovaryans	Kovaryans matrisi; PCA'nın olasılık temeli
21	Newton Yöntemi ve Konvekslik	İkinci-derece optimizasyon; konveks fonksiyonlar
22	Gradyan İnişi	$x \leftarrow x - \eta\nabla f$; line search; en temel optimizatör
23	Momentum ve Nesterov	Hızlandırılmış gradyan; ağır-top yöntemi
24	Doğrusal Programlama ve Oyunlar	Dualite, simpleks, iç-nokta yöntemleri
25	Stokastik Gradyan İnişi (SGD)	Mini-batch; derin öğrenmenin iş atı (konuk: S. Sra)
26	Sinir Ağı Yapısı ve ReLU	Parçalı-doğrusal fonksiyonlar; evrensellik
27	Geri Yayılım (Backprop)	Zincir kuralı = reverse-mode AD; matris-zinciri görüşü
30	Sirkülant Matrisler / Matris Tamamlama	Konvolüsyon, rank-1 tamamlama

#	Ders	Ana Fikir
31	Fourier Matrisi / Normal Matrisler	FFT; $AA^T = A^TA$ olan matrisler
32	CNN / Evrişim Kuralı / Kronecker	Ağırlık paylaşımı; Kronecker çarpımı/toplamı
33	Öğrenme Fonksiyonu F(x,v) / Uzaklık Matrisleri	NN'in öğrendiği fonksiyon; Gram, Cholesky
34	Procrustes / $Q = UV^T$	En yakın ortogonal matris; uzaklık matrisleri
35	Graf Kümeleme / Laplacian	k-means, spektral kümeleme, Fiedler vektörü
36	Edelman/Julia — İleri + Geri AD	Dual sayılar; backprop = $(I–L)^{-1}$ üçgen çözüm (SON DERS)

Not: Phase 2 pilotu Ders 1 (Kolon Uzayı) ile başlar. Dersler 2–36 aynı şablonla eklenecektir.

5 Notasyon

- **Matris ve vektör:** A matris, x vektör (kolon vektörü, NumPy/Julia/MATLAB konvansiyonu)
- **Kolon kombinasyonu:** $Ax = x_1a_1 + x_2a_2 + \cdots + x_na_n$ — a_k, A'nın k. kolonu
- **Kolon uzayı:** $C(A)$ — tüm Ax'lerin kümesi; **satır uzayı:** $C(A^T)$
- **Rank:** r — bağımsız kolon sayısı = bağımsız satır sayısı = $C(A)$'nın boyutu
- **Dış çarpım:** $a\,b^T$ — bir kolon çarpı bir satır = rank-1 matris (yapı taşı)
- **Beş faktorizasyon:** $A = CR$ (rank), $A = LU$ (eliminasyon), $A = QR$ (ortonormallik), $A = Q\Lambda Q^T$ (özdeğer), $A = U\Sigma V^T$ (tekil değer)
- **Özdeğer / tekil değer:** $Ax = \lambda x$; σ_i = A'nın tekil değerleri
- **Norm:** $\|x\|$ vektör normu, $\|A\|$ matris normu

Tüm matematik MathJax 3 ile render ediliyor.

6 Builder Eksen — Lineer Cebir → Makine Öğrenmesi

Her ders bu bağlantı katmanını taşır

18.065 kavramı	ML / veri bilimi karşılığı
$A = CR$ / düşük-rank	LoRA fine-tuning, model sıkıştırma, CUR
SVD + Eckart-Young	PCA, gürültü giderme, latent uzay, öneri sistemleri
Rank-1 = dış çarpım	attention QK^T, ağırlık güncellemeleri, yapı taşı
A^TA / kovaryans	PCA, en küçük kareler, Gram matrisi, kernel trick
Pozitif tanımlılık	konveks optimizasyon, kayıp yüzeyinin şekli
Gradyan inişi / SGD	sinir ağı eğitimi (tüm derin öğrenme)
Backprop = reverse-mode AD	`torch.autograd`, `loss.backward()`
Fourier / sirkülant	konvolüsyon, FFT, CNN ağırlık paylaşımı
Graf Laplacian	spektral kümeleme, graf sinir ağları (GNN)

Bir tek şey

Lineer cebir, bir matrise **bir bütün** olarak bakmayı öğretir: kolonlarının gerdiği uzay, o uzayın boyutu (rank), ve matrisin onu açığa çıkaran faktorizasyonu. Modern ML'in tamamı — sıkıştırmadan optimizasyona, konvolüsyondan attention'a — bu birkaç faktorizasyonun (CR, QR, $Q\Lambda Q^T$, SVD) üstünde durur. 18.065 bu köprüyü ders ders kurar.

7 Yazım Kuralları

- **Türkçe terminoloji + parantez içinde İngilizce orijinal** ilk geçtiğinde: “kolon uzayı (column space)”, “tekil değer ayrışımı (SVD)”, “düşük-rank (low-rank)”.
- **Strang’den alıntılar** İngilizce orijinal hâliyle, blockquote içinde, zaman damgasıyla verilir.
- **Builder Notu** callout’ları her ana bölüm sonunda; ML köprüsünü (geriye + ileriye) buraya yazıyoruz.
- **Figürler** tam, sayısal doğru bir `numpy` motoru kullanan executable hücrelerdir; kaynak kod `code-fold` ile katlanabilir (matematik + figür ön planda, kod erişilebilir).
- **Kontrol Soruları** collapse’lu — cevap kapalı başlar, okur kendi düşündükten sonra açar.
- **Egzersizler** cevapsız — en az bir elle-hesaplama + `numpy` doğrulaması.

> ⚠ Bu kitap Strang’in yerine geçmez
>
> Tek başına bu set yetmez — Strang’in tahtada matrisleri elle çevirerek anlattığı sezginin yerine geçemez. Önce videoyu izle, sonra ilgili dersi oku, son olarak küçük matrislerle **kendin hesapla**. Set videoyu **destekler**, ikame etmez.

8 Kolon Uzayı — A'nın Tüm Ax Vektörleri (A = CR)

Bir matrise bütün olarak bak: Ax kolonların kombinasyonudur, C(A) onların gerdiği uzaydır, A = CR bu yapıyı açığa çıkarır.

> **Bölüm bilgisi**
>
> - **Strang'in videosu:** YouTube — The Column Space of A Contains All Vectors Ax (≈52 dk)
> - **OCW sayfası:** MIT 18.065 — Video Lectures
> - **Okuma süresi:** ≈35 dk
> - **Hoca:** Gilbert Strang
> - **Önkoşul:** 18.06 / Phase 1 Lineer Cebir (kolon uzayı, rank, baz)

8.1 Bu Derste Ne Var?

18.065'in ilk dersi. Strang kursu "veriden öğrenme" üzerine açıyor ve hemen lineer cebirin kalbine giriyor: bir matrise **bir bütün** olarak bakmak. Bu ders, Phase 1 18.06'da gördüğümüz kolon uzayı, rank ve baz kavramlarını veri ve faktorizasyon gözüyle yeniden çerçeveliyor.

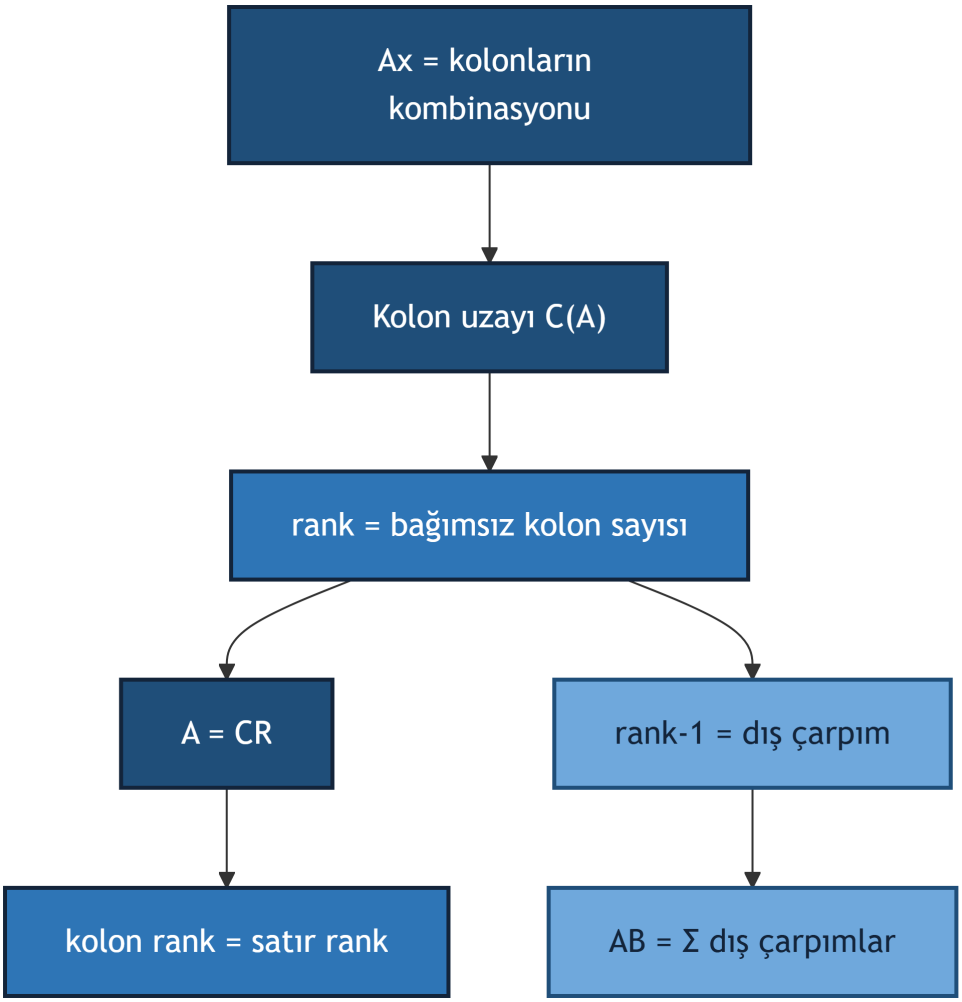
Üç temel fikir:

1. **Ax = A'nın kolonlarının kombinasyonu** — matris-vektör çarpımını "satır satır nokta çarpımı" olarak değil, "kolonların ağırlıklı toplamı" olarak görmek.
2. **Kolon uzayı $C(A)$ ve $A = CR$ faktorizasyonu** — bağımsız kolonlar (C) çarpı bir kombinasyon reçetesi (R).
3. **Kolon rank = satır rank** — lineer cebirin ilk büyük teoremi, doğrudan $A = CR$'den çıkıyor.

   "this is a great adventure for me… teaching a course that involves learning from data." — Strang, 0:23

Bu altı kavramın birbirine nasıl bağlandığını Şekil 8.1 özetliyor.

> **Builder Notu — Veri-Odaklı Cebire Giriş Kapısı**
>
> Bu dersin kavramları ML altyapısında nereye bağlanıyor:
>
> - **Ax = kolonların kombinasyonu** görüşü, GPU'daki matris çarpımının (GEMM) çalışma biçimidir: BLAS çarpımı satır-nokta-çarpımı yerine kolon ve blok kombinasyonları olarak yapar. "Matrise bütün olarak bakmak" performans sezgisidir.



Şekil 8.1: Ders 1 kavram haritası: kolon görüşünden A = CR ve rank teoremine.

- **A = CR ve rank**, düşük-rank yaklaşımın (Ders 7 Eckart-Young, PCA) ve LoRA gibi modern fine-tuning yöntemlerinin temelidir: büyük bir matrisi az sayıda bağımsız kolonla temsil etmek.
- **Dış çarpım (kolon × satır)** birikimi, tüm matris çarpımlarının atomik yapı taşıdır: $AB = \sum_k a_k b_k^T$.
- **Dev matrisi örnekleme** (rastgele x ile Ax), Ders 13'teki rastlantısal lineer cebirin habercisidir.

Tek cümle: bir matrisi "kolonlarının ürettiği uzay" olarak görmek, sıkıştırmadan rastlantısal cebire kadar tüm veri-odaklı lineer cebirin giriş kapısıdır.

8.2 Matris × Vektör — Satır Yolu mu, Kolon Yolu mu?

Strang ilk dersi tanıdık bir işle açıyor: bir matrisi bir vektörle çarpmak. Ama esas soru, bunu **nasıl** gördüğün.

Strang'in matrisi:

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 3 & 1 & 4 \\ 5 & 7 & 12 \end{pmatrix}$$

Ax çarpımını iki şekilde okuyabilirsin.

Satır yolu (alışılmış, low-level): her satırı x ile nokta çarpımına sok. Birinci bileşen $2x_1 + x_2 + 3x_3$, sonra ikinci, sonra üçüncü. Cevabı bileşen bileşen, parça parça verir.

Kolon yolu (Strang'in "doğru" yolu): sonucu kolonların ağırlıklı toplamı olarak gör.

$$Ax = x_1 \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix} + x_2 \begin{pmatrix} 1 \\ 1 \\ 7 \end{pmatrix} + x_3 \begin{pmatrix} 3 \\ 4 \\ 12 \end{pmatrix}$$

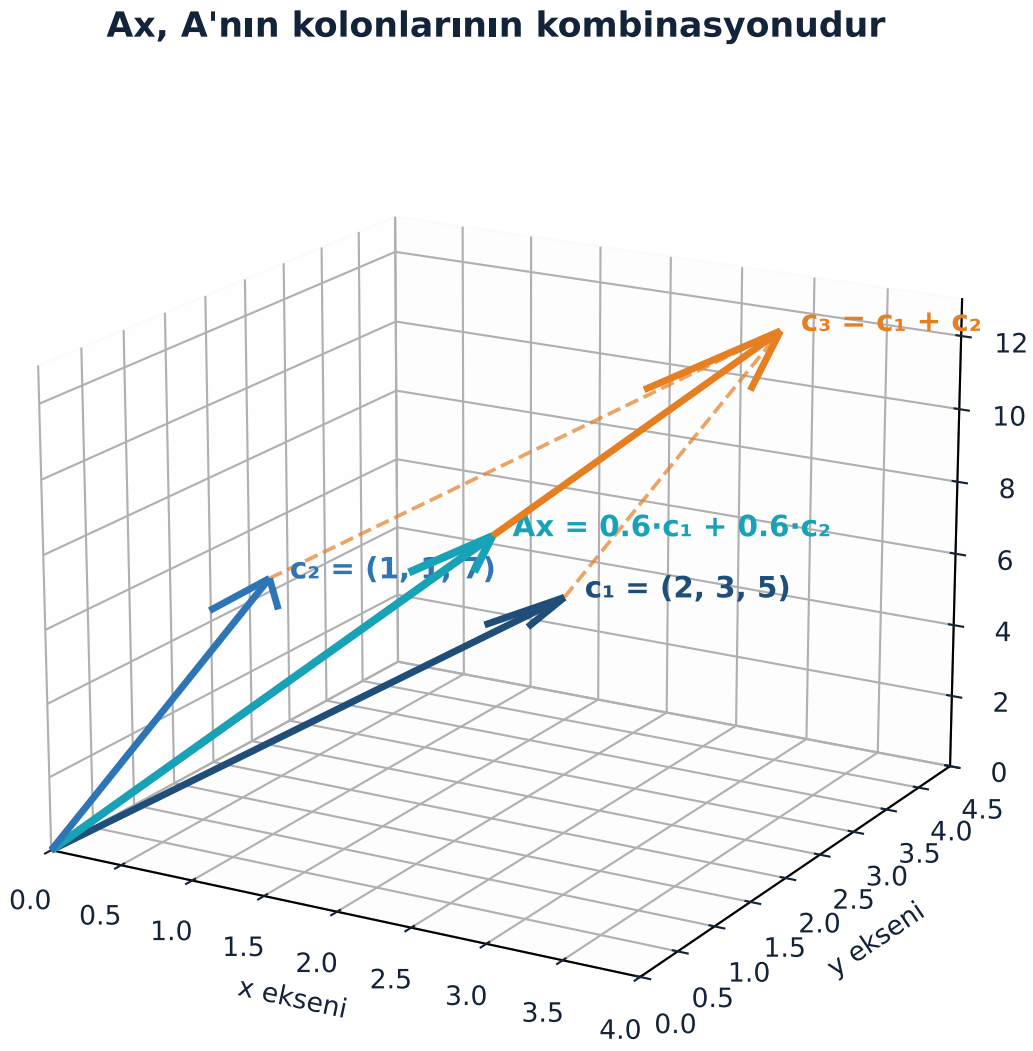
Aynı sayılar çıkar, ama artık her şeyi **bir kerede** görüyorsun: Ax, A'nın kolonlarının bir kombinasyonudur. Şekil 8.2 bu üç kolonu ve onların ağırlıklı toplamı olan Ax'i $\mathbb{R}^3$ uzayında gösteriyor.

> *"But do you know it the right way? Do you think of the multiplication the right way?"* — Strang, 4:29

> *"When I say Ax, you immediately think… it's a combination of the columns of A."* — Strang, 7:32

Builder Notu — Karışım Reçetesi

Kolon görüşü ML'de neden önemli: bir nöron katmanının çıktısı Wx, ağırlık matrisi W'nin kolonlarının girdiyle ölçeklenmiş toplamıdır. Her kolon, girdinin bir bileşeninin çıktıya "katkı yönü"dür. Bu görüş, embedding tablolarını (her kolon bir token'ın vektörü) ve GEMM'in kolon-blok yapısını anlamanın anahtarıdır.



Şekil 8.2: Ax, A'nın kolonlarının ağırlıklı toplamıdır. c_1 (navy) ve c_2 (çelik) bağımsız kolonlar; c_3 (orange) = $c_1 + c_2$ (kesikli paralelkenar tamamlama). Örnek Ax (teal) = $0.6 \cdot c_1 + 0.6 \cdot c_2$.

8.3 Kolon Uzayı C(A)

Tek bir x al, tek bir Ax çıkar. Şimdi **bütün** x'leri al — çıkan Ax'lerin hepsini bir araya topla. Bu sonsuz vektör kümesi neye benzer?

Bütün Ax'lerin kümesine A'nın **kolon uzayı** denir, $C(A)$ ile gösterilir. Tanım gereği bu, A'nın kolonlarının tüm kombinasyonlarının kümesidir — yani kolonların gerdiği uzay.

> *"And the beauty of linear algebra is that questions like this — you can answer them and you intuitively see it."* — Strang, 8:30

Bu A için kolon uzayı $\mathbb{R}^3$'ün tamamı mı? **Hayır** — bir düzlem. Çünkü üçüncü kolon ilk ikisinin toplamı; yeni bir yön getirmiyor. İlk iki kolon bir düzlem geriyor, üçüncü kolon o düzlemin içinde kalıyor. Bu düzlemi ve üzerine düşen rastgele Ax noktalarını Şekil 8.3 gösteriyor.

Eğer bir matrisin tüm kolonları aynı doğrultunun katlarıysa, kolon uzayı yalnızca bir **doğru** olur (rank 1). Rastgele bir 3×3 matriste ise kolonlar bağımsızdır, kolon uzayı tüm $\mathbb{R}^3$'tür ve matris tersinirdir.

> Builder Notu — Erişilebilir Bölge
>
> $C(A)$, "bu matrisle hangi çıktılara ulaşabilirim?" sorusunun cevabıdır. $Ax = b$ denkleminin çözümü olması için b'nin $C(A)$ içinde olması gerekir (Ders 9-10'da least squares tam da bu yüzden devreye girer: b kolon uzayında değilse, en yakın noktaya projekte ederiz).

8.4 Rank ve Rank-1 Matrisler

Kolon uzayının **boyutu**na rank denir. Bizim A'mızın kolon uzayı bir düzlem (2 boyutlu), yani rank 2. Bir doğru olsaydı rank 1, tüm $\mathbb{R}^3$ olsaydı rank 3 olurdu.

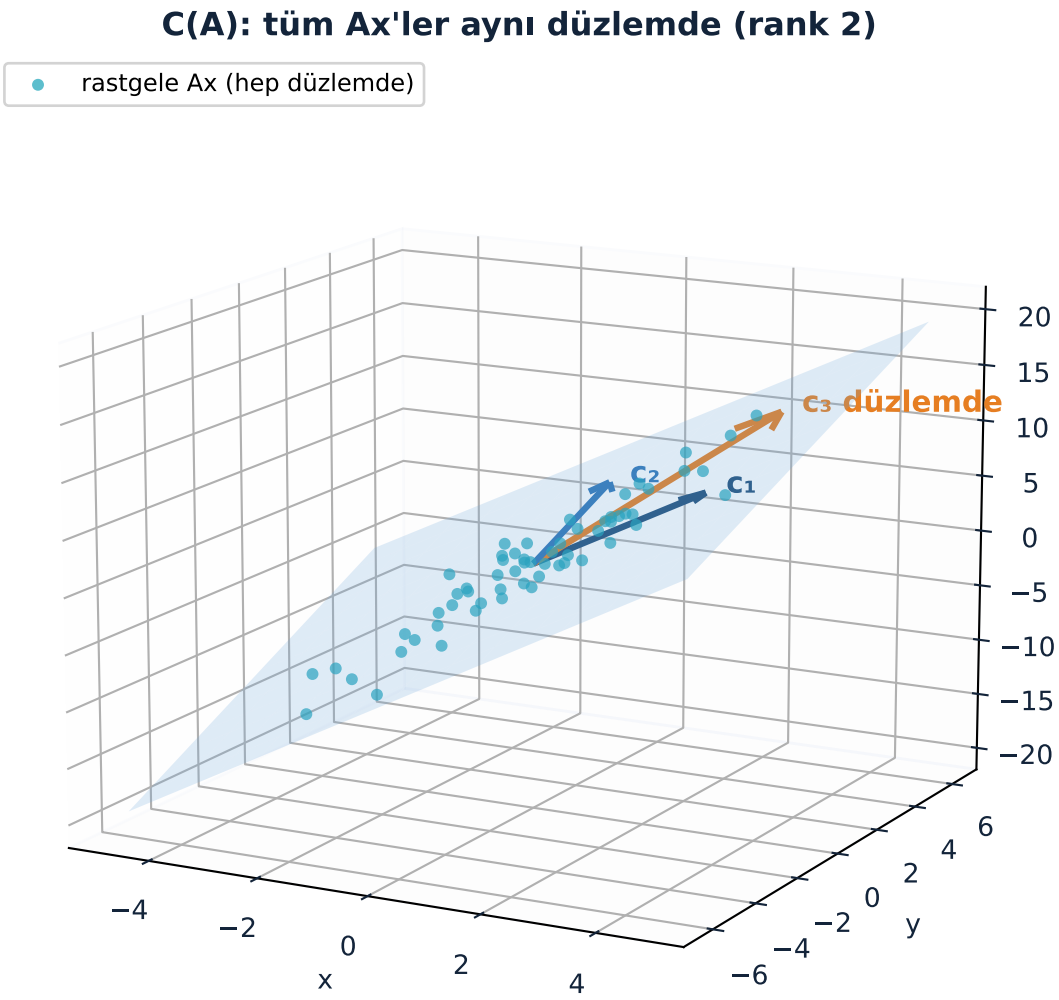
Üçüncü kolonun ilk ikisinin toplamı olması, yalnızca iki bağımsız kolon olduğunu söyler — ve rank, tam olarak bağımsız kolon sayısıdır.

Rank-1 matrisler lineer cebirin yapı taşıdır. Bir rank-1 matris, bir kolon vektörü çarpı bir satır vektörü olarak yazılır:

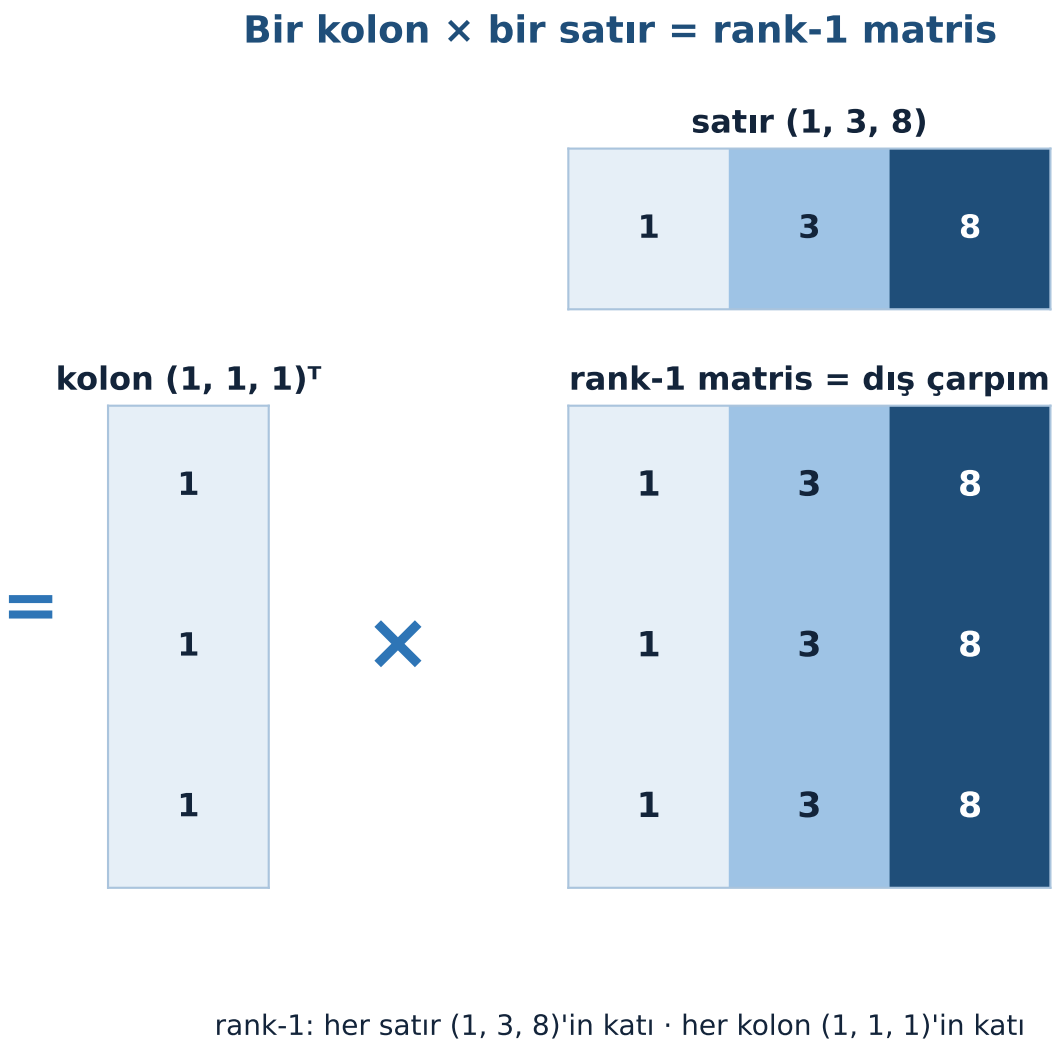
$$\begin{pmatrix}1\\1\\1\end{pmatrix}\begin{pmatrix}1 & 3 & 8\end{pmatrix} = \begin{pmatrix}1 & 3 & 8\\1 & 3 & 8\\1 & 3 & 8\end{pmatrix}$$

3×1 çarpı 1×3 boyutu 3×3 verir; sonucun rankı 1'dir (tüm satırlar (1,3,8)'in katı, tüm kolonlar (1,1,1)'in katı). Şekil 8.4 bu dış çarpımı bir kolon ile bir satırın çarpımı olarak görselleştiriyor.

> *"matrices like this are really the building blocks of linear algebra, they're the building blocks of data science. They're rank one matrices."* — Strang, 13:56



Şekil 8.3: C(A) bir düzlemdir (rank 2): c_1 ve c_2'nin gerdiği düzlem; bağımlı kolon $c_3 = c_1 + c_2$ düzlemde yatar ve 60 rastgele Ax noktasının tamamı bu düzlemin üzerindedir.



Şekil 8.4: Rank-1 matris = bir kolon × bir satır. $(1,1,1)^T$ kolonu ile (1,3,8) satırının dış çarpımı 3×3 matrisi kurar: her satır (1,3,8)'in, her kolon (1,1,1)'in katı.

> Builder Notu — Atomik Birim
>
> Rank-1 = kolon × satır = **dış çarpım**. SVD bir matrisi rank-1 dış çarpımların toplamı olarak yazar (Ders 6-7); en büyük birkaç terimi tutmak = düşük-rank sıkıştırma = PCA, LoRA, gömme sıkıştırma. "Rank-1 yapı taşı" sezgisi tüm kursun omurgasıdır.

8.5 Bağımsız Kolonlar ve Baz

Kolon uzayı için bir **baz** kuralım — en doğal yoldan, soldan sağa tarayarak.

Birinci kolon (2,3,5): sıfır değil, içeri al. İkinci kolon (1,1,7): birinciye bağlı değil, yeni bir yön — içeri al. Üçüncü kolon (3,4,12): ilk ikisinin toplamı, yeni bir şey katmıyor — **alma**. Geriye iki bağımsız kolon kaldı; bunlar kolon uzayının bir bazıdır.

> *"those two columns would be a basis for the column space."* — Strang, 15:51

Bu bağımsız kolonları yan yana koyup C matrisini oluştur:

$$C = \begin{pmatrix} 2 & 1 \\ 3 & 1 \\ 5 & 7 \end{pmatrix}$$

C doğrudan A'dan alındı; sadece bağımsız kolonları, soldan sağa tutarak.

> *"I kept only independent columns and I worked from left to right."* — Strang, 19:51

> Builder Notu — Hangi Kolonlar Önemli?
>
> Bu "soldan sağa, bağımsızsa tut" yordamı, sayısal lineer cebirdeki **kolon pivotlu QR** ve rank-açığa-çıkaran faktorizasyonların sezgisidir. CUR ve sütun-altküme seçimi gibi yöntemler tam da gerçek kolonları seçer — yorumlanabilirlik için (SVD'nin soyut tekil vektörlerinin aksine, C'nin kolonları gerçek veri sütunlarıdır).

8.6 A = CR Faktorizasyonu

C, A'nın bağımsız kolonlarını taşıyor (3×2). Şimdi bir R matrisi (2×3) kuralım: R, A'nın her kolonunun C'nin kolonlarından **nasıl** üretileceğini söylesin.

- A'nın 1. kolonu = 1·(C'nin 1. kolonu) + 0·(C'nin 2. kolonu) → R'nin 1. kolonu (1,0)
- A'nın 2. kolonu = 0·… + 1·… → R'nin 2. kolonu (0,1)
- A'nın 3. kolonu = 1·… + 1·… (çünkü $c_3 = c_1 + c_2$) → R'nin 3. kolonu (1,1)

$$R = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

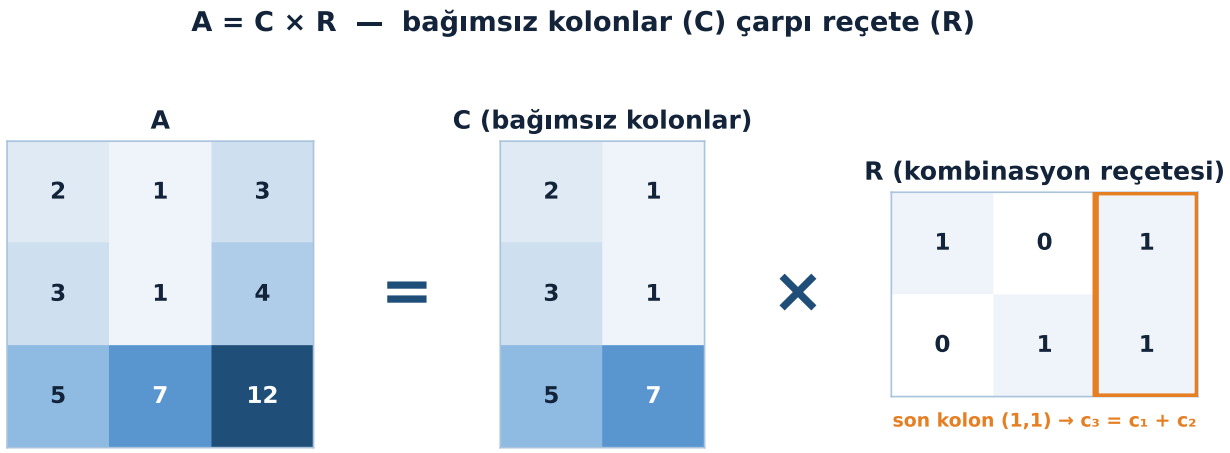
Birleştirince $A = CR$:

$$A = CR = \begin{pmatrix} 2 & 1 \\ 3 & 1 \\ 5 & 7 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 3 \\ 3 & 1 & 4 \\ 5 & 7 & 12 \end{pmatrix}$$

> *"this is a first matrix factorization. It's not– well, it is a famous one, actually."* — Strang, 21:04

Bu eski bir öğretim aracı gibi görünse de, modern büyük ölçekli sayısal lineer cebirde C×R (gerçek kolonlar çarpı gerçek satırlar) ciddi bir araç haline geldi. Şekil 8.5 bu ayrışmayı A, C ve R ısı haritalarıyla yan yana gösteriyor.

> *"C times R, columns times rows has become very, very important in large scale [numerical linear algebra]."* — Strang, 21:17



Şekil 8.5: A = C × R faktorizasyonu: A'nın bağımsız kolonları (C, 3×2) çarpı kombinasyon reçetesi (R, 2×3). R'nin birim bloğu bağımsız kolonları kopyalar; turuncu çerçeveli son kolon (1,1) bağımlı kolonu kurar ($c_3 = c_1 + c_2$).

> **Builder Notu — Tek Cümlede Lineer Cebir**
>
> $A = CR$, **yorumlanabilir düşük-rank** temsilidir: C'nin kolonları A'nın gerçek kolonlarıdır (örneğin gerçek hastalar, gerçek kelimeler), R ise geri kalan her şeyin bu temsilcilerden nasıl kurulacağının reçetesidir. Bu, CUR ayrışımının ve öneri sistemlerinde/genomikte tercih edilen "örnek-tabanlı" sıkıştırmanın çekirdeğidir.

8.7 İlk Büyük Teorem — Kolon Rank = Satır Rank

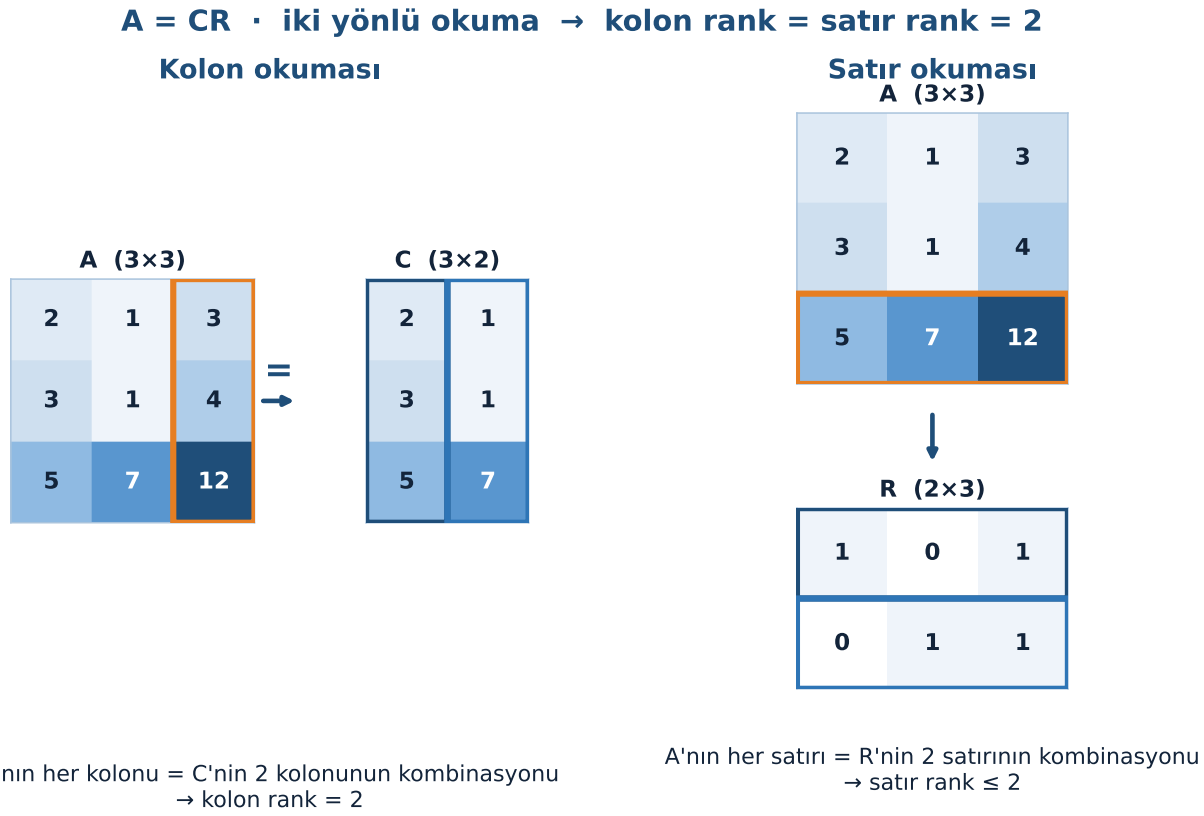
Strang burada heyecanlanıyor: $A = CR$ yazarken farkında olmadan lineer cebirin **ilk büyük teoremini** ispatlamış oluyoruz.

> *"[sitting here in front of me was] the first great theorem in linear algebra, the fact that the column rank equals the row rank."* — Strang, 23:44

Bizim A'mızda iki bağımsız kolon var (kolon rank = 2). Teorem diyor ki satır rank da 2 olmalı — yani satırlardan biri diğer ikisinin bir kombinasyonu. Matris kare ve tersinir değil (kolonlar bağımlı), o yüzden satırlar da bağımlı olmak zorunda.

> *"I believe that a combination of the rows gives 0."* — Strang, 24:10

İlk bakışta hangi satır kombinasyonunun sıfır verdiğini görmek zor — ama teorem garanti ediyor: bir matrisin bağımsız kolon sayısı, bağımsız satır sayısına **her zaman** eşittir. Neden eşit olduğunu bir sonraki bölümde $A = CR$'yi iki yönlü okuyarak göreceğiz. Şekil 8.6 bu iki yönlü okumayı — kolon okuması ve satır okuması — yan yana özetliyor.



Şekil 8.6: A = CR'nin iki yönlü okunması: kolon okuması (A'nın kolonları C'nin 2 kolonundan) ile satır okuması (A'nın satırları R'nin 2 satırından) aynı sonucu verir → kolon rank = satır rank = 2.

> Builder Notu — İki Yol, Aynı Sayı
>
> Rank, bir matrisin gerçek **serbestlik derecesi**dir: m×n bir matris, rankı r ise aslında yalnızca $r(m+n)$ civarı sayı içerir, mn değil. ML'de ağırlık ve veri matrislerinin "etkin rankı" ne kadar sıkıştırılabilir olduklarını belirler — düşük etkin rank, LoRA ve model budama (pruning) için açık kapıdır.

8.8 Satır Uzayı = C(A^T)

Kolon uzayı kolonların kombinasyonlarıydı. **Satır uzayı** ise satırların tüm kombinasyonlarıdır. Strang yeni bir harf icat etmek yerine zarif bir numara yapıyor: satırları transpoz ile kolon hâline getir, sonra eski tanıma geri dön.

$$\text{row space}(A) = C(A^T)$$

Yani satır uzayı = A^T'nin kolon uzayı. Böylece vektörleri hep kolon vektörü tutma alışkanlığını (NumPy, Julia, MATLAB konvansiyonu) bozmadan devam ediyoruz.

> *"So the row space of A is the column space of A transpose."* — Strang, 27:25

Strang araya önemli bir not düşüyor: bizim örneğimiz 3×3 kare olduğu için hem kolon uzayı hem satır uzayı $\mathbb{R}^3$'ün parçası. Ama gerçek veri kare gelmez — kolonlar örnekler (hastalar), satırlar öznitelikler (hastalıklar) olabilir; ikisi tamamen farklı boyutlu uzaylardadır.

> Builder Notu — Örnekler ve Öznitelikler
>
> Bir veri matrisinde kolon uzayı "örneklerin gerebildiği yön kümesi", satır uzayı ise "özniteliklerin gerebildiği yön kümesi"dir. PCA, kovaryans matrisi A^TA üzerinden tam da satır/kolon uzaylarının baskın yönlerini bulur. Kolon ve satır uzaylarının boyutunun eşit olması (rank), veri matrisinin gerçek bilgi içeriğinin tek bir sayıyla ölçülebilmesini sağlar.

8.9 İspat — CR'yi İki Yönlü Okumak

Kolon rank = satır rank neden doğru? İspat, $A = CR$ çarpımına **iki ayrı gözle** bakmakta saklı.

> *"The proof is really to look at this multiplication, C times R, two ways."* — Strang, 32:54

Birinci okuma (kolon kombinasyonu): A'nın her kolonu, C'nin kolonlarının R tarafından verilen kombinasyonudur. C'nin r tane kolonu var → A'nın kolonları r boyutlu bir uzayı gerer → kolon rank = r.

İkinci okuma (satır kombinasyonu): Aynı çarpım, A'nın her satırının R'nin satırlarının (C'deki sayılarla ağırlıklı) kombinasyonu olduğunu söyler. R'nin r tane satırı var → A'nın satırları en çok r boyutlu bir uzayı gerer → satır rank ≤ r.

İki okuma birlikte: kolon rank = r ve satır rank ≤ r; simetri ile (aynı argümanı A^T'ye uygula) satır rank = r. Demek ki **kolon rank = satır rank**. Tek bir faktorizasyon, teoremi açığa çıkarıyor.

$$A = CR \quad \Rightarrow \quad \text{col rank} = \text{row rank} = r$$

> Builder Notu — Bir Çarpım, İki Hikâye
>
> "Aynı nesneyi iki yönlü oku" tekniği, ML matematiğinde tekrar tekrar karşına çıkar: bir Gram matrisini hem örnekler hem öznitelikler üzerinden okumak (kernel trick), bir dış çarpımı hem ileri hem geri geçişte kullanmak (backprop). Bir çarpımın yapısını iki perspektiften görebilmek, çoğu türetmenin anahtarıdır.

8.10 rref ve CUR Bağlantısı

Bulduğumuz R aslında tanıdık bir nesne: A'nın **indirgenmiş satır echelon formu** (row reduced echelon form, rref). İçinde birim matris bloğu ve kalan kolonların doğru kombinasyonları oturur.

> *"It's called the row reduced echelon form of the matrix, and it's a big goal in 18.06."* — Strang, 38:49

Yani Phase 1 18.06'nın büyük hedeflerinden biri (rref), burada $A = CR$ faktorizasyonunun R parçası olarak yeniden karşımıza çıkıyor.

Strang bir alternatif daha gösteriyor: hem C'ye gerçek kolonları, hem de R'ye A'nın gerçek **satırlarını** koyarsan, ortada her şeyi düzelten küçük bir U matrisi gerekir. Bu, CUR faktorizasyonudur:

$$A = C\,U\,R$$

> *"[I] wrote a page about CUR."* — Strang, 40:24

> Builder Notu — Pratik vs Klasik
>
> CUR, SVD'ye yorumlanabilir bir alternatiftir: C ve R gerçek veri kolonları/satırları olduğundan (soyut tekil vektörler değil), sonuç insan-okunabilir kalır. Genomik, öneri sistemleri ve büyük seyrek matrislerde, "hangi gerçek özellikler önemli?" sorusuna SVD'den daha doğrudan cevap verir.

8.11 Dev Matrisleri Örnekleme

Strang dersin sonuna doğru ileriye bir köprü atıyor. Diyelim ki matrisin boyutu $10^5 \times 10^5$ — hızlı belleğe sığmaz, tüm girdilerle uğraşmak imkânsız. Ne yaparsın? **Örneklersin.**

> *"if you have a giant matrix, like size 10 to the 5th… you sample it."* — Strang, 35:43

Doğal fikir: rastgele bir x vektörü seç, Ax'e bak. Ax nerede yaşar?

> *"Ax is in the column space."* — Strang, 37:02

Yani rastgele tek tek kolon seçmek yerine, rastgele x ile **kolonların bir karışımını** (Ax) alırsın; 100 rastgele x, kolon uzayının neye benzediği hakkında çoğu zaman yeterli fikir verir. Strang araya lineer cebirin altın kuralını sıkıştırıyor: $ABCx$ çarpımı da A'nın kolon uzayındadır, çünkü o "A çarpı bir şey"dir.

> *"Putting parentheses in the right place is the key to linear algebra."* — Strang, 38:14

> Builder Notu — Petabyte'tan Megabyte'a
>
> "Rastgele x ile Ax al" fikri, doğrudan Ders 13'teki **rastlantısal lineer cebire** (randomized SVD, matris taslaklama/sketching) açılır. Modern büyük-ölçek ML'de dev matrislerin tam SVD'si yerine rastgele projeksiyonlarla düşük-rank yaklaşımı hesaplanır — temelinde tam bu "örnekle ve kolon uzayını yakala" sezgisi yatar.

8.12 Matris × Matris = Dış Çarpımlar Toplamı

Aynı "kolon görüşü"nü matris-matris çarpımına taşı. Alışılmış yol: AB'nin (i,j) girdisi, A'nın i. satırı ile B'nin j. kolonunun nokta çarpımı — başlangıç için iyi, ama yüzeysel.

> *"the deeper way is columns times row."* — Strang, 46:36

Derin yol: AB'yi, A'nın kolonları çarpı B'nin satırlarının toplamı olarak gör. A'nın k. kolonu (m×1) çarpı B'nin k. satırı (1×p), bir rank-1 matris (m×p) verir; bunları k boyunca topla:

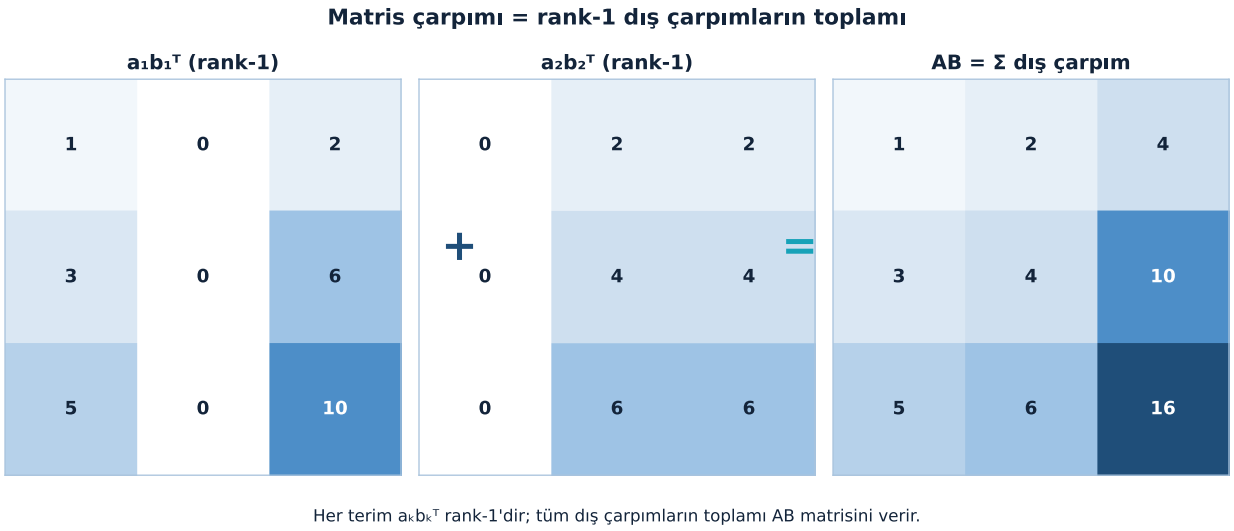
$$AB = \sum_{k=1}^{n} a_k \, b_k^T$$

Burada a_k = A'nın k. kolonu, b_k^T = B'nin k. satırı. Her terim bir rank-1 dış çarpım; AB bu dış çarpımların toplamı.

> *"So it's a sum of outer products."* — Strang, 48:48

Bu, Bölüm 1'deki Ax fikrinin matrise genellemesidir — ve rank-1 yapı taşı temasının doğal devamı. Şekil 8.7 bu toplamı rank-1 katmanlar olarak gösteriyor: $a_1 b_1^T + a_2 b_2^T = AB$.

> Builder Notu — Tek Matrisi N Katmanda Görmek
>
> Dış-çarpım görüşü ML'in her yerinde: attention skoru QK^T bir dış çarpımlar toplamıdır; LoRA güncellemesi $\Delta W = BA$ tam da düşük sayıda dış çarpımdır; optimizer'larda rank-1 güncellemeler (örneğin BFGS) bu yapıyı kullanır. "Çarpım = dış çarpımlar toplamı" görüşü, hem matematiği hem bellek erişim desenini (GPU'da) anlamanın anahtarıdır.



Şekil 8.7: Matris çarpımı dış çarpımlarla: AB, rank-1 katmanların ($a_1 b_1{}^T + a_2 b_2{}^T$) toplamıdır. Ortak renk ölçeği katmanlar ile sonucu kıyaslanabilir kılar.

8.13 Çarpım Maliyeti — mnp

Strang son bir dakikayı maliyete ayırıyor. m×n matris A çarpı n×p matris B kaç çarpma gerektirir?

Eski yol (nokta çarpımı): her girdi için n çarpma, ve sonuç m×p boyutunda → toplam $n \cdot (mp)$ çarpma.

Yeni yol (dış çarpım): her dış çarpım mp çarpma, ve n tane dış çarpım var → toplam $(mp) \cdot n$ çarpma.

$$n \cdot (mp) \;=\; mnp \;=\; (mp) \cdot n$$

İki yol da aynı sayıya, mnp'ye varır — hatta birebir aynı çarpmaları, sadece farklı sırada yapar.

> *"they're exactly the same multiplications, just a different order."* — Strang, 51:56

> **Builder Notu — Hızı Yön Belirler**
>
> mnp, matris çarpımının **FLOP sayısıdır** — ve derin öğrenmenin maliyetinin neredeyse tamamı buradan gelir. GPU'ların TFLOPS değerleri tam da bu mnp'yi ne kadar hızlı yaptığını ölçer. Eğitim/çıkarım bütçesini tahmin etmek, hangi katmanın pahalı olduğunu görmek ve "aynı çarpmalar farklı sıra" (yani çarpımı yeniden sıralayıp bellek/önbellek verimini artırma — GEMM tiling) hepsi bu tek sayının üstünde durur.

8.14 Bu Dersin Özeti

1. **Ax = kolonların kombinasyonu** — matris-vektör çarpımının "doğru" görüşü; satır-nokta-çarpımı değil.
2. **Kolon uzayı** $C(A)$ — tüm Ax'lerin kümesi; A'nın ulaşabildiği uzay.

3. **Rank** = kolon uzayının boyutu = bağımsız kolon sayısı.
4. **Rank-1 matris = kolon × satır** (dış çarpım) — lineer cebirin ve veri biliminin yapı taşı.
5. **A = CR faktorizasyonu** — C bağımsız kolonları, R kombinasyon reçetesini (rref) taşır.
6. **Kolon rank = satır rank** — ilk büyük teorem; $A = CR$'yi iki yönlü okuyarak kanıtlanır.
7. **Satır uzayı =** $C(A^T)$ — yeni bir harf icat etmeye gerek yok.
8. **AB = dış çarpımlar toplamı**, maliyeti mnp; dev matrisler rastgele x ile (Ax) örneklenir.

! Tek Bir Cümle

$A = CR$ — her matris, bağımsız kolonlarını (C) çarpı o kolonlardan geri kalan kolonları kuran reçeteyi (R) içerir; kolon rank = satır rank tam da bu yüzden doğrudur.

8.15 Kontrol Soruları

i Soru 1: Aşağıdaki matrisin A = CR faktorizasyonunu bul (C bağımsız kolonlar, R kombinasyon reçetesi).

$$A = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 0 & 1 & 1 \end{pmatrix}$$

Cevap:
Kolon 1 = (1,2,0) ve kolon 2 = (0,1,1) bağımsız. Kolon 3 = (1,3,1) = kolon 1 + kolon 2 → bağımlı, alınmaz.

$$C = \begin{pmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Doğrula: C·R = A ✓. Rank 2 (iki bağımsız kolon, kolon uzayı bir düzlem).

i Soru 2: Bu matriste kolon rank'ı ve satır rank'ı ayrı ayrı bul. Eşit mi çıkıyor?

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{pmatrix}$$

Cevap:
Kolon rank: ikinci kolon (2,4,6) = 2·(birinci kolon). Tek bağımsız kolon → kolon rank = 1.
Satır rank: tüm satırlar (1,2)'nin katı — (1,2), (2,4) = 2·(1,2), (3,6) = 3·(1,2). Tek bağımsız satır → satır rank = 1.
Eşit: kolon rank = satır rank = 1 ✓. (Genel kural: rank ≤ min(m,n); burada üst sınır min(3,2) = 2, gerçek rank 1.)

Soru 3: Aşağıdaki rank-1 matrisi bir kolon vektörü çarpı bir satır vektörü (dış çarpım) olarak yaz.

$$A = \begin{pmatrix} 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix}$$

Cevap:
Tüm kolonlar (2,3)'ün katı, tüm satırlar (1,2,3)'ün katı. Demek ki:

$$A = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$

2×1 çarpı 1×3 = 2×3, rank 1. Bu, SVD'nin tek bir rank-1 parçasının ta kendisidir (Ders 6-7'nin habercisi).

Soru 4: Bir m×n ağırlık matrisi rank r ise onu kaç sayıyla saklarsın? Bu neden LoRA'nın temeli?

Cevap:
Rank r matris $A = CR$ biçiminde yazılır: C boyutu m×r, R boyutu r×n. Toplam saklanan sayı $m \cdot r + r \cdot n = r(m + n)$.
Tam matris mn sayı tutar. $r \ll \min(m, n)$ olduğunda $r(m + n) \ll mn$ — büyük tasarruf.
Örnek: m = n = 1000, r = 8 → tam matris 10^6 sayı; düşük-rank temsil 8·2000 = 16.000 sayı (≈60× az).
LoRA tam da bunu yapar: ağırlık güncellemesini $\Delta W = BA$ (düşük-rank) olarak saklar ve dev modeli çok az parametreyle fine-tune eder. "Rank = gerçek serbestlik derecesi" sezgisinin doğrudan uygulamasıdır.

8.16 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. Aşağıdaki 3×4 matrisin $A = CR$ faktorizasyonunu bul. Bağımsız kolonları soldan sağa seç, sonra R'yi kur.

$$A = \begin{pmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 3 \\ 1 & 1 & 3 & 4 \end{pmatrix}$$

Egzersiz 2. İki aşamalı: (a) Aşağıdaki matrisin rankını bul. (b) Kolon uzayı için bir baz yaz. (c) $A = CR$'deki R matrisini kur.

$$A = \begin{pmatrix} 2 & 4 & 1 \\ 1 & 2 & 0 \\ 3 & 6 & 2 \end{pmatrix}$$

Egzersiz 3. Yapısal soru: Bir matrisin tüm kolonları bağımsızsa (rank = kolon sayısı), $A = CR$ faktorizasyonundaki R ne olur? Bir de: A'nın bir kolonu tamamen sıfırsa, o kolon C'ye girer mi, R'de nasıl görünür?

Egzersiz 4. Python ile doğrula:

```python
import numpy as np

A = np.array([[1, 0, 2, 1],
              [0, 1, 1, 3],
              [1, 1, 3, 4]], dtype=float)

print("rank:", np.linalg.matrix_rank(A))   # bağımsız kolon sayısı

# Rastgele x ile Ax her zaman kolon uzayında:
x = np.random.randn(4)
b = A @ x
print("Ax (kolon uzayında bir vektör):", b)

# 3. ve 4. kolonun ilk ikisinden kuruluşunu kontrol et:
c1, c2 = A[:, 0], A[:, 1]
print("2*c1 + c2 == kolon 3 ?", np.allclose(2*c1 + c2, A[:, 2]))
print("c1 + 3*c2 == kolon 4 ?", np.allclose(c1 + 3*c2, A[:, 3]))
```

Egzersiz 5. (Ders 2 habercisi.) Strang Ders 2'de matrisin **beş büyük faktorizasyonunu** tanıtıyor: A = LU (eliminasyon), A = QR (ortonormallik), $A = Q\Lambda Q^T$ (özdeğerler), $A = U\Sigma V^T$ (SVD) ve bu derste gördüğümüz A = CR (rank). Aşağıdaki matris için eliminasyon yapıp U üst üçgensel formunu bul; sonra $A = CR$ ile A = LU'nun **neyi açığa çıkardığını** (biri rankı, diğeri eliminasyon adımlarını) bir cümleyle karşılaştır.

$$A = \begin{pmatrix} 2 & 1 \\ 6 & 8 \end{pmatrix}$$

8.17 Sonraki Ders İçin Hazırlık

Ders 2: Matrisleri Çarpmak ve Çarpanlara Ayırmak

Ders 1'de tek bir faktorizasyon ($A = CR$) ve "matrise bütün olarak bakma" fikrini gördük. Ders 2 bunu genişletiyor: matris çarpımının farklı görünüşleri ve matrisin **beş temel faktorizasyonu**.

- Matris çarpımının dört farklı görünüşü (özellikle kolon × satır / dış çarpım)
- Beş büyük faktorizasyon: **LU, QR,** $Q\Lambda Q^T$**, SVD, CR**
- Her faktorizasyonun hangi soruyu cevapladığı (eliminasyon, ortonormallik, özdeğer, tekil değer, rank)

> ⚠ Ders 2 Öncesi Yapılacak
>
> - Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (LU önizlemesi).
> - Python'da birkaç matrisin rankını `np.linalg.matrix_rank` ile bul; rank-1 ve rank-2 matrisleri gözle ayırt et.
> - Ana cümleyi tekrar oku: "A = CR — bağımsız kolonlar çarpı kombinasyon reçetesi; kolon rank = satır rank."

8.18 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
Ax = kolon kombinasyonu	Ax, A'nın kolonlarının x ile ağırlıklı toplamıdır	7m32
Kolon uzayı $C(A)$	Tüm Ax'lerin kümesi; kolonların gerdiği uzay	8m17
Rank	Kolon uzayının boyutu, yani bağımsız kolon sayısı	13m56
Rank-1 = kolon × satır	Bir kolon çarpı bir satır; dış çarpım, yapı taşı	13m56
Baz	Bağımsız kolonlar; soldan sağa seçilir	15m51
A = CR	C bağımsız kolonlar, R kombinasyon reçetesi	21m04
Kolon rank = satır rank	İlk büyük teorem; bağımsız kolon sayısı = bağımsız satır sayısı	23m44
Satır uzayı = $C(A^T)$	Satırların kombinasyonları; transpozun kolon uzayı	27m25
CR iki yönlü okuma	Kolon ve satır rankının eşitliğinin ispatı	32m54
R = rref	R, indirgenmiş satır echelon formudur; 18.06'nın hedefi	38m49
CUR	Gerçek kolonlar C, gerçek satırlar R, orta matris U	40m24
Dev matris örnekleme	Rastgele x ile Ax kolon uzayını yoklar (randomized LA)	35m43
AB = dış çarpımlar toplamı	AB, kolon$_k$ çarpı satır$_k$ rank-1 parçaların toplamı	48m48
Maliyet mnp	m×n çarpı n×p çarpımının çarpma sayısı	51m56

8.19 ML Bağlantıları Özeti

1. **Ax = kolon kombinasyonu** → GEMM/BLAS matris çarpımını kolon ve blok kombinasyonları olarak yapar; performans sezgisinin temeli.
2. **Rank-1 = dış çarpım** → SVD, PCA ve LoRA hepsi rank-1 parçaların toplamı üstünde durur.
3. **A = CR / düşük-rank** → yorumlanabilir sıkıştırma; CUR, öneri sistemleri, genomik gerçek kolon/satır seçer.
4. **Rank = serbestlik derecesi** → ağırlık matrislerinin etkin rankı, model budama ve sıkıştırılabilirliği belirler.
5. **Kolon uzayı** $C(A)$ → $Ax = b$ ne zaman çözülür (b $\in C(A)$); değilse least squares projeksiyonu (Ders 9-11).

6. **Dev matris örnekleme (rastgele x)** → randomized SVD, matris taslaklama/sketching (Ders 13).
7. **AB = dış çarpımlar, maliyet** mnp → GPU TFLOPS, attention QK^T, eğitim/çıkarım maliyetinin fiziksel limiti.

! Tek bir şey alıp gideceksen

Bir matrise kolonları üstünden bak. Ax kolonların kombinasyonudur, $C(A)$ onların gerdiği uzaydır, rank o uzayın boyutudur — ve $A = CR$ bu yapıyı açığa çıkarıp kolon rank = satır rank teoremini bedavaya verir. Sıkıştırmadan rastlantısal cebire, LoRA'dan GPU'daki GEMM'e kadar tüm veri-odaklı lineer cebir bu tek görüşün üstünde durur.

9 Beş Faktorizasyon — Matrisleri Çarpmak ve Çarpanlara Ayırmak

Her faktorizasyon farklı bir soruyu cevaplar

> **Bölüm bilgisi**
>
> - **Video:** Multiplying and Factoring Matrices — Gilbert Strang, MIT 18.065
> - **OCW:** Lecture 2 — Multiplying and Factoring Matrices
> - **Okuma süresi:** ≈ 35 dk
> - **Önkoşul:** 18.06 (lineer cebir) + Ders 1 ($A = CR$ ve kolon görüşü)

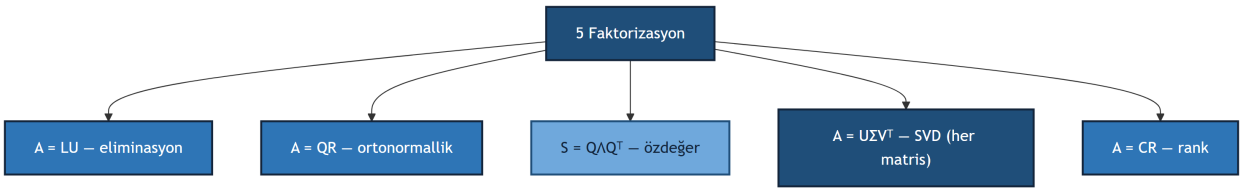
9.1 Bu Derste Ne Var?

Ders 1'de $A = CR$'yi ve "matrise kolonları üstünden bakma" fikrini gördük. Ders 2 bunu beş büyük faktorizasyona genişletiyor ve matris çarpımının "kolon × satır" görüşünü derinleştiriyor; sonunda lineer cebirin **temel teoremi** (dört alt-uzay) geliyor.

Üç temel fikir:

1. **Beş faktorizasyon:** $A = LU$, $A = QR$, $S = Q\Lambda Q^T$, $A = U\Sigma V^T$, $A = CR$ — her biri bir soruyu cevaplar.
2. **Spektral teorem:** simetrik S, rank-1 parçaların toplamıdır ($S = \sum_k \lambda_k q_k q_k^T$); matris çarpımı = dış çarpımlar toplamı temasının devamı.
3. **Dört temel alt-uzay:** $C(A)$, $C(A^T)$, $N(A)$, $N(A^T)$ — boyutları r, r, $n-r$, $m-r$.

Beş faktorizasyonun haritası Şekil 9.1'de: her dal bir matris ayrışımı, her ayrışım bir soru.



Şekil 9.1: Ders 2 kavram haritası: beş faktorizasyon ve her birinin merkezî fikri.

> *"…like LU or QR, would be the most used MATLAB commands in linear algebra."* — Strang, 1:29

Builder Notu — Faktorizasyon Haritası

- **Faktorizasyon = doğru soruyu sormak:** LU eliminasyonu, QR ortonormalliği, $Q\Lambda Q^T$ özdeğerleri, $U\Sigma V^T$ tekil değerleri, CR rankı açığa çıkarır. NumPy ve PyTorch'un lineer cebir modülleri bu beşinin üstüne kuruludur.
- **Spektral teorem → PCA:** kovaryans matrisi simetriktir; $S = \sum_k \lambda_k q_k q_k^T$ ayrışımı, ana bileşenlerin (en büyük λ'lar) tam tanımıdır.
- **SVD = her matris için:** dikdörtgen ağırlık matrisleri özdeğer ayrışımına sahip olmayabilir, ama SVD'si her zaman vardır — düşük-rank sıkıştırma, LoRA, PCA hepsi buradan.
- **Dört alt-uzay → çözülebilirlik:** $Ax = b$ ne zaman çözülür ($b \in C(A)$), çözüm ne zaman tektir ($N(A) = \{0\}$) — least squares ve regularizasyonun geometrik temeli.

Tek cümle: bir matrisi "hangi faktorizasyon hangi soruyu cevaplıyor" diye görmek, sayısal lineer cebirin ve ML araç kutusunun haritasıdır.

9.2 Beş Büyük Faktorizasyon

Strang dersi, bir matrisin beş temel çarpanlamasını sıralayarak açıyor — her biri farklı bir soruyu cevaplar:

$$A = LU, \quad A = QR, \quad S = Q\Lambda Q^T, \quad A = U\Sigma V^T, \quad A = CR$$

- $A = LU$ — eliminasyon (alt üçgensel çarpı üst üçgensel)
- $A = QR$ — ortonormallik (Gram-Schmidt; least squares)
- $S = Q\Lambda Q^T$ — simetrik matrisin özdeğer ayrışımı (spektral teorem)
- $A = U\Sigma V^T$ — tekil değer ayrışımı (SVD); her matris için
- $A = CR$ — rank (Ders 1)

> *"I want to illustrate that by the five key factorizations of matrices."* — Strang, 1:12

Builder Notu — numpy.linalg'in İskeleti

Bu beş satır, sayısal lineer cebirin tüm haritasıdır: `numpy.linalg` ve `torch.linalg` modüllerindeki her rutin (solve, lstsq, eig, svd, qr) bunlardan birine dayanır. Hangi problemde hangi faktorizasyonu çağıracağını bilmek, ML mühendisliğinin sessiz ama kritik bir becerisidir.

9.3 $A = LU$ — Eliminasyon

İlk faktorizasyon en tanıdık olanı: $Ax = b$ çözerken yaptığın satır işlemleri. Strang'in vurgusu, bu işlemlerin tam olarak bir matris çarpımına denk geldiği:

> *"all those row operations that you do are perfectly expressed by L times U."* — Strang, 17:20

L alt üçgensel (köşegende 1'ler, altında eliminasyon çarpanları), U üst üçgensel (köşegende pivotlar):

$$A = LU$$

Bu, Phase 1 18.06 Ders 4'ün tam konusudur. 18.065'te aynı çarpanlamayı yeni bir gözle — kolon çarpı satır, yani rank-1 soyma olarak — Bölüm 8'de yeniden göreceğiz.

> Builder Notu — solve() Ne Yapar?
>
> `numpy.linalg.solve(A, b)` arka planda $A = LU$ yapar (LAPACK getrf), sonra iki üçgensel sistemi çözer. Aynı A ile çok sayıda b çözeceksen LU'yu bir kez hesaplayıp sakla: n^3 yerine her b için n^2 maliyet.

9.4 $A = QR$ — Ortonormallik

İkinci faktorizasyon Q'nun ortonormal kolonlarına dayanır: kolonlar birbirine dik ve birim uzunlukta, yani $Q^TQ = I$.

> *"Orthogonal. The columns are orthogonal. Often orthonormal."* — Strang, 2:31

$$A = QR$$

Q ortonormal kolonları, R üst üçgensel matrisi taşır. Üretici algoritma Gram-Schmidt'tir (Phase 1 18.06 Ders 17). En büyük uygulaması least squares: normal denklemleri (A^TA) doğrudan çözmek yerine QR kullanmak sayısal olarak çok daha kararlıdır.

Gram-Schmidt'in bağımsız kolonları nasıl ortonormal Q'ya dönüştürdüğü Şekil 9.2'te görülüyor: ham kolonlardan izdüşüm çıkarılıp normalleştirilir, sonuç dik birim vektörlerdir.

> Builder Notu — Kararlı Yol
>
> QR, least squares'in (Ders 9) güvenilir yoludur. Derin öğrenmede ortonormal matrisler gradyan patlamasını/sönmesini önler (orthogonal initialization, ortonormal RNN ağırlıkları); QR ayrıca modern özdeğer algoritmalarının (QR iteration, Ders 12) çekirdeğidir.

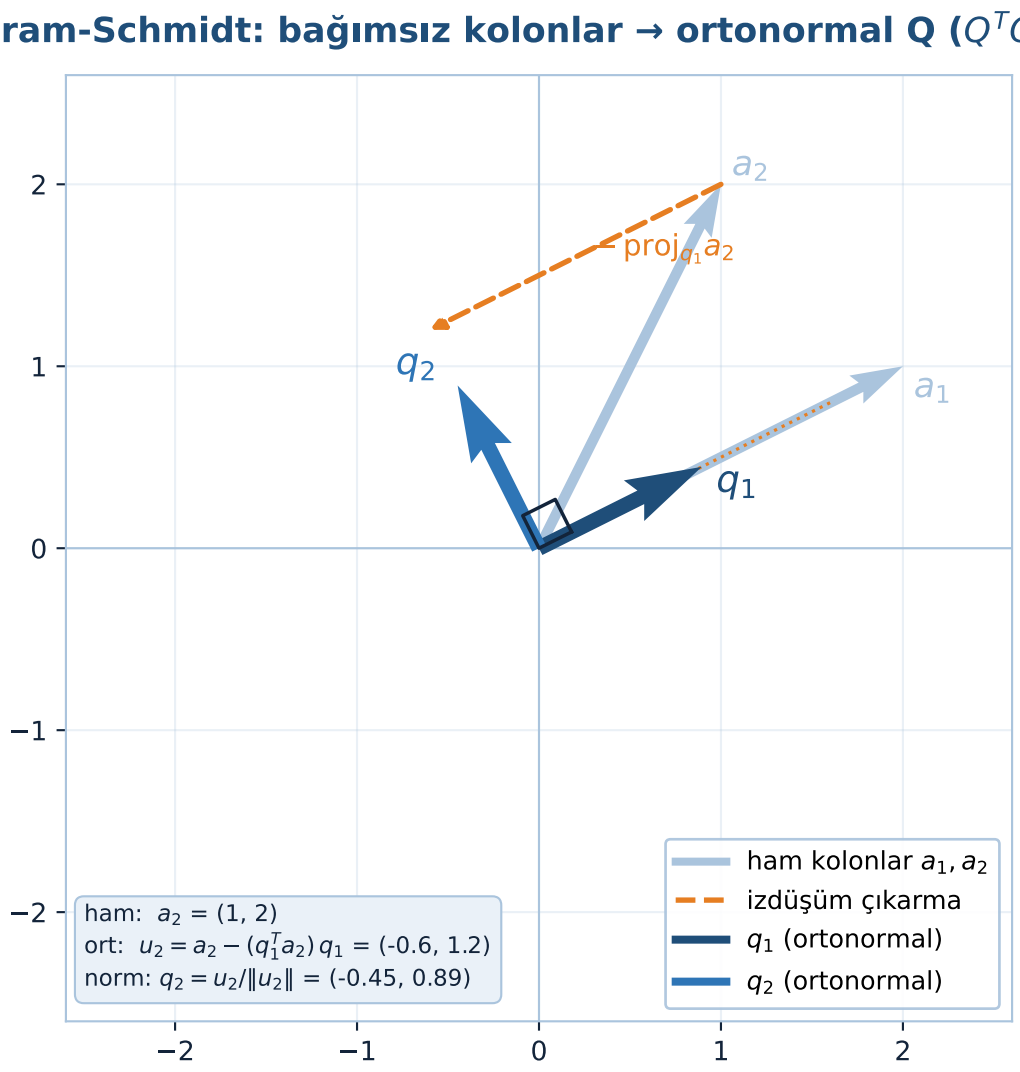
9.5 $S = Q\Lambda Q^T$ — Simetrik Matrisler ve Spektral Teorem

Üçüncü faktorizasyon simetrik matrisler içindir ($S = S^T$) ve Strang'e göre lineer cebirin en güzel sonucu:

$$S = Q\Lambda Q^T$$

Burada Λ köşegen özdeğer matrisi, Q ise özvektörleri taşır. İki olağanüstü özellik:

- **Özvektörler ortonormaldir** — Q ortogonal matris, $Q^TQ = I$.



Şekil 9.2: Gram-Schmidt ortogonalleştirme (A = QR). Bağımsız kolonlar a_1=(2,1) ve a_2=(1,2) (soluk) ortonormal q_1, q_2'ye dönüşür: $q_1 = a_1/\|a_1\|$ (navy); a_2'den q_1 üzerine izdüşümü çıkarıp normalleştirince $q_2 = (a_2 - \text{proj}_k\, a_2)/\|\cdot\|$ (çelik). Turuncu kesikli ok izdüşüm-çıkarmayı, orijindeki küçük kare $q_1 \perp q_2$ dikliğini ($Q^TQ = I$) gösterir.

- **Özdeğerler gerçektir** — λ'lar reel sayı.

  "Orthogonal matrices are the queens, and symmetric matrices are the kings." — Strang, 4:59
  "They're all real. So eigenvalues are real." — Strang, 5:35

> Builder Notu — Simetrik Krallar
>
> Simetrik matrisler ML'de her yerde: kovaryans matrisi, Gram matrisi (kernel), Hessian. Hepsi gerçek özdeğerli ve ortonormal özvektörlü — bu yüzden PCA, kernel yöntemleri ve ikinci-derece optimizasyon iyi tanımlıdır.

9.6 Spektral Teorem — Rank-1 Parçaların Toplamı

Ders 1'in "çarpım = dış çarpımlar toplamı" teması burada simetrik matrise uygulanıyor. $S = Q\Lambda Q^T$'yi kolon çarpı satır olarak açarsak, S bir rank-1 parçalar toplamına dönüşür:

"this is a sum of rank 1." — Strang, 8:21

$$S = \lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T + \cdots + \lambda_n q_n q_n^T = \sum_{k=1}^{n} \lambda_k q_k q_k^T$$

Her parça $\lambda_k q_k q_k^T$ simetrik bir rank-1 matristir. Doğruluğunu S'yi q_1 ile çarparak görürüz: ortonormallik sayesinde $q_k^T q_1 = 0$ $(k \neq 1)$ ve $q_1^T q_1 = 1$, yani tüm terimler düşer, geriye $\lambda_1 q_1$ kalır:

$$Sq_1 = \lambda_1 q_1 (q_1^T q_1) + \lambda_2 q_2 (q_2^T q_1) + \cdots = \lambda_1 q_1$$

"This is called the spectral theorem." — Strang, 10:39

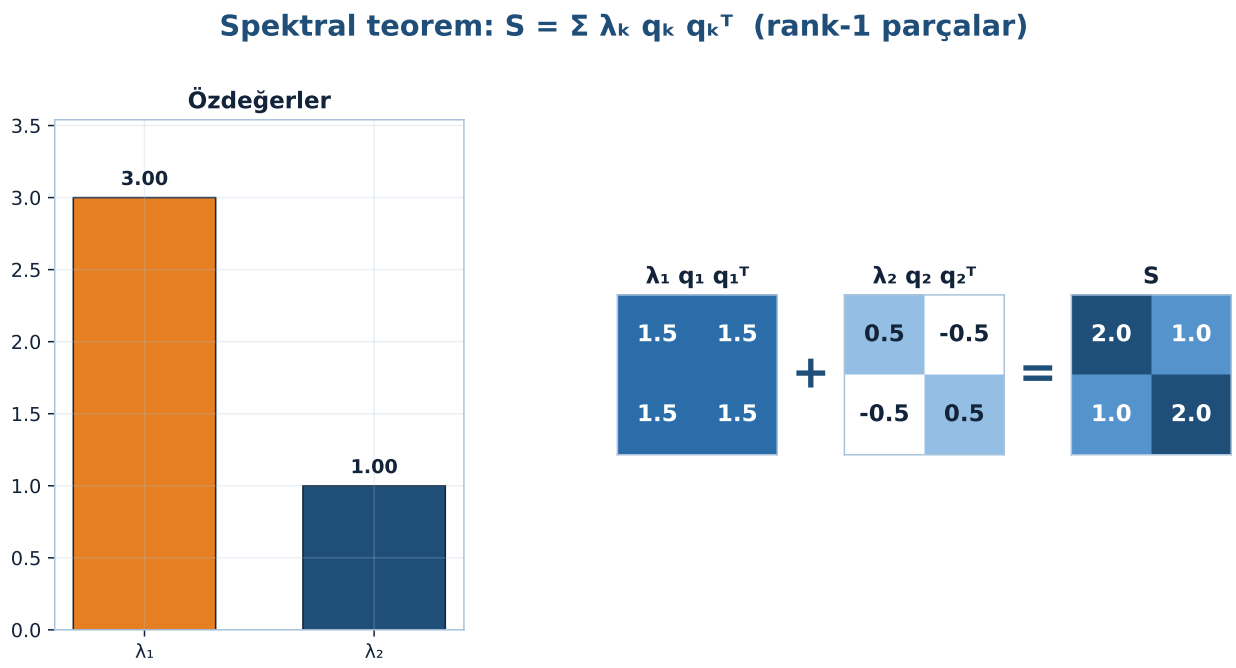
Spektral ayrışımın bileşenleri Şekil 9.3'de görselleştirilmiş: solda özdeğer çubukları (en büyüğü baskın), sağda rank-1 izdüşümler $\lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T = S$ olarak toplanıyor.

> Builder Notu — PCA'nın Ta Kendisi
>
> $S = \sum_k \lambda_k q_k q_k^T$ ayrışımı PCA'nın ta kendisidir: en büyük λ'lara karşılık gelen q_k yönleri verinin ana bileşenleridir. Birkaç büyük terimi tutmak = düşük-rank yaklaşım = boyut indirgeme. Aynı yapı Ders 7'de (Eckart-Young) SVD ile genelleşir.

9.7 $A = U\Sigma V^T$ — SVD, Her Matris İçin

Dördüncü faktorizasyon, kursun ve tüm veri biliminin temel taşı:

$$A = U\Sigma V^T$$



Şekil 9.3: Spektral teorem: simetrik S = QΛQ^{T}, özdeğer-ağırlıklı rank-1 izdüşümlerin toplamı olarak yazılır. Solda S = [[2,1],[1,2]] matrisinin özdeğerleri (λ_1=3 vurgulu, λ_2=1); sağda $\lambda_1 q_1 {q_1}^{T} + \lambda_2 q_2 {q_2}^{T}$ = S — her parça bir özyön boyunca rank-1 bir bileşen, en büyük özdeğer baskın.

U ve V iki **ayrı** ortogonal matris, Σ köşegen tekil değer matrisi. Özdeğer ayrışımının aksine SVD **dikdörtgen** matrisler için de çalışır.

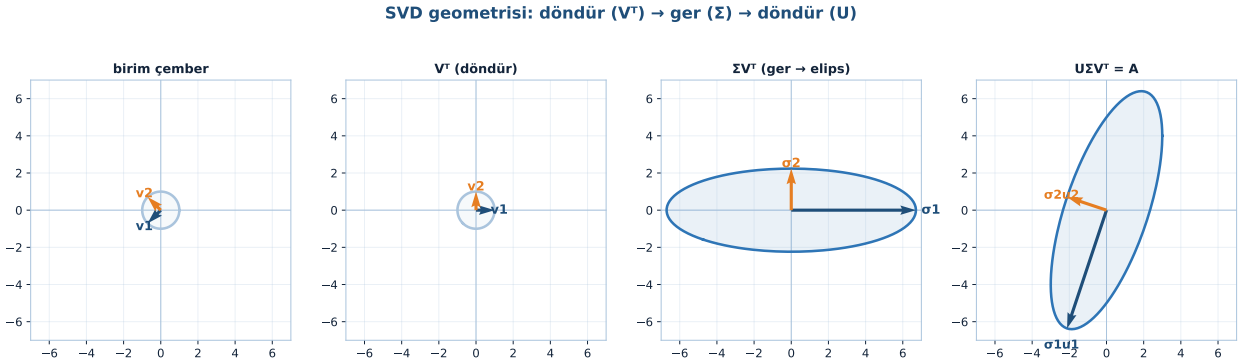
> *"it's going to be a foundational factorization [for this course and for all of data science]."* — Strang, 14:54 *"The point is it works for every matrix."* — Strang, 16:04

Simetrik özdeğer ayrışımı $Q\Lambda Q^T$ tek bir özvektör kümesi kullanır ve her matriste yeterli özvektör bulunmayabilir; SVD ise iki tekil vektör kümesi (U ve V) kullandığından bu sorun ortadan kalkar. Ayrıntısı Ders 6'da.

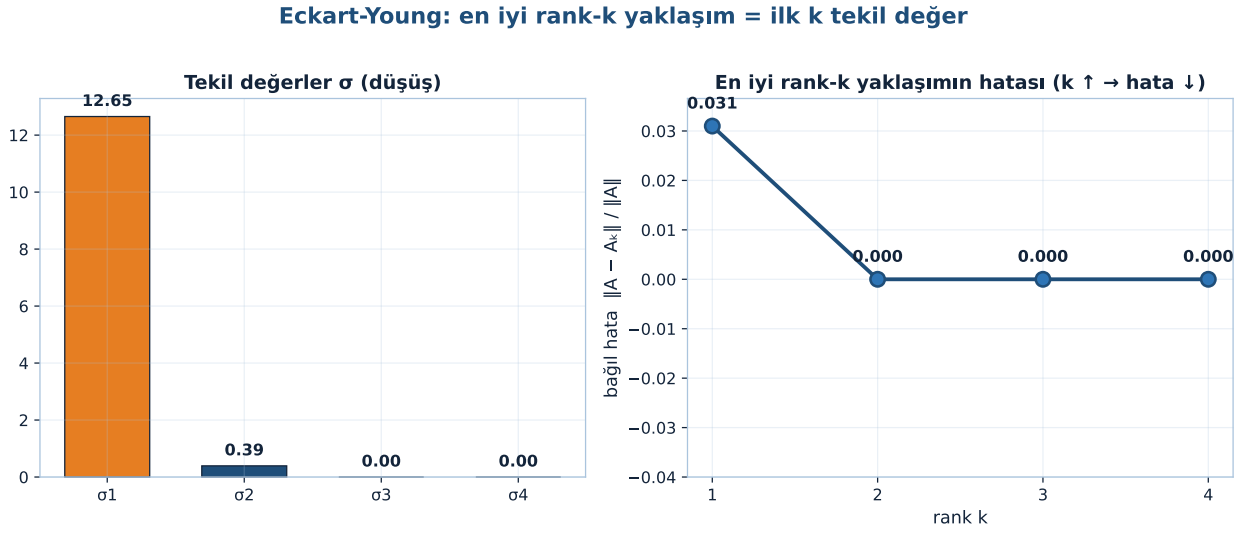
SVD'nin geometrisi Şekil 9.4'te üç adımla görülüyor: V^T döndürür, Σ gerer, U yeniden döndürür — birim çember eğik bir elipse dönüşür. Bu yapı en iyi düşük-rank yaklaşımı da verir; Şekil 9.5 tekil değer düşüşünü ve rank arttıkça hatanın düşmesini gösteriyor.

Builder Notu — Her Matrisin Hakkı

SVD, ağırlık matrislerinin sıkıştırılması (düşük-rank yaklaşım, LoRA), gürültü giderme, pseudoinverse ve PCA için kullanılır. "Her matrisin SVD'si vardır" garantisi, dikdörtgen ML ağırlıklarına (örneğin embedding katmanları) onu doğrudan uygulanabilir kılar.



Şekil 9.4: SVD geometrisi üç adımda: A = UΣV^T birim çemberi önce V^T ile döndürür (uzunluk korunur), sonra Σ ile eksenler boyunca gerer (eksen-hizalı elips, yarı-eksenler σ_1 ve σ_2), en son U ile yeniden döndürür. Sonuç A'nın çembere etkisi: eğik elips. v_1,v_2 sağ tekil vektörler (girdi eksenleri); $\sigma_1 u_1$,$\sigma_2 u_2$ çıktı tekil eksenleri (A=[[3,0],[4,5]], $\sigma_1 \approx 6.71$, $\sigma_2 \approx 2.24$).



Şekil 9.5: Eckart-Young düşük-rank yaklaşımı. Tekil değer düşüşlü 4×4 bir matriste σ hızla düşer (sol); en iyi rank-k yaklaşımın bağıl hatası $\|A - A_k\| / \|A\|$ k arttıkça düşer ve matrisin etkin rankında (burada k = 2) sıfıra iner (sağ). En iyi rank-k yaklaşım ilk k tekil değer/vektörden kurulur.

9.8 Matris Çarpımı = Kolon × Satır (Hatırlatma)

Spektral teoremde kullandığımız yapı taşı, Ders 1'in matris çarpımı görüşüdür: bir kolon çarpı bir satır, rank-1 bir matris verir.

$$\begin{pmatrix}1\\2\end{pmatrix}\begin{pmatrix}3 & 4\end{pmatrix}=\begin{pmatrix}3 & 4\\6 & 8\end{pmatrix}$$

Tüm kolonlar $(1,2)$'nin katı, tüm satırlar $(3,4)$'ün katı — rankı 1.

> *"Its rank is 1."* — Strang, 7:34 *"So those are the building blocks."* — Strang, 8:16

İki matrisin çarpımı AB, bu rank-1 yapı taşlarının (A'nın kolonları çarpı B'nin satırları) toplamıdır. Spektral teorem ve LU'nun rank-1 soyma görüşü, hep bu tek fikrin uygulamalarıdır.

> **Builder Notu — Ortak Dil**
>
> Kolon-çarpı-satır görüşü, GPU'da matris çarpımının (GEMM) ve attention'ın (QK^T) çalışma biçimidir. "Rank-1 yapı taşı" sezgisi SVD, PCA, LoRA ve spektral teoremin ortak dilidir.

9.9 $A = LU$'yu Rank-1 Soyarak Görmek

Strang LU'yu yeni bir gözle gösteriyor: eliminasyon, A'dan rank-1 parçaları sırayla soymaktır. 2×2 örnek:

$$A=\begin{pmatrix}2 & 3\\4 & 7\end{pmatrix}$$

Eliminasyon: satır 2'den 2 çarpı satır 1'i çıkar (çarpan $\ell_{21}=2$), pivotlu U çıkar:

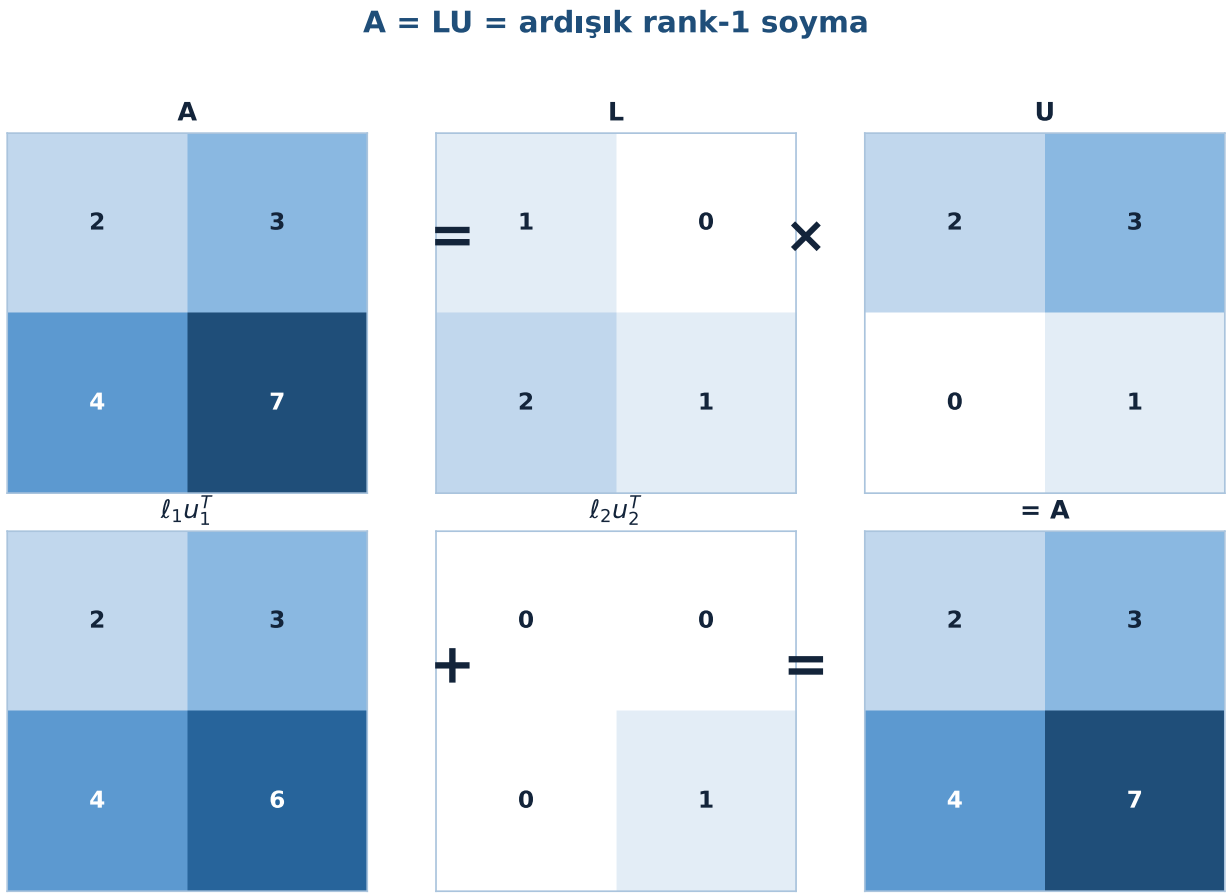
$$U=\begin{pmatrix}2 & 3\\0 & 1\end{pmatrix},\quad L=\begin{pmatrix}1 & 0\\2 & 1\end{pmatrix}$$

Şimdi $A=LU$'yu kolon çarpı satır olarak aç — birinci aşama, L'nin 1. kolonu çarpı U'nun 1. satırını soyar; geriye kalanı ikinci parça tamamlar:

$$A=\begin{pmatrix}1\\2\end{pmatrix}\begin{pmatrix}2 & 3\end{pmatrix}+\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0 & 1\end{pmatrix}=\begin{pmatrix}2 & 3\\4 & 6\end{pmatrix}+\begin{pmatrix}0 & 0\\0 & 1\end{pmatrix}$$

> *"[A is the first column of L times the first row of U, plus the rest] as the first column of l times the first row of u."* — Strang, 24:52

İlk rank-1 parça, A'nın ilk satır ve kolonunu doğru getirir; geriye bir boyut küçük bir problem (A_2) kalır. Bir sonraki eliminasyon adımı onu soyar. Böylece eliminasyon = ardışık rank-1 soyma. Bu soyma süreci Şekil 9.6'da gösteriliyor: üst sıra $A=L\times U$, alt sıra rank-1 katmanların ($\ell_1 u_1^T+\ell_2 u_2^T$) toplamı.



Şekil 9.6: A = LU çarpanlarına ayrılır; eşdeğer biçimde A, ardışık rank-1 parçaların toplamıdır: $A = \ell_1 u_1^T + \ell_2 u_2^T$. Üst sıra A = L × U çarpımını, alt sıra her rank-1 katmanı soyulduktan sonra A'nın geri toplanmasını gösterir.

> **Builder Notu — Soyarak Çözmek**
>
> Bu "rank-1 soyma" görüşü, blok-LU ve incremental/online matris ayrışımlarının temelidir. Aynı fikir SVD'de (en büyük tekil değerden başlayıp rank-1 parçaları soymak) düşük-rank yaklaşımı verir.

9.10 Dört Temel Alt-Uzay

Dersin sonunda Strang lineer cebirin temel teoremini veriyor: $m \times n$, rank r bir A matrisinin **dört temel alt-uzayı**.

> *"I invented the name four fundamental subspaces."* — Strang, 29:09

- **Kolon uzayı** $C(A)$ — A'nın kolonlarının kombinasyonları ($\mathbf{R}^m$ içinde).
- **Satır uzayı** $C(A^T)$ — A'nın satırlarının kombinasyonları ($\mathbf{R}^n$ içinde).
- **Null uzayı** $N(A)$ — $Ax = 0$ çözümleri ($\mathbf{R}^n$ içinde).
- **Sol null uzayı** $N(A^T)$ — $A^T y = 0$ çözümleri ($\mathbf{R}^m$ içinde).

> *"It's the set of solutions to Ax equal 0."* — Strang, 32:26

$C(A)$ ve $C(A^T)$ doğrudan matrisin sayılarından okunur; $N(A)$ ve $N(A^T)$ ise çözüm kümeleridir. Bir uzayda toplama ve sayıyla çarpma kapalıdır (lineer kombinasyon yine içindedir) — "uzay" olmanın tanımı budur.

Dört alt-uzayın büyük resmi Şekil 9.7'da: girdi uzayı $\mathbf{R}^n$ (satır uzayı $\perp$ null uzayı) ile çıktı uzayı $\mathbf{R}^m$ (kolon uzayı $\perp$ sol null uzayı) arasındaki A dönüşümü.

> **Builder Notu — Çözülebilirliğin Geometrisi**
>
> Dört alt-uzay, $Ax = b$'nin geometrisidir: $b \in C(A)$ ise çözüm vardır; $N(A) \neq \{0\}$ ise çözüm tek değildir (sonsuz çözüm). Least squares ($b \notin C(A)$ durumunda en yakın çözüm) ve minimum-norm çözüm ($N(A)$ içinde seçim) bu geometriden doğar — Ders 9-11'in temeli.

9.11 Boyutlar ve Rank

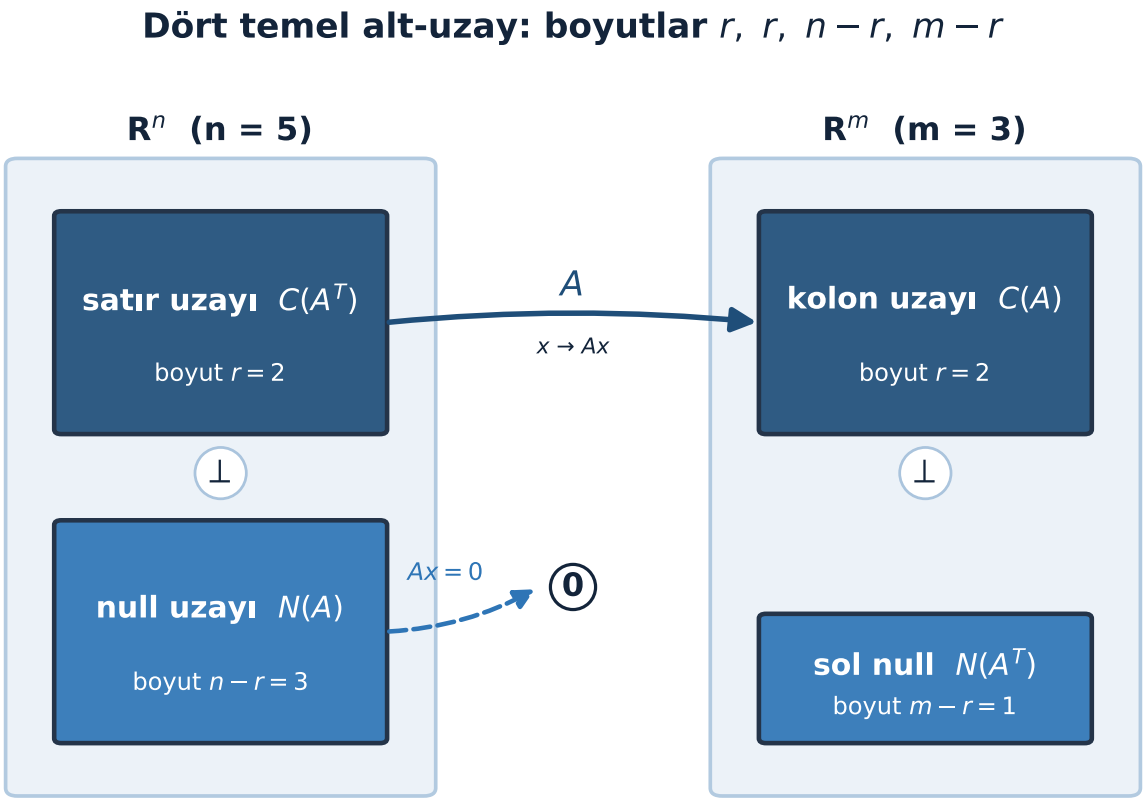
Dört alt-uzayın boyutları tek bir sayı etrafında düzenlenir: rank r.

$$\dim C(A) = \dim C(A^T) = r, \quad \dim N(A) = n - r, \quad \dim N(A^T) = m - r$$

Kolon uzayı ve satır uzayının boyutunun **aynı** olması (her ikisi de r), Ders 1'in ilk büyük teoremidir — burada temel teoremin parçası olarak tekrar karşımıza çıkıyor.

> *"those have the same dimension, same dimension."* — Strang, 31:23

Rank, kolon ve satır sayısını aşamaz: $r \leq \min(m, n)$. 50×100 bir matriste 100 kolonun bağımsız sayısı, 50 satırın bağımsız sayısına eşittir — sezgiye meydan okuyan ama her zaman doğru olan gerçek.



Şekil 9.7: Dört temel alt-uzay büyük resmi (m = 3, n = 5, r = 2). Sol tarafta girdi uzayı $\mathbf{R}^n$: satır uzayı $C(A^T)$ (boyut r) ile null uzayı $N(A)$ (boyut $n - r$) dik tümleyenler. Sağ tarafta çıktı uzayı $\mathbf{R}^m$: kolon uzayı $C(A)$ (boyut r) ile sol null uzayı $N(A^T)$ (boyut $m - r$). A, satır uzayını kolon uzayına birebir taşır; null uzayını sıfıra çöker.

Builder Notu — Serbestlik Derecesi

$n - r$ (null uzay boyutu), bir lineer sistemin "serbestlik derecesi"dir: kaç parametre serbest kalır. Eksik-belirlenmiş (underdetermined) ML problemlerinde (parametre > veri) bu boyut büyüktür; regularizasyon (ℓ^2 veya minimum norm) bu serbestliği daraltıp tek çözüm seçer.

9.12 Bu Dersin Özeti

1. **Beş faktorizasyon** — $A = LU$ (eliminasyon), $A = QR$ (ortonormallik), $S = Q\Lambda Q^T$ (özdeğer), $A = U\Sigma V^T$ (SVD), $A = CR$ (rank).
2. $A = LU$ — satır işlemlerinin matris formu; alt çarpı üst üçgensel.
3. $A = QR$ — ortonormal kolonlar ($Q^TQ = I$); least squares'in kararlı yolu.
4. $S = Q\Lambda Q^T$ — simetrik matris: gerçek özdeğerler, ortonormal özvektörler.
5. **Spektral teorem** — $S = \sum_k \lambda_k q_k q_k^T$; rank-1 parçaların toplamı (PCA'nın temeli).
6. $A = U\Sigma V^T$ **(SVD)** — her matris için; iki ortogonal matris, dikdörtgende de çalışır.
7. **Matris çarpımı = kolon × satır** — rank-1 yapı taşları (Ders 1 teması).
8. $A = LU$ **= ardışık rank-1 soyma** — eliminasyonun dış-çarpım görüşü.
9. **Dört temel alt-uzay** — $C(A)$, $C(A^T)$, $N(A)$, $N(A^T)$.
10. **Boyutlar** — r, r, $n - r$, $m - r$; kolon rank = satır rank.

Tek Bir Cümle

Beş faktorizasyon (LU, QR, $Q\Lambda Q^T$, $U\Sigma V^T$, CR) bir matrisi beş farklı soruya göre çözer; hepsinin ortak dili "matris = rank-1 parçaların toplamı"dır ve dört temel alt-uzay (boyutları r, r, $n - r$, $m - r$) bu yapının geometrisini verir.

9.13 Kontrol Soruları

Soru 1: Aşağıdaki simetrik matrisin spektral ayrışımını ($S = Q\Lambda Q^T$) bul.

$$S = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Cevap:
Karakteristik denklem: $\det(S - \lambda I) = (2 - \lambda)^2 - 1 = \lambda^2 - 4\lambda + 3 = (\lambda - 3)(\lambda - 1)$. Özdeğerler $\lambda = 3$ ve $\lambda = 1$ (ikisi de gerçek ✓).
$\lambda = 3$ için özvektör $(1, 1)$; $\lambda = 1$ için özvektör $(1, -1)$. Normalize edip Q'yu kur:

$$Q = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad \Lambda = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$$

Spektral toplam biçimi:

$$S = 3 \cdot \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + 1 \cdot \frac{1}{2} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Özvektörler ortogonal ✓ (dik), özdeğerler gerçek ✓ — simetrik matrisin iki garantisi.

Soru 2: Aşağıdaki matrisi bir kolon çarpı bir satır (rank-1 dış çarpım) olarak yaz ve rankını söyle.

$$A = \begin{pmatrix} 3 & 4 \\ 6 & 8 \end{pmatrix}$$

Cevap:
Tüm kolonlar $(1, 2)$'nin katı, tüm satırlar $(3, 4)$'ün katı:

$$A = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \begin{pmatrix} 3 & 4 \end{pmatrix}$$

Rank 1. Bu, spektral teoremdeki ve LU soymadaki rank-1 yapı taşının aynısıdır.

Soru 3: Neden her kare matrisin özdeğer ayrışımı olmayabilir ama SVD'si her zaman vardır? Bir örnek ver.

Cevap:
Özdeğer ayrışımı $A = Q\Lambda Q^T$ (veya genel $A = S\Lambda S^{-1}$) için yeterli sayıda bağımsız özvektör gerekir. Bazı matrislerde yoktur (defektif). Örnek:

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

Bu matrisin tek bir özdeğeri (0) ve tek bir bağımsız özvektörü var → diyagonalleştirilemez. Ayrıca dikdörtgen matrislerin (örneğin 2×3) özdeğeri tanımlı bile değildir.
SVD ($A = U\Sigma V^T$) ise tek özvektör kümesi yerine **iki** tekil vektör kümesi (U ve V) kullanır; bu yüzden defektif ve dikdörtgen dahil **her** matris için vardır.

Soru 4: 3×5 boyutunda, rank 2 bir A matrisi için dört temel alt-uzayın boyutlarını ver. Ax = b her b için çözülebilir mi?

Cevap:
$m = 3, n = 5, r = 2$:

$$\dim C(A) = 2, \quad \dim C(A^T) = 2, \quad \dim N(A) = 5 - 2 = 3, \quad \dim N(A^T) = 3 - 2 = 1$$

$C(A)$, $\mathbf{R}^3$ içinde 2 boyutlu bir alt-uzay (tüm $\mathbf{R}^3$ değil) → $Ax = b$ yalnızca $b \in C(A)$ ise çözülür, her b için DEĞİL.
Çözülebildiğinde de çözüm tek değildir: $N(A)$ 3 boyutlu olduğundan sonsuz çözüm vardır (genel çözüm = özel çözüm + $N(A)$). Bu, eksik-belirlenmiş sistemlerin tipik durumudur.

9.14 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy/scipy` ile kontrol et.

Egzersiz 1. Aşağıdaki matrisin $A = LU$ faktorizasyonunu bul (çarpanları kayıt et, L'yi doğrudan inşa et).

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 5 & 3 \\ 4 & 9 & 8 \end{pmatrix}$$

Egzersiz 2. İki aşamalı: (a) Aşağıdaki simetrik matrisin özdeğerlerini ve ortonormal özvektörlerini bul. (b) Spektral toplam $S = \lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T$ biçiminde yaz, $S \cdot q_1 = \lambda_1 q_1$ olduğunu doğrula.

$$S = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$$

Egzersiz 3. Aşağıdaki dikdörtgen matrisin neden özdeğer ayrışımı tanımsız ama SVD'si var olduğunu açıkla. $A^T A$ ve AA^T'nin boyutlarını yaz (tekil değerler bunların özdeğerlerinin karekökü).

$$A = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{pmatrix}$$

Egzersiz 4. Python ile beş faktorizasyonu gör:

```
import numpy as np
from scipy.linalg import lu, qr

A = np.array([[4.0, 3.0], [6.0, 3.0]])
S = np.array([[3.0, 1.0], [1.0, 3.0]])   # simetrik

P, L, U = lu(A)
Q, R = qr(A)
lam, V = np.linalg.eigh(S)               # simetrik için eigh
U2, sig, Vt = np.linalg.svd(A)

print("LU:", L, U, sep="\n")
print("QR:", Q, R, sep="\n")
print("eig(S):", lam, V, sep="\n")       # ortonormal V, gerçek lam
print("SVD:", U2, sig, Vt, sep="\n")
# Spektral toplam kontrolü:
print("rebuild S:", (V * lam) @ V.T)
```

Egzersiz 5. (Ders 3 habercisi.) Ders 3 Q matrislerine ($Q^T Q = I$) odaklanır. Aşağıdaki Q için $Q^T Q = I$ olduğunu doğrula; kolonların hem birim uzunlukta hem birbirine dik olduğunu göster. Bu, Gram-Schmidt'in (Ders 3) üreteceği türden bir matristir.

$$Q = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

9.15 Sonraki Ders İçin Hazırlık

Ders 3: Q'nun Ortonormal Kolonları $Q^TQ = I$ Verir

Ders 2'de beş faktorizasyonu tanıttık ve QR'a değindik. Ders 3, Q matrislerinin (ortonormal kolonlar) derinine iner — bu kursun ortogonallik temeli.

- Ortonormal kolonlar: $Q^TQ = I$ ne demek, neden uzunluk ve açı korur
- Ortogonal matrisler (kare Q): döndürme ve yansıma
- Örnekler: Householder, Givens, Fourier, Hadamard matrisleri
- Neden ortonormallik sayısal kararlılık ve ML init için kritik

⚠ Ders 3 Öncesi Yapılacak

- Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i ($Q^TQ = I$ doğrulaması).
- Python'da `np.linalg.qr` ile birkaç matrisi ayrıştır, Q'nun ortonormalliğini `Q.T @ Q` ile kontrol et.
- Ana cümleyi tekrar oku: "Beş faktorizasyon, matris = rank-1 parçaların toplamı."

9.16 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
Beş faktorizasyon	LU, QR, $Q\Lambda Q^T$, $U\Sigma V^T$, CR — her biri bir soruyu cevaplar	1m12
$A = LU$	Eliminasyon; alt üçgensel çarpı üst üçgensel	17m20
$A = QR$	Ortonormal kolonlar ($Q^TQ = I$); least squares	2m31
$S = Q\Lambda Q^T$	Simetrik: gerçek özdeğer, ortonormal özvektör	4m59
Özdeğerler gerçek	Simetrik matrisin garantisi	5m35
Spektral teorem	$S = \sum_k \lambda_k q_k q_k^T$; rank-1 parçaların toplamı	10m39
$A = U\Sigma V^T$ **(SVD)**	Her matris için; iki ortogonal matris	16m04
Rank-1 = kolon × satır	Dış çarpım; tüm çarpımların yapı taşı	7m34
$A = LU$ **rank-1 soyma**	Eliminasyon = ardışık rank-1 parça soyma	24m52
Dört temel alt-uzay	$C(A)$, $C(A^T)$, $N(A)$, $N(A^T)$	29m09
Boyutlar	r, r, $n-r$, $m-r$; kolon rank = satır rank	31m23

9.17 ML Bağlantıları Özeti

1. **Beş faktorizasyon** → `numpy.linalg/torch.linalg` araç kutusunun tamamı (solve, lstsq, eigh, svd, qr).
2. **Spektral teorem** → PCA; kovaryans matrisi $S = \sum_k \lambda_k q_k q_k^T$ olarak ana bileşenlere ayrılır.
3. **SVD** → her ağırlık matrisine uygulanır; düşük-rank sıkıştırma, LoRA, pseudoinverse.
4. **Rank-1 toplam** → SVD, PCA, spektral teoremin ortak dili; GEMM ve attention'ın yapısı.
5. **QR** → least squares'in kararlı çözümü; ortonormal ağırlık başlatması (orthogonal init).
6. **Dört alt-uzay** → $Ax = b$ çözülebilirliği ($b \in C(A)$) ve teklik ($N(A) = \{0\}$); least squares geometrisi.
7. $n - r$ **serbestlik derecesi** → eksik-belirlenmiş ML problemleri; regularizasyon bu serbestliği daraltıp tek çözüm seçer.

> **!** Tek bir şey alıp gideceksen
>
> Bir matrisi beş faktorizasyonla beş farklı soruya göre çözebilirsin — LU eliminasyonu, QR ortonormalliği, $Q\Lambda Q^T$ özdeğerleri, $U\Sigma V^T$ tekil değerleri, CR rankı açığa çıkarır. Hepsinin ortak dili "matris = rank-1 parçaların toplamı"dır; dört temel alt-uzay ise (boyutları r, r, $n - r$, $m - r$) bu yapının geometrisini verir.

10 Ortonormal Kolonlar — $Q^TQ = I$

Tek bir özdeşlik: uzunluk korunur, ters bedava gelir, bir aile açılır

> **Bölüm bilgisi**
>
> - **Video:** Orthonormal Columns in Q Give $Q^TQ = I$ — Gilbert Strang, MIT 18.065
> - **OCW:** Lecture 3 — Orthonormal Columns in Q Give $Q^TQ = I$
> - **Okuma süresi:** ≈ 35 dk
> - **Önkoşul:** Ders 2 (beş faktorizasyon, $A = QR$ ve $S = Q\Lambda Q^T$)

10.1 Bu Derste Ne Var?

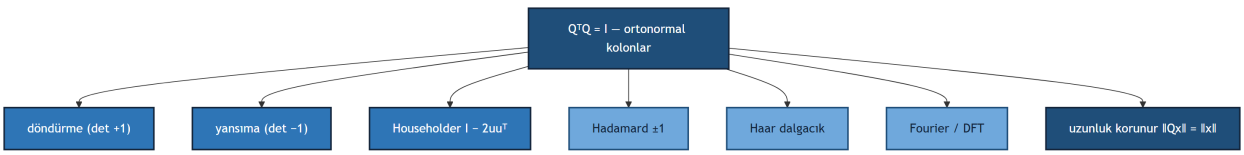
Ders 2'de beş faktorizasyon arasında QR'a değindik. Ders 3, Q matrislerinin — ortonormal kolonlu matrislerin — derinine iniyor. Tek bir özdeşlik ($Q^TQ = I$) her şeyi açar: uzunluk korunur, sayısal kararlılık gelir ve döndürme, yansıma, Householder, Hadamard, dalgacık, Fourier gibi bir ailenin kapısı aralanır.

Üç temel fikir:

1. $Q^TQ = I$ — ortonormal kolonların tek satırlık ifadesi; kare Q için bu ortogonal matristir ($Q^TQ = QQ^T = I$).
2. **Uzunluk korunur** — $\|Qx\| = \|x\|$; bu yüzden ortogonal matrisler sayısal algoritmaların gözdesidir (taşma yok).
3. **Ortogonal matris ailesi** — döndürme, yansıma, Householder ($I - 2uu^T$), Hadamard (± 1), Haar dalgacık, Fourier (DFT).

> *"…every time you see Q transpose Q, you've got the identity matrix."* — Strang, 0:47

Dersin tüm dallarını tek bir kavram haritasında topladık: Şekil 10.1 ortadaki $Q^TQ = I$ özdeşliğinden çıkan ortonormal kolon ailelerini ve onların ortak sonucu olan uzunluk korunumunu gösteriyor.



Şekil 10.1: Ders 3 kavram haritası: $Q^TQ = I$ ortak çatısı altında ortonormal kolon aileleri ve uzunluk korunumu.

Builder Notu — Açı ve Uzunluğu Koru

- **Uzunluk korunumu → gradyan kararlılığı:** ortonormal ağırlıklar gradyanı ne büyütür ne küçültür; orthogonal initialization ve ortonormal RNN/transformer katmanları gradyan patlama/sönmesini önler.
- $Q^TQ = I$ → **ucuz ters:** $Q^{-1} = Q^T$; ters almak bedavadır, normalizing flow ve invertible ağlarda kritik.
- **Householder ve Givens** — QR ve özdeğer algoritmalarının yapı taşları (LAPACK).
- **Fourier/DFT** — sinyal işleme, evrişim (Ders 32) ve FFT'nin temeli; ortogonal taban değişimi.

Tek cümle: $Q^TQ = I$, "açıyı ve uzunluğu koru" demektir — geometride döndürme ve yansıma, sayısal hesapta kararlılık, ML'de gradyan sağlığı.

10.2 Ortonormal Kolonlar → $Q^TQ = I$

Q'nun kolonları ortonormaldir: her biri birim uzunlukta (normalize) ve birbirine dik (ortogonal). Bu iki kelimelik özellik tek bir matris denklemine sıkışır:

$$Q^TQ = I$$

Neden? Q^TQ'nun (i, j) girdisi $q_i^Tq_j$'dir. Köşegende ($i = j$) \$q_i^{T} q_i = \$ uzunluk$^2 = 1 \rightarrow I$'nın 1'leri. Köşegen dışında ($i \neq j$) $q_i^Tq_j = 0$ (diklik) $\rightarrow I$'nın 0'ları.

> *"…every time you see Q transpose Q, you've got the identity matrix."* — Strang, 0:47

Builder Notu — İyi Koşullu Taban

$Q^TQ = I$, "kolonlar bağımsız ve dik" demektir — bu, bir tabanı en iyi koşullu (well-conditioned) hâline getirir. ML'de ortonormal taban, koordinatları gürültüye dayanıklı ve birbirinden bağımsız tutar (whitening, PCA sonrası ortonormal eksenler).

10.3 Ortogonal Matris (Kare Q)

Q kare olduğunda $Q^TQ = I$, Q'nun soldan tersinin Q^T olduğunu söyler. Kare matrislerde tek taraflı ters iki taraflı terstir, yani $QQ^T = I$ de doğrudur. Böyle bir matrise **ortogonal matris** denir:

$$Q^TQ = QQ^T = I \quad \Longrightarrow \quad Q^{-1} = Q^T$$

> *"an orthogonal matrix."* — Strang, 3:06

Tersinin transpozdan ibaret olması olağanüstü pratiktir: Q katsayılı her denklem, ters hesabı olmadan tek bir transpozla çözülür.

> **Builder Notu — Bedava Ters**
>
> $Q^{-1} = Q^T$ olması, ortogonal katmanların ileri ve geri geçişini simetrik kılar: backward pass'te Q^T ile çarpmak, forward'ın tam tersidir. Normalizing flow ve invertible ağlar bu özelliği kullanır (Jacobian determinantı ± 1).

10.4 Döndürme Matrisi

İlk örnek 2×2 döndürmedir. İlk kolon birim vektör ($\cos^2\theta + \sin^2\theta = 1$), ikinci kolon ona dik:

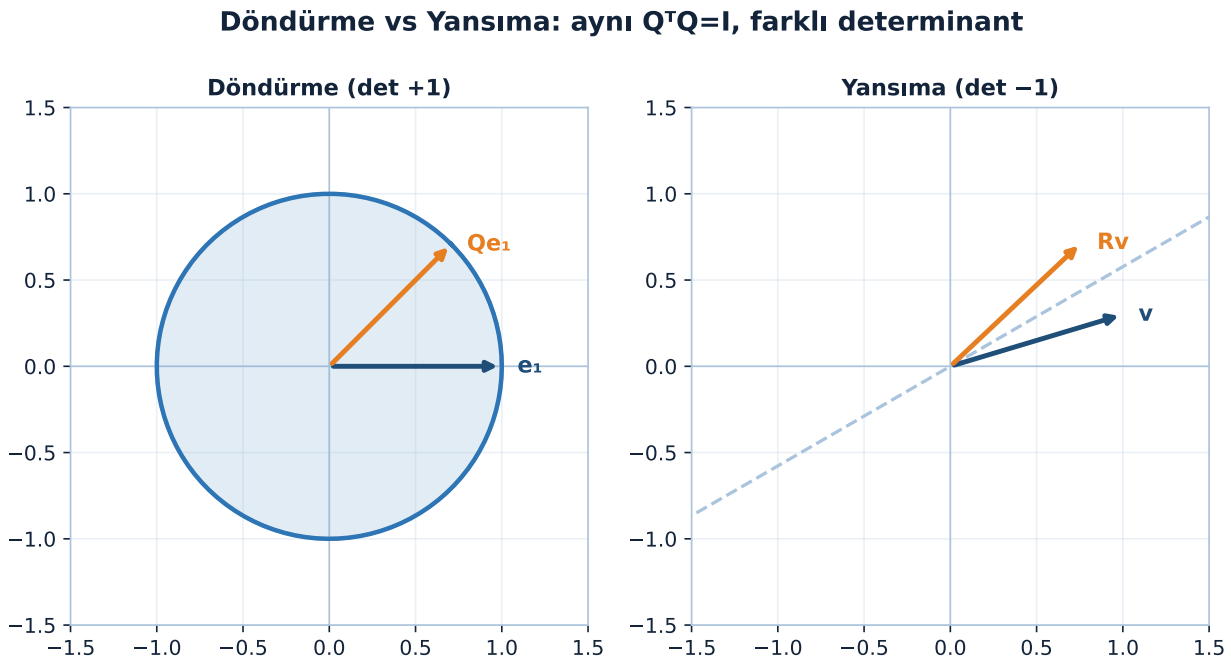
$$Q = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Bu matris tüm düzlemi θ açısıyla döndürür: $(1,0) \to (\cos\theta, \sin\theta)$, $(0,1) \to (-\sin\theta, \cos\theta)$.

> *"It's a rotation of the whole plane by theta."* — Strang, 5:00

Tersi $Q^{-1} = Q^T$, eksi işaretini alt köşeye taşır — yani $-\theta$ döndürmesi, beklendiği gibi.

Döndürme ve yansıma, aynı $Q^TQ = I$ özdeşliğini paylaşan iki temel ortogonal matristir; aralarındaki farkı determinant ele verir. Şekil 10.2 solda döndürmeyi (det $+1$, açı/yön korunur) sağda yansımayı (det -1, aynaya göre yön çevrilir) yan yana gösteriyor.



Şekil 10.2: Döndürme vs Yansıma: ikisi de $Q^TQ=I$ sağlar (ortogonal), ama döndürme det +1 ile açı/yön korur, yansıma det −1 ile aynaya göre yön çevirir.

> **Builder Notu — Düzlemi Döndür**
>
> Döndürme matrisleri, geometrik derin öğrenmede (rotation-equivariant ağlar), poz tahmininde (3B döndürmeler, SO(3)) ve veri artırmada (görüntü döndürme) doğrudan kullanılır. Determinantı $+1$ olan ortogonal matrisler "uygun döndürmeler"dir.

10.5 Uzunluk Korunur: ‖Qx‖ = ‖x‖

Ortogonal matrislerin en önemli özelliği: bir vektörün uzunluğunu değiştirmezler.

> *"It doesn't change length."* — Strang, 5:43

Kanıt tamamen $Q^TQ = I$'den çıkar. Kareleri karşılaştırmak kolay:

$$\|Qx\|^2 = (Qx)^T(Qx) = x^TQ^TQx = x^TIx = x^Tx = \|x\|^2$$

Parantezleri kaydırınca ortada $Q^TQ = I$ belirir ve düşer. Uzunluk korunduğu için ortogonal matrislerle çarparken **taşma (overflow) olmaz**:

> *"…no overflow can happen with orthogonal matrices. The lengths don't change."* — Strang, 6:20
> *"Every numerical algorithm is written to use orthogonal matrices wherever it can."* — Strang, 8:26

Bu korunumu somut bir örnekte görmek için Şekil 10.3'na bakın: $x = (3, 4)$ vektörü ve onun bir döndürme matrisiyle çarpılmış hâli Qx, aynı $\|x\| = 5$ yarıçaplı çember üzerinde kalır — yön döner, uzunluk değişmez.

> **Builder Notu — Gradyan Kararlılığı**
>
> Uzunluk korunumu = gradyan kararlılığı. Bir katman ortogonalse, ileri geçişte aktivasyon normu, geri geçişte gradyan normu sabit kalır — derin ağlarda patlama/sönme olmaz. Bu, ortogonal başlatma (orthogonal init) ve spektral normalizasyonun matematiksel gerekçesidir.

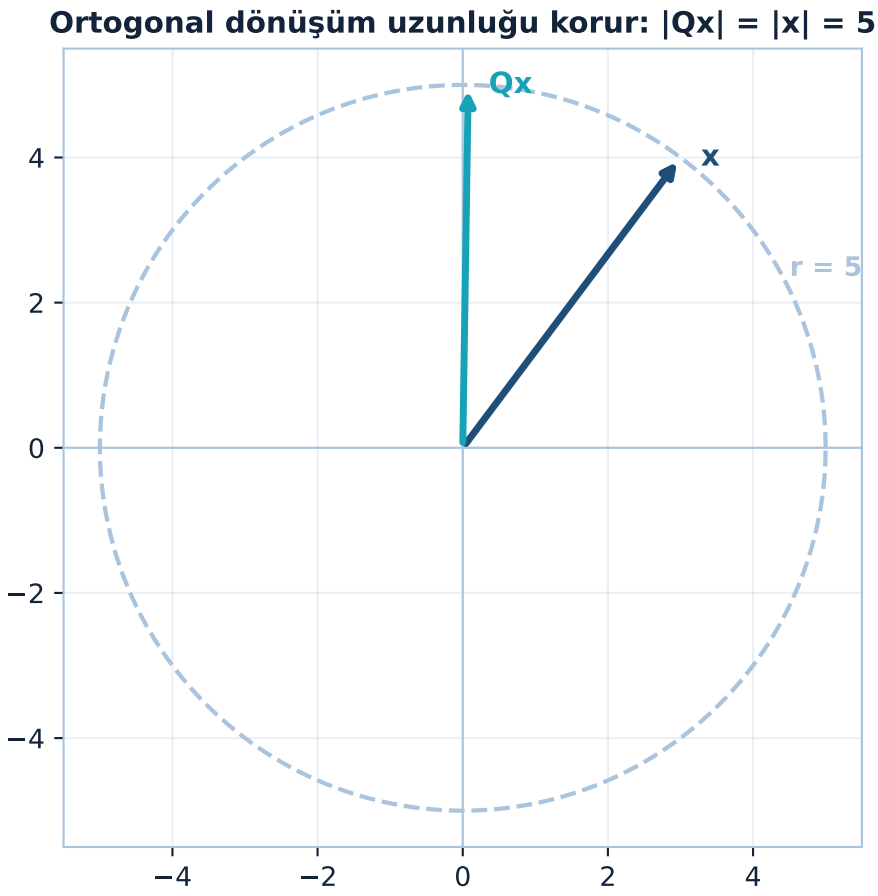
10.6 Yansıma Matrisi

Döndürmedeki tek bir işaret değişikliği farklı bir ortogonal matris verir. İlk kolon aynı $(\cos\theta, \sin\theta)$, ikinci kolonun işaretini değiştir:

$$\begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix}$$

Bu artık döndürme değil; simetriktir ve determinantı -1'dir. Düzlemi $\theta/2$ açısındaki bir aynaya göre yansıtır.

> *"This is a reflection matrix."* — Strang, 11:29



Şekil 10.3: Ortogonal dönüşüm uzunluğu korur: x = (3, 4) vektörü $\|x\| = 5$ uzunluğundadır ve bir döndürme matrisi Q ile çarpıldığında Qx aynı 5 yarıçaplı çember üzerinde kalır. $Q^TQ = I$ olduğu için $\|Qx\|^2 = (Qx)^T(Qx) = x^TQ^TQx = x^Tx = \|x\|^2$; yön değişir ama uzunluk korunur.

Determinant -1, yansımaların imzasıdır (döndürmelerde $+1$). Özdeğerlerinden biri -1 çıkar — ayna ekseninin tersi.

> Builder Notu — Ayna Simetrisi
>
> Yansımalar, ileride göreceğimiz Householder dönüşümlerinin 2×2 hâlidir. Geometrik ML'de parite/yansıma simetrisi (reflection-equivariance) modellenirken bu yapı kullanılır.

10.7 Householder Yansımaları

Strang yansımayı n boyuta taşıyor. Bir birim vektör u ($u^Tu = 1$) al ve şu matrisi kur:

$$H = I - 2uu^T$$

İçindeki uu^T bir kolon çarpı satır (rank-1) matristir. H hem simetriktir hem ortogonaldir.

> *"So these are Householder reflections."* — Strang, 14:53

Ortogonalliğini kareyle gör — H simetrik olduğu için $H^TH = H^2$:

$$H^2 = (I - 2uu^T)^2 = I - 4uu^T + 4u(u^Tu)u^T = I - 4uu^T + 4uu^T = I$$

Ortadaki $u^Tu = 1$ sayesinde son iki terim birbirini götürür.

> *"…this is a family of symmetric orthogonal matrices."* — Strang, 16:57

Householder dönüşümünün geometrisi Şekil 10.4'da görülüyor: birim normal u bir aynayı (ona dik doğruyu) tanımlar, bir v vektörü bu aynaya göre Hv'ye yansır — uzunluk korunur, $H^TH = I$.

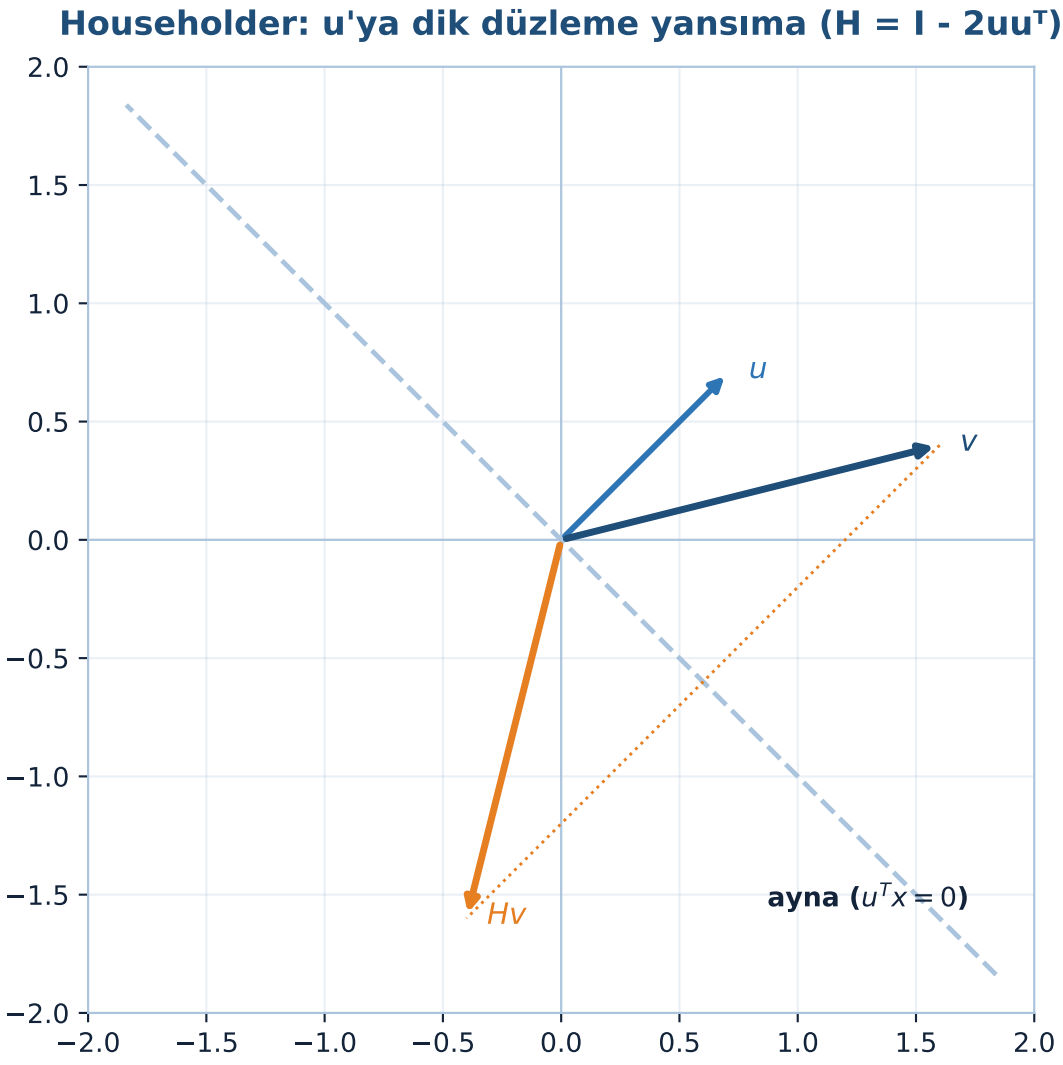
> Builder Notu — QR'ın Çekirdeği
>
> Householder yansımaları, QR faktorizasyonunun ve özdeğer algoritmalarının (LAPACK'in `geqrf`) çekirdeğidir — Gram-Schmidt'ten sayısal olarak daha kararlıdır. Bir vektörü tek adımda eksen yönüne yansıtarak kolonları sıfırlar.

10.8 Hadamard Matrisleri

Yalnızca $+1$ ve -1 girdili ortogonal matrisler. En küçüğü 2×2; büyükleri ikiye katlayarak kurulur:

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad H_4 = \begin{pmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{pmatrix}$$

Kolonlar diktir; ortonormal yapmak için $1/\sqrt{n}$ ile ölçeklenir (4×4 için $1/2$). Doğal soru: her boyutta var mı? Doubling yalnızca 2'nin katlarını verir, ama Strang daha cüretkar bir tahmin atıyor:

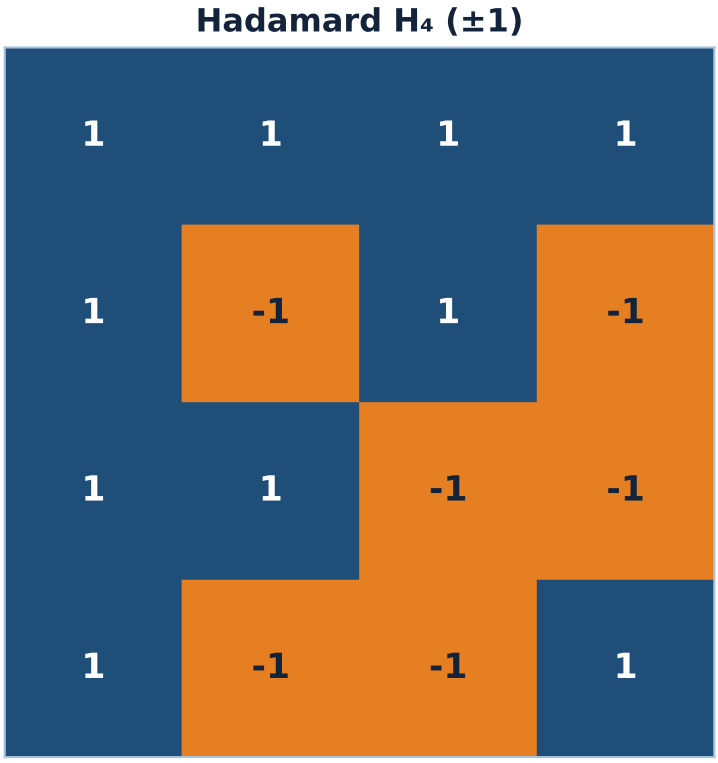


Şekil 10.4: Householder yansıması $H = I - 2uu^T$. Birim normal $u = (1,1)/\sqrt{2}$ (çelik) bir aynayı tanımlar: u'ya dik, orijinden geçen uzun kesik gri doğru ($u^T x = 0$). $v = (1.6, 0.4)$ (navy) bu aynaya göre $Hv = (-0.4, -1.6)$'ye (orange) yansır. Turuncu noktalı çizgi v ile Hv'yi birleştirir; aynayı dik kesip ortasından geçer — yansıma simetrisi. Uzunluk korunur ($\|Hv\| = \|v\|$), $H^T H = I$.

> *"…orthogonal matrix with orthogonal columns of every size n."* — Strang, 25:09

Tahmin: ± 1'li ortogonal bir matris, n'in 4'ün katı olduğu her boyutta vardır. Bazı boyutlar (örneğin 668) hâlâ tek tek aranır — sistematik bir üretim yöntemi yok.

H_4'ün ± 1 yapısı ve dik kolonları Şekil 10.5'da görülüyor: navy $+1$, turuncu -1; $H^TH = 4I$ olduğundan $1/2$ ile ölçeklenince ortonormal taban olur.



Şekil 10.5: Hadamard matrisi H_4: tüm girdileri ±1 (navy = +1, turuncu = −1), Sylvester ikilemesiyle kurulur. Kolonlar birbirine dik — $H^TH = 4I$ — yani ½ = 1/√4 ile ölçeklenince ortonormal taban olur. ±1 girdiler hızlı dönüşümlere (çarpma yok, yalnız toplama/çıkarma) ve kuantizasyona uygundur.

Builder Notu — Ucuz Ortogonal Dönüşüm

Hadamard matrisleri kodlama teorisi, hata düzeltme ve hızlı dönüşümlerde (fast Hadamard transform) kullanılır. ML'de rastgele projeksiyon ve boyut indirgemede (örneğin structured random features, Fastfood) ucuz ortogonal dönüşüm olarak iş görür.

10.9 Dalgacıklar (Haar)

Hadamard'a benzer ama bir farkla: dalgacıklar **kendini ölçekler** (self-scaling). Haar dalgacık matrisi W_4'ün kolonları farklı ölçeklerde ± 1 ve 0 içerir:

$$W_4 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 0 & -1 \end{pmatrix}$$

İlk kolon ortalama (hepsi 1), ikincisi büyük ölçekli fark, son ikisi küçük ölçekli farklar. Hadamard'dan farkı: sıfırlar sayesinde **seyrek** (sparse) ve çok-ölçeklidir.

> *"This is the Haar wavelet..."* — Strang, 30:17

Haar bunu 1910'da buldu; çok sonra (1988) Ingrid Daubechies daha pürüzsüz dalgacık ailelerini keşfetti.

W_4'ün çok-ölçekli ve seyrek yapısı Şekil 10.6'da görülüyor: kolonlar ortalama, büyük ölçek ve küçük ölçek rollerine ayrılır; küçük ölçek kolonları seyrektir (yarısı 0) ve $W^T W$ köşegendir (kolonlar dik).

> Builder Notu — Çok-Ölçekli Görüş
>
> Dalgacıklar JPEG2000 sıkıştırma, çok-ölçekli analiz ve gürültü gidermede kullanılır. Çok-ölçekli yapı, evrişimli ağların (CNN) havuzlama (pooling) piramidiyle aynı sezgiyi paylaşır: kaba ölçekten ince ölçeğe bilgi ayrıştırma.

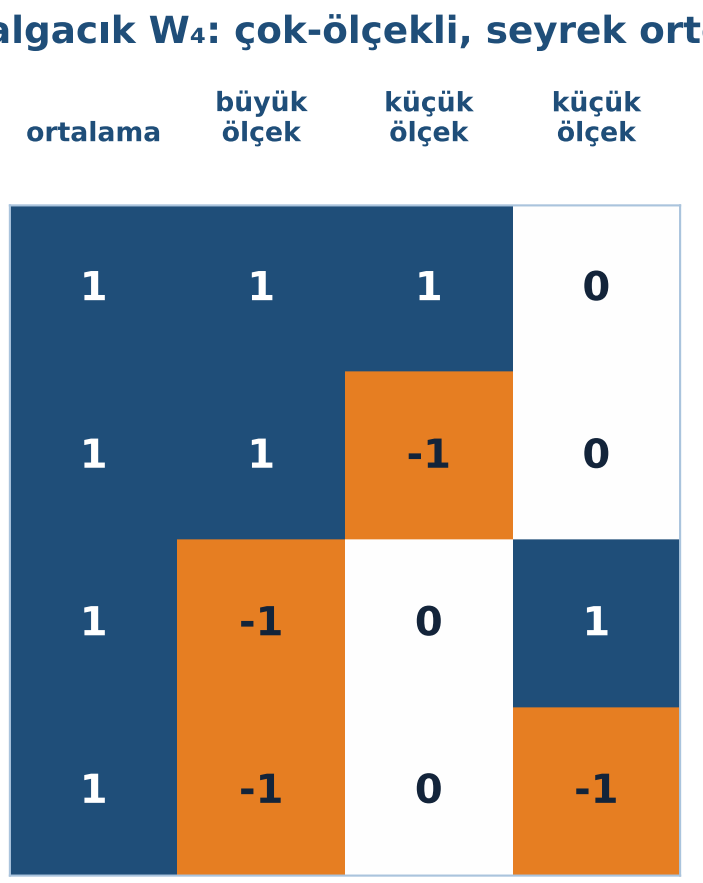
10.10 Ortogonal Özvektörler

Ortogonal matrisler nereden bol bol üretilir? En büyük kaynak, simetrik (ve ortogonal) matrislerin özvektörleridir.

> *"So the eigenvectors of a symmetric matrix [are orthogonal]."* — Strang, 35:09

Bir simetrik matris yaz, özvektörleri otomatik olarak ortogonal çıkar — uğraşmadan ortonormal bir taban elde edersin. Bu, Ders 4-5'in (özdeğerler, simetrik matrisler) habercisidir ve spektral teoremin (Ders 2) temelidir.

> Builder Notu — PCA Neden Çalışır
>
> "Simetrik matris → ortonormal özvektörler" garantisi, PCA'nın neden her zaman çalıştığını açıklar: kovaryans matrisi simetriktir, ana bileşenler (özvektörler) otomatik diktir. Aynı şey kernel matrisleri ve graf Laplacian'ları için de geçerli (Ders 35).



Şekil 10.6: Haar dalgacık W_4: dört ortogonal kolon, üç ölçekte. Kolon 1 sabit (ortalama / DC bileşeni), kolon 2 büyük ölçek (üst yarı − alt yarı), kolon 3-4 küçük ölçek; küçük ölçek kolonları seyrektir (yarısı 0) ve birbirine değmeyen blokları temsil eder. W^TW köşegendir, yani kolonlar dik — Haar matrisi çok-ölçekli ve seyrek bir ortogonal tabandır (dalgacık dönüşümünün çekirdeği).

10.11 Fourier Matrisi ve DFT

En önemli ortogonal taban Fourier'dir. Strang şaşırtıcı bir kaynaktan türetiyor: bir **permütasyon matrisinin** özvektörleri. Permütasyon matrisi birim matrisin satırları karıştırılmış hâlidir — kolonları açıkça ortogonaldir, yani bir Q'dur, dolayısıyla özvektörleri ortogonaldir. Bu özvektörler Fourier matrisi F'i oluşturur:

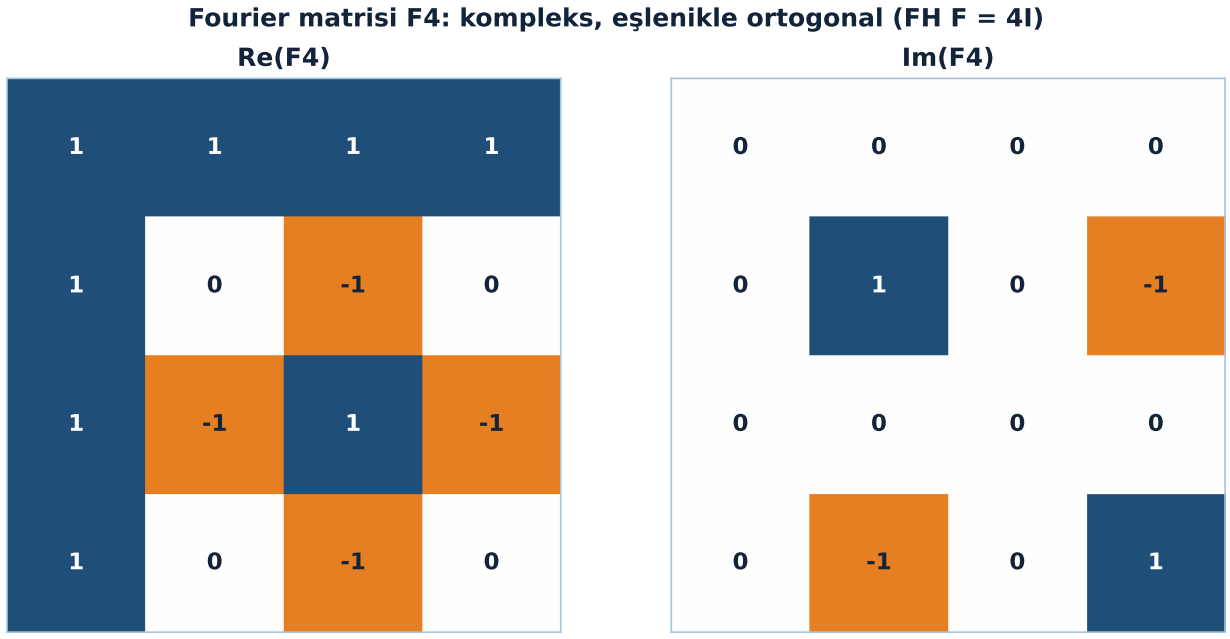
$$F = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & i^2 & i^3 \\ 1 & i^2 & i^4 & i^6 \\ 1 & i^3 & i^6 & i^9 \end{pmatrix}$$

> *"…they're tremendously useful. They're the heart of signal processing."* — Strang, 37:37

İlk kolon sabit (sıfır frekans), diğerleri i'nin (sanal birim) kuvvetleriyle artan frekanslar. Önemli uyarı: F kompleks olduğundan ortogonalliği test ederken **kompleks eşlenik** (conjugate) almak gerekir — her i'yi $-i$ ile değiştir, yoksa dik çıkmaz.

> *"…the complex conjugate of one is one."* — Strang, 46:15

Fourier matrisinin kompleks yapısı Şekil 10.7'da görülüyor: sol panel reel kısım $\mathrm{Re}(F_4)$, sağ panel sanal kısım $\mathrm{Im}(F_4)$. Kolonlar kompleks eşlenikle diktir: $F^H F = 4I$, yani $F/\sqrt{4}$ üniterdir.



Şekil 10.7: Fourier (DFT) matrisi F_4: girdileri i^{jk} ile kompleks (navy = +1, turuncu = −1, beyaz = 0). Sol panel reel kısım $\mathrm{Re}(F_4)$, sağ panel sanal kısım $\mathrm{Im}(F_4)$. İlk kolon ve ilk satır sabit (hepsi 1) — sabit/DC bileşeni. Kolonlar kompleks eşlenikle diktir: $F^H F = 4I$, yani $F/\sqrt{4}$ üniterdir (uzunluğu korur).

Builder Notu — Sinyal İşlemenin Kalbi

Ayrık Fourier dönüşümü (DFT) ve onun hızlı hâli FFT, sinyal işlemenin ve evrişimin (Ders 32) temelidir: evrişim, Fourier uzayında çarpmaya dönüşür. ML'de spektral yöntemler, ses/görüntü ön-işleme ve bazı dikkat (attention) hızlandırmaları DFT üzerine kuruludur.

10.12 Bu Dersin Özeti

1. $Q^TQ = I$ — ortonormal kolonların (birim + dik) tek satırlık ifadesi.
2. **Ortogonal matris** — kare Q için $Q^TQ = QQ^T = I$; $Q^{-1} = Q^T$.
3. **Döndürme** — 2×2, determinant $+1$, düzlemi θ döndürür.
4. **Uzunluk korunur** — $\|Qx\| = \|x\|$; taşma yok, sayısal algoritmaların gözdesi.
5. **Yansıma** — 2×2, determinant -1, aynaya göre yansıtır.
6. **Householder** — $H = I - 2uu^T$; simetrik + ortogonal, $H^2 = I$.
7. **Hadamard** — ± 1 girdili ortogonal matrisler; 4'ün katı boyut tahmini.
8. **Dalgacıklar (Haar)** — kendini ölçekleyen, seyrek $\pm 1/0$ matrisler.
9. **Ortogonal özvektörler** — simetrik/ortogonal matrislerden bedava gelir.
10. **Fourier (DFT)** — permütasyonun özvektörleri; kompleks, eşlenikle dik.

Tek Bir Cümle

$Q^TQ = I$ tek bir denklemdir ama "açıyı ve uzunluğu koru" demektir: geometride döndürme ve yansıma, sayısal hesapta taşmasız kararlılık, ML'de gradyan sağlığı — ve döndürme, Householder, Hadamard, dalgacık, Fourier hep bu tek özelliğin aileleridir.

10.13 Kontrol Soruları

Soru 1: Aşağıdaki Q için $Q^TQ = I$ olduğunu doğrula ve x = (3, 4) vektörünün uzunluğunu koruduğunu göster.

$$Q = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

Cevap:

$$Q^TQ = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = I$$

Uzunluk: $\|x\| = \sqrt{9+16} = 5$. $Qx = (1/\sqrt{2})(3-4, 3+4) = (1/\sqrt{2})(-1, 7)$, $\|Qx\|^2 = (1+49)/2 = 25$, $\|Qx\| = 5 = \|x\|$ ✓. (Bu Q, 45° döndürmedir.)

i Soru 2: u = (1/√2)(1, 1) için Householder matrisi H = I − 2uu^T'yi hesapla. Simetrik ve ortogonal olduğunu doğrula.

Cevap:

$$uu^T = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}, \quad H = I - 2uu^T = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} - \begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix} = \begin{pmatrix}0 & -1\\ -1 & 0\end{pmatrix}$$

Simetrik ✓ ($H^T = H$). Ortogonal: $H \cdot H = I$ (çünkü $u^T u = 1$). Bu H, koordinatları takas edip işaret değiştirir — u'ya dik doğruya göre yansıma.

i Soru 3: 90° döndürme matrisi tamamen gerçek olduğu hâlde özvektörleri neden kompleks çıkar?

$$Q = \begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}$$

Cevap:
Özdeğerler $\det(Q - \lambda I) = \lambda^2 + 1 = 0 \to \lambda = \pm i$ (kompleks). 90° döndürme, düzlemde **hiçbir** gerçek yönü sabit bırakmaz (her vektör döner), o yüzden gerçek özvektör yoktur. Özvektörler $(1, \mp i)$ gibi kompleks vektörlerdir. Bu, ortogonal matrislerin (S simetrikten farklı olarak) kompleks özvektörlü olabilmesinin nedenidir — Fourier matrisindeki i'nin kaynağı.

i Soru 4: Ortogonal başlatma (orthogonal init) derin ağlarda gradyan kararlılığına nasıl yardım eder? ‖Qx‖ = ‖x‖ ile bağ kur.

Cevap:
Bir katmanın ağırlığı ortogonalse, ileri geçişte aktivasyon normu korunur ($\|Qx\| = \|x\|$), geri geçişte de gradyan normu korunur (Q^T ile çarpım da uzunluk korur). Çok katmanlı bir ağda normlar ne katlanarak büyür (patlama) ne de katlanarak küçülür (sönme).
Genel bir ağırlık matrisinde her katman normu $\sigma_{\max}$ veya $\sigma_{\min}$ ile çarpar; L katman sonra σ^L patlar/söner. Ortogonalde tüm tekil değerler 1 olduğundan bu sorun ortadan kalkar. Pratikte `torch.nn.init.orthogonal_` bunu sağlar; RNN'lerde ve çok derin ağlarda kritiktir.

10.14 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. 60° döndürme matrisini yaz. $Q^T Q = I$ olduğunu ve determinantının $+1$ olduğunu doğrula.

$$Q = \begin{pmatrix}\cos 60^\circ & -\sin 60^\circ\\ \sin 60^\circ & \cos 60^\circ\end{pmatrix}$$

Egzersiz 2. $u = (1/\sqrt{3})(1, 1, 1)$ için 3×3 Householder matrisi $H = I - 2uu^T$'yi kur. Simetrik olduğunu ve $H^2 = I$ sağladığını göster. H'nin u'yu $-u$'ya gönderdiğini doğrula ($Hu = -u$).

Egzersiz 3. 4×4 Hadamard matrisi H_4'ün kolonlarının birbirine dik olduğunu (iç çarpımları 0) göster. Ortonormal yapmak için hangi sabitle ölçeklersin?

$$H_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Egzersiz 4. Python ile ortogonalliği ve uzunluk korunumunu test et:

```
import numpy as np

theta = np.pi / 5
Q = np.array([[np.cos(theta), -np.sin(theta)],
              [np.sin(theta),  np.cos(theta)]])

print("Q^T Q = I ?", np.allclose(Q.T @ Q, np.eye(2)))
print("det(Q) =", np.linalg.det(Q))          # +1 (döndürme)

x = np.array([3.0, 4.0])
print("|x| =", np.linalg.norm(x), " |Qx| =", np.linalg.norm(Q @ x))  # eşit

# Householder
u = np.array([1.0, 1.0, 1.0]); u = u / np.linalg.norm(u)
H = np.eye(3) - 2 * np.outer(u, u)
print("H simetrik ?", np.allclose(H, H.T))
print("H^2 = I ?", np.allclose(H @ H, np.eye(3)))
```

Egzersiz 5. (Ders 4 habercisi.) Ders 4 özdeğer ve özvektörlere geçer. Aşağıdaki simetrik matrisin özdeğerlerini bul; özvektörlerinin birbirine dik çıktığını doğrula (bu derste gördüğümüz "simetrik → ortogonal özvektörler" gerçeği).

$$S = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

10.15 Sonraki Ders İçin Hazırlık

Ders 4: Özdeğerler ve Özvektörler

Ders 3'ü "simetrik/ortogonal matrislerin özvektörleri ortogonaldir" gözlemiyle kapattık. Ders 4 doğrudan özdeğer ve özvektörlere giriyor — kursun özüne.

- Özdeğer/özvektör tanımı ve $Ax = \lambda x$
- Simetrik matrisler: gerçek özdeğerler, ortonormal özvektörler (spektral teoremin temeli)
- Pozitif tanımlılığa giriş (Ders 5'in hazırlığı)
- Neden özdeğerler ML'de her yerde: PCA, kararlılık, Hessian

⚠ Ders 4 Öncesi Yapılacak

- Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (simetrik matrisin özvektörleri).
- Python'da `np.linalg.eigh` ile birkaç simetrik matrisin özvektörlerini bul, `V.T @ V` ile ortonormalliklerini kontrol et.
- Ana cümleyi tekrar oku: "$Q^TQ = I$ — açıyı ve uzunluğu koru."

10.16 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
$Q^TQ = I$	Ortonormal kolonların (birim + dik) ifadesi	0m47
Ortogonal matris	Kare Q; $Q^TQ = QQ^T = I$, $Q^{-1} = Q^T$	3m06
Döndürme	2×2, determinant $+1$, düzlemi θ döndürür	5m00
Uzunluk korunur	$\|Qx\| = \|x\|$; $Q^TQ = I$'den çıkar	5m43
Taşma yok	Ortogonal matrisler sayısal algoritmaların gözdesi	6m20
Yansıma	2×2, determinant -1, aynaya yansıtır	11m29
Householder	$H = I - 2uu^T$; simetrik + ortogonal, $H^2 = I$	14m53
Hadamard	± 1 girdili ortogonal; 4'ün katı boyut tahmini	25m09
Haar dalgacık	Kendini ölçekleyen, seyrek $\pm 1/0$ matrisler	30m17
Ortogonal özvektörler	Simetrik/ortogonal matristen bedava gelir	35m09
Fourier / DFT	Permütasyonun özvektörleri; kompleks, eşlenikle dik	37m37

10.17 ML Bağlantıları Özeti

1. $Q^TQ = I$ → whitening ve ortonormal taban; koordinatlar bağımsız ve iyi koşullu.
2. **Uzunluk korunumu** → orthogonal init, spektral normalizasyon; derin ağda gradyan patlama/sönmesini önler.
3. $Q^{-1} = Q^T$ → normalizing flow ve invertible ağlar; ucuz ters, Jacobian determinantı ± 1.
4. **Householder/Givens** → QR ve özdeğer algoritmalarının çekirdeği (LAPACK).
5. **Hadamard** → hızlı rastgele projeksiyon, structured random features, boyut indirgeme.

6. **Dalgacıklar** → çok-ölçekli analiz, sıkıştırma; CNN havuzlama piramidiyle aynı sezgi.
7. **Fourier/DFT** → sinyal/ses/görüntü ön-işleme, evrişim (Ders 32), spektral yöntemler.

! Tek bir şey alıp gideceksen

$Q^TQ = I$ tek bir denklemdir ama "açıyı ve uzunluğu koru" demektir. Bu yüzden ortogonal matrisler uzunluğu değiştirmez ($\|Qx\| = \|x\|$), tersi transpozdur ($Q^{-1} = Q^T$) ve sayısal hesabın gözdesidir. Döndürme, yansıma, Householder, Hadamard, dalgacık ve Fourier — hepsi bu tek özelliğin aileleridir; ML'de gradyan kararlılığından evrişime kadar uzanır.

11 Özdeğerler ve Özvektörler

Ax = λx: bir matrisin doğal eksenleri, köşegenleştirme ve spektral teorem

> **Bölüm bilgisi**
>
> - **Video:** Eigenvalues and Eigenvectors — Gilbert Strang, MIT 18.065
> - **OCW:** Lecture 4 — Eigenvalues and Eigenvectors
> - **Okuma süresi:** ≈ 35 dk
> - **Önkoşul:** Ders 3 (ortonormal kolonlar, $Q^TQ = I$, "simetrik → ortogonal özvektörler")

11.1 Bu Derste Ne Var?

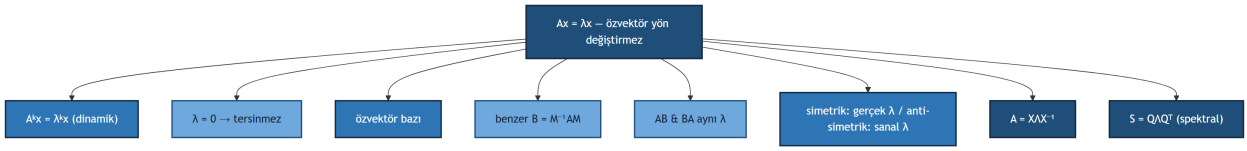
Ders 3'ü "simetrik matrislerin özvektörleri ortogonaldir" gözlemiyle kapatmıştık. Ders 4 doğrudan özdeğer ve özvektörlere giriyor: önce her kare matris A, sonra simetrik S, en sonunda özel olan pozitif tanımlı S (Ders 5).

Üç temel fikir:

1. $Ax = \lambda x$ — özvektör, A ile çarpılınca yön değiştirmeyen özel vektör; A^k, A^{-1}, e^{At} hepsi aynı özvektörü taşır, özdeğer λ^k, $1/\lambda$, $e^{\lambda t}$ olur.
2. **Köşegenleştirme** $A = X\Lambda X^{-1}$ — özvektörler X'i, özdeğerler Λ'yı oluşturur; benzer matrisler ($M^{-1}AM$) aynı özdeğeri paylaşır.
3. **Spektral teorem** $S = Q\Lambda Q^T$ — simetrik matris: gerçek özdeğer, ortonormal özvektör (Q ortogonal).

> *"Ax comes out some number times x."* — Strang, 1:51

Bu üç fikrin nasıl tek bir merkezden ($Ax = \lambda x$) dallandığını Şekil 11.1 özetliyor: dinamikten köşegenleştirmeye, simetri-anti-simetri ayrımından spektral teoreme kadar dersin bütün hatları.



Şekil 11.1: Ders 4 kavram haritası: Ax = λx merkezinden özdeğer dünyasının dalları — dinamik, köşegenleştirme ve spektral teorem.

Builder Notu — Doğal Eksenler

- $A^k = X\Lambda^k X^{-1}$ → **dinamik/kararlılık:** bir tekrarlı sistemin (RNN, power iteration, Markov) uzun-vadeli davranışı en büyük $|\lambda|$ tarafından belirlenir; $|\lambda| > 1$ patlar, $|\lambda| < 1$ söner.
- **Spektral teorem → PCA:** kovaryans simetriktir, özvektörleri (ana bileşenler) ortonormal, özdeğerleri (varyanslar) gerçektir.
- **Trace ve determinant** — $\sum \lambda = \text{trace}$, $\prod \lambda = \det$; modelde hızlı sağlama ve düzenlileştirme terimleri (nuclear/log-det) bu kimliklere dayanır.
- **Benzer matrisler** — taban değişimi özdeğeri korur; özellik mühendisliğinde ve ağ reparametrizasyonunda kritik.

Tek cümle: özvektörler bir matrisin "doğal eksenleridir" — o eksenlerde matris yalnızca ölçekler (λ ile), ve $A = X\Lambda X^{-1}$ tüm kuvvet/fonksiyon hesabını önemsizleştirir.

11.2 Özdeğer ve Özvektör: Ax = λx

Bir A matrisini çoğu vektörle çarpınca yön değişir. Ama bazı özel vektörlerde Ax, x ile **aynı doğrultuda** çıkar — yalnızca ölçeklenir:

$$Ax = \lambda x$$

Buradaki x özvektör, λ ise özdeğer. $n \times n$ bir matrisin (iyi durumda) n tane bağımsız özvektörü vardır.

> *"Ax comes out some number times x."* — Strang, 1:51

Bu "yön korunur, yalnızca ölçek değişir" sezgisi Şekil 11.2'de görülüyor: özvektör doğrultularında A vektörü kendi doğrusu üzerinde tutar, genel bir vektörde ise yön kayar.

Builder Notu — PCA'nın Sezgisi

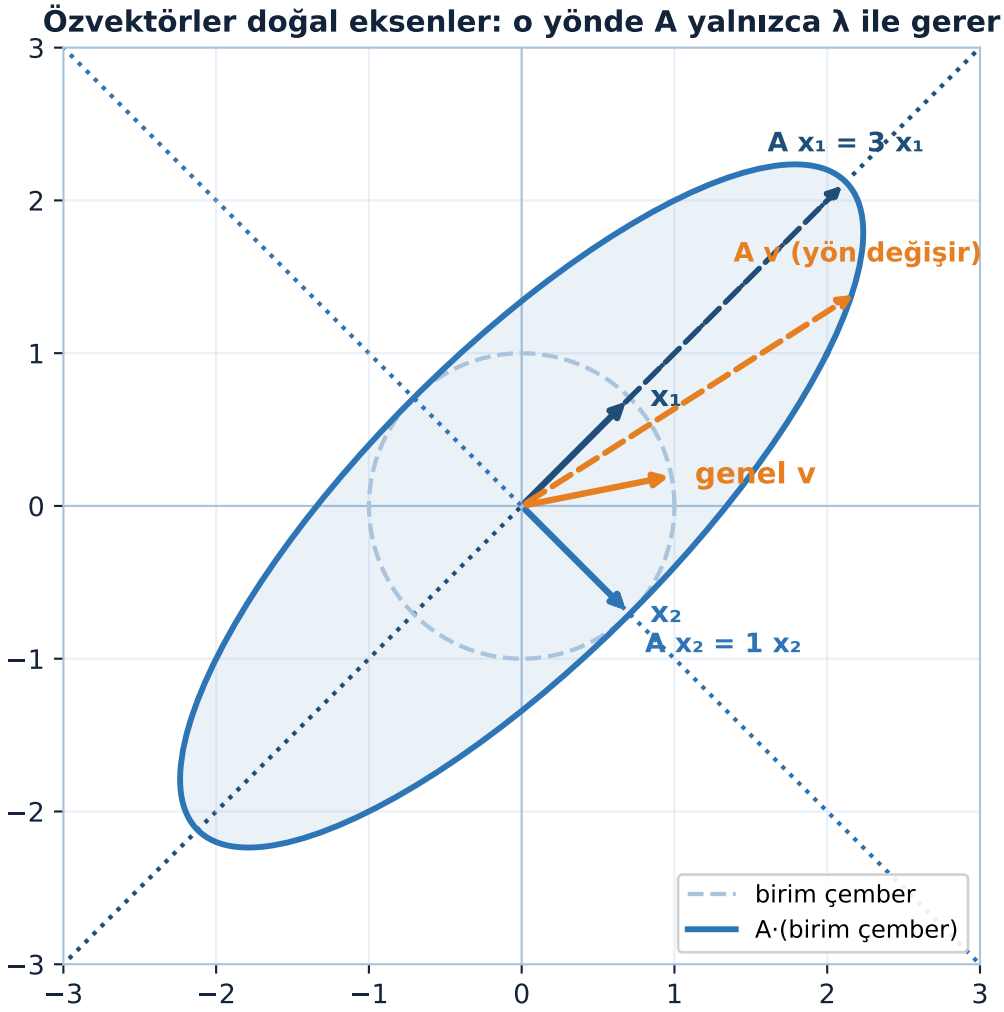
Özvektörler matrisin "doğal eksenleri"dir: o eksenlerde matris döndürmez, sadece gerer/sıkıştırır. PCA tam da veri kovaryansının bu doğal eksenlerini bulur; ana bileşenler en büyük özdeğerli özvektörlerdir.

11.3 Neden Yararlı: $A^k x = \lambda^k x$

Özvektörlerin gücü, A'nın kuvvetlerinde ortaya çıkar. A^2x'i hesapla: $A(Ax) = A(\lambda x) = \lambda(Ax) = \lambda^2 x$. Devam ederek:

$$A^2x = \lambda^2 x, \quad A^k x = \lambda^k x, \quad A^{-1}x = \frac{1}{\lambda}x, \quad e^{At}x = e^{\lambda t}x$$

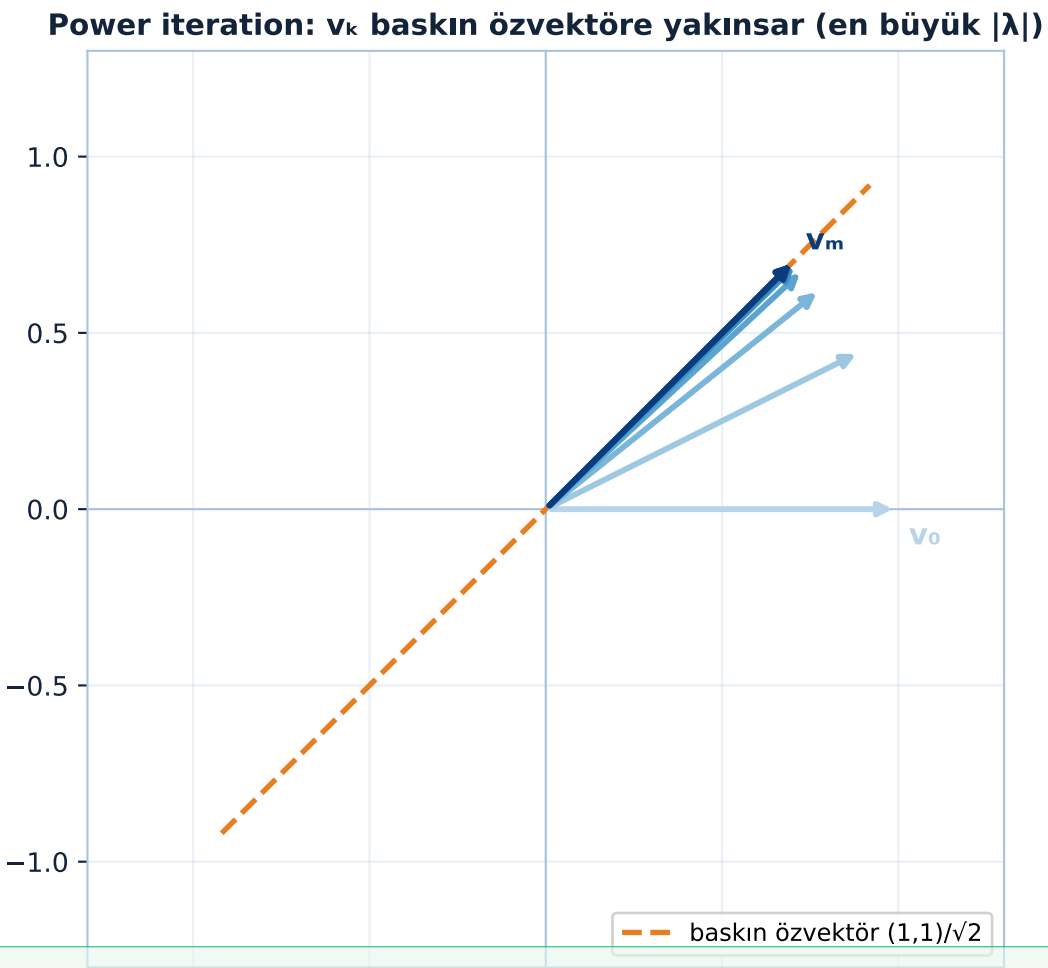
Özvektör değişmez; yalnızca özdeğer kuvvet alır. Matrisin herhangi bir kuvveti veya fonksiyonu (üstel dâhil) özvektör tabanında önemsizleşir.



Şekil 11.2: Özvektörler doğal eksenlerdir. $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ için $x_1 = (1, 1)$ yönünde $Ax_1 = 3x_1, x_2 = (1, -1)$ yönünde $Ax_2 = 1x_2$ — vektör yalnızca λ ile ölçeklenir, yönü korunur. Genel bir v vektöründe ise yön değişir.

“…and the eigenvalue is lambda squared.” — Strang, 4:28

Kuvvet aldıkça en büyük $|\lambda|$’nın yönünün baskın hâle gelişi Şekil 11.3’da somutlaşıyor: $v_k = A^k v_0$ yön dizisi baskın özvektöre yelpaze gibi yakınsar.



11.4 λ = 0 ve Tersinirlik

Özel durum: $\lambda = 0$ olabilir. O zaman $Ax = 0 \cdot x = 0$, yani özvektör A'nın **null uzayındadır** ve A tersinir değildir.

> *"A doesn't even have an inverse."* — Strang, 5:22

$A^{-1}x = (1/\lambda)x$ formülü tam da bu yüzden $\lambda = 0$'da çöker: $1/0$ tanımsızdır, çünkü tersi olmayan bir matrisin o yöndeki "tersi" yoktur. Özdeğerlerden biri 0 ise matris tekildir (singular).

> **Builder Notu — Sıfır Özdeğer = Tehlike**
>
> $\lambda = 0$ (veya çok küçük λ) = kötü koşullanma. Kovaryans veya Hessian matrisinde sıfıra yakın özdeğer, o yönde bilginin/eğriliğin yok olduğunu söyler — düzenlileştirme (λI ekleme, ridge) tam da bu sıfır özdeğerleri kaldırıp matrisi tersinir kılmak içindir.

11.5 Özvektör Bazı ve Fark Denklemleri

n bağımsız özvektör bir **baz** oluşturur. Herhangi bir v vektörünü bu bazda yaz:

$$v = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

A'nın k. kuvvetini uygulamak artık önemsiz — her parça kendi λ'sının kuvvetini alır:

$$A^k v = c_1 \lambda_1^k x_1 + c_2 \lambda_2^k x_2 + \cdots + c_n \lambda_n^k x_n$$

Bu, özvektörlerin icat ediliş amacıdır: fark denklemlerini ($v_{k+1} = Av_k$) ve sürekli zamanlı denklemleri ($dv/dt = Av \to v(t) = \sum_i c_i e^{\lambda_i t} x_i$) anında çözmek.

> *"...to be able to solve difference equations."* — Strang, 10:38

> **Builder Notu — Dinamiği Çözmek**
>
> Bu ayrışım, lineer dinamik sistemlerin (kontrol, difüzyon modelleri, lineerleştirilmiş RNN) çözümüdür. Bir başlangıç durumunu özvektör tabanına yansıt, her bileşeni λ^k ile evrimleştir, geri topla — eğitim dinamiğini ve kararlılığı analiz etmenin standart yolu.

11.6 Benzer Matrisler: B = M^{-1}AM

İki matris, tersinir bir M ile şöyle bağlıysa **benzer** (similar) denir:

$$B = M^{-1} A M$$

Benzer matrislerin anahtar özelliği: **aynı özdeğerlere** sahiptirler.

> *"They have the same eigenvalues."* — Strang, 13:56

Kanıt kısa: $M^{-1}AM$'in özvektörü y, özdeğeri λ olsun ($M^{-1}AM\,y = \lambda y$). Her iki tarafı M ile çarp → $A(My) = \lambda(My)$. Yani λ, A'nın da özdeğeridir; özvektör y'den My'ye değişti ama özdeğer aynı kaldı.

> **Builder Notu — Taban Değiştir, Özü Koru**
>
> Benzerlik = taban değişimi. M ile koordinat değiştirmek matrisin "özünü" (özdeğerlerini) korur, yalnızca temsili değişir. ML'de ağ reparametrizasyonu, beyazlatma (whitening) ve özellik dönüşümleri bu çerçevededir — doğru M, problemi köşegen (bağımsız) hâle getirir.

11.7 eig(A) Nasıl Hesaplanır

Özdeğerler pratikte `eig(A)` ile bulunur. Arka planda ne olur? Algoritma, ardışık iyi M'ler seçerek A'yı benzer matrislere taşır ve giderek üçgensel (simetrikte köşegen) hâle getirir; özdeğerler köşegende belirir.

> *"It brings the matrix to a triangular matrix."* — Strang, 16:13

Her M benzerliği koruduğundan özdeğerler boyunca değişmez; köşegen-dışı terimler küçülürken köşegende gerçek özdeğerler ortaya çıkar. Simetrik matrislerde bu süreç temizdir (köşegene yakınsar).

> **Builder Notu — QR İterasyonu**
>
> Bu, QR iterasyonunun (Ders 12) sezgisidir: ortogonal benzerlik dönüşümleriyle (Q'lar, Ders 3) köşegene yakınsama. Büyük matrislerde tam eig yerine Lanczos/güç iterasyonu gibi yalnızca baskın özdeğerleri bulan yöntemler kullanılır — PCA ve spektral kümelemenin pratiği.

11.8 AB ve BA Aynı Özdeğerlere Sahiptir

Hoş bir gerçek: herhangi iki A, B matrisi için AB ile BA aynı (sıfırdan farklı) özdeğerlere sahiptir.

> *"…the same non-zero ones… as BA."* — Strang, 19:58

Kanıt benzerlikten gelir: $M = B$ seç, o zaman $BA = B(AB)B^{-1}$, yani BA, AB'ye benzerdir → aynı özdeğerler:

$$BA = B(AB)B^{-1}$$

Dikkat: bu, özdeğerlerin çarpılabileceği anlamına gelmez. A'nın ve B'nin özdeğerlerini bilmek, AB'nin veya $A + B$'nin özdeğerlerini vermez — çünkü özvektörler farklıdır.

> **Builder Notu — Gram Matrisi Numarası**
>
> AB ile BA'nın özdeşliği, derin öğrenmede Gram matrisi numarasıdır: $m \times n$ veride XX^T ($m \times m$) ile X^TX ($n \times n$) aynı sıfırdan-farklı özdeğerlere sahiptir. Hangisi küçükse onu hesaplarsın (kernel

trick, SVD'nin iki tarafı) — büyük boyutta muazzam tasarruf.

11.9 Simetrik vs Anti-simetrik

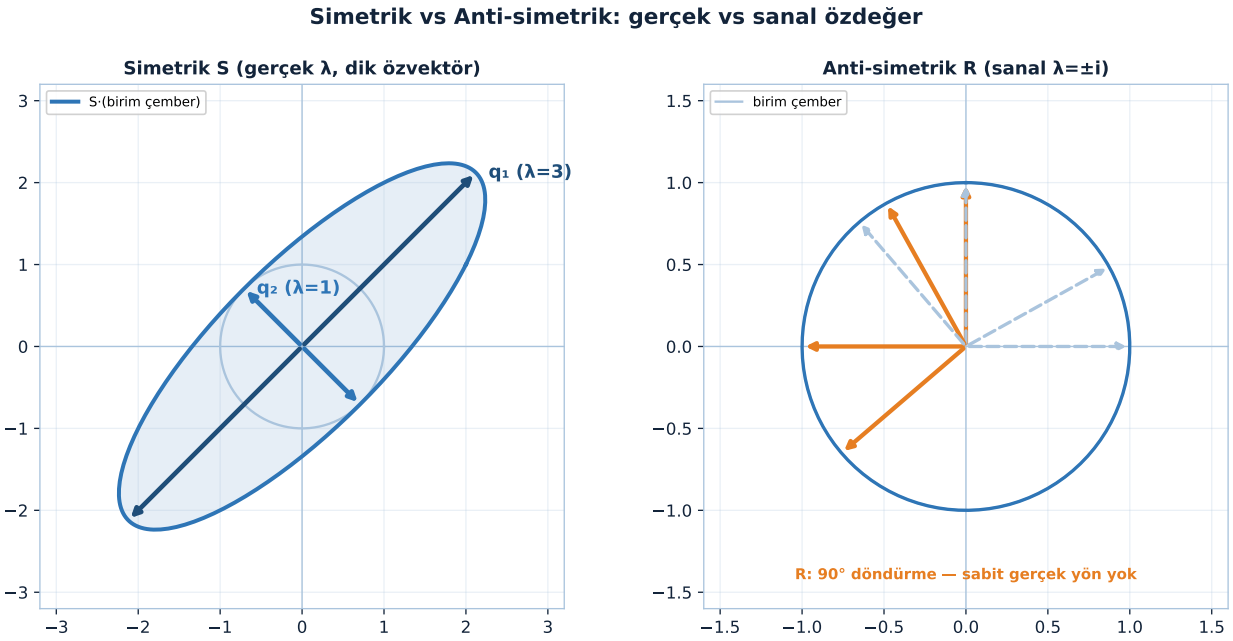
Şimdi simetrik matrise ($S = S^T$) özelleşelim. İki garanti: özdeğerler **gerçek**, özvektörler **ortogonal**. Karşıtını görmek için anti-simetrik matrise ($A^T = -A$) bak — onun özdeğerleri tamamen sanaldır. Klasik örnek 90° döndürme:

$$A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad \det(A - \lambda I) = \lambda^2 + 1 = 0 \;\Rightarrow\; \lambda = \pm i$$

> *"…have imaginary eigenvalues."* — Strang, 25:51

90° döndürme hiçbir gerçek vektörü kendi doğrultusunda bırakmaz, o yüzden gerçek özvektörü yoktur. Simetrik matrisin gerçek özdeğer garantisi, tam da bunun olamayacağını söyler.

Bu karşıtlık Şekil 11.4'de yan yana: solda simetrik S dik özvektör eksenleriyle bir elips gerer, sağda anti-simetrik R her vektörü döndürür ve hiçbir gerçek yönü sabit bırakmaz.



Şekil 11.4: Simetrik vs anti-simetrik matris: özdeğerlerin doğası geometriye yansır. Solda simetrik S = [[2,1],[1,2]] birim çemberi bir elipse gerer; iki ortonormal özvektör (q_1 λ=3, q_2 λ=1) dik eksenler oluşturur — gerçek özdeğerler, sabit gerçek yönler. Sağda anti-simetrik R = [[0,-1],[1,0]] her vektörü 90° döndürür; çember çember kalır ve hiçbir gerçek yön sabit kalmaz çünkü özdeğerler sanaldır (λ=±i).

> **Builder Notu — Güvenli Matrisler**
>
> Simetrik → gerçek özdeğer + ortonormal özvektör garantisi, ML'nin temel matrislerini (kovaryans, Gram, Hessian, graf Laplacian) güvenli kılar: özdeğerler yorumlanabilir gerçek sayılardır, özvektörler dik bir taban verir. Anti-simetrik kısım ise döndürmeyi (faz) kodlar.

11.10 2×2 Kontrolleri: Trace ve Determinant

Elle özdeğer hesabında iki hızlı sağlama var. Özdeğerlerin toplamı, köşegenin toplamına (trace) eşittir; çarpımları determinanta eşittir:

$$\text{trace}(A) = \sum_i \lambda_i, \qquad \det(A) = \prod_i \lambda_i$$

Anti-simetrik örnekte: trace $= 0 + 0 = 0 = i + (-i)$ ✓; det $= 1 = i \cdot (-i)$ ✓.

> *"…this number adding the diagonal is called the trace."* — Strang, 31:03

Bu iki kontrol her boyutta toplam/çarpım için geçerli; 2×2'de özdeğerleri tamamen belirler (iki denklem, iki bilinmeyen).

İki sağlamanın somut bir örnekte nasıl tuttuğu Şekil 11.5'te görülüyor: $A = \begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix}$ için özdeğerler 5 ve 2, iz 7, determinant 10.

> **Builder Notu — Toplu Sağlama**
>
> Trace ve determinant ML düzenlileştirmesinde doğrudan görünür: nuclear norm (tekil değerler toplamı), log-det terimi (Gauss olabilirliği, determinantal point process), ve $trace(A^{T}A) = $ Frobenius norm^2. Bu kimlikler özdeğerleri tek tek hesaplamadan toplu bilgi verir.

11.11 Köşegenleştirme: A = XΛX^{-1}

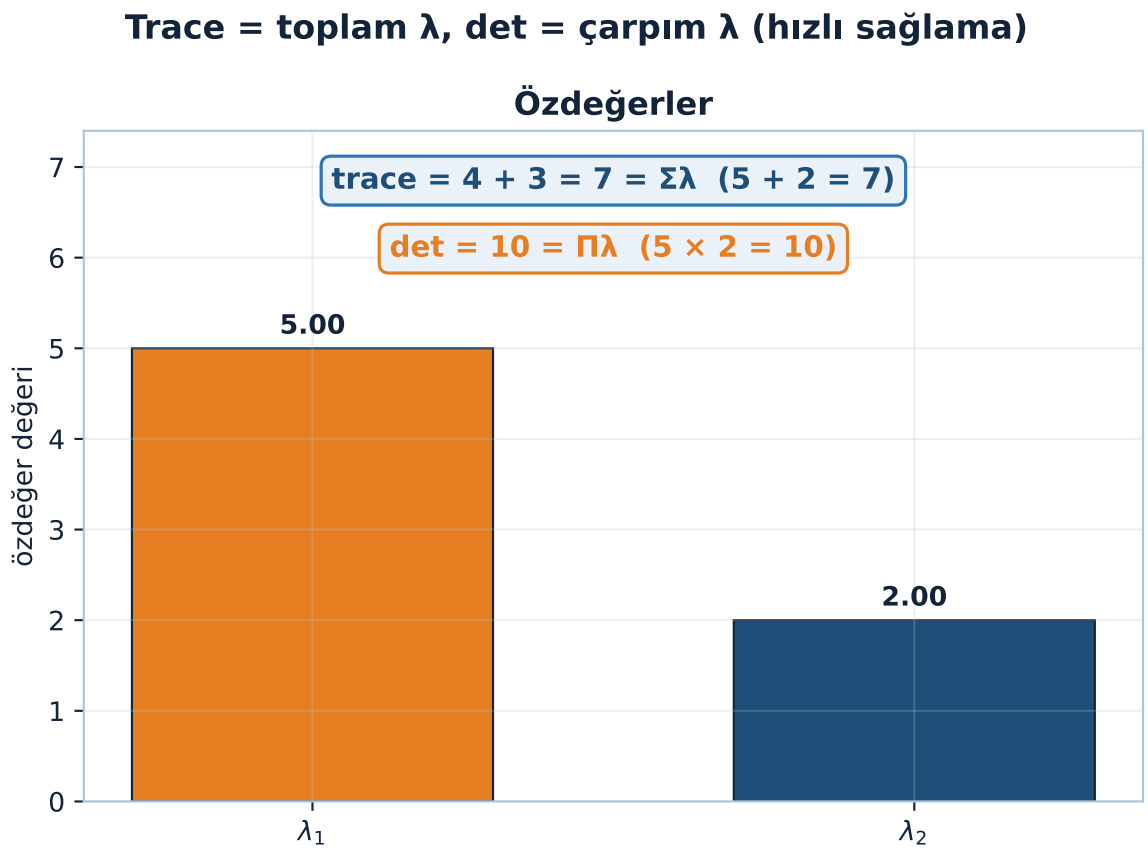
Tüm özvektörleri X matrisinin kolonlarına, özdeğerleri köşegen Λ'ya koy. $AX = X\Lambda$ ilişkisi şu kompakt forma dönüşür:

$$AX = X\Lambda \quad \Longrightarrow \quad A = X\Lambda X^{-1}$$

> *"…A equal x lambda x inverse."* — Strang, 45:39

Bu, A'nın Λ'ya benzer olduğunu ($M = X$) söyler. Kuvvet almak artık çok kolay — ortadaki $X^{-1}X = I$ düşer:

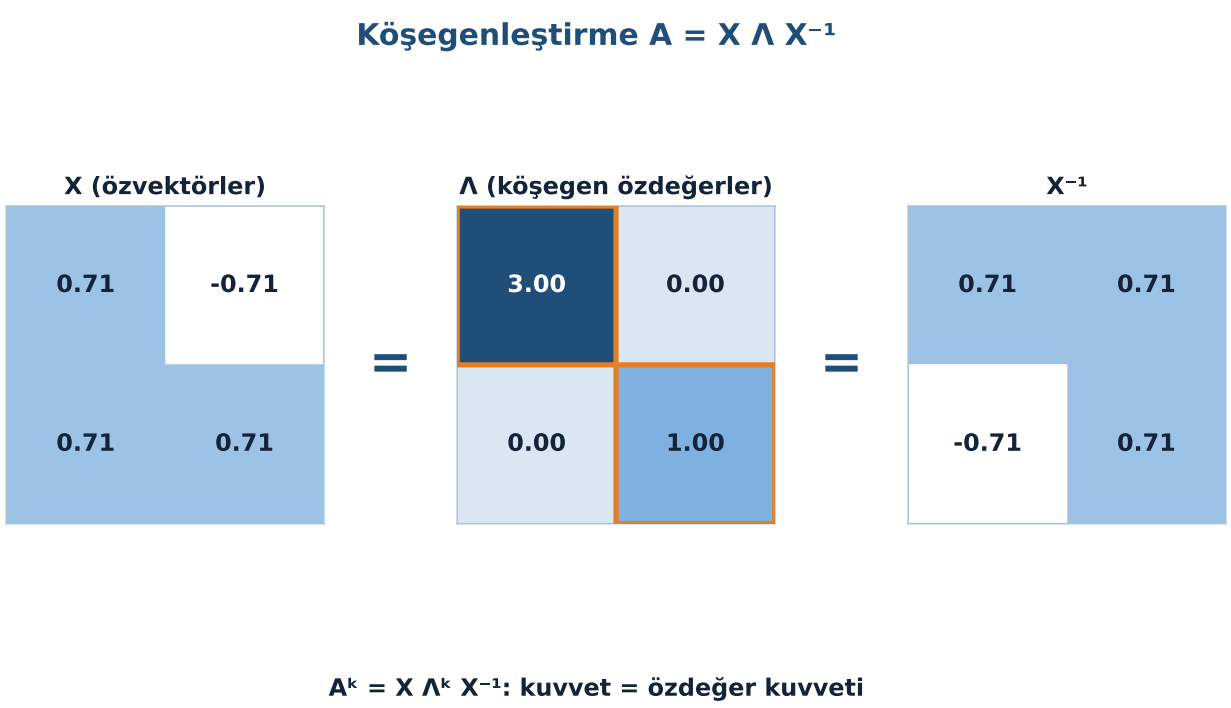
$$A^2 = X\Lambda X^{-1} \cdot X\Lambda X^{-1} = X\Lambda^2 X^{-1}, \quad A^k = X\Lambda^k X^{-1}$$



Şekil 11.5: $A = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix}$ için özdeğerler $\lambda_1 = 5,\ \lambda_2 = 2$. Karakteristik denklemi çözmeden iki hızlı sağlama: iz trace $= 4 + 3 = 7 = \sum \lambda$ ve determinant $\det = 10 = \prod \lambda$.

> *"…diagonalizing a matrix."* — Strang, 39:57

$A = X\Lambda X^{-1}$ ayrışmasının üçlü yapısı (özvektör X, köşegen Λ, ters X^{-1}) Şekil 11.6'da görülüyor: köşegen Λ vurgulanmış, ve kuvvet almanın neden Λ^k'ya indirgendiği netleşiyor.



Şekil 11.6: Köşegenleştirme A = XΛX^{-1}: A = [[2,1],[1,2]] matrisi, özvektörleri sütun olarak taşıyan X, özdeğerleri köşegene yerleştiren Λ (turuncu çerçeveyle vurgulanmış köşegen) ve X^{-1} üçlüsü olarak yazılır. Köşegen yapı kuvvet almayı kolaylaştırır: A^k = XΛ^{k}X^{-1}, yani matrisin kuvveti özdeğerlerin kuvvetine indirgenir.

> **Builder Notu — Kuvvetin Kapalı Formu**
>
> $A = X\Lambda X^{-1}$, lineer dinamiğin kapalı formudur: A^k'yı doğrudan hesaplamak yerine özdeğerleri kuvvetlersin. Diyagonalleştirilemeyen (defektif) matrisler bu yüzden tehlikelidir — SVD (Ders 6) bu sorunu olmayan evrensel alternatiftir.

11.12 Spektral Teorem: S = QΛQ^T

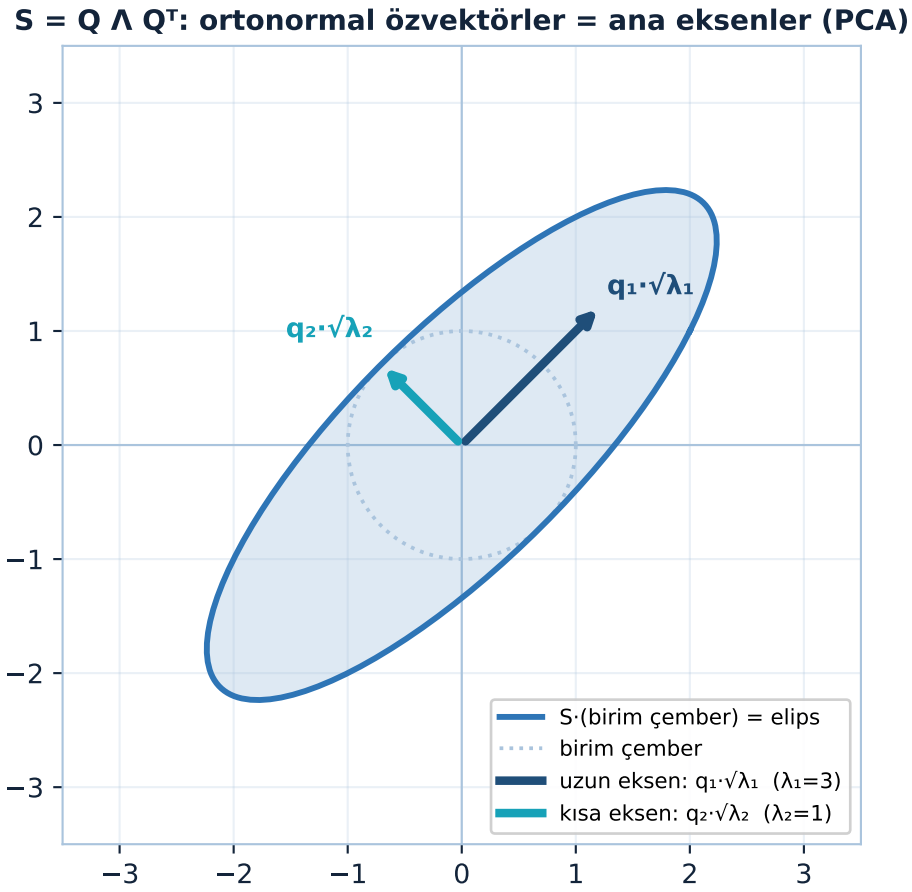
Matris simetrikse köşegenleştirme en güzel hâlini alır: özvektör matrisi X, ortogonal bir Q olur (özvektörler ortonormal), dolayısıyla $X^{-1} = Q^T$:

$$S = Q\Lambda Q^T, \qquad Q^{-1} = Q^T$$

> *"…this has the name spectral theorem."* — Strang, 48:27 *"…orthogonal eigenvectors, real eigenvalues."* — Strang, 48:46

Bu, Ders 2'de tanıttığımız spektral teoremin tam türetilişidir ve Ders 3'ün "simetrik → ortonormal özvektör" gözlemini kapatır. Her simetrik matris böyle görünür: gerçek özdeğerler (Λ) + ortonormal özvektörler (Q).

Spektral ayrışmanın geometrik anlamı Şekil 11.7'te görülüyor: ortonormal özvektörler q_1, q_2 elipsin ana eksenleridir, yarı-eksen uzunlukları özdeğerlerin karekökleriyle ölçeklenir — PCA'nın tam resmi.



Şekil 11.7: Simetrik $S = Q\Lambda Q^T$ spektral ayrışımı: ortonormal özvektörler q_1, q_2 elipsin (PCA) ana eksenleridir. Uzun eksen en büyük özdeğer ($\lambda_1 = 3$) yönünde, kısa eksen ise $\lambda_2 = 1$ yönünde uzanır; iki eksen diktir.

> **Builder Notu — PCA'nın Matematiği**
>
> $S = Q\Lambda Q^T$, PCA'nın matematiğidir: kovaryans matrisinin ortonormal özvektörleri (Q) ana bileşen eksenleri, özdeğerleri (Λ) o eksenlerdeki varyanslardır. Aynı ayrışım kernel PCA, spektral kümeleme (Ders 35) ve ikinci-derece optimizasyonun (Hessian) temelidir.

11.13 Bu Dersin Özeti

1. $Ax = \lambda x$ — özvektör yön değiştirmez, yalnızca λ ile ölçeklenir.
2. $A^k x = \lambda^k x$ — kuvvetler ve fonksiyonlar özvektör tabanında önemsizleşir.

3. $\lambda = 0 \rightarrow$ **tersinir değil** — özvektör null uzayında.
4. **Özvektör bazı** — $v = \sum_i c_i x_i$; fark/diferansiyel denklemleri çözer.
5. **Benzer matrisler** — $B = M^{-1}AM$, aynı özdeğerler.
6. AB **ve** BA — aynı sıfırdan-farklı özdeğerler.
7. **Simetrik** — gerçek özdeğer, ortogonal özvektör; **anti-simetrik** — sanal özdeğer.
8. **trace** $= \sum \lambda$**,** det $= \prod \lambda$ — hızlı sağlama.
9. $A = X\Lambda X^{-1}$ — köşegenleştirme; $A^k = X\Lambda^k X^{-1}$.
10. $S = Q\Lambda Q^T$ — spektral teorem (simetrik matris).

! Tek Bir Cümle

Özvektörler bir matrisin doğal eksenleridir — o eksenlerde matris yalnızca λ ile ölçekler; $A = X\Lambda X^{-1}$ bunu kullanıp her kuvvet ve fonksiyonu önemsizleştirir, ve simetrik matriste eksenler ortonormal olur ($S = Q\Lambda Q^T$).

11.14 Kontrol Soruları

i Soru 1: Aşağıdaki matrisin özdeğer ve özvektörlerini bul.

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Cevap:
$\det(A - \lambda I) = (2-\lambda)^2 - 1 = \lambda^2 - 4\lambda + 3 = (\lambda - 3)(\lambda - 1) \rightarrow \lambda = 3, 1$.
$\lambda = 3$: $(A - 3I)x = 0 \rightarrow (-1, 1)$ satırı $\rightarrow$ özvektör $(1, 1)$. $\lambda = 1$: özvektör $(1, -1)$. İki özvektör diktir (A simetrik olduğu için ✓).

i Soru 2: Aşağıdaki matrisin özdeğerlerini bul ve trace = Σλ, det = Πλ ile sağla.

$$A = \begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix}$$

Cevap:
trace $= 4+3 = 7$, det $= 4 \cdot 3 - 1 \cdot 2 = 10$. Karakteristik: $\lambda^2 - 7\lambda + 10 = (\lambda - 5)(\lambda - 2) \rightarrow \lambda = 5, 2$.
Sağlama: $5 + 2 = 7 = $ trace ✓; $5 \ 2 = 10 = $ det ✓. (Bu A simetrik değil, özvektörleri dik olmayabilir.)

i Soru 3: 90° döndürme matrisinin özdeğerleri neden gerçek değil? Hangi matris türü gerçek özdeğer garantiler?

$$A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

Cevap:
$\det(A - \lambda I) = \lambda^2 + 1 = 0 \rightarrow \lambda = \pm i$ (sanal). Döndürme hiçbir gerçek vektörü kendi doğrultusunda bırakmaz, gerçek özvektör yoktur. A anti-simetriktir ($A^T = -A$). **Simetrik** matrisler ($S = S^T$) ise her zaman gerçek özdeğer ve ortonormal özvektör garantiler (Ders 3'le bağ).

Soru 4: Bir tekrarlı sistem $v_k = A^k v_0$'da uzun-vadeli davranışı neden en büyük $|\lambda|$ belirler?

Cevap:
v_0'ı özvektör tabanında yaz: $v_0 = \sum_i c_i x_i$. O zaman $v_k = A^k v_0 = \sum_i c_i \lambda_i^k x_i$. k büyüdükçe en büyük $|\lambda|$'lı terim diğerlerini ezer (λ_i^k oranları 0'a veya ∞'a gider).
Sonuç: $|\lambda_{\max}| > 1 \to$ patlar, $|\lambda_{\max}| < 1 \to$ söner, $= 1 \to$ kararlı. Power iteration tam bunu kullanır (baskın özvektöre yakınsar); RNN'lerde gradyan patlama/sönmesi ve Markov zincirinin denge dağılımı aynı ilkeye dayanır.

11.15 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. Aşağıdaki matrisin özdeğerlerini ve özvektörlerini bul. Trace ve determinant ile sağla.

$$A = \begin{pmatrix} 5 & 4 \\ 1 & 2 \end{pmatrix}$$

Egzersiz 2. $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ matrisini $A = X\Lambda X^{-1}$ olarak köşegenleştir, sonra A^4'ü $X\Lambda^4 X^{-1}$ ile hesapla (doğrudan A^4 çarpımına gerek kalmadan).

Egzersiz 3. $A = \text{diag}(2, 3)$ ve $M = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ için $B = M^{-1}AM$ hesapla. B'nin özdeğerlerinin hâlâ 2 ve 3 olduğunu (benzerlik) doğrula; özvektörlerin değiştiğini göster.

Egzersiz 4. Python ile doğrula:

```
import numpy as np

A = np.array([[2.0, 1.0], [1.0, 2.0]])
lam, X = np.linalg.eig(A)
print("özdeğerler:", lam)              # 3, 1
print("trace =", np.trace(A), " Σλ =", lam.sum())     # eşit
print("det =", np.linalg.det(A), " Πλ =", lam.prod()) # eşit

# A = X Λ X^{-1} doğrula:
Lam = np.diag(lam)
print("A rebuild:", X @ Lam @ np.linalg.inv(X))

# A^4 = X Λ^4 X^{-1}:
print("A^4 (eig):", X @ np.diag(lam**4) @ np.linalg.inv(X))
print("A^4 (direct):", np.linalg.matrix_power(A, 4))
```

Egzersiz 5. (Ders 5 habercisi.) Ders 5 pozitif tanımlı matrislere geçer — tüm özdeğerleri pozitif olan simetrik matrisler. Aşağıdaki simetrik matrisin özdeğerlerini bul; hepsinin pozitif olup olmadığını kontrol et (pozitif tanımlı mı?).

$$S = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$$

11.16 Sonraki Ders İçin Hazırlık

Ders 5: Pozitif Tanımlı ve Yarı-Tanımlı Matrisler

Ders 4'te simetrik matrisin gerçek özdeğerli ve ortonormal özvektörlü olduğunu gördük. Ders 5, simetrik matrislerin en güzel ailesine odaklanır: **pozitif tanımlı** olanlar.

- Pozitif tanımlılık: tüm özdeğerler > 0; eşdeğer testler (pivotlar, alt-determinantlar, $x^T Sx > 0$)
- Yarı-tanımlı: özdeğerler ≥ 0
- Neden ML'de kritik: kovaryans, kernel, Hessian, kayıp fonksiyonu eğriliği

> ⚠ Ders 5 Öncesi Yapılacak
>
> - Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (pozitif özdeğer kontrolü).
> - Python'da `np.linalg.eigvalsh` ile birkaç simetrik matrisin özdeğer işaretlerini incele.
> - Ana cümleyi tekrar oku: "Özvektörler matrisin doğal eksenleri; $A = X\Lambda X^{-1}$."

11.17 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
$Ax = \lambda x$	Özvektör yön değiştirmez, λ ile ölçeklenir	1m51
$A^k x = \lambda^k x$	Kuvvetler/fonksiyonlar özvektör tabanında kolay	4m28
$\lambda = 0$	Matris tersinir değil; özvektör null uzayında	5m22
Özvektör bazı	$v = \sum_i c_i x_i$; fark/diferansiyel denklem çözer	10m38
Benzer matris	$B = M^{-1}AM$, aynı özdeğerler	13m56
eig algoritması	M'lerle üçgensele/köşegene taşıma	16m13
AB **ve** BA	Aynı sıfırdan-farklı özdeğerler	19m58
Anti-simetrik	$A^T = -A$; sanal özdeğerler (döndürme)	25m51
trace $= \sum \lambda$	Köşegen toplamı; det $= \prod \lambda$	31m03
$A = X\Lambda X^{-1}$	Köşegenleştirme; $A^k = X\Lambda^k X^{-1}$	45m39

Kavram	Tanım	Strang'de
$S = Q\Lambda Q^T$	Spektral teorem (simetrik matris)	48m27

11.18 ML Bağlantıları Özeti

1. $Ax = \lambda x \rightarrow$ PCA; veri kovaryansının doğal eksenleri (özvektörler) ana bileşenler.
2. $A^k x = \lambda^k x \rightarrow$ tekrarlı sistem kararlılığı; RNN gradyan patlama/sönmesi, power iteration, Markov dengesi.
3. $\lambda = 0$ **/ küçük** $\lambda \rightarrow$ kötü koşullanma; ridge/Tikhonov düzenlileştirmesi (λI ekleme) tam bunu giderir.
4. AB **ve** BA **aynı** $\lambda \rightarrow$ Gram matrisi numarası; XX^T ile $X^T X$'ten küçüğünü hesapla (kernel trick, SVD).
5. **Trace/determinant** $\rightarrow$ nuclear norm, log-det olabilirlik, Frobenius norm^2 = trace$(A^T A)$.
6. $A = X\Lambda X^{-1} \rightarrow$ lineer dinamiğin kapalı formu; defektif matris riski $\rightarrow$ SVD'ye geçiş (Ders 6).
7. $S = Q\Lambda Q^T \rightarrow$ PCA, kernel PCA, spektral kümeleme, Hessian analizi.

> **!** Tek bir şey alıp gideceksen
>
> Özvektörler bir matrisin doğal eksenleridir — o eksenlerde matris döndürmez, yalnızca λ ile gerer. $A = X\Lambda X^{-1}$ bu görüşü kapsüller ve her kuvvet/fonksiyon hesabını önemsizleştirir; simetrik matriste eksenler ortonormal olur ($S = Q\Lambda Q^T$), ki bu PCA'dan kararlılık analizine kadar ML'in çatısıdır.

12 Pozitif Tanımlı ve Yarı-Tanımlı Matrisler

$x^T Sx > 0$: beş eşdeğer test, enerji-kâsesi ve gradient descent'in zemini

> **Bölüm bilgisi**
>
> - **Video:** Positive Definite and Semidefinite Matrices — Gilbert Strang, MIT 18.065
> - **OCW:** Lecture 5 — Positive Definite and Semidefinite Matrices
> - **Okuma süresi:** ≈ 35 dk
> - **Önkoşul:** Ders 4 (özdeğer/özvektör, spektral teorem $S = Q\Lambda Q^T$, "simetrik → gerçek özdeğer / ortonormal özvektör")

12.1 Bu Derste Ne Var?

Bu ders, lineer cebir "highlights" turunu kapatıyor (beş ders) ve hepsini birbirine bağlıyor: özdeğer, enerji, $A^T A$, determinant, pivot — her biri **aynı** şeyin (pozitif tanımlılık) bir testidir. Strang ayrıca buradan derin öğrenmenin kalbine, **gradient descent**'e bir köprü atıyor.

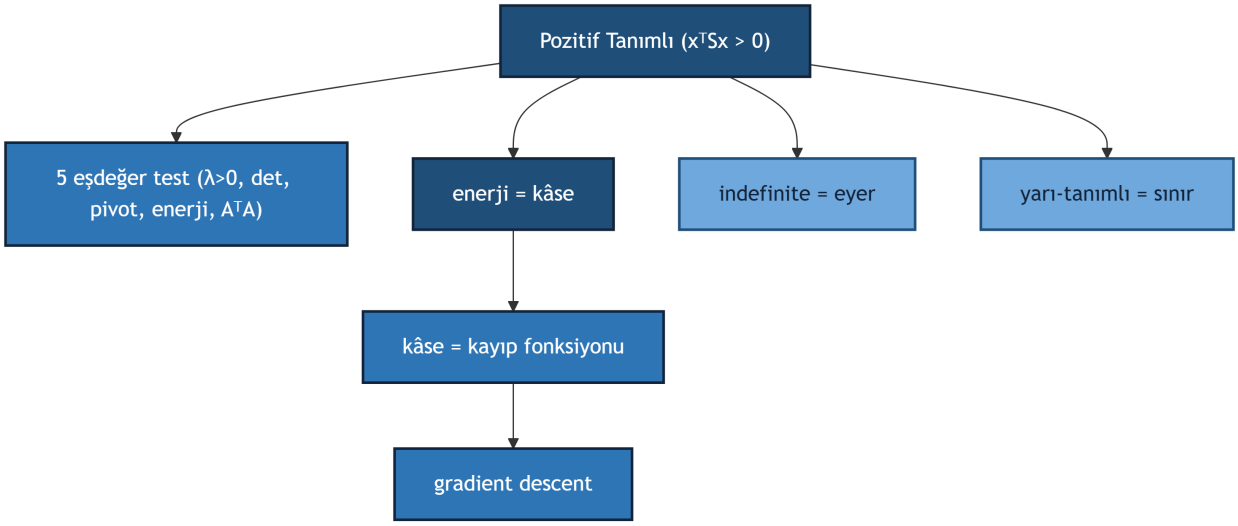
Üç temel fikir:

1. **Pozitif tanımlı = simetrik + tüm özdeğerler > 0** — ve buna eşdeğer beş test (özdeğer, leading determinant, pivot, enerji $x^T Sx$, $A^T A$).
2. **Enerji $x^T Sx > 0$ = kâse (bowl)** — pozitif tanımlı matris, yukarı açılan dışbükey bir yüzey verir; bu, bir kayıp fonksiyonudur.
3. **Gradient descent** — kâsenin dibini bulmak: en dik iniş yönünü (gradyan) izle; dar vadi sorunu özdeğer oranından gelir.

> *"…positive definite matrices. These are the best of the symmetric matrices."* — Strang, 3:10

Bu üç fikrin nasıl tek bir merkezden ($x^T Sx > 0$) dallandığını Şekil 12.1 özetliyor: beş eşdeğer testten enerji-kâsesine, oradan kayıp fonksiyonu ve gradient descent'e, ve sınır durumlara (eyer, yarı-tanımlı) kadar dersin bütün hatları.

> **Builder Notu — Optimizasyonun Zemini**
>
> - **Enerji $x^T Sx$ = kayıp fonksiyonu modeli:** derin öğrenmenin büyük hesabı bir enerjiyi minimize etmektir; pozitif tanımlı Hessian = yerel dışbükey kâse.
> - **Gradient descent** burada ilk kez tanıtılıyor (Ders 21-23'te derinleşir): birinci türevleri izle, ikinci türev (Hessian) hesaplama pahalı olduğundan kaçınılır.



Şekil 12.1: Ders 5 kavram haritası: pozitif tanımlılık ($x^T Sx > 0$) merkezinden enerji-kâsesi, gradient descent ve sınır durumlara (eyer, yarı-tanımlı) uzanan dallar.

- **Özdeğer oranı (kondisyon sayısı)** = kâsenin biçimi; dar vadi (büyük $\lambda_{\max}/\lambda_{\min}$) gradient descent'i yavaşlatır — momentum ve ön-koşullama (Ders 23) bunu düzeltir.
- $A^T A$ **her zaman pozitif yarı-tanımlı** — kovaryans, Gram, kernel matrislerinin neden hep "kâse" verdiğinin sebebi.

Tek cümle: pozitif tanımlılık, bir simetrik matrisin "yukarı açılan kâse" olduğunun beş eşdeğer testidir — ve bu kâse, tüm optimizasyonun ve derin öğrenmenin zeminidir.

12.2 Pozitif Tanımlı: Beş Eşdeğer Test

Pozitif tanımlı (PD) matris, **simetrik** ve tüm özdeğerleri **pozitif** olan matristir. Bunlar simetrik matrislerin en iyileridir.

> *"…positive definite matrices. These are the best of the symmetric matrices."* — Strang, 3:10

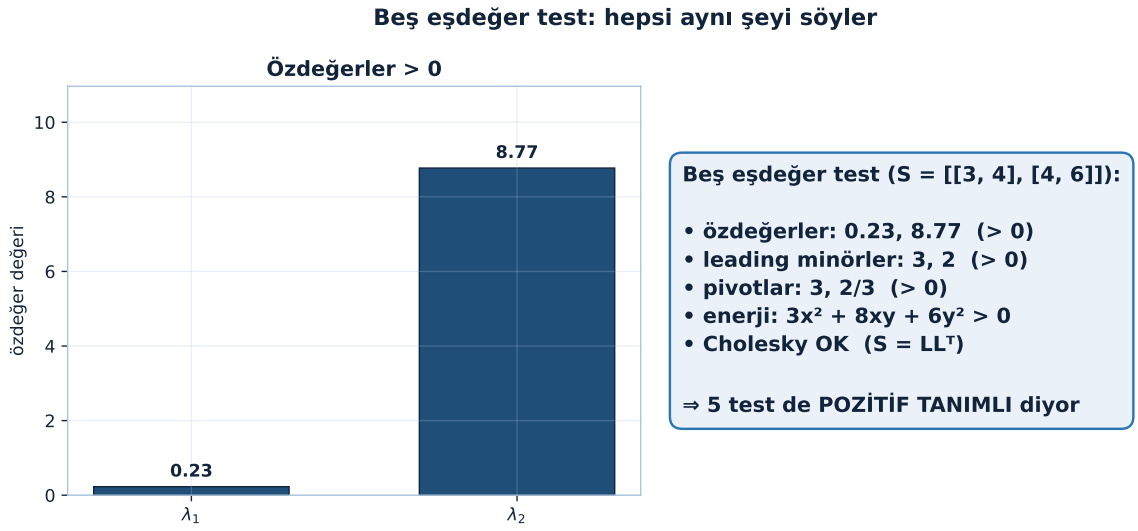
Özdeğer hesabı zor olduğundan beş eşdeğer test vardır — herhangi biri geçerse hepsi geçer:

1. Tüm özdeğerler $\lambda_i > 0$.
2. Tüm leading (sol-üst) alt-determinantlar > 0.
3. Tüm pivotlar > 0.
4. Enerji her $x \neq 0$ için pozitif: $x^T Sx > 0$.
5. $S = A^T A$ biçiminde, A'nın bağımsız kolonlarıyla yazılabilir.

Strang'e göre asıl tanım dördüncüsüdür (enerji):

$$x^T Sx > 0 \quad (\text{her } x \neq 0)$$

$S = \begin{pmatrix} 3 & 4 \\ 4 & 6 \end{pmatrix}$ için bu beş testi tek bir bakışta Şekil 12.2 gösteriyor: özdeğerler, leading minörler, pivotlar, enerji ve Cholesky — hepsi aynı kararı verir.



Şekil 12.2: $S = \begin{bmatrix} 3 & 4 \\ 4 & 6 \end{bmatrix}$ pozitif tanımlı: beş eşdeğer test de aynı kararı verir. Sol panel özdeğerleri ($\lambda_1 \approx 0.23,\ \lambda_2 \approx 8.77$, ikisi de > 0), sağ panel diğer dört testi gösterir: leading minörler $3, 2$; pivotlar $3, \frac{2}{3}$; enerji $3x^2 + 8xy + 6y^2 > 0$; Cholesky $S = LL^T$ başarılı.

> **Builder Notu — Beş Kapı, Tek Oda**
>
> Beş test, beş farklı kütüphane çağrısına karşılık gelir: `eigvalsh` (özdeğer), `cholesky` (pivot/A^TA), determinant. Pratikte pozitif tanımlılık Cholesky'nin başarısıyla test edilir — başarısızsa matris PD değildir, ucuz bir kontrol.

12.3 İndefinit Örnek

Hepsi pozitif sayılardan oluşan bir matris bile PD olmayabilir. Örnek:

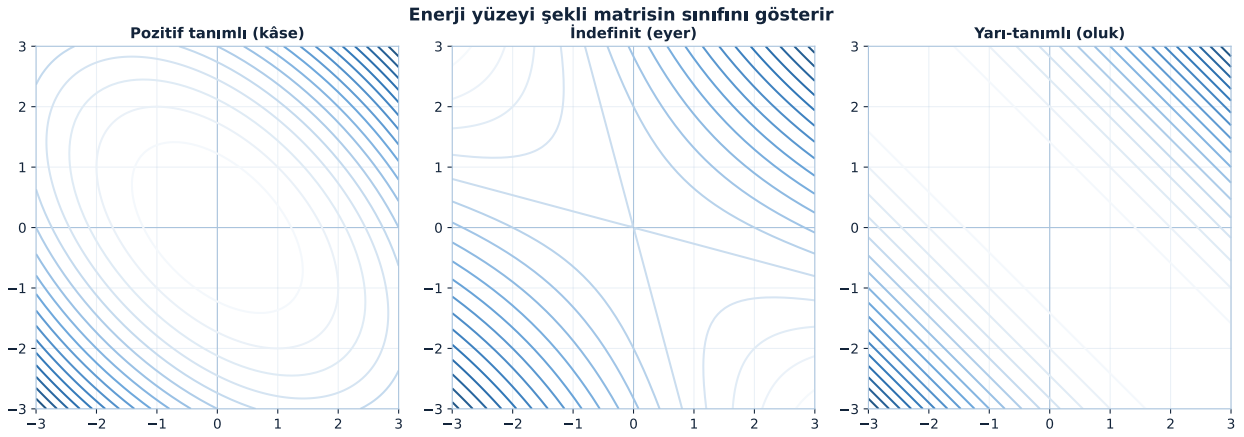
$$S = \begin{pmatrix} 3 & 4 \\ 4 & 5 \end{pmatrix}, \quad \det S = 15 - 16 = -1$$

Determinant özdeğerlerin çarpımıdır; negatif olması bir özdeğerin negatif olduğunu söyler. Yani bir özdeğer pozitif, biri negatif — bu **indefinit** (belirsiz) matristir.

> *"This matrix is an indefinite matrix."* — Strang, 5:31

Köşegen girdiyi artırmak ($5 \to 6$) matrisi pozitif tanımlı yapar; köşegene değer eklemek daima "daha pozitif" yönüne iter.

İndefinit, pozitif tanımlı ve yarı-tanımlı yüzeylerin nasıl farklılaştığını Şekil 12.3 üç panelde yan yana koyuyor: kâse, eyer ve oluk.



Şekil 12.3: Enerji $f(x) = x^T Sx$ yüzeyinin şekli matrisin sınıfını ele verir: **pozitif tanımlı** $S = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ iç içe elipsler çizer (yukarı açılan kâse, tek minimum); **indefinit** $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ hiperbolik konturlar verir (eyer noktası); **yarı-tanımlı** $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ paralel çizgiler çizer (düz bir oluk — bir yönde enerji sıfır).

> **Builder Notu — Eyer Tuzağı**
>
> İndefinit matris = eyer noktalı (saddle) yüzey: bazı yönlerde yukarı, bazılarında aşağı. Derin öğrenmede kayıp yüzeyinin eyer noktaları (Ders 19) tam da indefinit Hessian'a karşılık gelir — gradient descent'in takılabildiği yerler.

12.4 Leading Determinant Testi

İkinci test, sol-üst köşeden başlayan iç içe alt-matrislerin determinantlarına bakar. Pozitif tanımlı $\begin{pmatrix} 3 & 4 \\ 4 & 6 \end{pmatrix}$ için:

$$\det(3) = 3 > 0, \quad \det \begin{pmatrix} 3 & 4 \\ 4 & 6 \end{pmatrix} = 18 - 16 = 2 > 0$$

Yalnızca tam determinanta bakmak yetmez: $\begin{pmatrix} -3 & 4 \\ 4 & -6 \end{pmatrix}$ da determinantı 2 verir ama 1×1 testini ($-3 < 0$) geçemez. Bu yüzden n özdeğer için n leading determinantın **hepsi** kontrol edilir.

> **Builder Notu — Erken Uyarı**
>
> Leading minör testi, kovaryans matrislerinin geçerliliğini doğrulamada kullanılır. Negatif bir leading minör, tahmin edilen kovaryansın bozuk (PD değil) olduğunu erken yakalar — Gauss süreçleri ve Kalman filtrelerinde sayısal sağlık kontrolü.

12.5 Pivot Testi

Üçüncü test eliminasyon pivotlarına bakar. $\begin{pmatrix} 3 & 4 \\ 4 & 6 \end{pmatrix}$ için ilk pivot 3'tür; ikinci satırdan $4/3$ çarpı birinci satırı çıkarınca ikinci pivot ortaya çıkar:

$$\text{pivot}_1 = 3, \qquad \text{pivot}_2 = \frac{\det_2}{\det_1} = \frac{2}{3}$$

Her iki pivot da pozitif → matris PD. Güzel bağ: her pivot, ardışık leading determinantların oranıdır ($2/3 = 2 \div 3$).

> *"So the pivots… are the 3 and the 2/3."* — Strang, 8:43

> **Builder Notu — Cholesky'nin Sırrı**
>
> Pivotların pozitifliği = Cholesky ayrışımının ($S = LL^T$) var olması. Cholesky, PD matrisleri çözmenin en hızlı ve kararlı yoludur; Bayesian çıkarım, Gauss süreçleri ve normal denklemler bunu kullanır. Bir pivot ≤ 0 çıkarsa Cholesky çöker — matris PD değildir.

12.6 Enerji Testi: $x^T Sx > 0$ (Kâse)

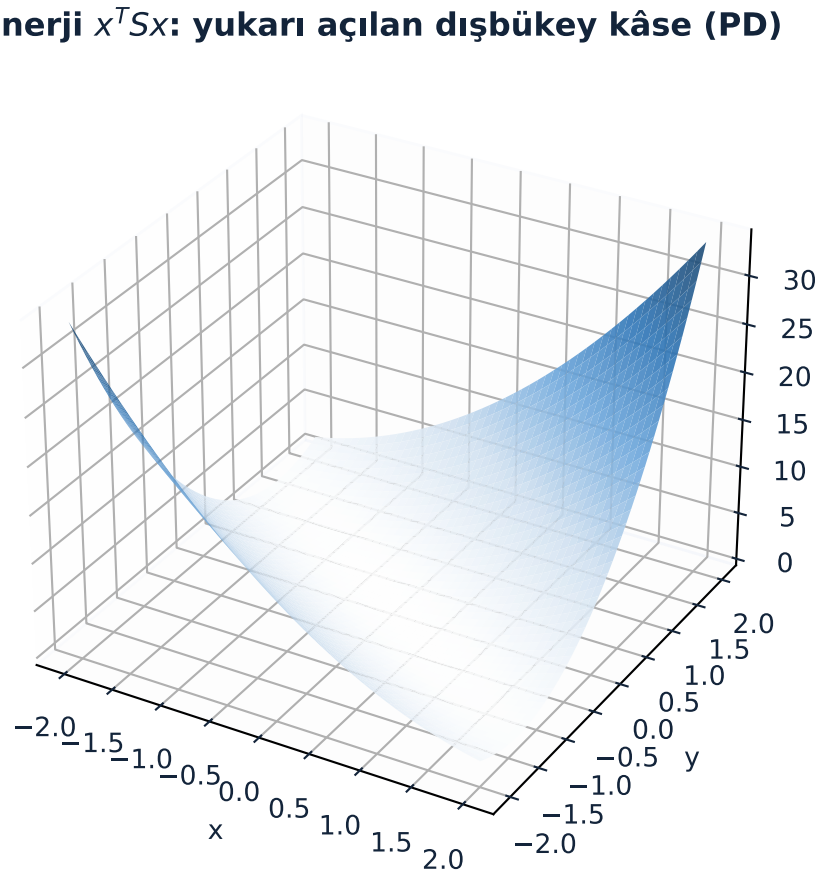
Strang'in asıl tanımı enerjidir. $\begin{pmatrix} 3 & 4 \\ 4 & 6 \end{pmatrix}$ matrisinin enerjisini açalım:

$$x^T Sx = 3x^2 + 8xy + 6y^2$$

Köşegen girdiler $(3, 6)$ kareli terimleri (hep pozitif), köşegen-dışı $(4 + 4 = 8)$ çapraz terimi verir. Çapraz terim negatife dönebilir ama kareli terimler onu bastırırsa fonksiyon her yerde pozitif kalır — grafik yukarı açılan bir **kâse**dir.

$$f(x, y) = 3x^2 + 8xy + 6y^2 > 0 \quad \text{for } (x, y) \neq (0, 0)$$

Bu yukarı açılan kâsenin üç boyutlu hâli Şekil 12.4'de görülüyor: tek bir minimum (orijin), her yöne tırmanan dışbükey bir yüzey.



Şekil 12.4: Enerji $x^T Sx$ ($S = \begin{bmatrix} 3 & 4 \\ 4 & 6 \end{bmatrix}$): yukarı açılan dışbükey kâse. Tek minimum orijinde — pozitif tanımlı (PD) matrisin imzası.

> **Builder Notu — İdeal Kayıp**
>
> $x^T Sx$ kuadratik formdur; PD ise grafiği dışbükey kâsedir, tek minimumu vardır (orijin). Bu kâse, ikinci-derece bir kayıp fonksiyonunun ta kendisidir — optimizasyonun "ideal" senaryosu.

12.7 Kâse = Kayıp Fonksiyonu

Strang burada lineer cebirden derin öğrenmeye köprü atıyor. O kâse, minimize edilen bir kayıp fonksiyonudur:

> *"This is what deep learning is about. This could be a loss function that you minimize."* — Strang, 14:25

Pratikte saf kareler olmaz; lineer terim ($-x^T b$) kâseyi merkezden kaydırır, lineer-olmayan terimler yüzeyi dalgalandırır. Ama temel model budur:

$$f(x) = \tfrac{1}{2} x^T Sx - x^T b$$

> *"…this is the model."* — Strang, 15:48

Bu fonksiyon dışbükey (convex): kuadratik için dışbükeylik = pozitif tanımlılık (veya sınırda yarı-tanımlılık). 100.000+ değişkenli kayıpların minimumunu bulmak, derin öğrenmenin en büyük hesabıdır.

> **Builder Notu — Eğitim Bu**
>
> "Kayıp = kâse" modeli, eğitimin neden işe yaradığının çekirdeğidir. Kayıp yerel olarak PD Hessian'a sahipse (dışbükey kâse), gradient descent dibe iner. Gerçek derin ağlarda yüzey dışbükey değildir (eyer noktaları, çoklu minimum) — ama yerel kâse sezgisi hâlâ rehberdir.

12.8 Gradient Descent

Kâsenin dibini (minimum) nasıl bulursun? Bir başlangıç noktasından, en dik iniş yönünü — gradyanın tersini — izle:

$$x_{k+1} = x_k - \eta \nabla f(x_k)$$

Gradyan ∇f tüm birinci türevlerin vektörüdür ve o noktada en hızlı çıkış yönünü gösterir; eksisi en hızlı iniştir. Adım boyu (η, learning rate) ve ne kadar gidileceği (line search) kritik kararlardır.

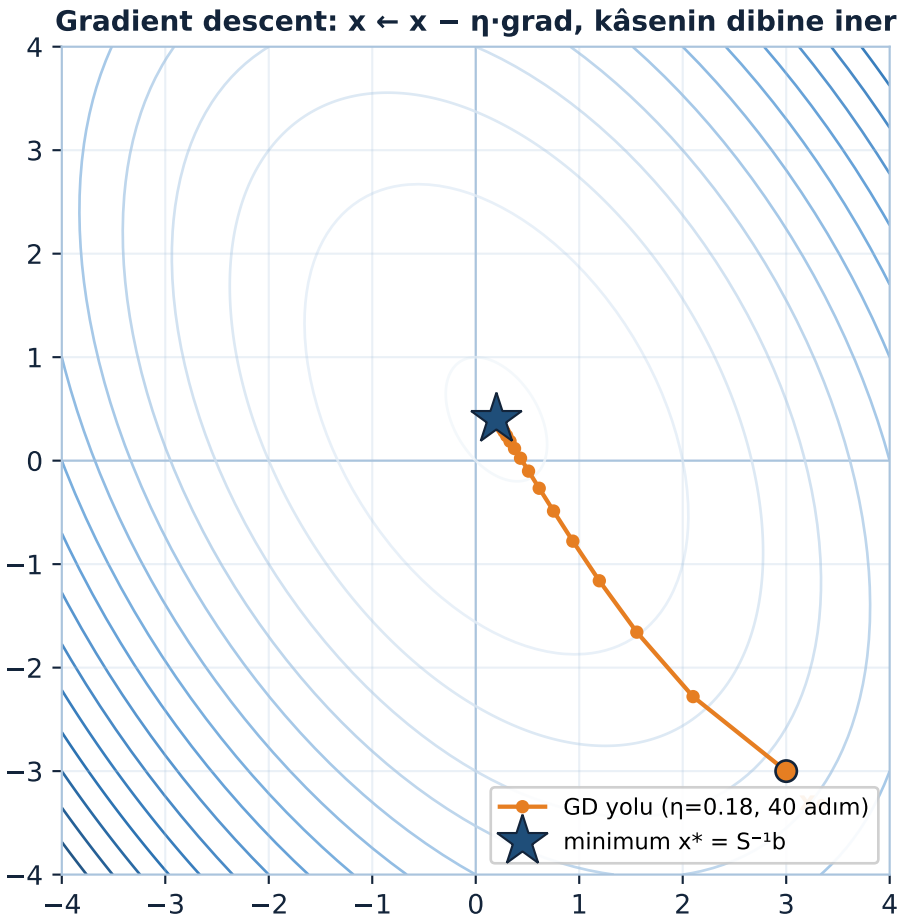
> *"…I would do a gradient descent."* — Strang, 22:09

Sorun: kâse dar bir vadiyse (bir özdeğer büyük, biri küçük), adımlar vadiyi enlemesine geçip karşı yamaca tırmanır; ileri-geri zikzak çizerek dibe çok yavaş yaklaşır.

> *"…if you're going down a narrow valley…"* — Strang, 26:30

Strang ikinci türev (Hessian) hesaplamaktan kaçınıldığını vurgular — 100.000+ değişkende çok pahalıdır; bu yüzden makine öğrenmesi çoğunlukla birinci türevle (gradyan) sınırlıdır.

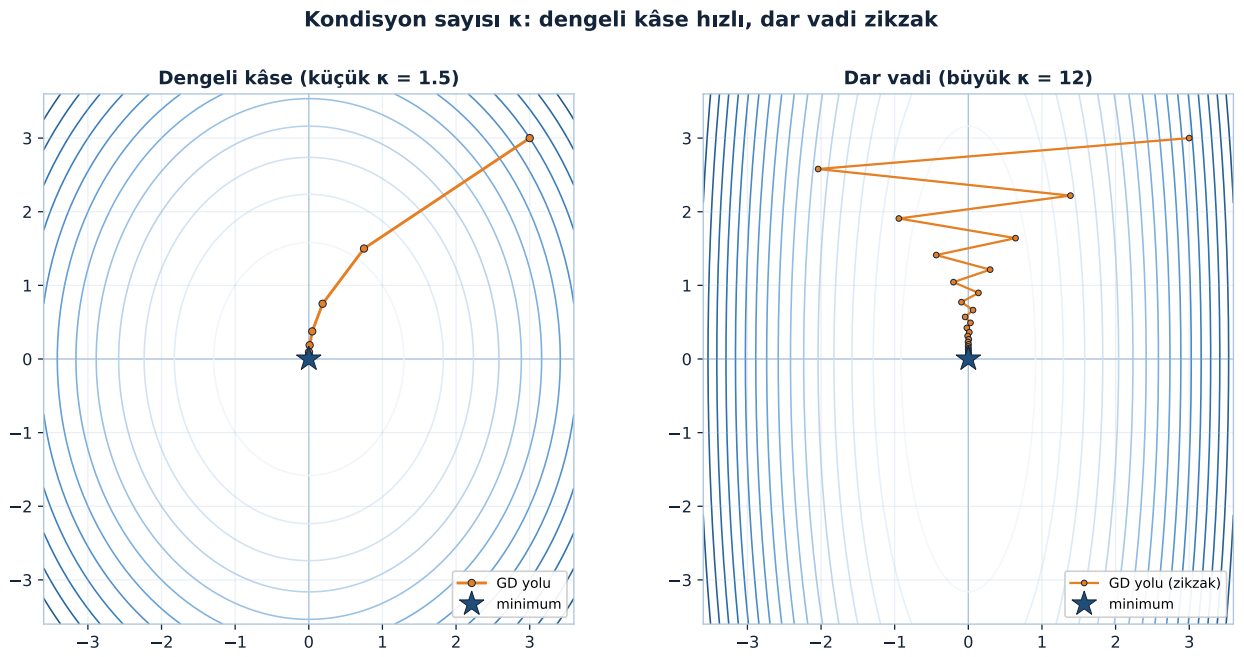
Gradient descent'in dışbükey kâsenin dibine $-\eta\nabla f$ adımlarıyla nasıl indiğini Şekil 12.5 gösteriyor: turuncu yol başlangıç noktasından $x^* = S^{-1}b$ minimumuna yakınsar.



Şekil 12.5: Gradient descent dışbükey kâsede: $f(x) = \frac{1}{2}x^T Sx - x^T b$ enerjisinin kontur çizgileri (S pozitif tanımlı) bir kâse oluşturur. $x_{k+1} = x_k - \eta\nabla f$ adımı her seferinde $-\nabla f$ yönünde, yani en dik iniş yönünde ilerler ve turuncu yol $x_0 = (3, -3)$ noktasından kâsenin dibindeki minimum $x^* = S^{-1}b$ (navy yıldız) noktasına yakınsar.

Bu zikzak sorununun kaynağı kondisyon sayısıdır: $\kappa = \lambda_{\max}/\lambda_{\min}$ büyüdükçe iniş yavaşlar. Şekil 12.6 dengeli kâse ile dar vadiyi yan yana koyarak bunu somutlaştırıyor.

> **Builder Notu — İnişin Merkezi**
>
> Bu, derin öğrenmenin merkezî algoritmasıdır (Ders 21-23'te momentum, SGD ile derinleşir). Dar vadi sorunu = yüksek kondisyon sayısı ($\lambda_{\max}/\lambda_{\min}$); momentum, Adam ve ön-koşullama (preconditioning) tam bu zikzakı azaltmak için icat edildi.



Şekil 12.6: Kondisyon sayısı $\kappa = \lambda_{\max}/\lambda_{\min}$ inişin hızını belirler. SOL: dengeli kâse ($\kappa = 1.5$), neredeyse dairesel eş-yükselti eğrileri; gradient descent birkaç adımda doğrudan dibe iner. SAĞ: dar vadi ($\kappa = 12$), uzun-ince elipsler; aynı yöntem vadinin duvarları arasında zikzak çizerek yavaş yakınsar.

12.9 Pozitif Tanımlı Matrislerle İşlemler

Beş test, PD matrislerle yapılan işlemleri kolayca yanıtlar:

- **S + T:** İki PD matrisin toplamı PD mi? Enerji testiyle anında: her x için $x^T(S+T)x = x^TSx + x^TTx > 0$. Evet.

$$x^T(S+T)x = x^TSx + x^TTx > 0$$

  "What about S plus T?" — Strang, 30:44

- **S^{-1}:** PD'nin tersi PD mi? Özdeğer testiyle: S'nin özdeğerleri $\lambda > 0$ ise S^{-1}'in özdeğerleri $1/\lambda > 0$. Evet.
- **Q^TSQ:** Q ortogonal ise Q^TSQ PD mi? Hem benzerlik (aynı özdeğer) hem enerji ($y = Qx$ ile $y^TSy > 0$) ile: Evet.

> **Builder Notu — Dışbükeyi Koru**
>
> "Doğru testi seç" becerisi ML'de işe yarar: $S+T$'nin PD'liği, iki dışbükey kaybın toplamının dışbükey olduğunu söyler (düzenlileştirme ekleme); S^{-1}'in PD'liği, ters kovaryans (precision matrix) matrisinin geçerliliğini garanti eder. Eigenvalue testi yerine enerji testi çoğu ispatı tek satıra indirir.

12.10 Yarı-Tanımlı (Semidefinite): Sınır Durum

Pozitif tanımlı ile indefinit arasındaki sınır yarı-tanımlıdır (positive semidefinite): özdeğerler ≥ 0, en az biri $= 0$. $\begin{pmatrix} 3 & 4 \\ 4 & 16/3 \end{pmatrix}$ tam sınırdadır:

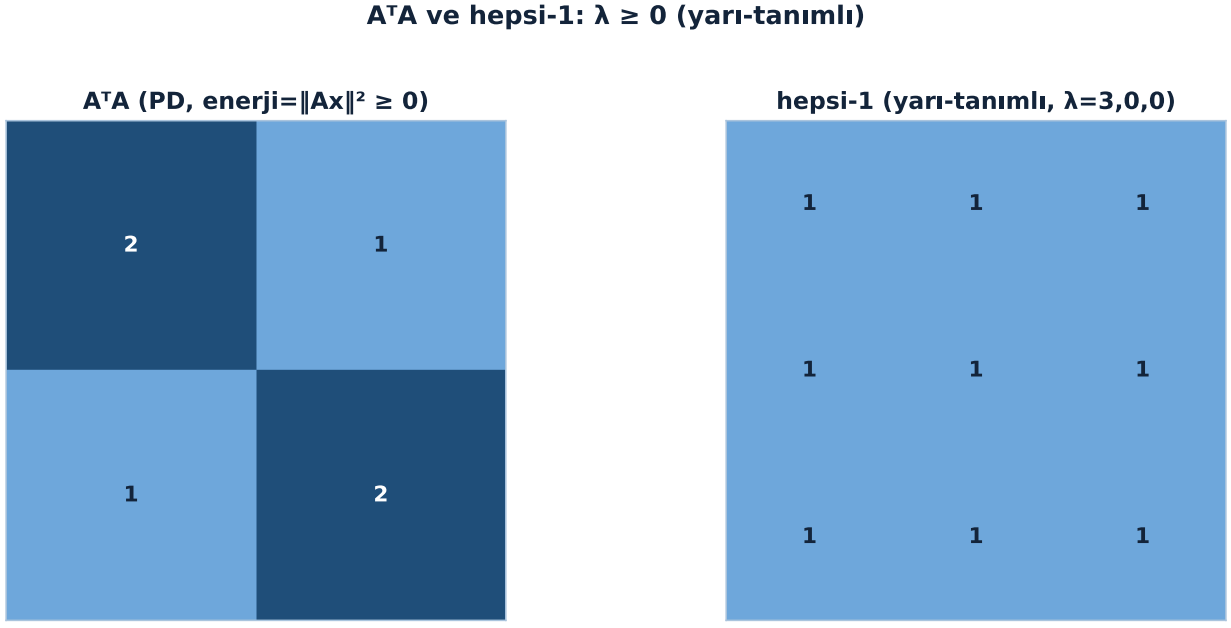
$$\det \begin{pmatrix} 3 & 4 \\ 4 & 16/3 \end{pmatrix} = 16 - 16 = 0$$

> *"Semidefinite is the borderline."* — Strang, 38:06

Determinant $0 \rightarrow$ bir özdeğer 0 (matris tekil); trace pozitif olduğundan diğeri pozitif. Klasik örnek hepsi-1 matrisi: rank 1 olduğundan özdeğerleri $3, 0, 0$'dır ve bir rank-1 yapı taşı olarak yazılır:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = 3 \cdot \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$

Yarı-tanımlılığın iki yüzünü — $A^T A$'nın daima ≥ 0 enerjisi ve hepsi-1 matrisinin rank-1 yapısı — Şekil 12.7 yan yana koyuyor.



Şekil 12.7: Yarı-tanımlılığın iki yüzü. Solda $A^T A$: $A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$ tam kolon ranklı olduğu için $A^T A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ pozitif tanımlıdır ($\lambda = 1, 3 > 0$); enerji $x^T A^T A x = \|Ax\|^2 \geq 0$ asla negatif olamaz. Sağda hepsi-1 matrisi: rank-1 olduğu için iki özdeğeri sıfırdır ($\lambda = 3, 0, 0$) — pozitif yarı-tanımlı, sınırdaki durum. Her ikisinde de $\lambda \geq 0$.

Builder Notu — Sınırdaki Kovaryans

Yarı-tanımlılık ML'de her yerde: kovaryans ve Gram (A^TA) matrisleri her zaman en az yarı-tanımlıdır (özdeğerler ≥ 0), çünkü bağımlı kolonlar 0 özdeğer üretir. Düzenlileştirme ($S + \varepsilon I$) bu 0 özdeğerleri pozitife çekip matrisi tam PD ve tersinir yapar.

12.11 Bu Dersin Özeti

1. **Pozitif tanımlı** = simetrik + tüm özdeğerler > 0.
2. **Beş eşdeğer test** — özdeğer, leading determinant, pivot, enerji, A^TA.
3. **İndefinit** — det < 0 (karışık işaretli özdeğer); eyer noktası.
4. **Leading determinant testi** — sol-üst minörlerin hepsi > 0.
5. **Pivot testi** — eliminasyon pivotları > 0; pivot = ardışık determinant oranı.
6. **Enerji** $x^TSx > 0$ — yukarı açılan dışbükey kâse; asıl tanım.
7. **Kâse = kayıp fonksiyonu** — derin öğrenmenin minimize ettiği yüzey.
8. **Gradient descent** — en dik iniş; dar vadi (özdeğer oranı) yavaşlatır.
9. **PD işlemleri** — $S + T$, S^{-1}, Q^TSQ hepsi PD (doğru testle tek satır).
10. **Yarı-tanımlı** — sınır durum: $\lambda \geq 0$, det $= 0$ (örn. hepsi-1, rank 1).

Tek Bir Cümle

Pozitif tanımlılık, bir simetrik matrisin "yukarı açılan kâse" ($x^TSx > 0$) olduğunun beş eşdeğer testidir; bu kâse her optimizasyonun ve derin öğrenmenin kayıp yüzeyidir, gradient descent de o kâsenin dibini arar.

12.12 Kontrol Soruları

Soru 1: [[2, 1], [1, 2]] matrisi pozitif tanımlı mı? Üç testle göster.

$$S = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Cevap:
Leading determinantlar: $2 > 0$ ve det $= 4 - 1 = 3 > 0$ ✓. Özdeğerler: $\lambda = 3, 1$ (ikisi de > 0) ✓. Enerji: $x^TSx = 2x^2 + 2xy + 2y^2 = (x + y)^2 + x^2 + y^2 > 0$ ✓. Üç test de PD diyor.

Soru 2: [[1, 2], [2, 1]] pozitif tanımlı mı? Hepsi pozitif sayı olmasına rağmen.

$$S = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

Cevap:
det $= 1 - 4 = -3 < 0$ → özdeğerlerin çarpımı negatif → bir özdeğer negatif ($\lambda = 3, -1$). Bu **indefinit**. Pozitif girdiler PD garantisi vermez; önemli olan pozitif özdeğer/determinant/pivottur. Enerji

$x^T Sx = x^2 + 4xy + y^2$, örneğin $(1, -1)$'de $1 - 4 + 1 = -2 < 0$.

Soru 3: [[1, 1], [1, 1]] matrisi hangi sınıfta? Enerjisini açıkça yaz.

$$S = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Cevap:
det $= 0 \rightarrow$ tekil; özdeğerler 2 ve 0. Enerji: $x^T Sx = x^2 + 2xy + y^2 = (x + y)^2 \geq 0$, ama $(1, -1)$'de tam 0. Negatif olmuyor ama sıfır olabiliyor $\rightarrow$ **pozitif yarı-tanımlı** (semidefinite). Rank 1 (hepsi-1 yapı taşı).

Soru 4: Yüksek kondisyon sayısı (λ_max/λ_min büyük) gradient descent'i neden yavaşlatır?

Cevap:
Enerji kâsesinin biçimi özdeğerlerle belirlenir: büyük özdeğer dik yön, küçük özdeğer sığ yön $\rightarrow$ uzun, dar bir vadi. Gradyan en dik yöne (dik duvara) işaret eder, vadinin uzun ekseni boyunca değil; bu yüzden adımlar vadiyi enlemesine geçip zikzak çizer, dibe çok yavaş yaklaşır.
$\kappa = \lambda_{\text{max}}/\lambda_{\text{min}} = 1$ (çembersel kâse) ise tek adımda dibe iner. κ büyüdükçe yakınsama yavaşlar. Momentum, Adam ve ön-koşullama (Ders 23) bu zikzakı azaltır; eğitimde özellik ölçekleme/normalizasyon κ'yı küçük tutmak içindir.

12.13 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. $\begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$ matrisi pozitif tanımlı mı? Leading determinant, pivot ve enerji testlerinin üçüyle de göster.

Egzersiz 2. Hangi c değerleri için $\begin{pmatrix} c & 2 \\ 2 & c \end{pmatrix}$ pozitif tanımlıdır? (Leading determinant testini kullan: $c > 0$ ve $c^2 - 4 > 0$.)

$$S = \begin{pmatrix} c & 2 \\ 2 & c \end{pmatrix}$$

Egzersiz 3. Herhangi bir A matrisi için $A^T A$'nın daima en az pozitif yarı-tanımlı olduğunu enerji testiyle göster. (İpucu: $x^T(A^T A)x = (Ax)^T(Ax) = \|Ax\|^2 \geq 0$.) A'nın kolonları bağımsızsa neden tam pozitif tanımlı olur?

Egzersiz 4. Python ile testleri uygula:

```
import numpy as np

S = np.array([[4.0, 2.0], [2.0, 3.0]])
```

```
print("özdeğerler:", np.linalg.eigvalsh(S))     # hepsi > 0 ?
try:
    L = np.linalg.cholesky(S)                    # PD ise başarılı
    print("Cholesky OK -> pozitif tanımlı")
except np.linalg.LinAlgError:
    print("Cholesky başarısız -> PD değil")

# Enerji: rastgele x'lerde xᵀSx > 0 ?
for _ in range(3):
    x = np.random.randn(2)
    print("xᵀSx =", x @ S @ x)
```

Egzersiz 5. (Ders 6 habercisi.) Ders 6 SVD'ye geçer. Aşağıdaki dikdörtgen A için A^TA'yı hesapla; simetrik ve yarı-tanımlı olduğunu doğrula. A^TA'nın özdeğerleri, A'nın tekil değerlerinin **kareleridir** (σ^2) — SVD'nin temeli.

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}$$

12.14 Sonraki Ders İçin Hazırlık

Ders 6: Tekil Değer Ayrışımı (SVD)

Ders 5'i "ready for the SVD" diyerek kapattık. Ders 6, kursun ve tüm veri biliminin temel taşına geçer: SVD.

- Her matris (dikdörtgen dâhil) için $A = U\Sigma V^T$
- A^TA ve AA^T'nin özdeğerleri ($= \sigma^2$) ve özvektörleri (V ve U)
- Tekil değerler $\sigma \geq 0$; pozitif yarı-tanımlılıkla bağ
- Neden SVD = düşük-rank yaklaşımın, PCA'nın ve sıkıştırmanın merkezi

> ⚠ Ders 6 Öncesi Yapılacak
>
> - Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (A^TA ve σ^2).
> - Python'da `np.linalg.svd` ile birkaç dikdörtgen matrisi ayrıştır; σ değerlerini A^TA'nın özdeğerlerinin karekökleriyle karşılaştır.
> - Ana cümleyi tekrar oku: "Pozitif tanımlılık = yukarı açılan kâse ($x^TSx > 0$)."

12.15 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
Pozitif tanımlı	Simetrik + tüm özdeğerler > 0; beş eşdeğer test	3m10
İndefinit	det < 0, karışık işaretli özdeğer; eyer noktası	5m31
Leading determinant	Sol-üst alt-determinantların hepsi > 0	6m23
Pivot testi	Eliminasyon pivotları > 0; ardışık determinant oranı	8m43
Enerji $x^T Sx > 0$	Yukarı açılan dışbükey kâse; asıl tanım	14m25
Kâse = kayıp	Derin öğrenmenin minimize ettiği yüzey modeli	15m48
Gradient descent	En dik iniş; dar vadi (özdeğer oranı) yavaşlatır	22m09
$S+T$, S^{-1}, $Q^T SQ$	Hepsi pozitif tanımlı (doğru testle tek satır)	30m44
Yarı-tanımlı	Sınır durum: $\lambda \geq 0$, det $= 0$ (örn. hepsi-1)	38m06
SVD'ye hazır	Sonraki büyük adım; $A^T A$ özdeğerleri $= \sigma^2$	45m08

12.16 ML Bağlantıları Özeti

1. **Enerji** $x^T Sx > 0$ **= dışbükey kâse** → ikinci-derece kayıp fonksiyonunun ideal biçimi; tek minimum.
2. **Kâse = kayıp** → derin öğrenme bir enerjiyi minimize eder; PD Hessian = yerel dışbükeylik.
3. **Gradient descent** → derin öğrenmenin merkezî algoritması; dar vadi → momentum/Adam (Ders 23).
4. **Kondisyon sayısı** ($\lambda_{\text{max}}/\lambda_{\text{min}}$) → yakınsama hızını belirler; normalizasyon/ön-koşullama κ'yı küçültür.
5. $A^T A \geq 0$ **(yarı-tanımlı)** → kovaryans, Gram, kernel matrisleri hep kâse verir; düzenlileştirme ($+\varepsilon I$) tam PD yapar.
6. $S+T$ **PD** → dışbükey kayıpların toplamı dışbükey; düzenlileştirme terimi eklemenin gerekçesi.
7. **Cholesky (pivot** > 0**)** → PD matrisleri çözmenin en hızlı/kararlı yolu; Bayesian çıkarım, GP.

> **!** Tek bir şey alıp gideceksen
>
> Pozitif tanımlılık, bir simetrik matrisin "yukarı açılan kâse" ($x^T Sx > 0$) olduğunun beş eşdeğer testidir. Bu kâse, optimizasyonun ve derin öğrenmenin kayıp yüzeyidir; gradient descent o kâsenin dibini arar ve kâse ne kadar dengeli (özdeğerler birbirine yakın) ise o kadar hızlı iner.

13 Tekil Değer Ayrışımı (SVD)

A = UΣV$^{\mathrm{T}}$: her matrise uygulanır — döndür, ger, yeniden döndür ve verinin en önemli parçası

> **Bölüm bilgisi**
>
> - **Video:** Singular Value Decomposition (SVD) — Gilbert Strang, MIT 18.065
> - **OCW:** Lecture 6 — Singular Value Decomposition (SVD)
> - **Okuma süresi:** ≈ 38 dk
> - **Önkoşul:** Ders 5 (pozitif tanımlılık, $A^TA \geq 0$, "simetrik → gerçek özdeğer / ortonormal özvektör", spektral teorem $S = Q\Lambda Q^T$)

13.1 Bu Derste Ne Var?

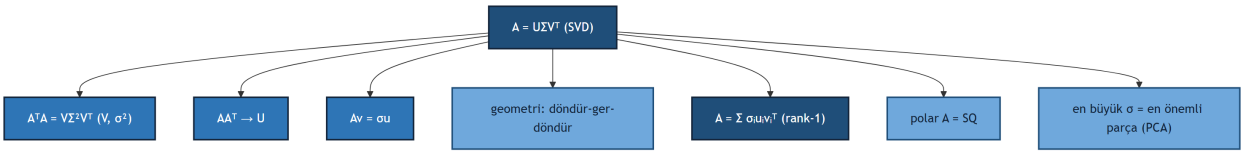
Strang'in deyimiyle "büyük gün": kursun ve tüm veri biliminin temel taşı SVD. Özdeğerler dikdörtgen matriste çalışmaz; SVD bunu **iki** tekil vektör kümesiyle çözer ve **her** matrise uygulanır.

Üç temel fikir:

1. $A = U\Sigma V^T$ — her matris (dikdörtgen dâhil) için; U sol tekil vektörler, V sağ tekil vektörler, Σ tekil değerler ($\sigma \geq 0$).
2. A^TA **anahtarı** — V, A^TA'nın özvektörleri; σ^2, A^TA'nın özdeğerleri; U ise AA^T'nin özvektörleri (Ders 5'in pozitif yarı-tanımlılığı burada ödedi).
3. **Geometri** — her matris bir döndürme × germe × döndürme'dir; çemberi elipse çevirir.

> *"…this is a big day mathematically speaking…"* — Strang, 0:22

SVD'nin merkezî denkleminden ($A = U\Sigma V^T$) tüm hatların nasıl dallandığını Şekil 13.1 özetliyor: A^TA ve AA^T özvektörlerinden $Av = \sigma u$ ilişkisine, döndür-ger-döndür geometrisine, rank-1 toplamına, polar ayrışıma ve PCA'ya kadar dersin bütün dalları.



Şekil 13.1: Ders 6 kavram haritası: A = UΣV$^{\mathrm{T}}$ (SVD) merkezinden A$^{\mathrm{T}}$A/AA$^{\mathrm{T}}$ özvektörlerine, döndür-ger-döndür geometrisine, rank-1 toplamına, polar ayrışıma ve PCA'ya uzanan dallar.

Builder Notu — Veri Biliminin Merkezi

- **SVD = veri biliminin merkezi:** en büyük birkaç σ ile rank-1 parça ($\sigma_i u_i v_i^T$) tutmak = düşük-rank yaklaşım = PCA, LoRA, sıkıştırma, gürültü giderme (Ders 7 Eckart-Young).
- **Her matrise uygulanır:** dikdörtgen ağırlık matrisleri (embedding, lineer katman) özdeğer ayrışımına sahip olmayabilir ama SVD'si her zaman vardır.
- **A^TA'dan kaçın:** pratik hesapta A^TA kurmak kondisyon sayısını kareler (yuvarlama hatası); gerçek SVD algoritmaları farklıdır.
- **Polar ayrışım** $A = SQ$ — her matris simetrik × ortogonal; SVD'den türer (mühendislikte gerinim = germe + dönme).

Tek cümle: SVD her matrisi "döndür, ger, yeniden döndür" ($U\Sigma V^T$) olarak çözer ve en büyük tekil değerler verinin en önemli parçasını verir.

13.2 Neden SVD: Dikdörtgen Matris, İki Vektör Kümesi

Özdeğerler kare matrisler içindir. A dikdörtgense Ax, x'ten farklı boyutta çıkar ($\mathbb{R}^n \to \mathbb{R}^m$), yani $Ax = \lambda x$ imkânsızdır. Kare olsa bile genel bir matrisin özvektörleri kompleks veya dik olmayabilir. SVD bu sorunları **iki** ayrı vektör kümesiyle çözer.

> *"…this is a big day mathematically speaking…"* — Strang, 0:22

Özdeğerlerde tek küme (Q) vardı; SVD'de m boyutunda sol tekil vektörler (U) ve n boyutunda sağ tekil vektörler (V) olmak üzere iki küme var.

> *"There are two sets of singular vectors, not one."* — Strang, 2:23

Builder Notu — Her Ağırlığa Uygulanır

Bu "iki küme" fikri SVD'yi evrensel kılar: her ağırlık matrisi (dikdörtgen embedding, lineer katman) için tanımlıdır — özdeğer ayrışımının aksine hiçbir koşul gerektirmez.

13.3 A = UΣV^T

SVD, herhangi bir A matrisini üç çarpana ayırır:

$$A = U\Sigma V^T$$

U ($m \times m$) ve V ($n \times n$) ortogonal matrisler (ortonormal kolonlar), Σ köşegen tekil değer matrisi. Köşegendeki değerler $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$ hepsi negatif olmayan **tekil değerler**dir. Bu, simetrik matrisin $S = Q\Lambda Q^T$ ayrışımının her matrise genellenmiş hâlidir.

> **Builder Notu — Tek Satır SVD**
>
> $A = U\Sigma V^T$, NumPy/PyTorch'ta `np.linalg.svd` ile tek satırda gelir. Tekil değerlerin azalan sırada olması, "en önemli yön en başta" demektir — düşük-rank sıkıştırmanın temeli.

13.4 A^TA Anahtarı: V ve σ^2

SVD'nin matematiği A^TA üzerinden yürür. Bu matris simetrik ve pozitif yarı-tanımlıdır (Ders 5).

> *"…the key is that A transpose A is a great matrix."* — Strang, 4:09

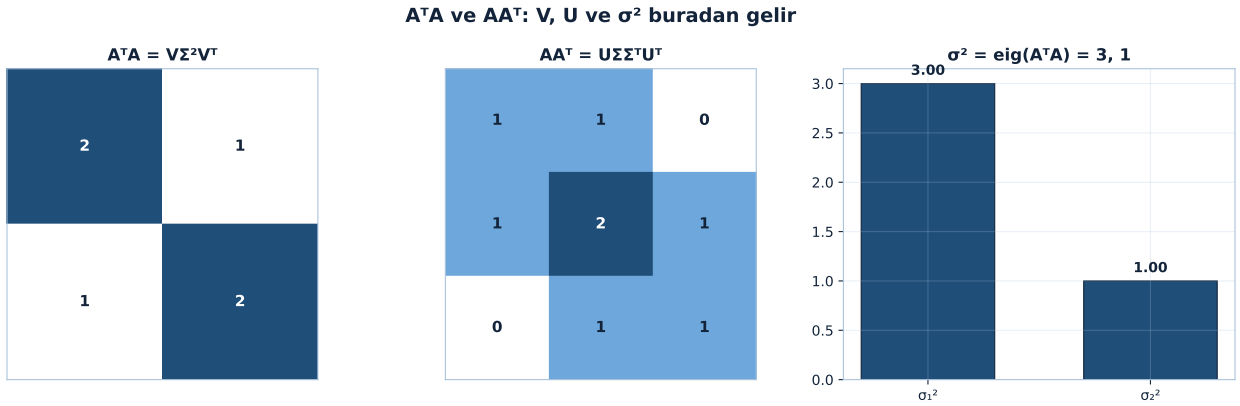
$A = U\Sigma V^T$ varsayıp A^TA'yı açarsak, ortadaki $U^TU = I$ düşer:

$$A^TA = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma^TU^TU\Sigma V^T = V\Sigma^2V^T$$

Bu tam olarak bir $Q\Lambda Q^T$ biçimidir. Yani V, A^TA'nın özvektörleri; σ^2 (tekil değerlerin karesi), A^TA'nın özdeğerleridir.

> *"…the v's are the eigenvectors of A transpose A."* — Strang, 15:20

Aynı simetrik özayrışımın diğer yönden (AA^T) U'yu nasıl verdiğini, ve σ^2'nin A^TA'nın özdeğerlerinden geldiğini Şekil 13.2 üç panelde gösteriyor: $A^TA = V\Sigma^2V^T$, $AA^T = U\Sigma\Sigma^TU^T$ ve $\sigma^2 = 3, 1$.



Şekil 13.2: A^TA (2×2) ve AA^T (3×3) simetrik matrislerinin özayrışımı SVD'yi verir: $A^TA = V\Sigma^2V^T$'den V ve σ^2 (özdeğerler), $AA^T = U\Sigma\Sigma^TU^T$'den U gelir. Burada $\sigma^2 = \text{eig}(A^TA) = 3, 1$.

> **Builder Notu — PCA'nın Özü**
>
> $\sigma^2 = A^TA$'nın özdeğeri bağlantısı, Ders 5'in "$A^TA \geq 0$" sonucunu ödetir: özdeğerler ≥ 0 olduğundan karekökleri (σ) gerçektir. PCA bunu doğrudan kullanır: kovaryans = $(1/n)A^TA$, tekil değerler = standart sapma yönündeki ölçekler.

13.5 AA^T: U

Aynı numarayı diğer yönden yap. AA^T'yi açınca bu sefer $V^TV = I$ düşer:

$$AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T$$

Yani U, AA^T'nin özvektörleridir. A^TA ($n \times n$) ve AA^T ($m \times m$) farklı boyutlarda ama **aynı sıfırdan-farklı özdeğerlere** sahiptir (Ders 4'teki AB/BA gerçeği) — eksik olanlar sıfırdır. Güzel simetri: V bir taraftan, U diğer taraftan gelir.

> Builder Notu — İki Yön, Aynı σ
>
> A^TA ile AA^T'nin aynı sıfırdan-farklı özdeğerleri olması, hangisinin küçük olduğunu hesaplama özgürlüğü verir: 10000×50 veride 50×50 olan A^TA'yı çöz (kernel trick'in SVD karşılığı). Büyük veri matrislerinde muazzam tasarruf.

13.6 Av = σu ve Ortogonallik

SVD'nin tanımlayıcı ilişkisi, $Ax = \lambda x$'in yerini alır:

$$Av_i = \sigma_i u_i, \qquad u_i = \frac{Av_i}{\sigma_i}$$

> *"A times one set of singular vectors gives me a number of times the other set of singular vectors."*
> — Strang, 9:02

Sihir şudur: A^TA'nın ortogonal özvektörleri (V) alınır, A ile çarpılır, ve çıkan Av'ler de ortogonaldir. Kanıt — $u_1^Tu_2$'yi hesapla, ortada A^TA belirir, v_2 onun özvektörü olduğundan $\sigma_2^2 v_2$ verir:

$$u_1^Tu_2 = \frac{(Av_1)^T(Av_2)}{\sigma_1\sigma_2} = \frac{v_1^T(A^TA)v_2}{\sigma_1\sigma_2} = \frac{\sigma_2^2\,(v_1^Tv_2)}{\sigma_1\sigma_2} = 0$$

Son adımda $v_1^Tv_2 = 0$ (ortonormallik). Demek ki satır uzayındaki ortogonal v'ler, kolon uzayında ortogonal Av'lere gider.

> *"…orthogonal v's in the row space, orthogonal Av's over in column space."* — Strang, 26:06

> Builder Notu — Kazanç Yönleri
>
> "Bir ortogonal taban (V), A ile başka bir ortogonal tabana (U) gider" — SVD'nin kalbi budur. Veri matrisinde V girdi özelliklerinin, U çıktı örneklerinin doğal eksenleridir; σ ikisi arasındaki kazancı verir.

13.7 Hesaplama Uyarısı: A^TA'dan Kaçın

A^TA ispat için harika ama pratikte tehlikeli. 5000×10000 bir A için A^TA kurmak hem pahalıdır hem de hataları büyütür:

> *"…you would not go this A transpose A route."* — Strang, 26:58

Sebep: A^TA kurmak matrisin **kondisyon sayısını kareler** — yuvarlama hatasına karşı kırılganlık iki katına çıkar.

> *"…condition number gets squared."* — Strang, 28:02

Gerçek SVD algoritmaları A^TA'yı hiç kurmadan, doğrudan A üzerinde çalışır (bidiagonalizasyon + örtük QR).

> Builder Notu — Sayısal Altın Kural
>
> "A^TA kurma" kuralı sayısal ML'nin altın kuralıdır: normal denklemler (A^TA) yerine QR veya SVD kullan; least squares'te `lstsq` bunu otomatik yapar. Kondisyon sayısının karelenmesi, küçük tekil değerlerin bilgisini yok eder.

13.8 Geometri: Döndürme × Germe × Döndürme

SVD'nin üç çarpanı üç geometrik işlemdir: V ortogonal (döndürme), Σ köşegen (germe), U ortogonal (döndürme). Yani her matris çarpımı şudur:

$$A = U\Sigma V^T$$

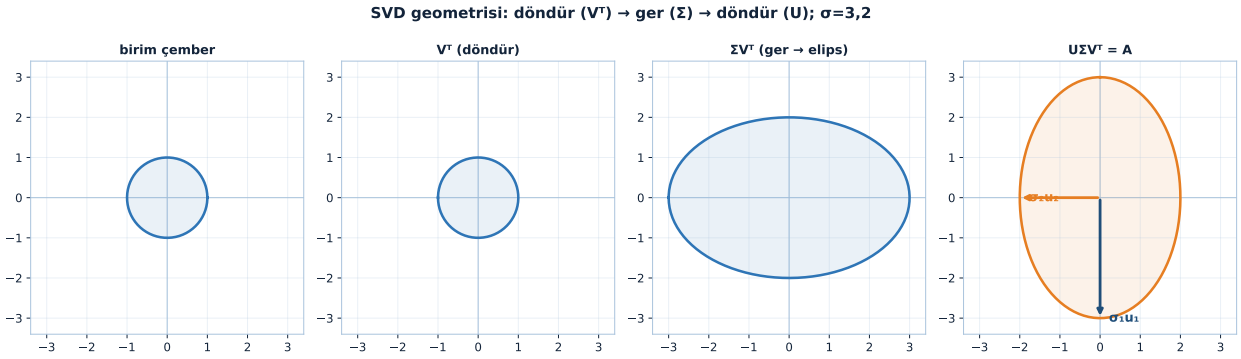
Birim çember önce V^T ile **döner** (uzunluk değişmez), sonra Σ ile her eksende σ_i kadar **gerilir** (çember → elips), sonra U ile yeniden **döner**.

> *"…into a rotation times a stretch times a different rotation…"* — Strang, 32:40

U ne zaman V'ye eşit olur? A simetrik **pozitif tanımlı** olduğunda — o zaman SVD tam olarak $S = Q\Lambda Q^T$'dir ($Q = U = V$, $\Lambda = \Sigma$). Pozitif tanımlı matrisler bu yüzden "en iyileridir": iki döndürme aynıdır.

Bu üç adımı — döndür, ger, döndür — Şekil 13.3 birim çember üzerinde tek tek izliyor: V^T döndürür, Σ elipse gerer, U yeniden döndürür; son panelde tekil eksenler $\sigma_1 u_1$ ve $\sigma_2 u_2$ elipsin ana ve yan eksenlerini gösterir.

> Builder Notu — Katmanın Anatomisi
>
> "Döndür-ger-döndür", bir lineer katmanın geometrik anatomisidir. Σ'daki tekil değerler katmanın her yöndeki kazancını verir; en büyük σ Lipschitz sabitidir (spektral norm). Spektral normalizasyon σ_1'i sınırlayarak GAN ve sağlam ağları kararlı kılar.



Şekil 13.3: SVD geometrisi: birim çember V^T ile döndürülür, Σ ile gerilerek elipse dönüşür, U ile yeniden döndürülür. Sonuç $A = U\Sigma V^T$; tekil eksenler $\sigma_1 u_1$ ve $\sigma_2 u_2$ elipsin ana ve yan eksenlerini verir.

13.9 Parametre Sayımı ve Determinant

Sayım güzel bir tutarlılık kontrolü verir. 2×2 bir A'da 4 sayı (a, b, c, d) vardır; sağ tarafta da 4 olmalı: Σ'da 2 tekil değer + V döndürmesinde 1 açı + U döndürmesinde 1 açı = 4. 3×3'te $9 = 3$ (Σ) + 3 (V, roll-pitch-yaw) + 3 (U). Determinant da temiz: ortogonal matrislerin determinantı ± 1 olduğundan

$$|\det A| = \sigma_1 \sigma_2 \cdots \sigma_n$$

Özdeğerlerin çarpımı gibi tekil değerlerin çarpımı da $|\det|$'i verir — ama tek tek $\sigma \neq |\lambda|$ (tekil değerler özdeğerleri "dıştan sarar").

> **Builder Notu — Hacim Ölçeği**
>
> Tekil değerlerin çarpımı = hacim ölçeği (Jacobian determinantı). Normalizing flow'larda $\log|\det| = \sum \log \sigma_i$ olasılık hesabına girer; düşük-rank katmanlarda küçük σ'lar hacmi (bilgiyi) sıkıştırır.

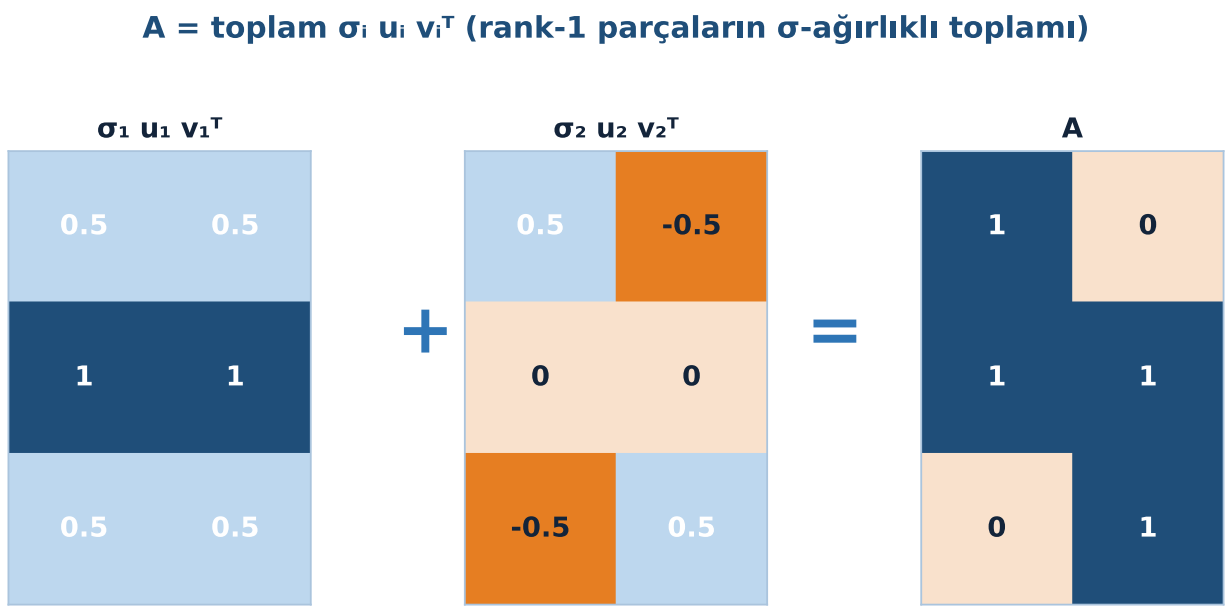
13.10 İndirgenmiş vs Tam SVD

SVD'nin iki boyu vardır. **İndirgenmiş** (reduced) form yalnızca r (rank) sıfırdan-farklı tekil değeri tutar: U ($m \times r$), Σ ($r \times r$, hepsi > 0), V^T ($r \times n$). **Tam** (full) form U'yu $m \times m$, V'yi $n \times n$ ortogonal matrise tamamlar ve Σ'ya null uzayı için sıfırlar ekler.

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_r u_r v_r^T$$

Bu rank-1 toplam, SVD'nin en kullanışlı yazılışıdır: A, r tane rank-1 parçanın σ-ağırlıklı toplamıdır. Sıfırlar hiçbir şey katmaz; gerçek bilgi r parçadadır.

Bu σ-ağırlıklı rank-1 toplamı Şekil 13.4 somut bir 3×2 matriste gösteriyor: $\sigma_1 u_1 v_1^T$ en çok bilgiyi taşır, $\sigma_2 u_2 v_2^T$ kalanı tamamlar, ikisinin toplamı tam olarak A'dır.



Şekil 13.4: $A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T$. SVD, matrisi rank-1 parçaların σ-ağırlıklı toplamına çözer: ilk parça en büyük σ_1 ile en çok bilgiyi taşır, ikinci parça kalanı tamamlar; toplamları tam olarak A'yı verir.

> **Builder Notu — Depolama Tasarrufu**
>
> İndirgenmiş SVD = depolama tasarrufu: $m \times n$ matris yerine $r(m+n)$ sayı. Rank-1 toplam, düşük-rank yaklaşımın (en büyük k parçayı tut) ve LoRA'nın (ΔW = düşük-rank) doğrudan formudur. `np.linalg.svd(A, full_matrices=False)` indirgenmiş formu verir.

13.11 Polar Ayrışım: A = SQ

SVD'den bedava çıkan ünlü bir ayrışım: polar ayrışım. Karmaşık sayının $r \cdot e^{i\theta}$ formu gibi, her matris bir simetrik ($r \leftrightarrow S$) çarpı bir ortogonal ($e^{i\theta} \leftrightarrow Q$) olarak yazılır:

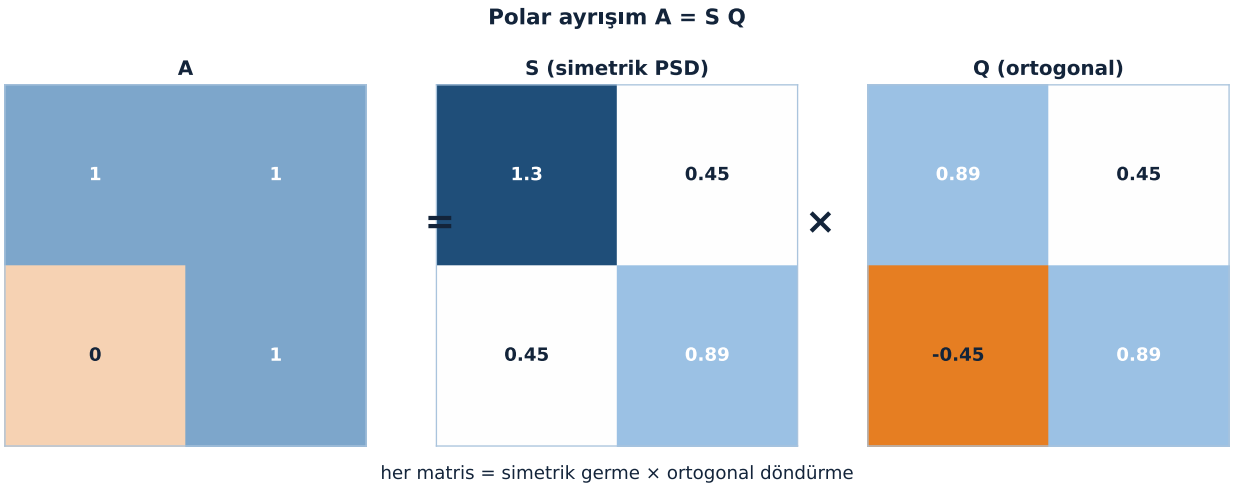
$$A = U\Sigma V^T = (U\Sigma U^T)(UV^T) = SQ$$

Araya $U^TU = I$ sokup parantezleri kaydırdık: $S = U\Sigma U^T$ simetrik pozitif yarı-tanımlı, $Q = UV^T$ ortogonal.

> *"…It's called the polar decomposition of a matrix."* — Strang, 46:17

Mühendislik dilinde: her gerinim (strain), bir simetrik germe çarpı bir iç döndürmedir.

Polar ayrışımın iki parçasını — simetrik germe S ve ortogonal döndürme Q — Şekil 13.5 somut bir matriste gösteriyor: $A = SQ$, tıpkı karmaşık sayının yarıçap × dönüş ayrışımı gibi.



Şekil 13.5: Polar ayrışım $A = SQ$. Her kare matris, bir simetrik PSD germe ($S = U\Sigma U^T$) ile bir ortogonal döndürmenin ($Q = UV^T$) çarpımına ayrılır. Burada $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ için S simetriktir ($S = S^T$, özdeğerleri ≥ 0) ve Q ortogonaldir ($Q^TQ = I$). Bu, karmaşık sayılardaki $z = r\,e^{i\theta}$ (yarıçap × dönüş) ayrışımının matris karşılığıdır.

> **Builder Notu — En Yakın Ortogonal**
>
> Polar ayrışım, en yakın ortogonal matrisi bulmanın yoludur ($Q = UV^T$) — Procrustes hizalaması (Ders 34), 3B poz/döndürme kestirimi ve ağırlıkları ortogonale yansıtma (ortogonal düzenlileştirme) bunu kullanır.

13.12 Veri: En Önemli Rank-1 Parça

Strang dersi veriye bağlayarak kapatıyor. Büyük bir veri matrisinde bir kısım sinyal, bir kısım gürültüdür. En önemli parça hangisi?

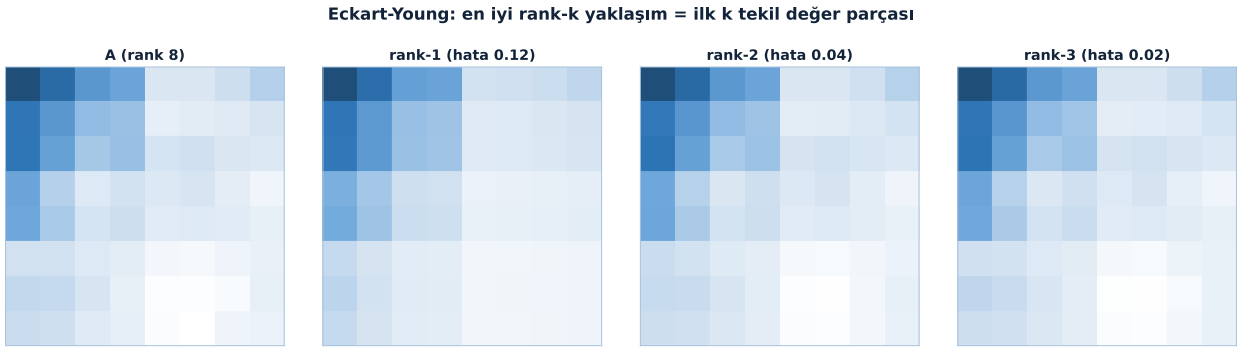
> *"…the most important part of the matrix."* — Strang, 50:42

Cevap, SVD'nin rank-1 toplamındaki **en büyük tekil değerli** parçadır:

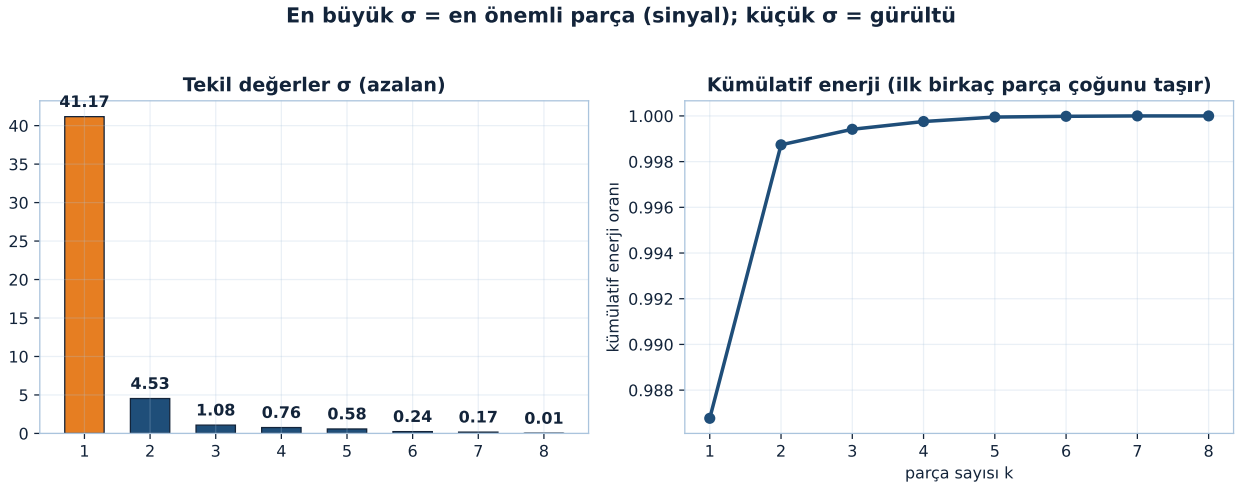
$$A \approx \sigma_1 u_1 v_1^T \quad \text{(largest single piece)}$$

σ'lar azalan sırada olduğundan $\sigma_1 u_1 v_1^T$ matrisin en güçlü yönüdür. İlk k parçayı tutmak = en iyi rank-k yaklaşım = PCA. Bu, Ders 7'nin (Eckart-Young) tam konusudur.

Bunu Şekil 13.6 somut bir görüntü matrisinde gösteriyor: tam matristen başlayarak yalnızca ilk k tekil değer parçası tutulduğunda baskın yapı daha rank-3'te neredeyse tümüyle korunur. Tekil değerlerin hızlı düşüşünü ve kümülatif enerjinin ilk birkaç parçada nasıl toplandığını ise Şekil 13.7 ortaya koyuyor — en büyük σ sinyali taşır, küçük σ'lar gürültüdür.



Şekil 13.6: Eckart-Young teoremi görsel olarak. Soldaki A tam (rank 8) yapılı bir matris; sağına doğru yalnızca ilk k tekil değer parçası tutularak elde edilen en iyi rank-k yaklaşımlar ($A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$). Başlıklardaki bağıl hata $\|A - A_k\|/\|A\|$ hızla küçülür (0.12 → 0.04 → 0.02): büyük tekil değerler sinyali taşır, küçük olanlar gürültüdür. Daha rank-3'te matrisin baskın yapısı neredeyse tümüyle korunur — düşük-rank sıkıştırmanın özü budur.



Şekil 13.7: Tekil değerlerin hızlı düşüşü ve kümülatif enerji: en büyük σ sinyalin çoğunu taşır, küçük σ'lar gürültüdür.

Builder Notu — Sinyal vs Gürültü

"En büyük σ = en önemli yön" sezgisi, boyut indirgemenin (PCA), görüntü/model sıkıştırmanın ve gürültü gidermenin temelidir: küçük σ'lı parçalar genellikle gürültüdür, atılır. LoRA bunu fine-tuning'e uygular (düşük-rank güncelleme).

13.13 Bu Dersin Özeti

1. **SVD** — özdeğerlerin dikdörtgen/genel matrise genellenmesi; iki tekil vektör kümesi.
2. $A = U\Sigma V^T$ — U, V ortogonal; Σ köşegen, $\sigma \geq 0$.
3. $A^TA = V\Sigma^2V^T$ — V özvektörler, σ^2 özdeğerler.
4. $AA^T = U\Sigma\Sigma^TU^T$ — U özvektörler; aynı sıfırdan-farklı özdeğerler.
5. $Av = \sigma u$ — tanımlayıcı ilişki; ortogonal v'ler → ortogonal Av'ler.
6. A^TA**'dan kaçın** — kondisyon sayısı karelenir; pratikte doğrudan A.
7. **Geometri** — döndürme × germe × döndürme; çember → elips.
8. $|\det A| = \prod\sigma$; parametre sayımı ($2 \times 2 = 1 + 2 + 1$).
9. **İndirgenmiş SVD** — $A = \sum\sigma_i u_i v_i^T$, r rank-1 parça.
10. **Polar ayrışım** — $A = SQ$ (simetrik × ortogonal).
11. **En büyük** σ — verinin en önemli parçası (PCA habercisi).

Tek Bir Cümle

SVD her matrisi $A = U\Sigma V^T$ olarak "döndür, ger, yeniden döndür" diye çözer; V ve U, A^TA ve AA^T'nin ortonormal özvektörleri, σ'lar da $ ^{2} = $ özdeğer ilişkisinden gelir — ve en büyük tekil değerler verinin en önemli (en düşük-rank) parçasını verir.

13.14 Kontrol Soruları

Soru 1: Aşağıdaki matrisin SVD'sini (U, Σ, V) bul. (İpucu: A^TA köşegen.)

$$A = \begin{pmatrix} 0 & 2 \\ 3 & 0 \end{pmatrix}$$

Cevap:

$A^TA = \begin{pmatrix} 9 & 0 \\ 0 & 4 \end{pmatrix}$ → özdeğerler $9, 4 \to \sigma_1 = 3, \sigma_2 = 2$. $V = I$ (özvektörler e_1, e_2). $u_1 = Ae_1/\sigma_1 = (0,3)/3 = (0,1)$; $u_2 = Ae_2/\sigma_2 = (2,0)/2 = (1,0)$.

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Tekil değerler $3, 2$ (pozitif), özdeğerler ise $\pm\sqrt{6}$ idi — tekil değerler özdeğerlerden farklı.

Soru 2: A = [[1,0],[1,1],[0,1]] (3×2) için A^TA'yı bul; tekil değerlerin $\sigma^2 = A^TA$ özdeğeri olduğunu doğrula.

$$A^TA = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Cevap:
A^TA'nın özdeğerleri 3 ve 1 (det testi: $(2-\lambda)^2 - 1$). Demek ki $\sigma_1 = \sqrt{3}$, $\sigma_2 = 1$. Tekil değerlerin kareleri $(3, 1)$ tam olarak A^TA'nın özdeğerleridir ✓. V, bu özvektörlerdir: $(1, 1)/\sqrt{2}$ ve $(1, -1)/\sqrt{2}$.

Soru 3: SVD ne zaman özdeğer ayrışımına ($Q\Lambda Q^T$) eşit olur?

Cevap:
A simetrik **pozitif tanımlı** olduğunda. O zaman $U = V = Q$ ve $\Sigma = \Lambda$; iki döndürme aynıdır. Genel simetrikte (negatif özdeğerli) tekil değerler $\sigma = |\lambda|$ olur ve U ile V işaretle ayrışır. Pozitif tanımlılık, $\sigma = \lambda$ olmasını sağlayan koşuldur — bu yüzden "en iyi matrisler"dir.

Soru 4: Bir görüntü matrisini sıkıştırmak için neden en büyük tekil değerli parçalar tutulur?

Cevap:
$A = \sum \sigma_i u_i v_i^T$ rank-1 parçaların σ-ağırlıklı toplamıdır. σ'lar azalan sırada olduğundan ilk k parça matrisin enerjisinin (Frobenius normunun) çoğunu taşır; küçük σ'lı parçalar genelde gürültü/ayrıntıdır. İlk k parçayı tutmak (rank-k yaklaşım), $m \times n$ yerine $k(m+n)$ sayı saklar ve Eckart-Young teoremine (Ders 7) göre **en iyi** rank-k yaklaşımdır. JPEG benzeri sıkıştırma, PCA ve LoRA hep bu ilkeye dayanır.

13.15 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. Aşağıdaki matrisin SVD'sini bul (A^TA üzerinden V ve σ, sonra $u = Av/\sigma$).

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Egzersiz 2. $A = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix}$ matrisinin rank'ı kaçtır? İndirgenmiş SVD'sini (tek rank-1 parça) yaz. Tam SVD'de kaç sıfır tekil değer olur?

Egzersiz 3. Egzersiz 1'deki A'nın SVD'sinden polar ayrışımını $A = SQ$ kur ($S = U\Sigma U^T$, $Q = UV^T$). S'nin simetrik, Q'nun ortogonal olduğunu doğrula.

Egzersiz 4. Python ile SVD'yi keşfet:

```
import numpy as np

A = np.array([[1.0, 0.0], [1.0, 1.0], [0.0, 1.0]])
```

```
U, s, Vt = np.linalg.svd(A, full_matrices=False)
print("tekil değerler:", s)
print("σ² :", s**2)
print("eig(AᵀA):", np.linalg.eigvalsh(A.T @ A))   # σ² ile eşleşir

# Yeniden kur: A = U Σ Vᵀ
print("A rebuild:", U @ np.diag(s) @ Vt)

# En büyük rank-1 parça:
print("σ1 u1 v1ᵀ:", s[0] * np.outer(U[:, 0], Vt[0]))
```

Egzersiz 5. (Ders 7 habercisi.) Ders 7 Eckart-Young teoremine geçer: en iyi rank-k yaklaşım, SVD'nin ilk k parçasıdır. Aşağıdaki matrisin SVD'sini bul ve en iyi rank-1 yaklaşımını ($\sigma_1 u_1 v_1^T$) yaz. Orijinalden ne kadar uzakta (Frobenius)?

$$A = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$$

13.16 Sonraki Ders İçin Hazırlık

Ders 7: Eckart-Young — A'ya En Yakın Rank-k Matris

Ders 6'da SVD'yi kurduk ve "en büyük σ = en önemli parça" sezgisini gördük. Ders 7 bunu teoreme dönüştürür.

- Eckart-Young: en iyi rank-k yaklaşım = SVD'nin ilk k parçası (her normda)
- PCA: merkezlenmiş verinin SVD'si; ana bileşenler
- Düşük-rank yaklaşımın hata ölçüsü (atılan σ'lar)
- Neden bu, sıkıştırma ve gürültü gidermenin matematiksel temeli

⚠ Ders 7 Öncesi Yapılacak

- Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (rank-1 yaklaşım).
- Python'da `np.linalg.svd` ile bir görüntüyü/matrisi farklı k değerleriyle yeniden kur; bilginin nasıl korunduğunu gözlemle.
- Ana cümleyi tekrar oku: "SVD = döndür, ger, yeniden döndür ($U\Sigma V^T$)."

13.17 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
SVD	Özdeğerlerin dikdörtgen/genel matrise genellenmesi	0m22

Kavram	Tanım	Strang'de
İki tekil vektör kümesi	U (sol, m boyut), V (sağ, n boyut)	2m23
A^TA **anahtarı**	SVD'nin matematiği; simetrik pozitif yarı-tanımlı	4m09
$Av = \sigma u$	Tanımlayıcı ilişki; $Ax = \lambda x$'in yerini alır	9m02
$V =$ **eig**(A^TA)**, $ ^{2} = $ özdeğer**	$A^TA = V\Sigma^2V^T$; U ise eig(AA^T)	15m20
Ortogonal v **→ ortogonal** Av	SVD'nin kalbi; satır uzayından kolon uzayına	26m06
A^TA**'dan kaçın**	Kondisyon sayısı karelenir; doğrudan A kullan	28m02
Geometri	Döndürme × germe × döndürme; çember → elips	32m40
Polar ayrışım	$A = SQ$; simetrik çarpı ortogonal	46m17
En önemli parça	En büyük σ'lı rank-1 parça (PCA habercisi)	50m42

13.18 ML Bağlantıları Özeti

1. $A = U\Sigma V^T$ → her ağırlık/veri matrisine uygulanır; PCA, sıkıştırma, pseudoinverse.
2. $\sigma^2 =$ **eig**(A^TA) → PCA'nın özü; kovaryans özdeğerleri = tekil değer kareleri.
3. $Av = \sigma u$ **(ortogonal → ortogonal)** → V girdi eksenleri, U çıktı eksenleri, σ kazanç.
4. A^TA**'dan kaçın** → sayısal kararlılık; least squares'te QR/SVD, normal denklem değil.
5. **$ _1 = $ spektral norm** → Lipschitz sabiti; spektral normalizasyon (GAN, sağlamlık).
6. **Polar** $A = SQ$ → en yakın ortogonal matris; Procrustes, poz kestirimi, ortogonal düzenlileştirme.
7. **En büyük** σ **rank-1 parça** → düşük-rank sıkıştırma, PCA, LoRA, gürültü giderme.

> **!** Tek bir şey alıp gideceksen
>
> SVD her matrisi $A = U\Sigma V^T$ olarak çözer — V^T ile döndür, Σ ile ger, U ile yeniden döndür. V ve U, A^TA ve AA^T'nin ortonormal özvektörleridir, σ'lar da $ ^{2} = $ özdeğer ilişkisinden gelir. En büyük tekil değerli rank-1 parçalar verinin en önemli kısmıdır — bu yüzden SVD, PCA'dan LoRA'ya tüm düşük-rank veri biliminin temel taşıdır.

14 Eckart-Young — En Yakın Rank-k Matris

Matris normları, $\ell 1$ seyreklik ve PCA

> **Bölüm bilgisi**
>
> **Video:** Eckart-Young: The Closest Rank k Matrix to A · **OCW:** MIT 18.065 Lecture 7 · **Okuma süresi:** ~36 dk · **Eğitmen:** Gilbert Strang · **Önkoşul:** Ders 6 (SVD).

14.1 Bu Derste Ne Var?

Ders 6'da SVD'yi kurduk. Ders 7 onun en güçlü sonucunu veriyor: bir matrisin en önemli bilgisi en büyük birkaç tekil değerdedir, ve **Eckart-Young teoremi** bunu kesinleştirir. Yanında matris normları ve PCA geliyor.

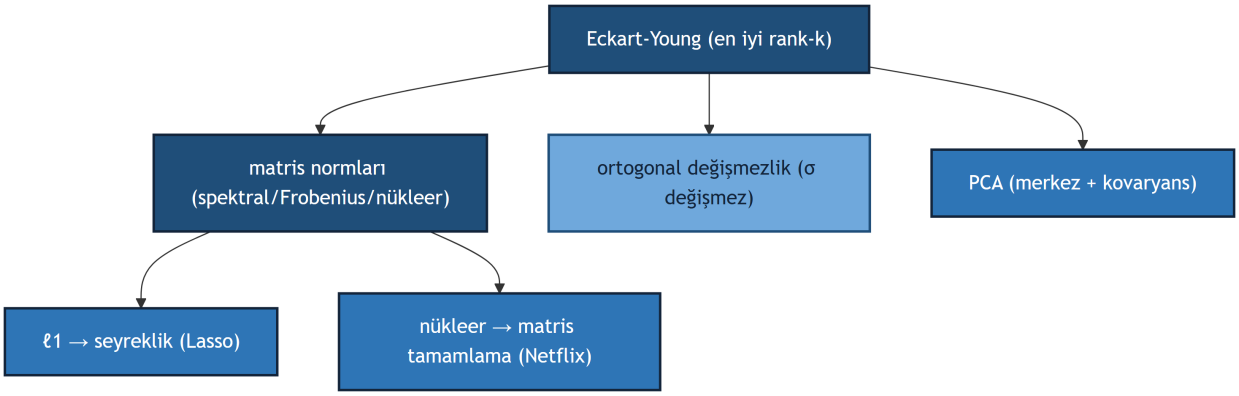
Üç temel fikir:

1. **Eckart-Young** — SVD'nin ilk k parçası (A_k), A'ya en yakın rank-k matristir; her başka rank-k matris daha uzaktır.
2. **Matris normları** — spektral (σ_1), Frobenius ($\sqrt{\sum \sigma_i^2}$), nükleer ($\sum \sigma_i$); üçü de yalnız tekil değerlere bağlı ve Eckart-Young hepsinde geçerli.
3. **PCA** — veriyi merkezle, en iyi doğruyu bul; PCA least squares değildir (dik uzaklık, dikey değil).

> *"This lecture is about principal component analysis, PCA…"* — Strang, 0:25

Şekil 14.1 bu dersin haritasını veriyor: merkezde Eckart-Young, ondan normlara, $\ell 1$ seyrekliğe, nükleer norm ile matris tamamlamaya, PCA'ya ve ortogonal değişmezliğe uzanan dallar.

> **Builder Notu — Sıkıştırmanın Kalbi**
>
> - **Eckart-Young = sıkıştırmanın garantisi:** ilk k tekil parçayı tutmak en iyi rank-k yaklaşımdır — görüntü sıkıştırma, model budama, LoRA, gürültü giderme.
> - **$\ell 1$ normu → seyreklik:** Lasso, compressed sensing, öznitelik seçimi — $\ell 1$ minimizasyonu çoğu bileşeni sıfır yapar (yorumlanabilirlik).
> - **Nükleer norm → matris tamamlama:** Netflix yarışması, öneri sistemleri, eksik MRI verisi — düşük-rank tamamlama.
> - **PCA → boyut indirgeme:** merkezlenmiş verinin SVD'si; ana bileşenler en yüksek varyans yönleridir.



Şekil 14.1: Ders 7 kavram haritası: Eckart-Young (en iyi rank-k) merkezinden matris normlarına (spektral/Frobenius/nükleer), $\ell 1$ seyrekliğine (Lasso), nükleer norm ile matris tamamlamaya (Netflix), PCA'ya ve ortogonal değişmezliğe uzanan dallar.

Tek cümle: SVD'nin ilk k parçası (Eckart-Young) bir matrisin en iyi düşük-rank özetidir — ve bu, sıkıştırmadan öneri sistemlerine, PCA'dan LoRA'ya tüm veri biliminin temel hamlesidir.

14.2 1. PCA ve En Önemli k Parça

SVD, A'yı r tane rank-1 parçaya ayırır (her parça $\sigma_i u_i v_i^T$, U ve V ortonormal). Ama büyük bir matriste **tüm** bilgi değil, **önemli** bilgi gerekir.

> *"This lecture is about principal component analysis, PCA…"* — Strang, 0:25

Strang'in vurgusu: makine öğrenmesinde tüm eğitim verisini ezberlemek "öğrenmek" değildir. Önemli olan en büyük k tekil değerdeki bilgidir. Rank binlerce olabilir ama ilk 100 parça çoğu zaman yeter.

Builder Notu — Önemli Bilgiyi Tut

"Önemli bilgi = en büyük tekil değerler" ilkesi, derin öğrenmenin sıkıştırma yönünü tanımlar: parametreleri ezberlemek yerine düşük-rank yapıyı yakalamak. LoRA, model budama ve damıtma (distillation) hep bu sezgidir.

14.3 2. Eckart-Young Teoremi

Dersin tek teoremi: SVD'nin ilk k parçası (A_k), A'ya **en yakın** rank-k matristir.

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

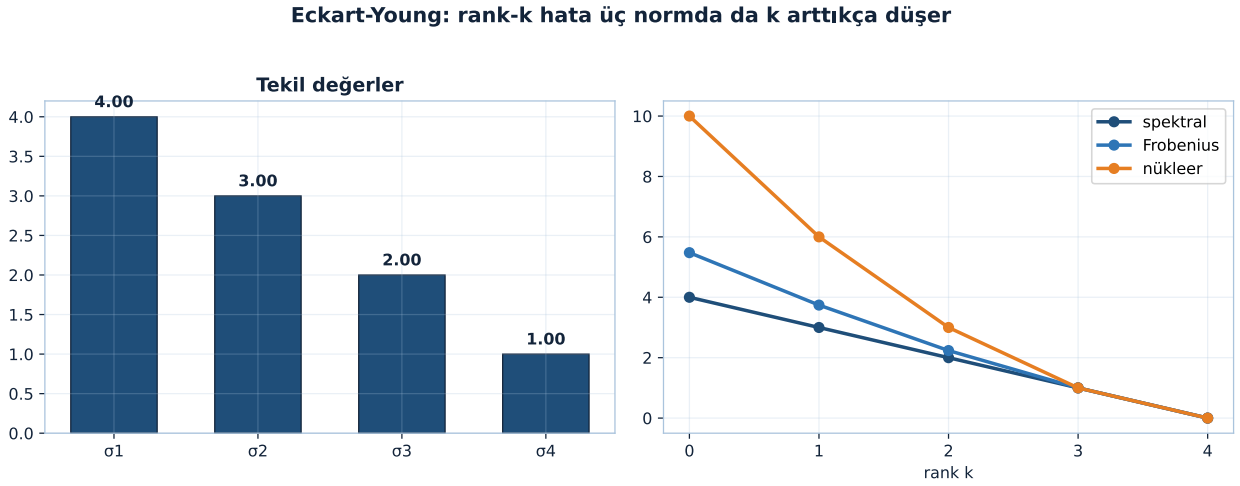
> *"…is the best approximation to A of rank k."* — Strang, 2:33

Resmî ifade: rank'ı k olan herhangi başka bir B matrisi için, hata A_k'nin hatasından küçük olamaz:

$$\|A - B\| \geq \|A - A_k\| \quad (\text{rank}(B) \leq k)$$

> *"…it's often called the Eckart-Young Theorem."* — Strang, 3:36

Bu, SVD'nin neden "mükemmel" olduğunu söyler: en iyi düşük-rank özeti doğrudan ilk k tekil parçadan gelir, ayrıca aramaya gerek yok.



Şekil 14.2: Eckart-Young, üç matris normunda da geçerlidir. Solda $A = \text{diag}(4, 3, 2, 1)$ matrisinin tekil değerleri. Sağda rank-k yaklaşımın hatası: spektral hata $= \sigma_{k+1}$, Frobenius hatası $= \sqrt{\sum_{i>k} \sigma_i^2}$, nükleer hata $= \sum_{i>k} \sigma_i$. Üçü de k arttıkça monoton düşer ve $k = 4$'te (tam rank) sıfıra iner — en iyi rank-k yaklaşımı her normda ilk k tekil değer parçasıdır.

Şekil 14.2 bunu somutlaştırır: $A = \text{diag}(4, 3, 2, 1)$ için her üç norm da rank k arttıkça monoton düşer ve tam rankta sıfıra iner.

> **Builder Notu — En İyisi Garanti**
>
> Eckart-Young, sıkıştırmanın optimallik garantisidir: ilk k parçayı tutmak, tüm rank-k matrisler arasında en iyisidir. Görüntü sıkıştırma, gürültü giderme ve düşük-rank katman yaklaşımları bu teoreme dayanır — "en büyüğü tut" hamlesi kanıtlı en iyidir.

14.4 3. Vektör Normları (ℓ1, ℓ2, ℓ∞)

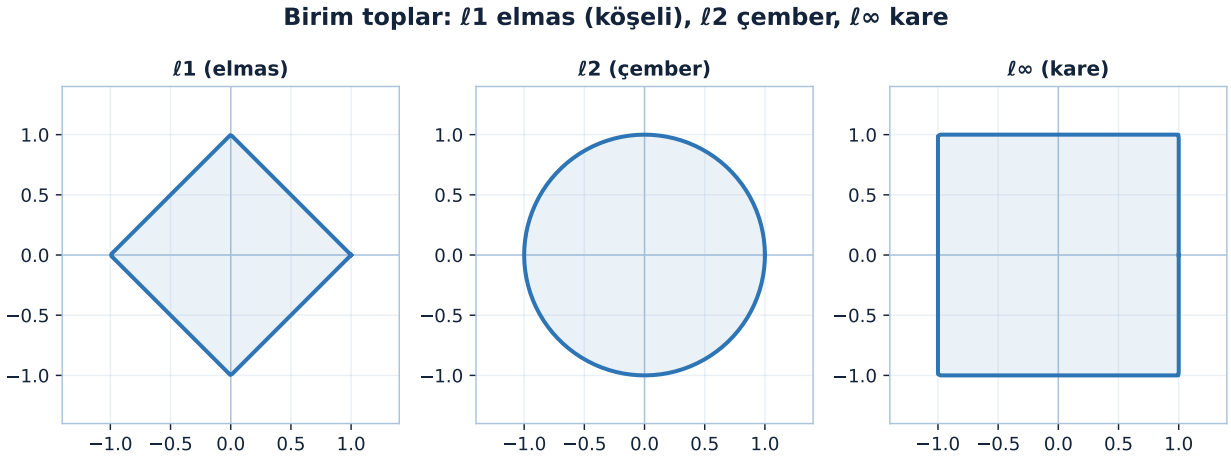
Teoremdeki çift çubuk $\|\cdot\|$ bir **norm** — matrisin/vektörün büyüklüğünün ölçüsü.

> *"…the norm of the matrix."* — Strang, 4:13

Önce vektör normları. Üç temel norm:

$$\|v\|_2 = \sqrt{v_1^2 + \cdots + v_n^2}, \quad \|v\|_1 = |v_1| + \cdots + |v_n|, \quad \|v\|_\infty = \max_i |v_i|$$

$\ell 2$ alışılmış uzunluk (Pythagoras hipotenüsü), $\ell 1$ mutlak değerlerin toplamı, $\ell\infty$ en büyük bileşen. İndeks (1, 2, ∞) hangi kuvvetin alınıp kök edildiğini söyler.



Şekil 14.3: Üç vektör normunun birim topu ($\|v\|_p \leq 1$ kümesi). Soldan sağa: ℓ_1 topu bir elmastır — sınırı eksenler üzerindeki köşelerde sivridir, bu köşeli geometri en seyrek (sparse) çözümü üretir (Lasso). ℓ_2 topu bir çemberdir — her yöne pürüzsüz ve simetriktir, köşesi yoktur. ℓ_∞ topu bir karedir — bileşenlerin en büyüğü 1'i geçemez. Üç top da aynı uçlardan (± 1 eksenlerde) geçer ama aradaki şişkinlik farkı, hangi normu seçtiğimizin neden önemli olduğunu gösterir.

Şekil 14.3 üç birim topu yan yana koyar: $\ell 1$ elmas (köşeli), $\ell 2$ çember (pürüzsüz), $\ell\infty$ kare. Köşelerin yeri bir sonraki bölümün anahtarı.

> **Builder Notu — Üç Uzunluk Ölçüsü**
>
> Bu üç norm ML'de farklı işler görür: $\ell 2$ (ridge, ağırlık zayıflatma), $\ell 1$ (Lasso, seyreklik), $\ell\infty$ (en kötü durum, sağlamlık/adversarial). Norm seçimi çözümün karakterini belirler.

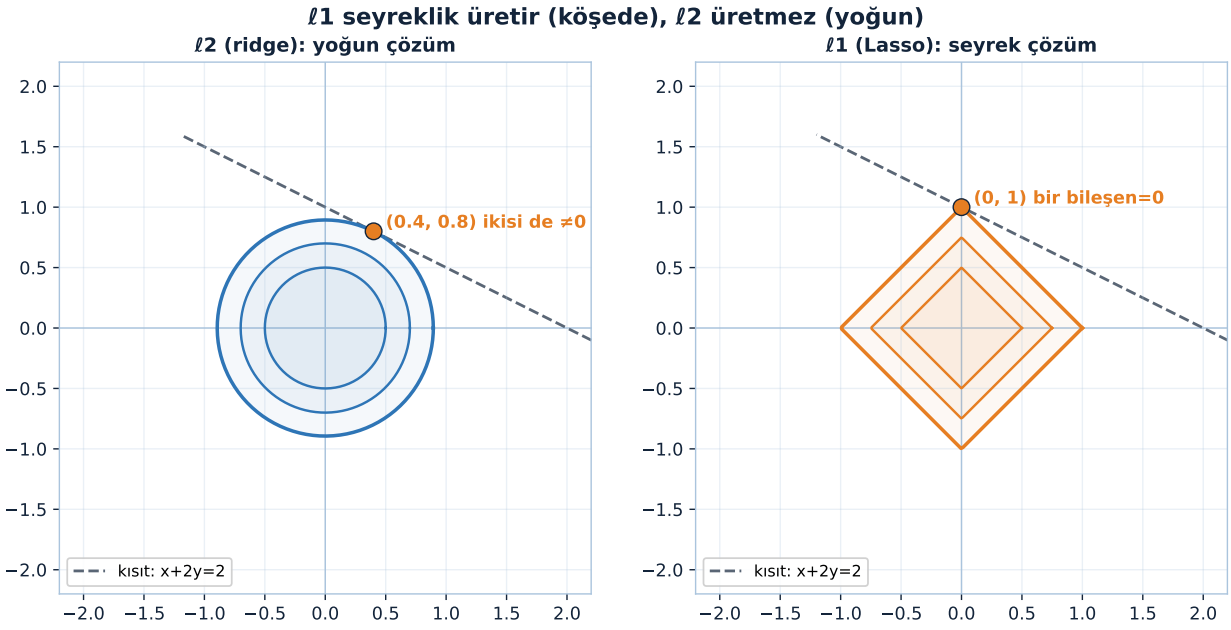
14.5 4. ℓ1 ve Seyreklik

$\ell 1$ normu son yıllarda çok önemli hâle geldi çünkü onunla minimize edildiğinde kazanan vektör **seyrek** çıkar — çoğu bileşeni sıfır.

> *"…the minimizing vector turns out to be sparse."* — Strang, 8:16

$\ell 2$ (Gauss'un least squares'i) küçük ama çok sayıda sıfırdan-farklı bileşen verir (kareler küçükleri cezalandırmaz). $\ell 1$ ise birkaç sıfırdan-farklı bileşen verir — bu da yorumlanabilirdir: hangi bileşenlerin gerçekten önemli olduğunu görürsün.

Şekil 18.3 geometriyi gösterir: $\ell 2$ topu doğruya eksen-dışı bir noktada teğet (yoğun çözüm), $\ell 1$ elması ise bir köşede değer — bu köşe seyrek çözümü üretir.



Şekil 14.4: $\ell 1$ ve $\ell 2$ minimizasyonun geometrisi. Gri kesik kısıt doğrusu x+2y=2 her panelde aynı. Solda $\ell 2$ (ridge) çemberleri büyüyerek doğruya teğet olur; teğet noktası eksen dışıdır (0.4, 0.8) ve iki bileşen de sıfırdan farklı — yoğun çözüm. Sağda $\ell 1$ (Lasso) elmasları büyüyerek doğruya değer; ilk dokunuş bir köşede (0, 1) gerçekleşir ve bir bileşen sıfır olur — seyrek çözüm. Köşeli $\ell 1$ topu seyrekliği üretir, yuvarlak $\ell 2$ topu üretmez.

> **Builder Notu — Sıfırların Gücü**
>
> $\ell 1 \rightarrow$ seyreklik, modern ML'in temel taşıdır: Lasso regresyonu (öznitelik seçimi), compressed sensing (az ölçümle sinyal kurtarma), seyrek kodlama. "$\ell 1$ seyreklik üretir" gerçeği, yorumlanabilir ve verimli modellerin matematiksel sebebidir.

14.6 5. Matris Normları (Spektral, Frobenius, Nükleer)

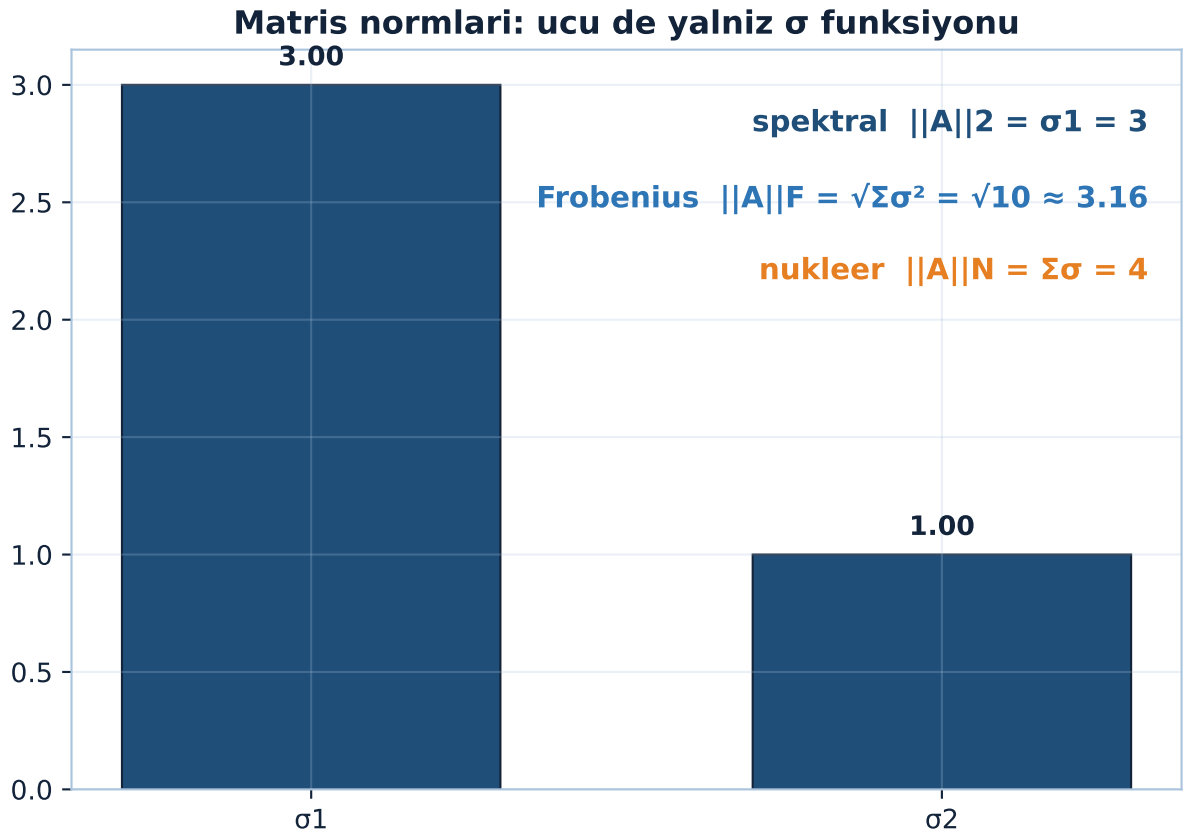
Vektör normları matrise üç şekilde taşınır — ve üçü de tekil değerlerle hesaplanır:

$$\|A\|_2 = \sigma_1, \quad \|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2} = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}, \quad \|A\|_N = \sigma_1 + \cdots + \sigma_r$$

Spektral norm ($\ell 2$) en büyük tekil değer σ_1'dir. **Frobenius normu** tüm girdilerin karelerinin toplamının karekökü dür (= tekil değerlerin karelerinin toplamının karekökü). **Nükleer norm** tekil değerlerin toplamıdır.

> *"…named after Frobenius."* — Strang, 11:54

> *"It's called the nuclear norm."* — Strang, 13:05



Şekil 14.5: Matris normlari: spektral, Frobenius ve nukleer norm — ucu de yalniz tekil degerlerin (σ) bir fonksiyonu, bu yuzden Eckart-Young her ucunde gecerli.

Normların ortak özellikleri: homojenlik ($\|cA\| = |c| \cdot \|A\|$) ve üçgen eşitsizliği ($\|A + B\| \leq \|A\| + \|B\|$).

Şekil 14.5 üç normu $M = \text{diag}(3, 1)$ üzerinde sayısallaştırır: spektral $= 3$, Frobenius $= \sqrt{10} \approx 3.16$, nükleer $= 4$ — hepsi yalnız tekil değerlerden.

> **Builder Notu — σ'nun Üç Dili**
>
> Üç norm üç rol oynar: spektral norm = Lipschitz sabiti (kararlılık, GAN); Frobenius = en yaygın matris "büyüklüğü" (ağırlık zayıflatma, trace$(A^T A)$); nükleer = rankın dışbükey gevşemesi (düşük-rank optimizasyon, matris tamamlama).

14.7 6. Üç Norm da Yalnız Tekil Değerlere Bağlı

Eckart-Young teoremi tam olarak bu üç norm için doğrudur. Sebep: üçü de yalnızca tekil değerlere bağlıdır.

> *"These norms depend only on the singular values."* — Strang, 15:56

Spektral $= \sigma_1$, Frobenius $= \sqrt{\sum \sigma_i^2}$, nükleer $= \sum \sigma_i$ — hepsi yalnız σ'ların fonksiyonu. Bu yüzden "en büyük k σ'yı tut" hamlesi her üçünde de en iyi sonucu verir; başka (σ'ya bağlı olmayan) bir norm uydurulsa Eckart-Young çalışmayabilirdi.

> **Builder Notu — Neden SVD Çözer**
>
> "Tekil değere bağlı norm" sınıfı (unitarily invariant norms), düşük-rank optimizasyonun matematiksel zeminidir. Bir kayıp bu sınıftaysa, çözümü SVD ile bulunur — matris tamamlama ve robust PCA bu özelliği kullanır.

14.8 7. Nükleer Norm: Netflix, Matris Tamamlama, MRI

Nükleer norm ($\sum \sigma_i$), $\ell 1$'in matris karşılığıdır ve eksik veriyi tamamlamada parlar. Klasik örnek **Netflix yarışması**: kullanıcı × film puan matrisi, çoğu girdisi eksik (herkes her filmi izlemedi). Eksik girdileri doldurmak = matris tamamlama.

> *"It's called the nuclear norm."* — Strang, 13:05

Nükleer normu minimize eden tamamlama, **düşük-rank** bir matris bulur (az sayıda gizli beğeni faktörü) — öneri sisteminin ta kendisi. Aynı yöntem MRI'da kullanılır: çocuk uzun süre durmadığında eksik kalan görüntü verisi, nükleer norm ile tamamlanır.

> **Builder Notu — Eksiği Tamamla**
>
> Nükleer norm = rankın dışbükey gevşemesi: rank'ı doğrudan minimize etmek NP-zor, ama nükleer norm dışbükeydir ve düşük-rank çözümler üretir. Öneri sistemleri, robust PCA (gürültü + düşük-rank ayrıştırma) ve görüntü onarımı bunu kullanır.

14.9 8. Örnek ve Ortogonal Değişmezlik

Köşegen örnek: A = diag(4, 3, 2, 1), rank 4. En iyi rank-2 yaklaşım en büyük iki değeri tutar:

$$A = \text{diag}(4, 3, 2, 1) \;\Rightarrow\; A_2 = \text{diag}(4, 3, 0, 0)$$

"Köşegende daha yakın" bir B (örn. 3.5'ler) denemek cazip ama düşük rank için ödenen köşegen-dışı bedel onu daha uzağa atar. Peki köşegen örnek özel mi? Hayır: $A = U\Sigma V^T$ ile her iki yana ortogonal matris çarpsan bile **tekil değerler değişmez** (4, 3, 2, 1 kalır).

> *"…my whole problem is orthogonally invariant."* — Strang, 27:11

Çünkü Q ortogonal ise QA'nın SVD'si $(QU)\Sigma V^T$'dir; QU yine ortogonal, Σ aynı. Normlar da değişmez ($\|Qv\| = \|v\|$ gibi). Yani köşegen örnek tüm matrisleri temsil eder. Bu monoton düşüşün üç norm için sayısal kanıtını yukarıda Şekil 14.2'da görmüştük.

> **Builder Notu — Tabandan Bağımsız**
>
> Ortogonal değişmezlik, normların ve Eckart-Young'un neden taban seçiminden bağımsız olduğunu söyler. Veriyi döndürmek (ortonormal dönüşüm) bilgisini değiştirmez — whitening, PCA eksenleri ve ortonormal gömülerin matematiksel güvencesi.

14.10 9. PCA: Veriyi Merkezle

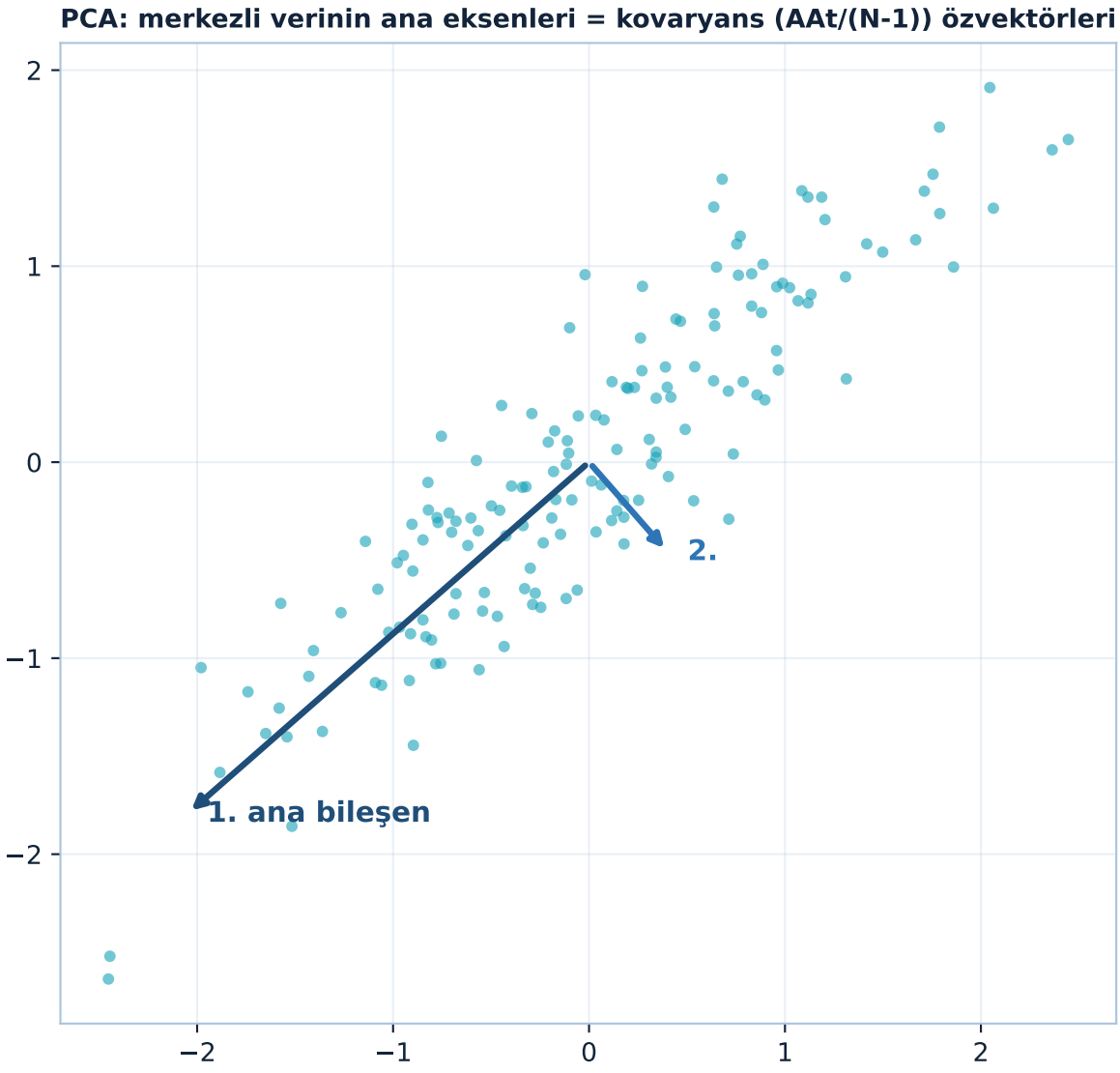
PCA, veri matrisindeki en iyi doğrusal ilişkiyi bulur. Örnek: her sütun bir kişi, satır 1 boy, satır 2 yaş. İlk adım istatistikçinin yaptığıdır: her satırdan ortalamasını çıkarıp **mean = 0** yap (veriyi merkezle).

$$A_{\text{merkezli}} = A - (\text{satir ortalamalari})$$

Merkezlenince noktalar orijin etrafında toplanır (küçük yaşlar negatif, büyükler pozitif olur). Artık en iyi doğru orijinden geçer ve aranan şey, bu bulutun ana yönüdür.

Şekil 14.6 merkezlenmiş bulutu ve onun ana eksenlerini gösterir: en uzun ok birinci ana bileşen (en büyük varyans yönü), kısa ok ona dik.

> **Builder Notu — Sıfıra Çek**
>
> Merkezleme, PCA'nın zorunlu ön adımıdır: kovaryans, merkezlenmiş veriden hesaplanır. Sinir ağlarında girdi normalizasyonu (mean çıkarma, ölçekleme) aynı sezgidir — optimizasyonu kolaylaştırır, kâseyi merkezler (Ders 5).



Şekil 14.6: PCA: merkezlenmis verinin ana eksenleri, kovaryans matrisinin (AAt/(N-1)) ozvektorleridir. En uzun ok (1. ana bilesen) en buyuk varyans yonunu, ikinci ok ona dik yonu gosterir.

14.11 10. PCA ≠ Least Squares

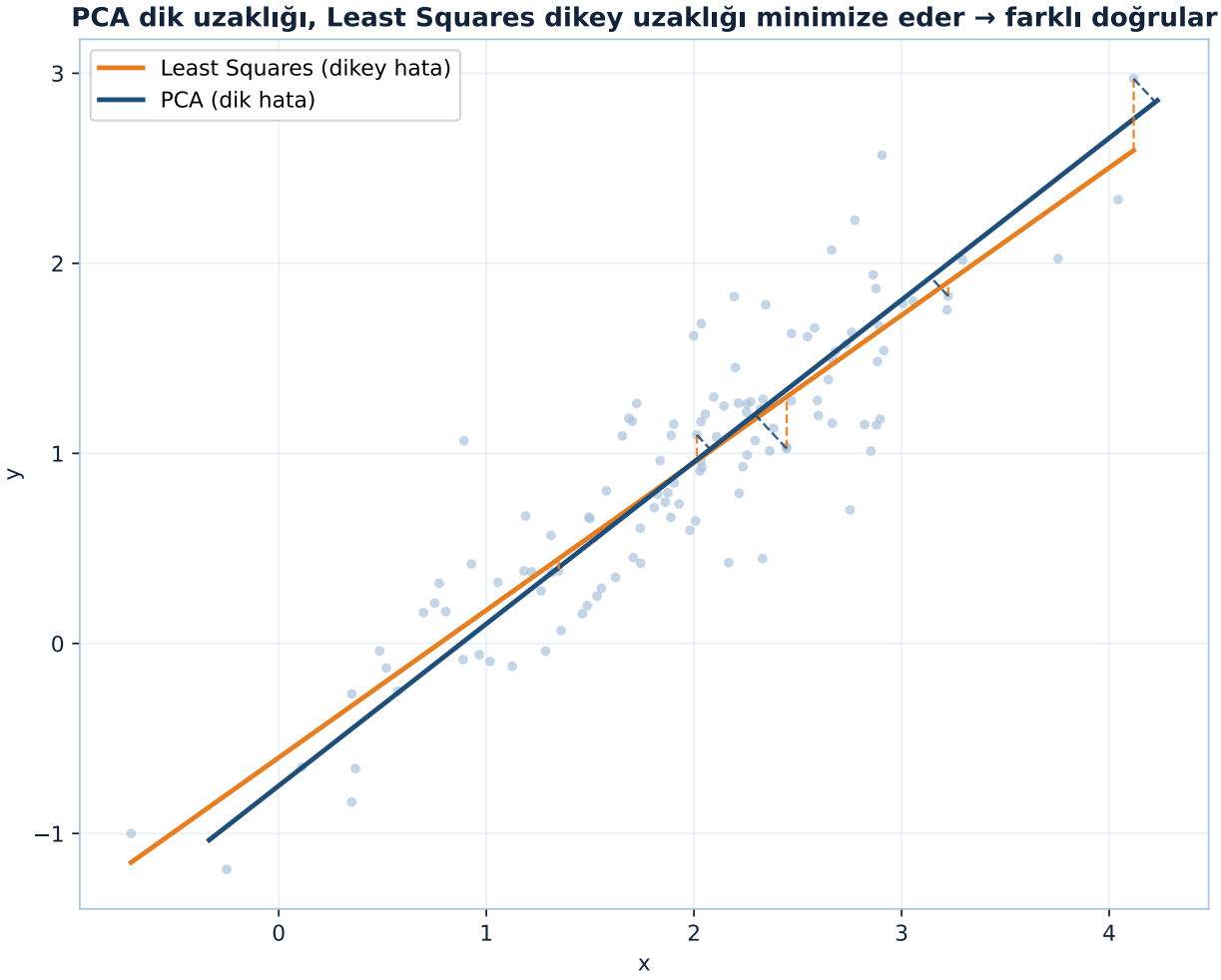
"En iyi doğru" deyince akla least squares gelir ama PCA farklıdır. Fark, hatanın **nasıl ölçüldüğü**:

> *"…in PCA, you're measuring perpendicular to the line."* — Strang, 38:17

Least squares dikey hatayı (b − Ax) ölçer ve normal denklemleri çözer:

$$A^T A\hat{x} = A^T b \quad \text{(least squares, dikey hata)}$$

PCA ise her noktanın doğruya **dik** uzaklığını ölçüp kareler toplamını minimize eder. İki farklı problem, iki farklı doğru — ve PCA'nın cevabı SVD'den (tekil vektörlerden) gelir, normal denklemlerden değil.



Şekil 14.7: PCA ve Least Squares aynı veri bulutuna farklı doğrular uydurur. Least Squares (turuncu) her noktanın doğruya **dikey** (y-yönündeki) uzaklığını minimize eder; bu, y'yi x'in fonksiyonu olarak modellemenin doğal seçimidir. PCA (lacivert) ise her noktanın doğruya **dik** (ortogonal) uzaklığını minimize eder ve doğru, merkezlenmiş verinin birinci ana bileşeni (kovaryansın baskın özvektörü) yönündedir. Kesik çizgiler birkaç nokta için iki uzaklık türünü gösterir. İki ölçüt farklı şeyleri cezalandırdığı için eğimler de farklı çıkar — bu yüzden iki doğru çakışmaz.

Şekil 14.7 aynı buluta iki doğru uydurur: PCA (lacivert) dik uzaklığı, Least Squares (turuncu) dikey uzaklığı minimize eder — farklı ölçüt, farklı doğru.

> Builder Notu — Dik mi Dikey mi
>
> PCA (dik mesafe, simetrik) ile regresyon (dikey mesafe, y'yi x'ten tahmin) farklı sorulardır: regresyon tahmin için, PCA yapı/boyut indirgeme için. İkisini karıştırmak yaygın bir hatadır; PCA bir tahmin modeli değil, verinin ana eksenlerini bulan bir araçtır.

14.12 11. Kovaryans Matrisi ve En İyi Yön

Merkezlenmiş veriden sonra istatistikçi varyansa bakar. İki ölçüt (boy, yaş) olduğundan **kovaryans matrisi** 2×2'dir ve aralarındaki bağı içerir (burada A merkezlenmiş veridir):

$$C = \frac{1}{N-1} AA^T$$

> *"…covariance matrix, and it will be 2 by 2."* — Strang, 42:06

Köşegen girdiler boy ve yaşın ayrı varyansları, köşegen-dışı ise aralarındaki kovaryans (birlikte değişim). En iyi doğrunun yönü, bu kovaryans matrisinin (yani AA^T'nin) en büyük özvektörü = A'nın birinci tekil vektörüdür. Böylece PCA SVD'ye bağlanır.

> Builder Notu — Verinin Ana Yönü
>
> Kovaryans $= \frac{1}{N-1} AA^T$ ve PCA = onun özvektörleri, istatistiğin ML ile buluştuğu yerdir. Birinci ana bileşen en yüksek varyans yönü; boyut indirgeme ilk birkaç bileşeni tutar. Aynı matris LDA, Gauss modelleri ve beyazlatmada görünür.

14.13 Bu Dersin Özeti

1. **PCA** — bir veri matrisinin en önemli bilgisini en büyük tekil değerlerden çıkarma.
2. **Eckart-Young** — A_k (ilk k SVD parçası) en iyi rank-k yaklaşımdır.
3. **Vektör normları** — $\ell 2$ (uzunluk), $\ell 1$ (mutlak toplam, seyreklik), $\ell\infty$ (maksimum).
4. **$\ell 1$ → seyreklik** — Lasso, compressed sensing; çoğu bileşen sıfır.
5. **Matris normları** — spektral σ_1, Frobenius $\sqrt{\sum \sigma_i^2}$, nükleer $\sum \sigma_i$.
6. **Üç norm da yalnız σ'ya bağlı** — Eckart-Young hepsinde geçerli.
7. **Nükleer norm** — Netflix, matris tamamlama, MRI (düşük-rank).
8. **Ortogonal değişmezlik** — $U\Sigma V^T$ tekil değerleri ve normları değiştirmez.
9. **PCA ≠ least squares** — dik uzaklık, dikey değil.
10. **Kovaryans** $AA^T/(N-1)$ — en iyi yön = birinci tekil vektör.

! Tek Bir Cümle

Eckart-Young teoremi, SVD'nin ilk k parçasının bir matrisin en iyi rank-k özeti olduğunu (spektral, Frobenius ve nükleer normların üçünde de) söyler; PCA bunu merkezlenmiş veriye uygular ve en büyük tekil vektör verinin ana eksenini verir.

14.14 Kontrol Soruları

i Soru 1: A = diag(5, 2) için en iyi rank-1 yaklaşımı ve hatasını (spektral norm) bul.

Cevap: Tekil değerler 5 ve 2. En iyi rank-1 = en büyüğü tut: $A_1 = \text{diag}(5, 0)$. Hata $A - A_1 = \text{diag}(0, 2)$, spektral normu $\sigma_2 = 2$. Eckart-Young: başka hiçbir rank-1 matris 2'den yakın olamaz.

i Soru 2: Tekil değerleri σ = 3, 1 olan bir A için spektral, Frobenius ve nükleer normları hesapla.

Cevap: Spektral: $\|A\|_2 = \sigma_1 = 3$. Frobenius: $\|A\|_F = \sqrt{\sigma_1^2 + \sigma_2^2} = \sqrt{9+1} = \sqrt{10} \approx 3.16$. Nükleer: $\|A\|_N = \sigma_1 + \sigma_2 = 4$. Üçü de yalnız tekil değerlerin fonksiyonu — bu yüzden Eckart-Young üçünde de geçerli.

i Soru 3: PCA ile least squares neden farklı doğrular bulur?

Cevap: Least squares dikey hatayı (y yönünde, b − Ax) minimize eder — y'yi x'ten tahmin etme problemi, asimetrik. PCA ise noktaların doğruya **dik** uzaklığını minimize eder — simetrik, x ve y'ye eşit davranır. Farklı hata ölçüsü → farklı doğru. PCA'nın cevabı SVD'nin birinci tekil vektörü; least squares'inki $A^T A\hat{x} = A^T b$.

i Soru 4: $\ell 1$ minimizasyonu neden seyrek çözüm verir, $\ell 2$ neden vermez? (Geometrik sezgi.)

Cevap: $\ell 1$ "topu" bir elmas (köşeleri eksenlerde); bir kısıt doğrusuna değdiğinde genelde bir **köşede** değer → bazı bileşenler tam sıfır (seyrek). $\ell 2$ topu bir çember (köşesiz); kısıta teğet noktası eksende olmak zorunda değildir → tüm bileşenler küçük ama sıfırdan farklı.
Bu yüzden Lasso ($\ell 1$) öznitelik seçer (sıfırlar üretir), ridge ($\ell 2$) ise tüm ağırlıkları küçültür ama sıfırlamaz. Seyreklik isteniyorsa $\ell 1$.

14.15 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. A = diag(6, 4, 1) için en iyi rank-1 ve rank-2 yaklaşımlarını yaz. Her birinin hatasını (spektral norm) bul.

Egzersiz 2. Aşağıdaki matrisin spektral, Frobenius ve nükleer normlarını hesapla.

$$A = \begin{pmatrix} 3 & 0 \\ 0 & 4 \end{pmatrix}$$

Egzersiz 3. Simetrik pozitif tanımlı bir S için nükleer normun trace'e eşit olduğunu göster ($\|S\|_N = \sum \sigma_i = \sum \lambda_i = \text{trace}\, S$). Neden bu yalnız PD matrisler için doğru?

Egzersiz 4. Python ile düşük-rank yaklaşımı keşfet:

```
import numpy as np

A = np.diag([4.0, 3.0, 2.0, 1.0])
U, s, Vt = np.linalg.svd(A)

k = 2
Ak = U[:, :k] @ np.diag(s[:k]) @ Vt[:k, :]
print("A_2 =\n", Ak)
print("hata (spektral) =", np.linalg.norm(A - Ak, 2))      # = σ_{k+1} = 2
print("hata (Frobenius) =", np.linalg.norm(A - Ak, 'fro'))  # = √(σ3²+σ4²)
print("3 norm:", s[0], np.sqrt((s**2).sum()), s.sum())      # spektral, Frob, nükleer
```

Egzersiz 5. (Ders 8 habercisi.) Ders 8 normları derinleştirir. Aşağıdaki matrisin spektral normunu (en büyük tekil değer) $A^T A$'nın en büyük özdeğerinin karekökü olarak bul.

$$A = \begin{pmatrix} 2 & 0 \\ 1 & 2 \end{pmatrix}$$

14.16 Sonraki Ders İçin Hazırlık

Ders 8: Vektör ve Matris Normları

Ders 7'de üç normu tanıttık (Eckart-Young için). Ders 8 normları sistematik olarak ele alır.

- ℓp normları ve birim toplar (geometri)
- Matris normları: operatör (indüklenmiş) normlar vs Frobenius
- Spektral norm = en büyük tekil değer; nükleer norm = rankın dışbükey gevşemesi
- Norm seçiminin optimizasyon ve seyreklikteki rolü

> ⚠ Ders 8 Öncesi Yapılacak
>
> - Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (spektral norm).
> - Python'da `np.linalg.norm` ile farklı `ord` değerlerini (2, 'fro', 'nuc', 1, np.inf) dene.
> - Ana cümleyi tekrar oku: "Eckart-Young — SVD'nin ilk k parçası en iyi rank-k özettir."

14.17 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
PCA	En büyük tekil değerlerde önemli bilgi	0m25
Eckart-Young	A_k (ilk k SVD parçası) = en iyi rank-k yaklaşım	2m33
Matris normu	Matrisin büyüklüğünün ölçüsü $\|A\|$	4m13
Spektral norm ($\ell 2$)	En büyük tekil değer σ_1	5m18
$\ell 1$ → seyreklik	Minimize eden vektör seyrek (çoğu sıfır)	8m16
Frobenius normu	$\sqrt{\sum a_{ij}^2} = \sqrt{\sum \sigma_i^2}$; en yaygın büyüklük	11m54
Nükleer norm	$\sum \sigma_i$; matris tamamlama, rankın gevşemesi	13m05
Üç norm σ'ya bağlı	Eckart-Young üçünde de geçerli	15m56
Ortogonal değişmezlik	$U\Sigma V^T$ tekil değerleri ve normları korur	27m11
PCA ≠ least squares	Dik uzaklık (PCA) vs dikey (LS)	38m17
Kovaryans $AA^T/(N-1)$	En iyi yön = birinci tekil vektör	42m06

14.18 ML Bağlantıları Özeti

1. **Eckart-Young** → görüntü/model sıkıştırma, LoRA, budama; "en büyüğü tut" kanıtlı en iyi.
2. **$\ell 1$ → seyreklik** → Lasso, compressed sensing, öznitelik seçimi.
3. **Spektral norm (σ_1)** → Lipschitz sabiti; spektral normalizasyon (GAN, sağlamlık).
4. **Frobenius** → ağırlık zayıflatma, trace$(A^T A)$; en yaygın matris düzenlileştirmesi.
5. **Nükleer norm** → matris tamamlama (Netflix, öneri), robust PCA, MRI onarımı.
6. **PCA** → boyut indirgeme, görselleştirme, gürültü giderme; kovaryansın özvektörleri.
7. **Ortogonal değişmezlik** → whitening, ortonormal gömüler; taban seçimi bilgisini değiştirmez.

> **!** Eğer Bu Dersten Tek Bir Şey Alıp Gidersen
>
> **Eckart-Young teoremi, SVD'nin ilk k parçasının (en büyük k tekil değer) bir matrisin en iyi rank-k yaklaşımı olduğunu söyler — spektral, Frobenius ve nükleer normların üçünde de. PCA bunu merkezlenmiş veriye uygular: en büyük tekil vektör verinin ana eksenidir. Sıkıştırmadan öneri sistemlerine, Lasso'dan LoRA'ya tüm düşük-rank veri bilimi bu teoremin üstünde durur.**

15 Vektör ve Matris Normları

ℓp ailesi, birim topların geometrisi, konvekslik ve üç matris normu

> **Bölüm bilgisi**
>
> **Video:** Norms of Vectors and Matrices · **OCW:** MIT 18.065 Lecture 8 · **Okuma süresi:** ~36 dk · **Eğitmen:** Gilbert Strang · **Önkoşul:** Ders 7 (Eckart-Young, matris normlarına giriş).

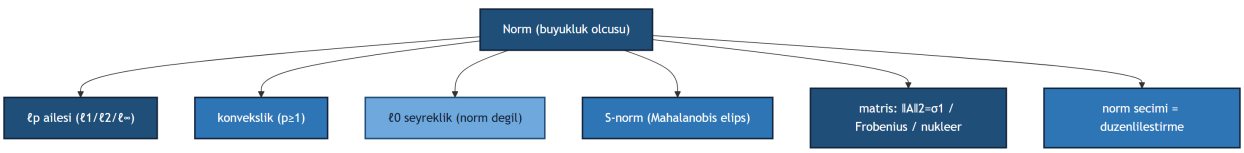
15.1 Bu Derste Ne Var?

Ders 7'de üç normu Eckart-Young için tanıttık. Ders 8 normları sistematik işliyor: ℓ^p ailesi, birim topların geometrisi, konvekslik ve matris normları. (Strang dersi eğlenceli bir olasılık gözlemiyle açıyor — "probability matching": insanlar yanlı bir parada bile %75 tura tahmin eder; bu kursun konusu değil ama hoş bir kenar not.)

Üç temel fikir:

1. ℓ^p **normları ve birim toplar** — ℓ^2 çember, ℓ^1 elmas, ℓ^∞ kare; konvekslik normun anahtarıdır (ℓ^0 ve $p < 1$ norm değildir).
2. $\ell^1 \rightarrow$ **seyreklik** — kısıtlı minimizasyonda ℓ^1 kazananı bir köşede oturur (en çok sıfır); ℓ^2 (ridge) yaymaz.
3. **Matris normları** — spektral $\|A\|_2 = \sigma_1$ (en büyük büyütme çarpanı), Frobenius $\sqrt{\sum \sigma_i^2}$, nükleer $\sum \sigma_i$.

Aşağıdaki kavram haritası dersin parçalarını nasıl bağladığını gösterir (Şekil 15.1).



Şekil 15.1: Ders 8 kavram haritası: 'Norm (büyüklük ölçüsü)' merkezinden ℓp ailesine (ℓ1/ℓ2/ℓ∞), konvekslığe (p≥1), ℓ0 seyrekliğine (norm değil), ağırlıklı S-normuna (Mahalanobis elipsi), üç matris normuna (spektral=σ1 / Frobenius / nükleer) ve norm seçiminin düzenlileştirme demek olduğu sonucuna uzanan dallar.

Builder Notu — Büyüklüğün Dili

- **Birim top geometrisi → düzenlileştirme seçimi:** ℓ^1 elmasın köşeleri eksenlerde (Lasso, seyreklik); ℓ^2 çemberi yayar (ridge); ℓ^∞ kutusu en kötü durum (adversarial).
- **Matris 2-normu $= \sigma_1 =$ Lipschitz sabiti:** bir katmanın maksimum büyütme çarpanı; spektral normalizasyon bunu sınırlar (GAN, sağlamlık).
- **Nükleer norm → düşük-rank:** Srebro varsayımı — gradient descent fazla-parametreli ağlarda nükleer normu küçük çözümü seçer (implicit regularization).
- **Konvekslik** — norm = konveks birim top; optimizasyonun tek-minimum garantisi (Ders 5 kâse).

Tek cümle: norm bir vektör/matrisin büyüklük ölçüsüdür; hangi normu seçtiğin (ℓ^1 seyrek, ℓ^2 düzgün, nükleer düşük-rank) çözümün karakterini belirler.

15.2 1. Norm Nedir: Büyüklüğün Ölçüsü

Norm, bir vektörün, matrisin, tensörün ya da fonksiyonun **büyüklüğünü** ölçen bir araçtır.

> *"…a norm is a way to measure the size of a vector or the size of a matrix…"* — Strang, 4:41

Üç temel kural: pozitiflik ($\|v\| \geq 0$, sıfır yalnız $v = 0$'da), homojenlik ($\|cv\| = |c| \cdot \|v\|$ — vektörü iki katına çıkarırsan norm da iki katına çıkar) ve üçgen eşitsizliği ($\|v + w\| \leq \|v\| + \|w\|$). Bu üç kuralı sağlayan her ölçü bir normdur.

Builder Notu — Üç Kural

Norm, ML'de "ne kadar büyük/uzak" sorusunun cevabıdır: kayıp (tahmin − hedef normu), düzenlileştirme (ağırlık normu), yakınsama (gradyan normu). Hangi normu seçtiğin problemin karakterini belirler.

15.3 2. ℓp Normları (p = 1, 2, ∞)

Bir ailenin üyeleri: ℓ^p normu, p. kuvvetlerin toplamının p. köküdür:

$$\|v\|_p = \left(|v_1|^p + |v_2|^p + \cdots + |v_n|^p\right)^{1/p}$$

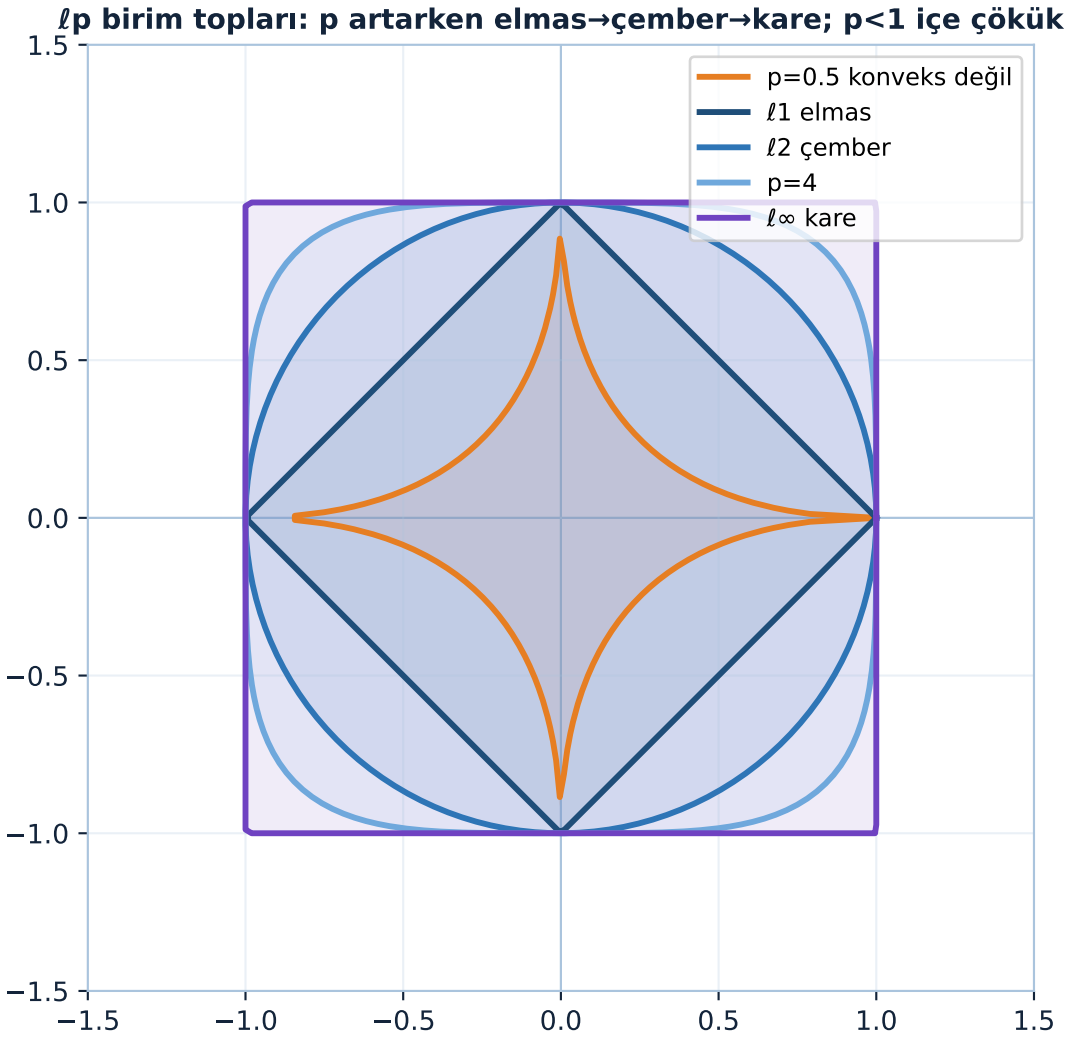
Üç önemli üye:

$$\|v\|_2 = \sqrt{\sum v_i^2}, \quad \|v\|_1 = \sum |v_i|, \quad \|v\|_\infty = \max_i |v_i|$$

ℓ^2 alışılmış uzunluk, ℓ^1 mutlak değerlerin toplamı, ℓ^∞ ($p \to \infty$ limiti) en büyük bileşeni seçer. ℓ^1 özellikle önemlidir:

> *"…it plays a very significant part now in compressed sensing."* — Strang, 5:58

Birim topların tek bir eksende üst üste çizimi p'nin geometriyi nasıl değiştirdiğini gösterir (Şekil 15.2): p arttıkça birim top elmastan çembere, oradan kareye "şişer".



Şekil 15.2: ℓp birim toplarının tek eksende üst üste çizimi. p arttıkça birim top elmastan (ℓ1) çembere (ℓ2), oradan kareye (ℓ∞) dönüşür; p<1 olduğunda kenarlar içe çöker ve birim top konveksliğini yitirir (norm değildir).

> **Builder Notu — ℓp Ailesi**
>
> ℓ^p ailesi ML'de farklı düzenlileştiriciler: ℓ^2 (ridge, ağırlık zayıflatma — düzgün küçültür), ℓ^1 (Lasso — seyrekleştirir), ℓ^∞ (en kötü-durum, adversarial sağlamlık). p seçimi çözümün geometrisini belirler.

15.4 3. ℓ0 ve Seyreklik

Seyreklikte asıl istenen, sıfırdan-farklı bileşen **sayısı**dır:

$$\|v\|_0 = \#\{\, i : v_i \neq 0 \,\}$$

"…the number of non-zero components." — Strang, 7:50

Ama bu bir norm **değildir**: homojenlik kuralını çiğner. $2v$'nin sıfırdan-farklı bileşen sayısı v ile aynıdır, yani $\|2v\|_0 = \|v\|_0 \neq 2\|v\|_0$. Bu yüzden seyreklik problemleri pratikte ℓ^1 ile (en yakın gerçek norm) çözülür — ℓ^0'ı doğrudan minimize etmek NP-zordur.

> Builder Notu — Sayılamayan İdeal
>
> ℓ^0 "tam seyreklik" idealdir ama hesaplanamaz (kombinatoryal). ℓ^1, ℓ^0'ın dışbükey gevşemesidir ve aynı seyrek çözümü çoğu zaman verir — compressed sensing ve Lasso'nun teorik temeli. "Norm değil ama hedef" olan ℓ^0, ℓ^1 ile yaklaşılır.

15.5 4. Birim Toplar Geometrisi

Bir normu anlamanın en iyi yolu, $\|v\| = 1$ olan vektörleri çizmektir (birim top). 2B'de:

- ℓ^2**:** çember, $v_1^2 + v_2^2 = 1$.
- ℓ^1**:** elmas (köşeleri eksenlerde: $(\pm 1, 0)$, $(0, \pm 1)$), $|v_1| + |v_2| = 1$.
- ℓ^∞**:** kare, $\max(|v_1|, |v_2|) = 1$.

"It's a diamond." — Strang, 11:26

p arttıkça şekil elmastan ($p = 1$) çembere ($p = 2$) oradan kareye ($p = \infty$) doğru "şişer" (Şekil 15.2). Köşelerin yeri kritiktir: ℓ^1 elmasının köşeleri eksenlerdedir — bu, seyrekliğin geometrik kaynağıdır.

> Builder Notu — Köşeler Konuşur
>
> Birim top geometrisi, düzenlileştirmenin neden işe yaradığını gösterir: ℓ^1 elmasının sivri köşeleri eksenleri "yakalar" (bazı bileşenler tam sıfır), ℓ^2 çemberinin köşesizliği yaymaya yol açar. Bu basit resim, Lasso ile ridge'in farkını tek bakışta verir.

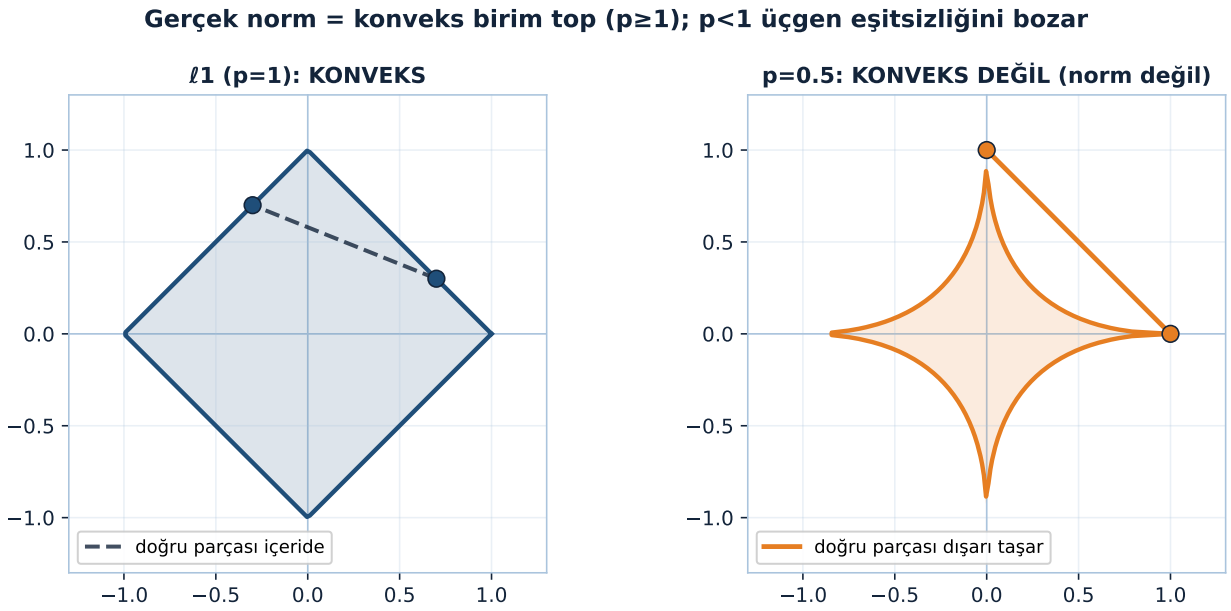
15.6 5. Konvekslik: Normun Anahtarı

Çember, elmas ve karenin ortak özelliği nedir? **Konvekslik** (dışbükeylik) — iki nokta arasındaki doğru parçası şeklin içinde kalır.

"…This is a true norm as the convex unit [ball]." — Strang, 17:09

$p < 1$ için (örneğin $p = 1/2$) birim top içe çöker — konveks değildir, ve üçgen eşitsizliği bozulur, yani gerçek bir norm olmaz. ℓ^0 ise tamamen eksenlere büzülür (konveks değil). Demek ki "gerçek norm" ile "konveks birim top" aynı şeydir; geçerli aralık $p \geq 1$'dir.

Şekil 15.3 bunu yan yana gösterir: ℓ^1 elmasının içinde kalan doğru parçası karşısında $p = 0.5$ yıldızının dışına taşan doğru parçası.



Şekil 15.3: Bir normun birim topunun konveks olması zorunludur. Solda ℓ_1 (p=1) elması: kenar üzerindeki iki noktayı birleştiren doğru parçası (gri kesik) tamamen topun içinde kalır — küme konvekstir, üçgen eşitsizliği sağlanır. Sağda p=0.5 ile çizilen yıldız (astroid): eksenler üzerindeki iki köşeyi birleştiren doğru parçası (turuncu) topun dışına taşar; bu küme konveks değildir, dolayısıyla p<1 için $\|v\|_{0.5}$ bir norm değildir (üçgen eşitsizliği bozulur).

> **Builder Notu — Konveks Cennet**
>
> Konvekslik, optimizasyonun cennetidir: konveks bir norm-cezası, tek bir küresel minimum garanti eder (Ders 5 kâsesi). Bu yüzden ML ℓ^0 (konveks değil, NP-zor) yerine ℓ^1 (konveks) kullanır — seyrekliği konveks bir problemle elde etmek.

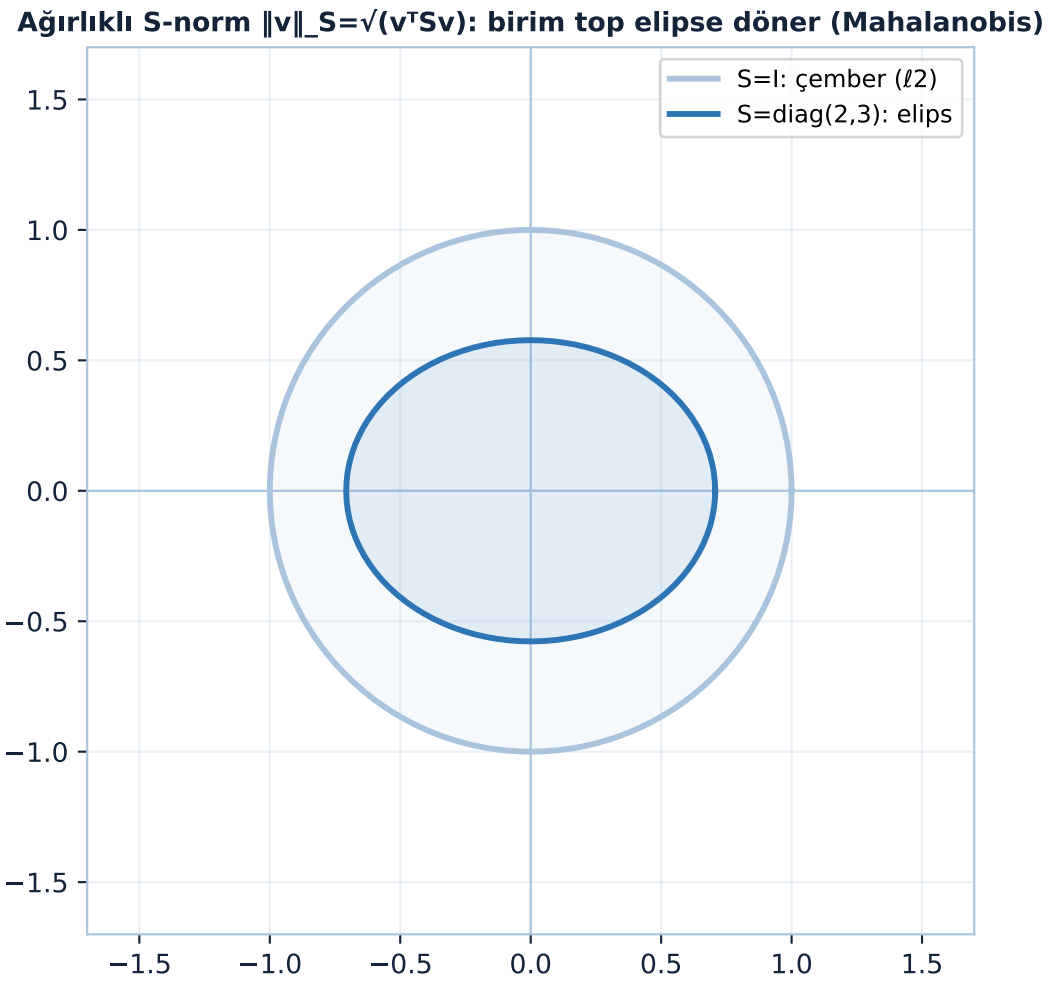
15.7 6. Ağırlıklı S-Normu

Pozitif tanımlı bir S matrisi yeni bir norm tanımlar — enerjinin karekökü:

$$\|v\|_S = \sqrt{v^T S v}$$

> *"…The energy. That's the energy in the vector v."* — Strang, 18:53

$S = I$ alınca ℓ^2 normuna döner (çember). Farklı bir S (örneğin $\mathrm{diag}(2,3)$) birim topu bir **elipse** dönüştürür: $2v_1^2 + 3v_2^2 = 1$. Büyük ağırlık (3) o eksende daha az ilerlemene izin verir. Bu "ağırlıklı norm", probleme uygun ölçekler seçmeni sağlar (Şekil 15.4).



Şekil 15.4: Ağırlıklı S-norm $\|v\|_S=\sqrt{(v^T Sv)}$: birim top elipse döner (Mahalanobis).

> **Builder Notu — Probleme Göre Ölç**
>
> S-normu (Mahalanobis uzaklığı), istatistik ve ML'de her yerde: kovaryansın tersiyle ölçeklenmiş uzaklık, anormallik tespiti, beyazlatma. Hessian'ı S olarak almak (ikinci-derece yöntemler) kâseyi çembere çevirir — Newton/doğal gradyanın sezgisi.

15.8 7. Kısıtlı Minimizasyon: ℓ1 vs ℓ2

Temel problem: $Ax = b$ kısıtı altında $\|x\|$'i minimize et. Norm seçimi sonucu kökten değiştirir. ℓ^2'de buna ridge regresyon, ℓ^1'de **basis pursuit** denir.

> *"…it has a famous name, basis pursuit."* — Strang, 24:02

Geometrik resim: çözümler bir doğru üzerinde. Normu (birim topu) orijinden şişir, doğruya ilk değdiği nokta kazanır. ℓ^2 çemberi doğruya **en yakın** noktada değer (dik ayak). ℓ^1 elması ise sivri **köşesiyle** değer — ve köşe eksende olduğundan kazanan bir bileşeni sıfır olan seyrek vektördür (Şekil 15.5).

> *"The winner has the most zeros."* — Strang, 28:58

> **Builder Notu — Lasso vs Ridge**
>
> Bu resim Lasso (ℓ^1) ile ridge (ℓ^2) farkının tamamıdır: ℓ^1 köşesi eksene değer → öznitelik seçimi (seyreklik); ℓ^2 teğeti → tüm ağırlıklar küçük ama sıfırdan farklı. Seyreklik/yorumlanabilirlik istiyorsan ℓ^1, düzgün küçültme istiyorsan ℓ^2.

15.9 8. Matris 2-Normu = σ1

Matris normu, vektör normundan türetilir: A'nın bir vektörü ne kadar "büyütebildiği" (büyütme çarpanı):

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

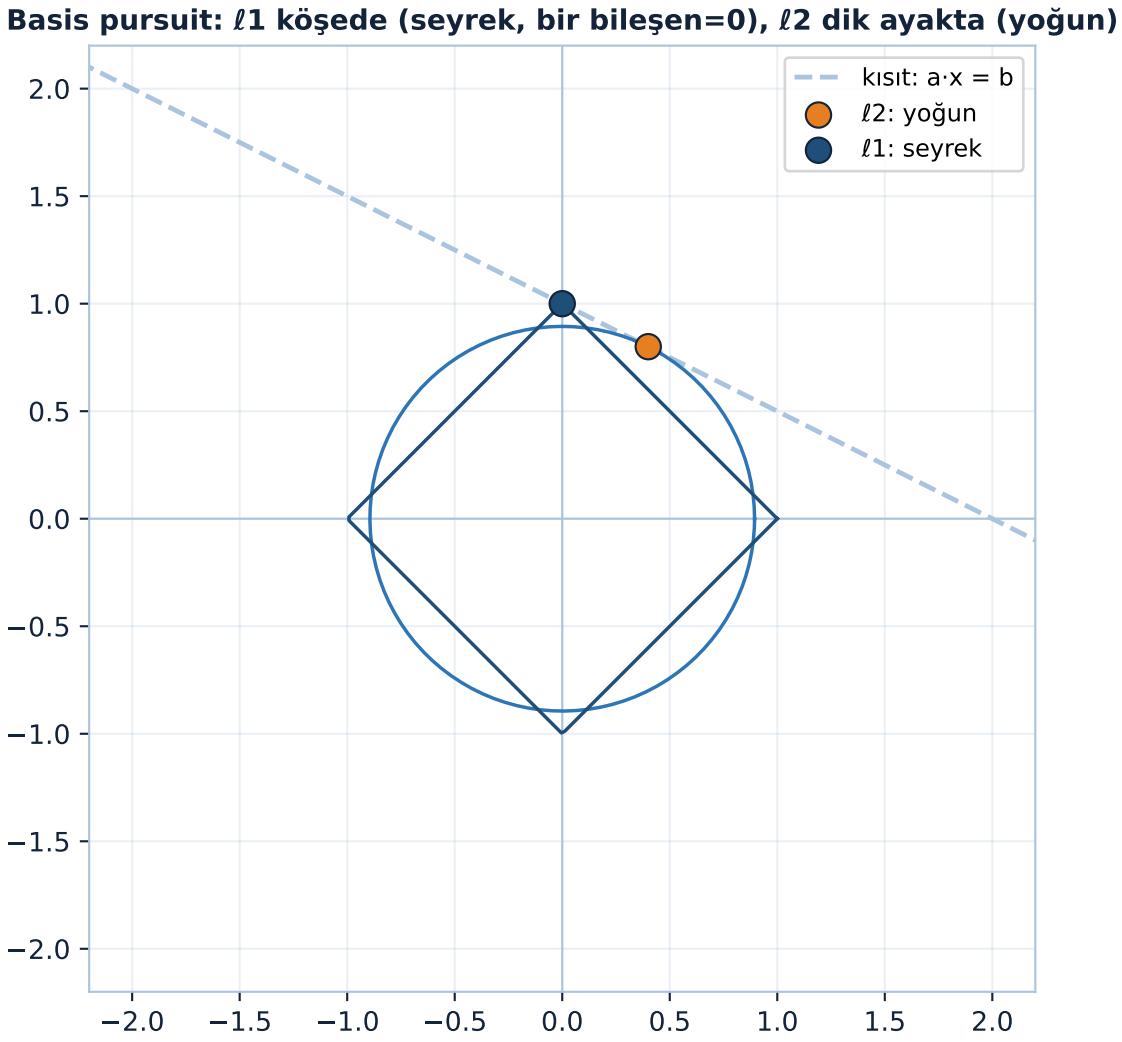
> *"…the answer will be the maximum blow-up."* — Strang, 34:34

En büyük büyütme çarpanı σ_1'dir ve onu sağlayan x birinci sağ tekil vektör v_1'dir (özvektör değil — A simetrik olmayabilir).

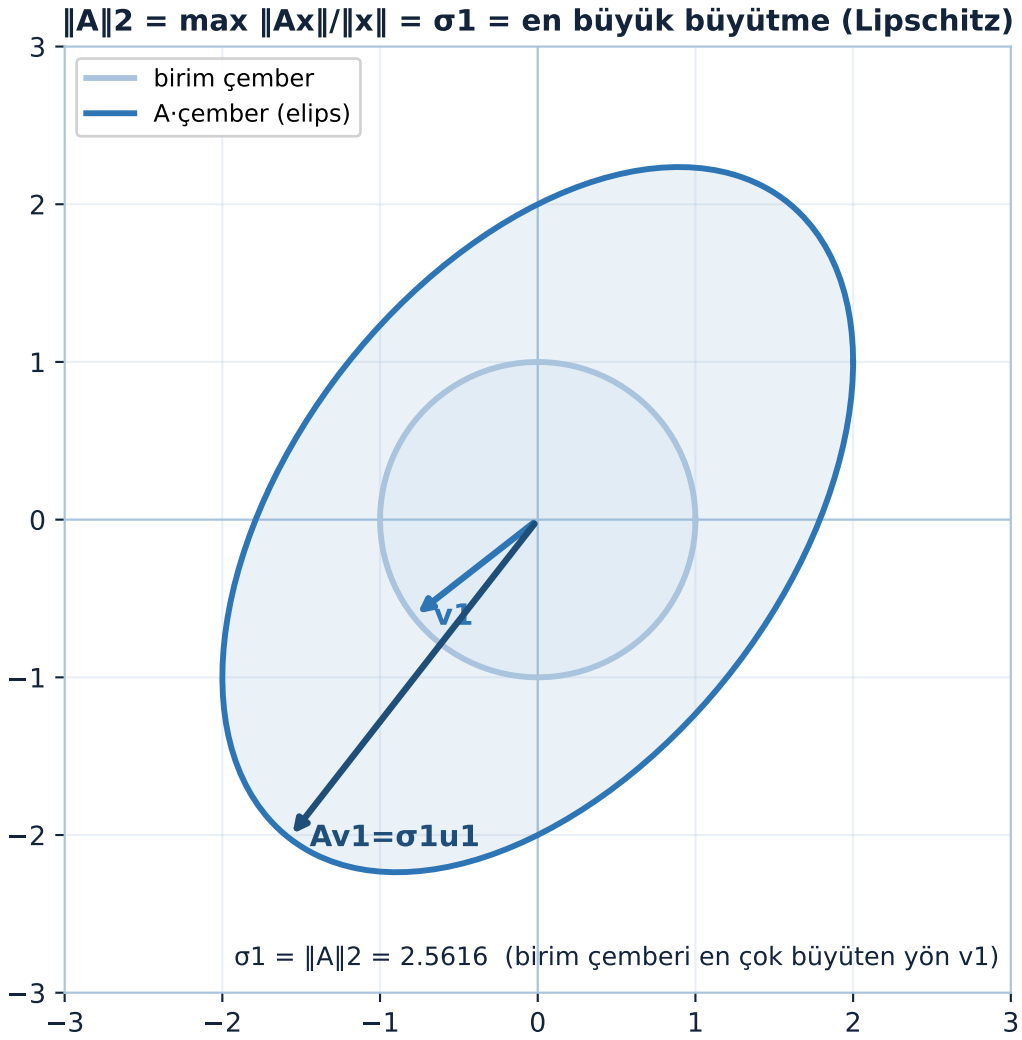
> *"…the x that has the biggest blow-up factor…"* — Strang, 36:14

Yerine koy: $x = v_1$ iken $Av_1 = \sigma_1 u_1$, $\|v_1\| = \|u_1\| = 1$, yani oran $= \sigma_1$ (Şekil 15.6):

$$\|A\|_2 = \frac{\|Av_1\|}{\|v_1\|} = \frac{\|\sigma_1 u_1\|}{1} = \sigma_1$$



Şekil 15.5: Basis pursuit: aynı kısıt doğrusu (a·x = b) üzerinde $\ell 1$ ve $\ell 2$ normunu minimize etmek. Büyüyen $\ell 1$ elması doğruya bir köşesinden değer → çözüm eksende (bir bileşen sıfır = seyrek). Büyüyen $\ell 2$ çemberi doğruya teğet değer → çözüm eksen dışı dik ayakta (yoğun). Seyreklik $\ell 1$'in köşelerinden gelir.



Şekil 15.6: $\|A\|_2 = \max \|Ax\|/\|x\| = \sigma_1$: birim çember A altında bir elipse döner; en büyük büyütme yönü v1, görüntüsü Av1 = σ1·u1 elipsin en uzun yarı-eksenidir. σ1 = ‖A‖2 ≈ 2.5616, katmanın Lipschitz sabiti.

> **Builder Notu — Katmanın Lipschitz'i**
>
> $\|A\|_2 = \sigma_1$ bir katmanın Lipschitz sabiti — girdideki değişimi en fazla kaç kat büyütebileceği. Spektral normalizasyon σ_1'i 1'e sabitler; GAN kararlılığı, adversarial sağlamlık ve gradyan patlamasını kontrol etmenin doğrudan yolu.

15.10 9. Frobenius Normu

İkinci matris normu, matrisi uzun bir vektör gibi düşünüp tüm girdilerin karelerinin toplamının kareköküdür — ve bu, tekil değerlerin karelerinin toplamına eşittir:

$$\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2} = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2}$$

> *"…the square root of the sum of the squares of all the sigmas."* — Strang, 40:58

Neden eşit? SVD'de $A = U\Sigma V^T$; ortogonal U ve V Frobenius normunu değiştirmez (uzunluk korur), geriye köşegen Σ kalır, onun girdileri de σ'lardır. Üç matris normunun aynı σ vektöründen geldiği Şekil 15.7'da görülür.

> **Builder Notu — Ucuz Büyüklük**
>
> Frobenius en yaygın matris "büyüklüğü"dür: ağırlık zayıflatma ($\|W\|_F^2$ cezası), gradyan normu, $\|A\|_F^2 = \text{trace}(A^T A)$. Hesaplaması ucuz (SVD gerekmez), bu yüzden derin öğrenmede varsayılan matris düzenlileştiricisidir.

15.11 10. Nükleer Norm ve Srebro Varsayımı

Üçüncü matris normu, tekil değerlerin toplamıdır — matrisin ℓ^1 normu gibidir:

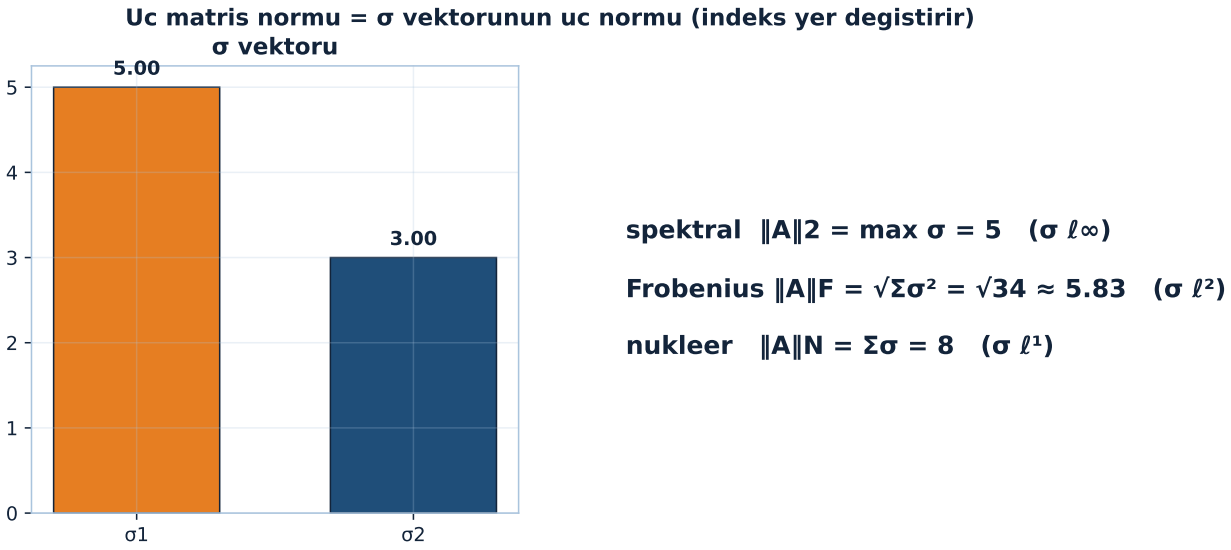
$$\|A\|_N = \sigma_1 + \sigma_2 + \cdots + \sigma_r$$

Hoş bir örüntü: üç matris normu, σ vektörünün üç normudur — spektral $=$ max σ (ℓ^∞), Frobenius $= \sqrt{\sum \sigma^2}$ (ℓ^2), nükleer $= \sum \sigma$ (ℓ^1). İndeksler matrise geçince yer değiştirir (Şekil 15.7).

Srebro'nun (derin öğrenme teorisyeni) varsayımı: fazla-parametreli ağlarda (ağırlık $>$ örnek) birçok minimum vardır; gradient descent bunlardan **nükleer normu en küçük** olanı seçer.

> *"…picks out the weights that minimize the nuclear norm."* — Strang, 46:00

> **Builder Notu — Gizli Düzenlileştirme**
>
> Nükleer norm $=$ düşük-rank teşviki (rankın dışbükey gevşemesi): matris tamamlama (Netflix), robust PCA. Srebro varsayımı **implicit regularization**'ın çekirdeğidir: GD açıkça düzenlileştirme olmadan



Şekil 15.7: Üç matris normu = σ vektörünün üç normu: spektral $\|A\|_2$ = max σ ($\ell\infty$), Frobenius $\|A\|\_F$ = $\sqrt{\Sigma\sigma^2}$ (ℓ^2), nükleer $\|A\|\_N$ = Σσ (ℓ^1). İndeks p yer değiştirir.

bile düşük-rank/küçük-norm çözümlere yönelir — derin öğrenmenin neden genelleştiğine dair önemli bir ipucu.

15.12 Bu Dersin Özeti

1. **Norm** — büyüklük ölçüsü; pozitiflik + homojenlik + üçgen eşitsizliği.
2. ℓ^p **normları** — ℓ^1 (toplam), ℓ^2 (uzunluk), ℓ^∞ (maksimum).
3. ℓ^0 — sıfırdan-farklı sayısı; norm değil (homojenlik çiğnenir); seyreklik hedefi.
4. **Birim toplar** — ℓ^2 çember, ℓ^1 elmas, ℓ^∞ kare.
5. **Konvekslik** — gerçek norm = konveks birim top; $p < 1$ norm değildir.
6. **S-normu** — $\sqrt{v^T S v}$; ağırlıklı, elips (Mahalanobis).
7. **Kısıtlı min** — ℓ^1 seyrek köşe (basis pursuit), ℓ^2 ridge.
8. $\|A\|_2 = \sigma_1$ — maksimum büyütme çarpanı (Lipschitz).
9. **Frobenius** — $\sqrt{\sum \sigma_i^2} = \sqrt{\sum a_{ij}^2}$; en yaygın.
10. **Nükleer** — $\sum \sigma_i$; düşük-rank, Srebro varsayımı.

! Tek Bir Cümle

Norm bir vektör/matrisin büyüklük ölçüsüdür; birim topunun geometrisi (ℓ^1 elmas seyrek köşeli, ℓ^2 çember düzgün, nükleer düşük-rank) hangi çözümün kazanacağını belirler — bu yüzden norm seçimi ML'de düzenlileştirme kararının ta kendisidir.

15.13 Kontrol Soruları

i Soru 1: v = (3, −4) vektörünün ℓ1, ℓ2 ve ℓ∞ normlarını hesapla.

$\ell^2 = \sqrt{3^2 + 4^2} = \sqrt{25} = 5$ (alışılmış uzunluk). $\ell^1 = |3| + |-4| = 7$ (mutlak toplam). $\ell^\infty = \max(3, 4) = 4$ (en büyük bileşen). Her zaman $\ell^\infty \le \ell^2 \le \ell^1$ (burada $4 \le 5 \le 7$).

i Soru 2: Tekil değerleri σ = 5, 3 olan bir A için spektral, Frobenius ve nükleer normları hesapla. σ vektörünün hangi normlarına denk?

Spektral $\|A\|_2 = \sigma_1 = 5$ (σ vektörünün ℓ^∞'u). Frobenius $= \sqrt{25 + 9} = \sqrt{34} \approx 5.83$ (σ'nın ℓ^2'si). Nükleer $= 5 + 3 = 8$ (σ'nın ℓ^1'i). Üç matris normu $= \sigma = (5, 3)$ vektörünün ℓ^∞, ℓ^2, ℓ^1 normları — indeksler yer değiştirmiş.

i Soru 3: ℓ0 neden norm değildir? p = 1/2 neden gerçek bir norm vermez?

ℓ^0 (sıfırdan-farklı sayısı) **homojenliği** çiğner: $\|2v\|_0 = \|v\|_0 \neq 2\|v\|_0$ (ölçeklemek sıfır sayısını değiştirmez). $p = 1/2$'nin birim topu **konveks değildir** (içe çöker), bu yüzden üçgen eşitsizliği bozulur — norm olmaz. Gerçek norm aralığı $p \ge 1$'dir (konveks birim top).

i Soru 4: Seyrek öznitelik seçimi için hangi norm cezasını seçersin? Geometrik nedenini açıkla.

ℓ^1 (Lasso). Sebep geometrik: kısıt kümesi ℓ^1 elmasının sivri **köşesine** değer, köşeler eksenlerde olduğundan bazı bileşenler tam sıfır olur (seyrek). ℓ^2 çemberi köşesiz olduğundan teğet noktası genelde eksende değildir → tüm bileşenler küçük ama sıfırdan farklı (ridge). İdeal seyreklik ℓ^0'dır ama hesaplanamaz; ℓ^1 onun dışbükey, çözülebilir vekilidir.

15.14 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. $v = (1, -2, 2)$ için ℓ^1, ℓ^2 ve ℓ^∞ normlarını hesapla. $\ell^\infty \le \ell^2 \le \ell^1$ olduğunu doğrula.

Egzersiz 2. Aşağıdaki rank-1 matrisin spektral, Frobenius ve nükleer normlarını bul. (İpucu: tek bir tekil değer var.)

$$A = \begin{pmatrix} 3 & 4 \\ 0 & 0 \end{pmatrix}$$

Egzersiz 3. 2B'de ℓ^1, ℓ^2, ℓ^∞ birim toplarını çiz. Hangisi diğerini içerir? (İpucu: aynı yarıçapta $\ell^1 \subseteq \ell^2 \subseteq \ell^\infty$ mı, tersi mi?) Konveks olmayan bir "birim top" örneği ver ($p < 1$).

Egzersiz 4. Python ile normları keşfet:

```
import numpy as np

v = np.array([1.0, -2.0, 2.0])
print("ℓ1:", np.linalg.norm(v, 1), " ℓ2:", np.linalg.norm(v, 2), " ℓ∞:", np.linalg.norm(v, np.inf
A = np.array([[3.0, 4.0], [0.0, 0.0]])
print("spektral:", np.linalg.norm(A, 2))      # σ1
print("Frobenius:", np.linalg.norm(A, 'fro'))
print("nükleer:", np.linalg.norm(A, 'nuc'))    # Σσ
print("σ:", np.linalg.svd(A, compute_uv=False))
```

Egzersiz 5. (Ders 9 habercisi.) Ders 9 least squares'in dört çözüm yolunu işler. Aşağıdaki tutarsız sistem için (3 denklem, 1 bilinmeyen) normal denklem $A^TA\hat{x} = A^Tb$'yi kurup $\hat{x}$'i bul. Bu, ℓ^2 (en küçük kareler) çözümüdür.

$$A = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

15.15 Sonraki Ders İçin Hazırlık

Ders 9: En Küçük Kareleri Çözmenin Dört Yolu

Ders 8'de normları tamamladık. Ders 9, $Ax = b$'nin çözümü olmadığında "en iyi" x'i bulmanın (least squares) dört farklı yolunu işler.

- Normal denklemler ($A^TA\hat{x} = A^Tb$)
- QR ile (sayısal kararlılık)
- SVD / pseudoinverse ile (en genel)
- Gram-Schmidt ve doğrudan geometri (projeksiyon)

⚠ Ders 9 Öncesi Yapılacak

- Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (normal denklem).
- Python'da `np.linalg.lstsq` ile birkaç tutarsız sistemi çöz; ℓ^2 hatasını gözlemle.
- Ana cümleyi tekrar oku: "Norm seçimi (ℓ^1 seyrek, ℓ^2 düzgün) çözümün karakterini belirler."

15.16 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
Norm	Büyüklük ölçüsü; pozitiflik + homojenlik + üçgen	4m41
ℓ^p **normları**	$(\sum\|v_i\|^p)^{1/p}$; ℓ^1, ℓ^2, ℓ^∞	5m58

Kavram	Tanım	Strang'de
ℓ^0	Sıfırdan-farklı sayısı; norm değil (homojenlik)	7m50
Birim toplar	ℓ^2 çember, ℓ^1 elmas, ℓ^∞ kare	11m26
Konvekslik	Gerçek norm = konveks birim top; $p \geq 1$	17m09
S-normu	$\sqrt{v^T S v}$; ağırlıklı, elips (Mahalanobis)	18m53
Kısıtlı min ℓ^1	Basis pursuit; kazanan seyrek (köşe)	24m02
Matris 2-normu	Maksimum büyütme çarpanı $= \sigma_1$ (Lipschitz)	36m14
Frobenius	$\sqrt{\sum a_{ij}^2} = \sqrt{\sum \sigma_i^2}$; en yaygın	40m58
Nükleer norm	$\sum \sigma_i$; düşük-rank, Srebro varsayımı	46m00

15.17 ML Bağlantıları Özeti

1. $\ell^1 \rightarrow$ **seyreklik** $\rightarrow$ Lasso, compressed sensing, öznitelik seçimi (elmas köşesi).
2. $\ell^2 \rightarrow$ **ridge** $\rightarrow$ ağırlık zayıflatma; düzgün küçültür, sıfırlamaz.
3. $\ell^\infty \rightarrow$ en kötü-durum, adversarial sağlamlık (kare birim top).
4. $\|A\|_2 = \sigma_1 \rightarrow$ Lipschitz sabiti; spektral normalizasyon (GAN, sağlamlık).
5. **Frobenius** $\rightarrow$ ağırlık zayıflatma cezası, gradyan normu; en yaygın matris düzenlileştirmesi.
6. **Nükleer norm** $\rightarrow$ matris tamamlama, robust PCA; implicit regularization (Srebro).
7. **Konvekslik** $\rightarrow$ norm cezaları konveks problem verir; tek küresel minimum.

> **!** Eğer bu dersten tek bir şey alıp gidersen
>
> Norm bir vektör/matrisin büyüklük ölçüsüdür, ama asıl güç birim topunun geometrisindedir: ℓ^1 elmasının köşeleri seyreklik üretir (Lasso), ℓ^2 çemberi düzgün küçültür (ridge), ℓ^∞ kutusu en kötü durumu ölçer. Matris tarafında $\|A\|_2 = \sigma_1$ (Lipschitz), Frobenius $= \sqrt{\sum \sigma_i^2}$, nükleer $= \sum \sigma_i$ (düşük-rank). Hangi normu seçtiğin, ML'de düzenlileştirme kararının ta kendisidir.

16 En Küçük Kareleri Çözmenin Dört Yolu

Pseudoinverse, normal denklemler, QR ve iteratif yöntemler — hepsi aynı projeksiyonda buluşur

> **Bölüm bilgisi**
>
> **Video:** Four Ways to Solve Least Squares Problems · **OCW:** MIT 18.065 Lecture 9 · **Okuma süresi:** ~36 dk · **Eğitmen:** Gilbert Strang · **Önkoşul:** Ders 8 (vektör ve matris normları).

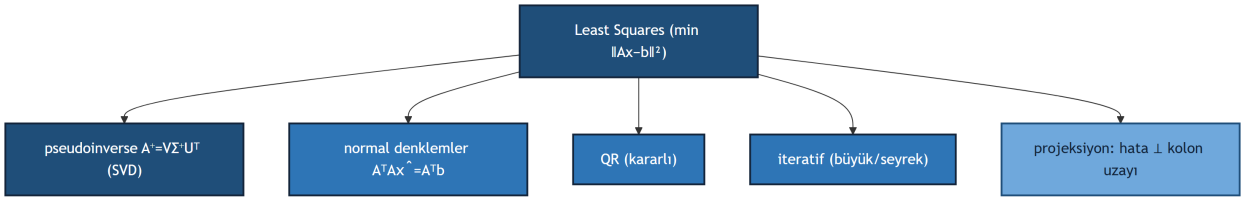
16.1 Bu Derste Ne Var?

$Ax = b$ çoğu zaman çözülemez — ölçümler gürültülü, denklem sayısı bilinmeyenden çok. **Least squares** en iyi $\hat{x}$'i bulur. Ders 9 bunun dört yolunu işliyor; hepsi SVD ve dört temel alt-uzay üzerinde buluşuyor.

Üç temel fikir:

1. **Pseudoinverse** $A^+ = V\Sigma^+U^T$ — SVD'den; satır/kolon uzayında gerçek ters, null uzaylarında sıfır. Her matris için tanımlı.
2. **Normal denklemler** $A^TA\hat{x} = A^Tb$ — Gauss'un least squares'i; A'nın kolonları bağımsızsa çalışır; geometrik olarak b'nin kolon uzayına projeksiyonu.
3. **Dört yol** — pseudoinverse (SVD), normal denklemler, QR/Gram-Schmidt, ve büyük/seyrek sistemler için iteratif yöntemler.

Bu dört yolun nasıl birbirine bağlandığını Şekil 16.1 özetliyor: hepsi tek bir least squares probleminden dallanır ve aynı dik izdüşümde buluşur.



Şekil 16.1: Ders 9 kavram haritası: Least Squares (min $\|Ax-b\|^2$) merkezinden pseudoinverse (SVD), normal denklemler, QR (kararlı), iteratif (büyük/seyrek) ve projeksiyon (hata $\perp$ kolon uzayı) ilkesine uzanan dört çözüm yolu.

> *"…what are these four ways…"* — Strang, 5:21

Builder Notu — Regresyonun Çekirdeği

- **Least squares = regresyonun çekirdeği:** lineer regresyon, en küçük kareler kaybı; ML'in en temel uyum problemi.
- **Pseudoinverse** A^+ — rank-eksik ve dikdörtgen durumlarda "en iyi ters"; minimum-norm çözüm verir (Ders 11).
- $A^T A\hat{x} = A^T b$ — pratikte en yaygın; ama $A^T A$ kurmak kondisyonu kareler (Ders 6), o yüzden QR/SVD tercih edilir.
- **Projeksiyon geometrisi** — b'yi kolon uzayına düşür; hata $b - A\hat{x}$ kolon uzayına diktir (ortogonallik ilkesi).

Tek cümle: çözülemeyen $Ax = b$ için least squares, b'yi A'nın kolon uzayına projekte eder; bu projeksiyonu pseudoinverse, normal denklemler veya QR ile bulursun — hepsi aynı $\hat{x}$'e varır.

16.2 1. Least Squares Problemi

En küçük kareler problemi basittir: $Ax = b$ denklemin **çözümü yoktur** (A dikdörtgen veya b kolon uzayında değil). Tipik durum: çok sayıda gürültülü ölçüm, az sayıda bilinmeyen.

> *"…the least squares problem is simply, you have an equation, Ax equals b."* — Strang, 23:05

Çözüm olmayınca ne yaparız? Gauss'un yolunu izleriz: hatanın ℓ^2 normunun karesini minimize et.

$$\min_x \|Ax - b\|^2 = (Ax - b)^T (Ax - b)$$

Bu, çözülebilir bir problem verir ve Strang bunun dört çözüm yolunu işler.

> *"…what are these four ways…"* — Strang, 5:21

Builder Notu — En İyiyi Bul

Least squares, lineer regresyonun ve en temel uyum probleminin çekirdeğidir. ℓ^2 kaybı (kareli hata), derin öğrenmedeki MSE'nin atasıdır; "çözümü yok → en iyiyi bul" yaklaşımı tüm optimizasyonun zeminidir.

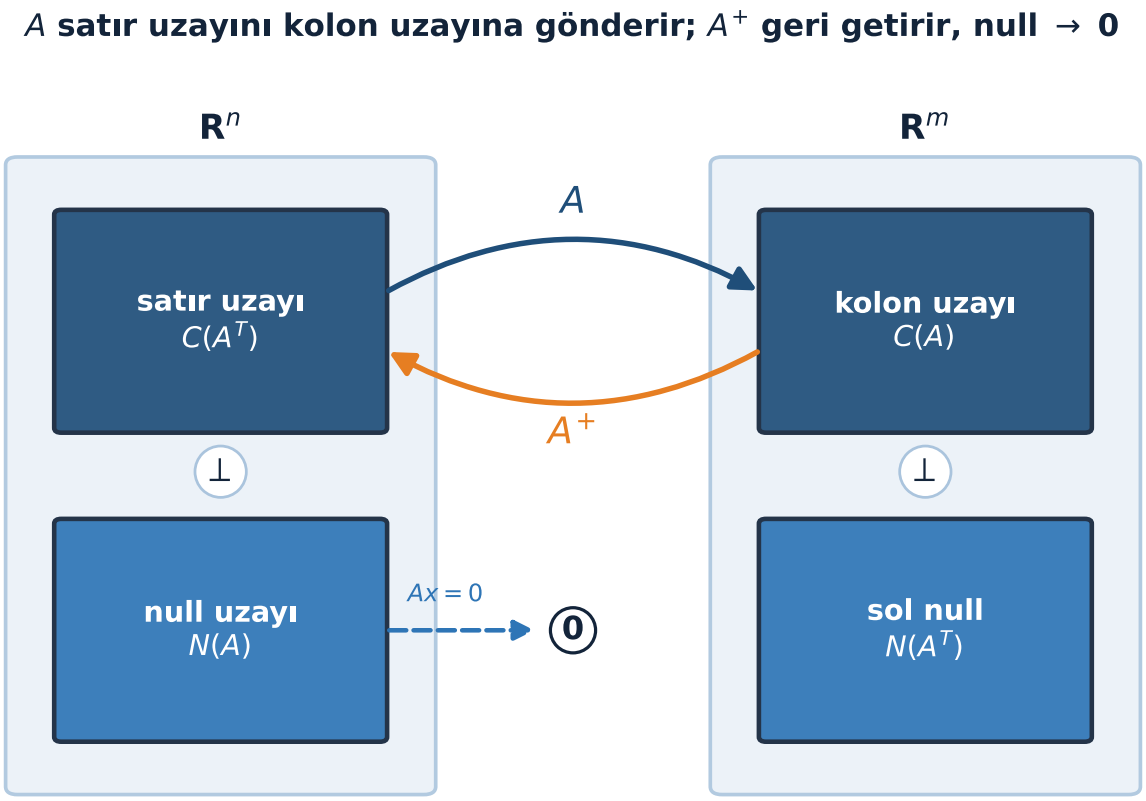
16.3 2. Pseudoinverse A⁺

İlk araç pseudoinverse'tir (sözde-ters). $m \times n$ bir A için A^+ matrisi $n \times m$'dir ve $A^+ A$'yı mümkün olduğunca birim matrise yaklaştırır.

> *"…I'm going to get as near to the identity as I can. That's the idea… of the pseudo inverse."* — Strang, 5:55

A tersinirse $A^+ = A^{-1}$ olur. Ama A dikdörtgense veya null uzayı varsa (kolonlar bağımlı), tam ters yoktur. Dört temel alt-uzay resmiyle: A, satır uzayını kolon uzayına tersinir biçimde gönderir (üst yarı); A^+ bunu geri getirir. Null uzayındaki vektörler 0'a gider — geri getirilemez, A^+ onları 0'a gönderir.

Bu haritayı Şekil 16.2 gösteriyor: A satır uzayını kolon uzayına gönderir, A^+ geri taşır, null uzayı sıfıra çöker.



Şekil 16.2: Dört temel alt-uzay ve A/A^+ haritası. Sol tarafta girdi uzayı $\mathbf{R}^n$: satır uzayı $C(A^T)$ (navy) ile null uzayı $N(A)$ (steel) dik tümleyenler. Sağ tarafta çıktı uzayı $\mathbf{R}^m$: kolon uzayı $C(A)$ (navy) ile sol null uzayı $N(A^T)$ (steel). A satır uzayını kolon uzayına gönderir (navy ok); pseudoinverse A^+ kolon uzayından satır uzayına geri taşır (turuncu ok); null uzayı sıfıra çöker ($Ax = 0$).

> Builder Notu — Tersi Olmayanın Tersi
>
> A^+, "tersi olmayan matrisin en iyi tersi"dir. ML'de rank-eksik tasarım matrisleri, eksik-belirlenmiş sistemler ve minimum-norm çözümlerde (Ders 11) kullanılır — regresyonun en genel hâli.

16.4 3. $A^+ = V\Sigma^+U^T$ (SVD ile)

Pseudoinverse için temiz formül SVD'den gelir, çünkü SVD her matris için vardır.

> *"…start with the SVD. Because the SVD works for any matrix."* — Strang, 14:34

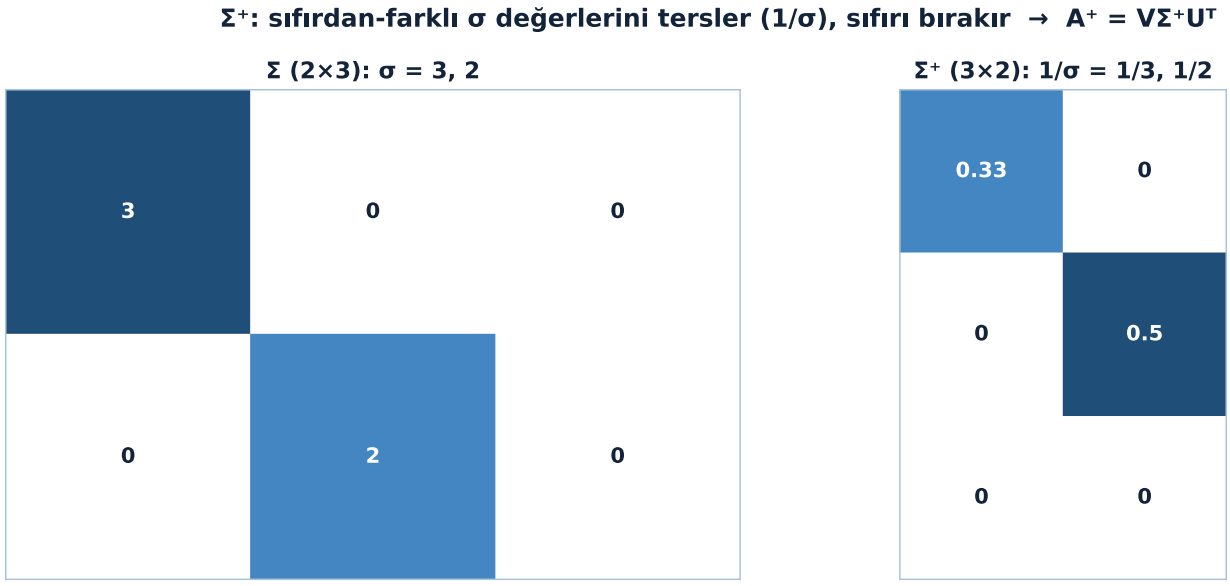
$A = U\Sigma V^T$ ise, üç çarpanı tersine çevir (ortogonal U, V için ters = transpoz):

$$A^+ = V\Sigma^+ U^T$$

Buradaki Σ^+, sıfırdan-farklı tekil değerleri ters çevirir, sıfırları sıfır bırakır:

$$\Sigma^+ = \text{diag}\left(\frac{1}{\sigma_1}, \dots, \frac{1}{\sigma_r}, 0, \dots, 0\right)$$

$\Sigma^+\Sigma$ çarpımı, ilk r köşegende 1, gerisi 0 olan bir projeksiyondur — yapabileceğimizin en iyisi. Bu tersleme kuralını Şekil 16.3 gösteriyor.



Şekil 16.3: Pseudoinverse'in kalbi: Σ^+ köşegen matrisi, Σ'nın sıfırdan-farklı her σ değerini tersler ($\sigma \to 1/\sigma$), sıfırları olduğu gibi bırakır. Burada Σ (2×3) için σ = 3, 2 iken Σ^+ (3×2) için 1/3, 1/2 olur. Bu yüzden $A^+ = V\Sigma^+U^T$ her matris için tanımlıdır.

> **Builder Notu — SVD Her Zaman Var**
>
> $A^+ = V\Sigma^+U^T$, `np.linalg.pinv`'in tam olarak hesapladığıdır. Küçük tekil değerleri (gürültü) eşikleyip atmak (truncated SVD), düzenlileştirilmiş bir pseudoinverse verir — ters problemlerde ve gürültülü regresyonda kararlılık sağlar.

16.5 4. Yol 1: x̂ = A⁺b

En genel çözüm doğrudan pseudoinverse ile gelir:

$$\hat{x} = A^{+}b = V\Sigma^{+}U^{T}b$$

Bu yol **her** durumda çalışır — A dikdörtgen, rank-eksik, null uzaylı olsa bile. Null uzayı varsa bu, sonsuz çözüm arasından **minimum-norm** olanı seçer (Ders 11). Diğer üç yol, A'nın bağımsız kolonlu olmasını gerektirir; pseudoinverse bu koşula ihtiyaç duymaz.

> Builder Notu — En Güvenli Çözüm
>
> $\hat{x} = A^{+}b$, `np.linalg.lstsq`/`pinv`'in döndürdüğü en güvenli çözümdür: rank-eksik durumlarda bile patlamaz, minimum-norm cevabı verir. Ters problemler, az-örnekli regresyon ve eksik-belirlenmiş sistemler için varsayılan.

16.6 5. Yol 2: Normal Denklemler

İkinci yol, A'nın bağımsız kolonları varsa en yaygın olanıdır. $\|Ax - b\|^2$ minimize edilince türev sıfırlanır ve **normal denklemler** çıkar:

> *"We follow Gauss's advice to get the best we can."* — Strang, 30:20

$$A^{T}A\,\hat{x} = A^{T}b$$

> *"A transpose A is going to come from there…"* — Strang, 31:58

A'nın bağımsız kolonları varsa $A^{T}A$ tersinirdir ve çözüm $\hat{x} = (A^{T}A)^{-1}A^{T}b$ olur.

> *"If A has independent columns…"* — Strang, 37:58

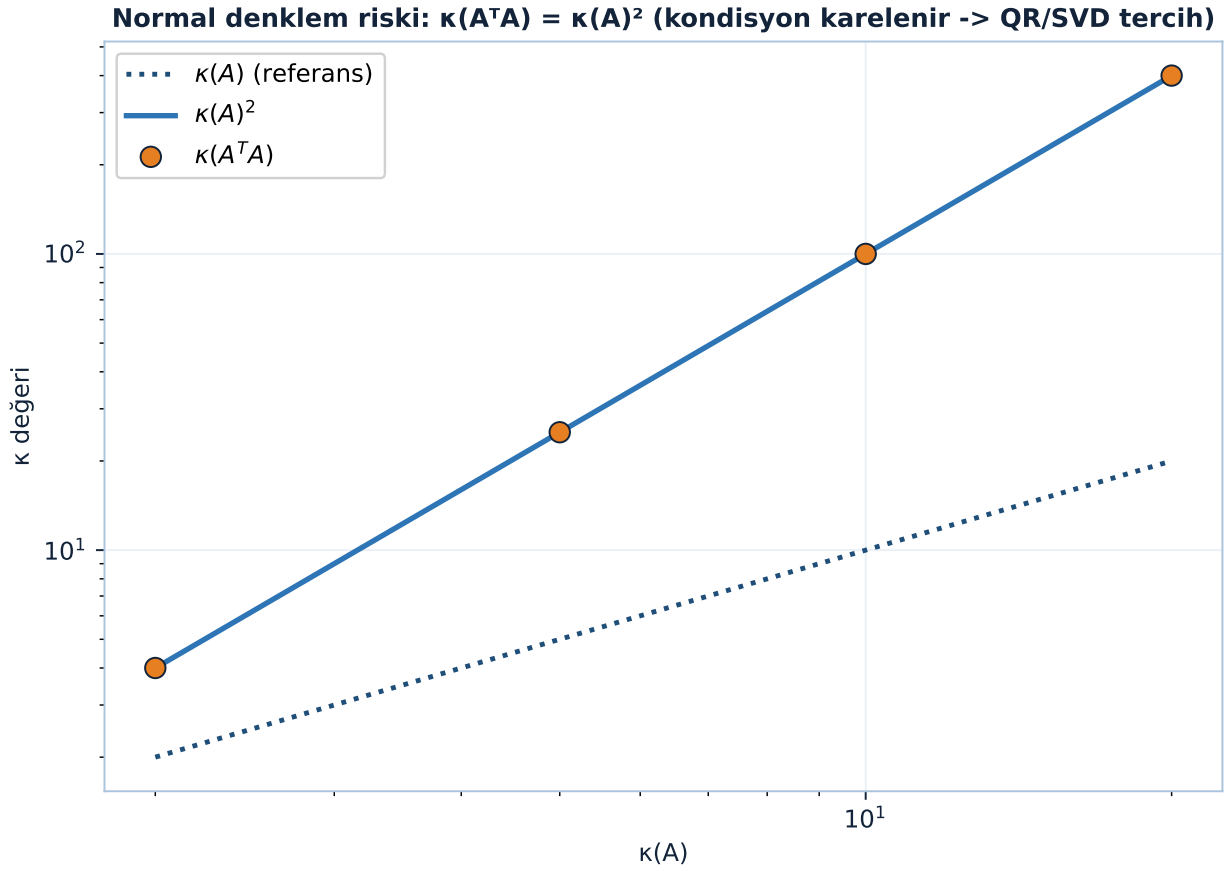
Bu, istatistikteki lineer regresyondur; çoğu pratik problemde matrisi kurup doğrudan çözersin. Ama bu basitliğin bir bedeli var: $A^{T}A$ kurmak kondisyon sayısını karelendiriyor, ki bunu Şekil 16.4 net biçimde gösteriyor.

> Builder Notu — Gauss'un Yolu
>
> Normal denklemler basit ama $A^{T}A$ kondisyon sayısını kareler (Ders 6) — sayısal olarak riskli. Bu yüzden ciddi problemlerde QR (Yol 3) veya SVD (Yol 1) tercih edilir; `lstsq` arka planda normal denklem kurmaz, QR/SVD kullanır.

16.7 6. Geometri: Kolon Uzayına Projeksiyon

Normal denklemlerin geometrisi nettir. b genelde A'nın kolon uzayında değildir (o yüzden çözüm yok). En iyi $A\hat{x}$, b'nin kolon uzayına **dik izdüşümüdür**:

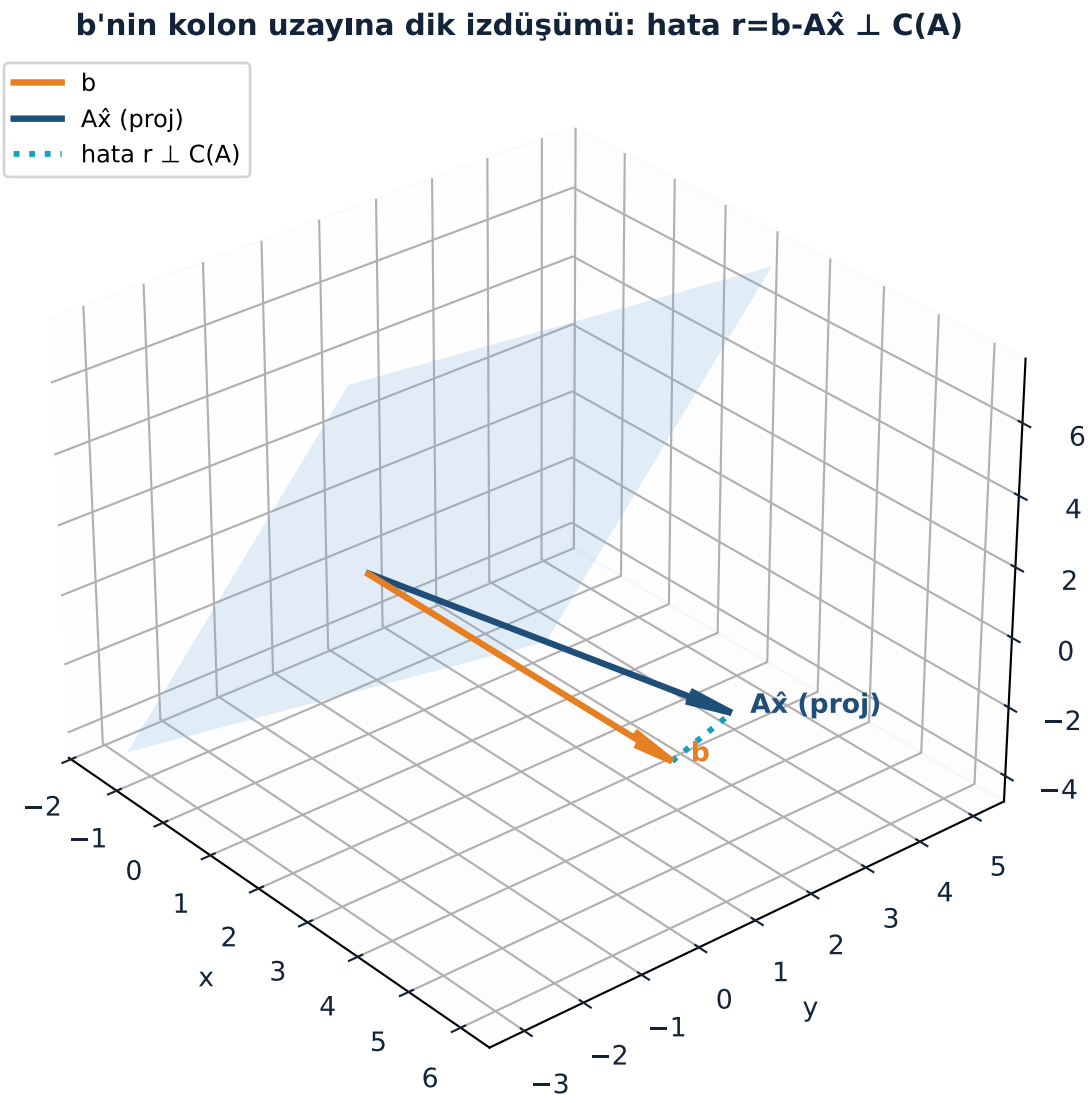
> *"…It's the projection."* — Strang, 35:41



Şekil 16.4: Normal denklemler $A^T A\hat{x} = A^T b$ kurulduğunda matrisin kondisyon sayısı **karelenir**: $\kappa(A^T A) = \kappa(A)^2$. Şekil, artan kondisyonlu birkaç köşegen matris $A = \mathrm{diag}(1, c)$ için ölçülen $\kappa(A^T A)$ değerlerini (turuncu) $\kappa(A)^2$ teorik eğrisiyle (çelik) kıyaslar; noktalar tam eğri üzerine oturur. Lacivert kesik çizgi $\kappa(A)$ referansıdır. Log-log eksende ikisi paralel kayık doğrulardır — bu yüzden büyük kondisyonlu problemlerde $A^T A$ kurmak sayısal hatayı katlar ve QR/SVD tercih edilir.

$$A\hat{x} = \text{proj}_{C(A)}(b), \qquad (b - A\hat{x}) \perp C(A)$$

Hata vektörü $b - A\hat{x}$, kolon uzayına diktir — bu da $A^T(b - A\hat{x}) = 0$, yani $A^T A\hat{x} = A^T b$ demektir. Geometri (diklik) ile cebir (normal denklem) aynı şeyi söyler. Bu **ortogonallik ilkesi**, least squares'in kalbidir. Şekil 16.5 bu izdüşümü üç boyutta gösteriyor: b düzlemin dışında, $A\hat{x}$ düzlemin üzerinde, hata vektörü düzleme dik.



Şekil 16.5: b'nin kolon uzayı $C(A)$'ya dik izdüşümü. Turuncu vektör gözlem $b = (6, 0, 0)$'ı, lacivert vektör izdüşüm $A\hat{x} = (5, 2, -1)$'i gösterir; bu nokta düzlemin **üzerinde** yatar. Teal kesik çizgi artık $r = b - A\hat{x} = (1, -2, 1)$'dir ve $C(A)$'ya **diktir** ($r \perp C(A)$, yani r her iki kolona ortogonaldir). Least squares çözümü $\hat{x} = (5, -3)$ tam olarak bu dik izdüşümü veren $\hat{x}$'tir: $\|b - A\hat{x}\|$ en küçük olur çünkü r düzleme dik düşer.

> Builder Notu — Diklik İlkesi
>
> "Hata, kolon uzayına diktir" ilkesi her yerde: Kalman filtresi (yenilik artığı $\perp$ geçmiş), Gauss-Markov teoremi, ve sinir ağı son katmanının lineer çözümü. Projeksiyon = en iyi tahmin = ortogonallik koşulu.

16.8 7. Doğru Uydurma Örneği

Least squares'in ünlü örneği: gürültülü noktalara en iyi doğruyu ($C + Dx$) uydurmak.

> *"…the famous example of least squares is fit a straight line to the b's."* — Strang, 25:02

Her ölçüm x_i noktasında bir b_i değeri var; bilinmeyenler C ve D (iki sütun → rank 2). Sistem:

$$A = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix}, \quad x = \begin{pmatrix} C \\ D \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

m ölçüm, 2 bilinmeyen → çok fazla denklem, çözüm yok. Noktalar tam doğru üstündeyse çözüm vardır; gürültü varsa (gerçek hayat) yoktur → en iyi C, D'yi least squares ile bul. Şekil 16.6 bunu gösteriyor: noktalar saçılı, en iyi doğru dikey hataların karelerini minimize ediyor.

> Builder Notu — İlk Model
>
> Bu, ML'deki en basit modeldir: tek özellikli lineer regresyon. Çok özellikli hâli (A'nın çok sütunu) aynı normal denklemle çözülür. Polinom uydurma, trend kestirimi ve baseline modeller hep bu yapıdır.

16.9 8. Yol 3: QR / Gram-Schmidt

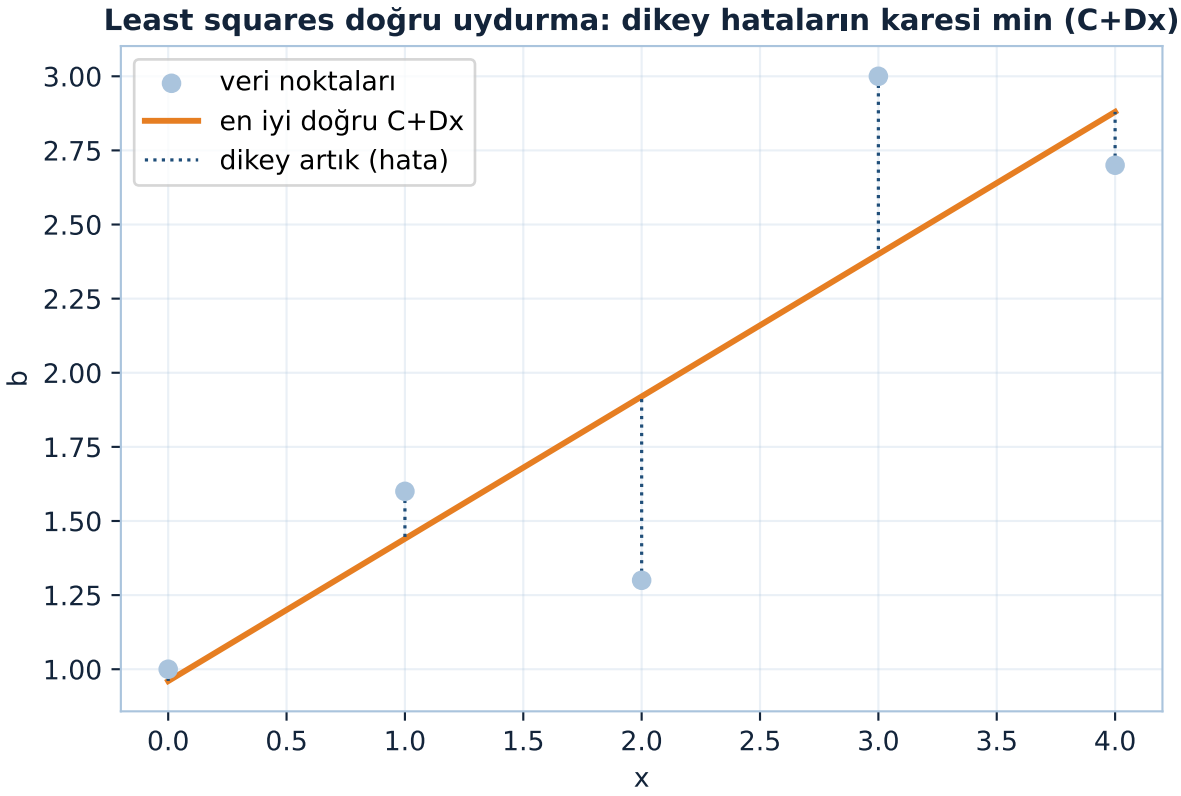
Üçüncü yol, normal denklemin sayısal sorununu aşar: önce A'nın kolonlarını ortonormal yap (Gram-Schmidt), yani $A = QR$.

> *"…Gram-Schmidt is a way… to get [orthogonal] columns."* — Strang, 47:11

$A = QR$ yerine koyunca $A^TA = R^TQ^TQR = R^TR$ olur (çünkü $Q^TQ = I$), ve normal denklem sadeleşir:

$$A = QR \;\Rightarrow\; R\hat{x} = Q^Tb$$

R üst üçgensel olduğundan geri yerine koymayla çözülür; A^TA hiç kurulmaz, kondisyon karelenmez. Bu, pratik least squares'in standart yoludur.



Şekil 16.6: Least squares doğru uydurma: dikey hataların karesi minimize edilir (C+Dx).

Builder Notu — Sayısal Güvenlik

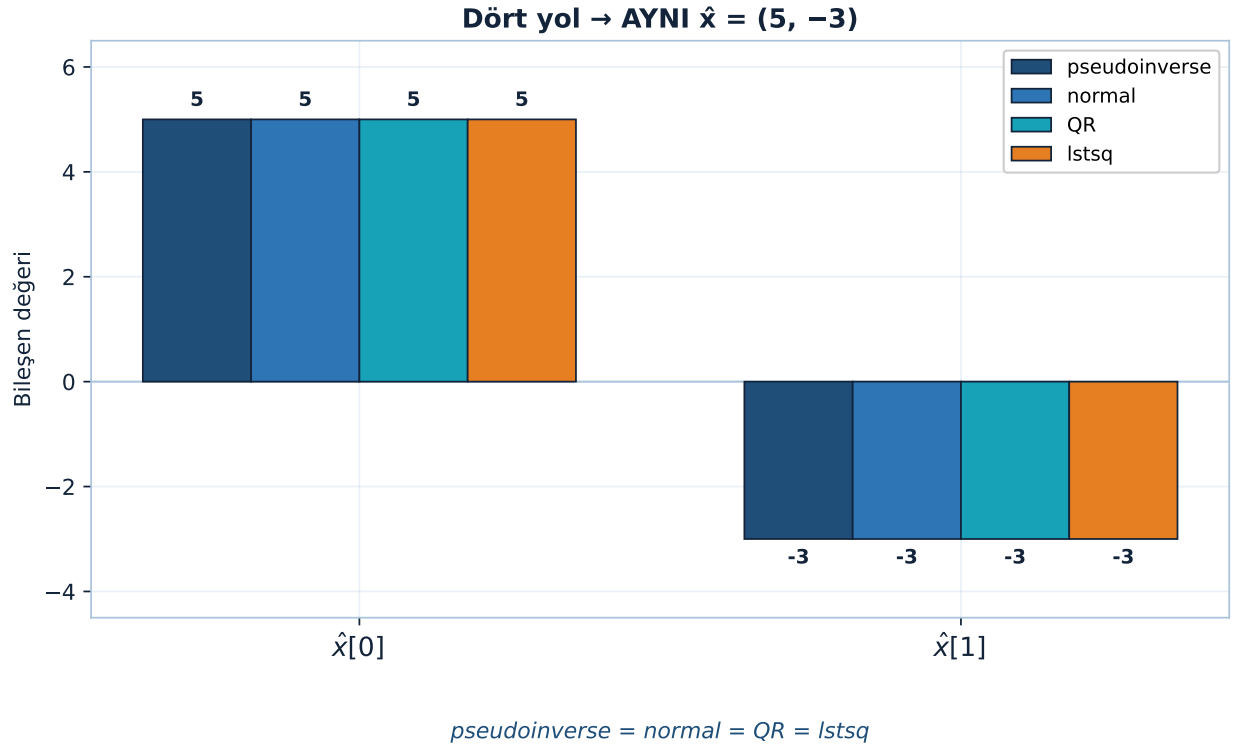
QR, `numpy.linalg.lstsq`'in (ve LAPACK'in) varsayılan yoludur: kararlı, A^TA'dan kaçınır. Householder yansımaları (Ders 3) QR'ı kararlı üretir. Ortonormallik = sayısal güvenlik.

16.10 9. Pseudoinverse = Normal Denklem (Bağımsız Kolonlarda)

Dört yol aynı $\hat{x}$'e varır. A'nın bağımsız kolonları varsa (null uzay yok), pseudoinverse tam olarak normal denklem çözümüne eşittir:

$$A^+ = (A^TA)^{-1}A^T \quad \text{(left inverse)}$$

Bu A^+, A'yı soldan tersler ($A^+A = I$) ama sağdan terslemez ($AA^+ \neq I$ — kolon uzayına projeksiyon). SVD formülü $A^+ = V\Sigma^+U^T$ ise hiçbir koşul gerektirmez ve rank-eksikte de geçerlidir; bağımsız kolonlarda ikisi çakışır. Dört yolun da aynı $\hat{x}$'e vardığını Şekil 16.7 sayısal olarak doğruluyor.



Şekil 16.7: Dört yol → aynı x̂ = (5, −3): pseudoinverse, normal denklemler, QR ve lstsq özdeş çözümü verir.

Builder Notu — Sol Ters

"Sol ters var ama sağ ters yok" durumu, ince-uzun (tall) tasarım matrislerinin tipik hâlidir: çok örnek, az parametre. $A^+A = I$ (parametreler kurtarılır) ama AA^+ projeksiyon (veri tam kurtarılamaz, gürültü düşer) — regresyonun matematiksel imzası.

16.11 10. Yol 4: Büyük Sistemler İçin İteratif

Dördüncü yol, A çok büyük veya seyrek olduğunda devreye girer (Strang bu dersin sonunda zamana yetişemedi, ileri derslerde işlenir). Doğrudan A^TA veya QR çözmek pahalı/imkânsızsa, **iteratif yöntemler** (conjugate gradient ve benzeri) çözümü adım adım yaklaşır.

$$\text{Devasa/seyrek A} \;\rightarrow\; \text{iteratif (conjugate gradient)}$$

Bu yöntemler matrisi hiç açıkça kurmaz; yalnızca A ile çarpım (matrix-vector product) kullanır — milyon-boyutlu sistemlerde tek pratik seçenek.

> **Builder Notu — Devasa Sistemler**
>
> İteratif çözücüler (conjugate gradient, LSQR), büyük-ölçek ML ve bilimsel hesabın belkemiğidir: yalnızca Ax çarpımı gerektirdiklerinden seyrek/yapısal matrislerde belleğe sığar. Gradient descent (Ders 21-23) bu ailenin optimizasyon akrabasıdır.

16.12 Bu Dersin Özeti

1. **Least squares** — çözülemeyen $Ax = b$ için $\|Ax - b\|^2$'yi minimize et.
2. **Pseudoinverse** A^+ — satır/kolon uzayında ters, null uzaylarında 0; her matris için.
3. $A^+ = V\Sigma^+U^T$ — SVD'den; Σ^+ sıfırdan-farklı σ'ları tersler, sıfırları bırakır.
4. **Yol 1:** $\hat{x} = A^+b$ — en genel; rank-eksikte minimum-norm çözüm.
5. **Yol 2:** $A^TA\hat{x} = A^Tb$ — normal denklemler; bağımsız kolonlar gerekir.
6. **Geometri** — $A\hat{x} = b$'nin kolon uzayına projeksiyonu; hata $\perp$ kolon uzayı.
7. **Doğru uydurma** — $C + Dx$; m ölçüm, 2 bilinmeyen.
8. **Yol 3: QR** — $A = QR$, $R\hat{x} = Q^Tb$; A^TA'dan kaçınır, kararlı.
9. $A^+ = (A^TA)^{-1}A^T$ — bağımsız kolonlarda pseudoinverse = normal denklem.
10. **Yol 4: iteratif** — büyük/seyrek için conjugate gradient.

> **Tek Bir Cümle**
>
> Çözülemeyen $Ax = b$ için least squares, b'yi A'nın kolon uzayına dik olarak projekte eder (hata $\perp$ kolon uzayı); bu $\hat{x}$'i pseudoinverse (SVD), normal denklemler veya QR ile bulursun — hepsi aynı en-iyi çözüme varır.

16.13 Kontrol Soruları

i Soru 1: (1,1), (2,2), (3,2) noktalarına en iyi doğruyu (C + Dx) normal denklemlerle uydur.

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}$$

Cevap: $A^TA = \begin{pmatrix} 3 & 6 \\ 6 & 14 \end{pmatrix}$, $A^Tb = (5, 11)$. Normal denklem: $3C + 6D = 5$, $6C + 14D = 11$. Çözüm: $D = 1/2$, $C = 2/3$. En iyi doğru: $y = 2/3 + (1/2)x$. (Noktalar tam doğru üstünde değil, bu en küçük kareler uyumu.)

i Soru 2: A = [[1, 2], [2, 4]] (rank 1) için pseudoinverse A^+'ı bul.

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$$

Cevap: $A = cc^T$, $c = (1, 2)$. Tek tekil değer $\sigma_1 = \|c\|^2 = 5$, tekil vektör $u = v = c/\sqrt{5}$. $A^+ = (1/\sigma_1)vu^T = (1/25)cc^T = A/25$:

$$A^+ = \frac{1}{25}\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$$

Kontrol: A^+A satır uzayına projeksiyon (birim değil, çünkü rank 1).

i Soru 3: Least squares çözümünde hata vektörü b – Ax̂ neden kolon uzayına diktir?

Cevap: $A\hat{x}$, b'nin kolon uzayına en yakın noktasıdır (projeksiyon). En yakın nokta, hatanın uzaya **dik** olduğu noktadır (Pythagoras: dik olmayan her yön daha uzun hata verir). Diklik: kolon uzayının her vektörü Ay için $(Ay)^T(b - A\hat{x}) = 0 \to A^T(b - A\hat{x}) = 0 \to A^TA\hat{x} = A^Tb$. Yani ortogonallik ilkesi doğrudan normal denklemleri verir.

i Soru 4: Ne zaman pseudoinverse (A^+), ne zaman normal denklem (A^TA) kullanırsın?

Cevap: Normal denklem / QR: A'nın bağımsız kolonları varsa (tall, full-rank). QR sayısal olarak daha kararlı, pratik seçim. **Pseudoinverse (SVD):** A rank-eksik veya çok kötü koşullu ise — null uzayı varken sonsuz çözüm arasından minimum-norm olanı seçer, patlamaz. Çok büyük/seyrek A için ise iteratif yöntemler (Yol 4). Pratikte `np.linalg.lstsq` bu kararı senin için verir (QR/SVD tabanlı).

16.14 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. $(0, 6)$, $(1, 0)$, $(2, 0)$ noktalarına en iyi doğruyu $(C + Dx)$ normal denklemlerle uydur. Hata vektörünü ve onun kolon uzayına dik olduğunu doğrula.

Egzersiz 2. Aşağıdaki matrisin pseudoinverse'ini SVD ile bul. A^+A ve AA^+'yı hesapla; hangisi birim, hangisi projeksiyon?

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$$

Egzersiz 3. A bağımsız kolonluysa $A^+ = (A^TA)^{-1}A^T$ olduğunu göster. $A^+A = I$ (sol ters) ama $AA^+ \neq I$ (projeksiyon) olduğunu açıkla.

Egzersiz 4. Python ile dört yolu karşılaştır:

```
import numpy as np

A = np.array([[1.0, 0.0], [1.0, 1.0], [1.0, 2.0]])
b = np.array([6.0, 0.0, 0.0])

x_normal = np.linalg.solve(A.T @ A, A.T @ b)    # Yol 2
x_pinv   = np.linalg.pinv(A) @ b                # Yol 1
x_lstsq  = np.linalg.lstsq(A, b, rcond=None)[0] # QR/SVD
print("normal:", x_normal, " pinv:", x_pinv, " lstsq:", x_lstsq)  # aynı

# Hata kolon uzayına dik mi?
r = b - A @ x_lstsq
print("A^T(b − Ax̂ ) =", A.T @ r)   # ≈ 0
```

Egzersiz 5. (Ders 10 habercisi.) Ders 10 $Ax = b$'nin tüm zorluklarını tarar (kare/dikdörtgen, tam/eksik rank, iyi/kötü koşullu). Aşağıdaki üç matris için hangi durumda olduğunu söyle: tersinir mi, least squares mi, minimum-norm mu gerekir?

$$A_1 = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}, \; A_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \; A_3 = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$

16.15 Sonraki Ders İçin Hazırlık

Ders 10: Ax = b'nin Zorluklarına Genel Bakış

Ders 9'da least squares'i ve dört yolu işledik. Ders 10, $Ax = b$'nin tüm olası durumlarını sistematik tarar — hangi durumda hangi yöntemin gerektiğini.

- Kare/dikdörtgen, tam/eksik rank kombinasyonları
- Tersinir, least squares, minimum-norm, her ikisi
- İyi vs kötü koşullu sistemler
- Gram-Schmidt ve sayısal yöntemlere köprü

⚠ Ders 10 Öncesi Yapılacak

- Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (durum sınıflandırma).
- Python'da `np.linalg.matrix_rank` ile birkaç matrisin rankını ve durumunu incele.
- Ana cümleyi tekrar oku: "Least squares = b'yi kolon uzayına projekte etmek; hata $\perp$ kolon uzayı."

16.16 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
Least squares	Çözülemeyen $Ax = b$ için $\min \|Ax - b\|^2$	23m05
Pseudoinverse A^+	En iyi ters; satır/kolon uzayında, null $\to 0$	5m55
$A^+ = V\Sigma^+U^T$	SVD'den; Σ^+ sıfırdan-farklı σ'ları tersler	14m34
Gauss least squares	ℓ^2 hatasını minimize et	30m20
Normal denklemler	$A^TA\hat{x} = A^Tb$; bağımsız kolonlar	31m58
Projeksiyon	$A\hat{x} = b$'nin kolon uzayına izdüşümü; hata $\perp$	35m41
Doğru uydurma	$C + Dx$; m ölçüm, 2 bilinmeyen	25m02
Bağımsız kolon	A^TA tersinir, Gauss çalışır	37m58
Yol 3: QR	$A = QR$, $R\hat{x} = Q^Tb$; A^TA'dan kaçınır	47m11
Dört yol	A^+, normal denklem, QR, iteratif	5m21

16.17 ML Bağlantıları Özeti

1. **Least squares** → lineer regresyonun çekirdeği; MSE kaybının atası.
2. **Pseudoinverse** A^+ → rank-eksik/dikdörtgen regresyon; minimum-norm çözüm.
3. **Normal denklemler** → klasik regresyon; ama A^TA kondisyonu kareler (dikkat).
4. **QR** → kararlı least squares; `lstsq`'in varsayılanı; ortonormal güvenlik.
5. **Projeksiyon / ortogonallik** → Kalman filtresi, Gauss-Markov, son katman lineer çözümü.
6. **Truncated SVD** A^+ → gürültülü/ters problemlerde düzenlileştirilmiş çözüm.
7. **İteratif (conjugate gradient)** → büyük/seyrek sistemler; yalnız Ax çarpımı gerektirir.

❗ Eğer bu dersten tek bir şey alıp gidersen

Çözümü olmayan $Ax = b$ için least squares, b'yi A'nın kolon uzayına dik olarak projekte eder — hata vektörü kolon uzayına diktir (ortogonallik ilkesi), ki bu doğrudan normal denklemleri ($A^TA\hat{x} = A^Tb$)

verir. Aynı en-iyi $\widehat{x}$'e dört yoldan varılır: pseudoinverse (SVD, en genel), normal denklemler, QR (kararlı) ve büyük sistemler için iteratif.

17 Ax = b'nin Zorluklarına Genel Bakış

Bir lineer denklem doktoru — boyut, rank ve kondisyon sayısı hangi aleti seçeceğini söyler

> **Bölüm bilgisi**
>
> **Video:** Survey of Difficulties with Ax = b · **OCW:** MIT 18.065 Lecture 10 · **Okuma süresi:** ~36 dk · **Eğitmen:** Gilbert Strang · **Önkoşul:** Ders 9 (least squares, pseudoinverse ve dört çözüm yolu).

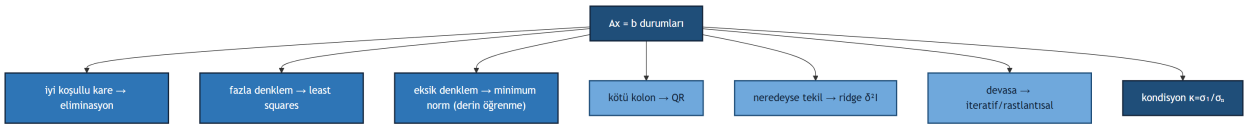
17.1 Bu Derste Ne Var?

Bir *lineer denklem doktoru*: Ders 10, $Ax = b$'nin tüm olası durumlarını (boyut, rank, koşullanma) sistematik tarar ve her birinde ne yapılacağını reçete eder. Bu, önceki dokuz dersi tek bir haritada birleştiren bir özet.

Üç temel fikir:

1. **Durum haritası** — kare/iyi koşullu (eliminasyon), fazla denklem (least squares), eksik denklem (minimum norm), neredeyse tekil (düzenlileştirme), çok büyük (iteratif/randomized).
2. **Kondisyon sayısı** $\kappa = \sigma_1/\sigma_n$ — bir matrisin "ne kadar zor" olduğunun ölçüsü; büyükse tehlike.
3. **Düzenlileştirme → pseudoinverse** — ceza terimi $\delta^2\|x\|^2$ ekle; $\delta \to 0$ limitinde çözüm tam pseudoinverse'e gider.

Bütün bu durumların tek bir merkezden nasıl dallandığını Şekil 17.1 özetliyor: $Ax = b$ tek bir problem değil, her biri kendi aletini isteyen bir problemler ailesidir.



Şekil 17.1: Ders 10 kavram haritası: Ax = b durumları merkezinden çözüm yollarına dallar — iyi koşullu kare (eliminasyon), fazla denklem (least squares), eksik denklem (minimum norm, derin öğrenme), kötü kolon (QR), neredeyse tekil (ridge δ^2I), devasa (iteratif/rastlantısal); ayrıca kondisyon sayısı $\kappa=\sigma_1/\sigma_n$ zorluğun ölçüsü.

> *"…the subject of today's class, it's Ax = b."* — Strang, 0:25

Builder Notu — Lineer Denklem Doktoru

- **Durum haritası = pratik karar ağacı:** problemin boyut/rank/koşuluna göre doğru aletle (backslash, lstsq, pinv, conjugate gradient) eşleştir.
- **Eksik-belirlenmiş = derin öğrenme:** ağırlık > örnek; sonsuz çözüm, hangisini seçtiğin (implicit bias) genellemeyi belirler.
- **Ridge düzenlileştirme** — $\delta^2 I$ eklemek matrisi tersinir kılar; $\delta \to 0$'da pseudoinverse, istatistikte ridge regresyon/Tikhonov.
- **Ölçeğe göre alet:** küçük (doğrudan), büyük (iteratif/Krylov), devasa (rastlantısal LA — örnekleme).

Tek cümle: $Ax = b$'nin zorluğu boyut, rank ve kondisyon sayısıyla belirlenir; her durumun kendi çözüm aleti vardır ve düzenlileştirme ($\delta^2 I$) neredeyse tekil problemleri pseudoinverse'e doğru yumuşatır.

17.2 1. Ax = b Sözlüğü: Durumlar Haritası

Strang bu dersi bir *tanı kılavuzu* gibi kuruyor: $Ax = b$ farklı boyut, rank ve koşullarda çok farklı problemlerdir — her biri farklı bir çözüm ister.

> *"…the subject of today's class, it's Ax = b."* — Strang, 0:25

Kolaydan zora doğru durumlar: (0) pseudoinverse her zaman bir cevap verir; (1) iyi koşullu kare; (2) fazla denklem; (3) eksik denklem; (4) kötü kolonlar; (5) neredeyse tekil; (6) çok büyük; (7) devasa. Bu ders, semptoma göre reçeteyi eşleştiren bir sözlüktür.

Matrisin şekli ve rankı doğru aleti nasıl seçtiriyor, Şekil 17.2 bu tanı akışını gösteriyor: kare tam-rank → eliminasyon, tall → least squares, wide → minimum norm, neredeyse tekil → ridge.

Builder Notu — Tanı Kılavuzu

Bu harita pratik bir karar ağacıdır: problemin boyut/rank/koşuluna bakıp doğru aleti seç — `solve` (backslash), `lstsq`, `pinv`, conjugate gradient. ML mühendisliğinde "hangi çözücü" kararı sessizce buradan gelir.

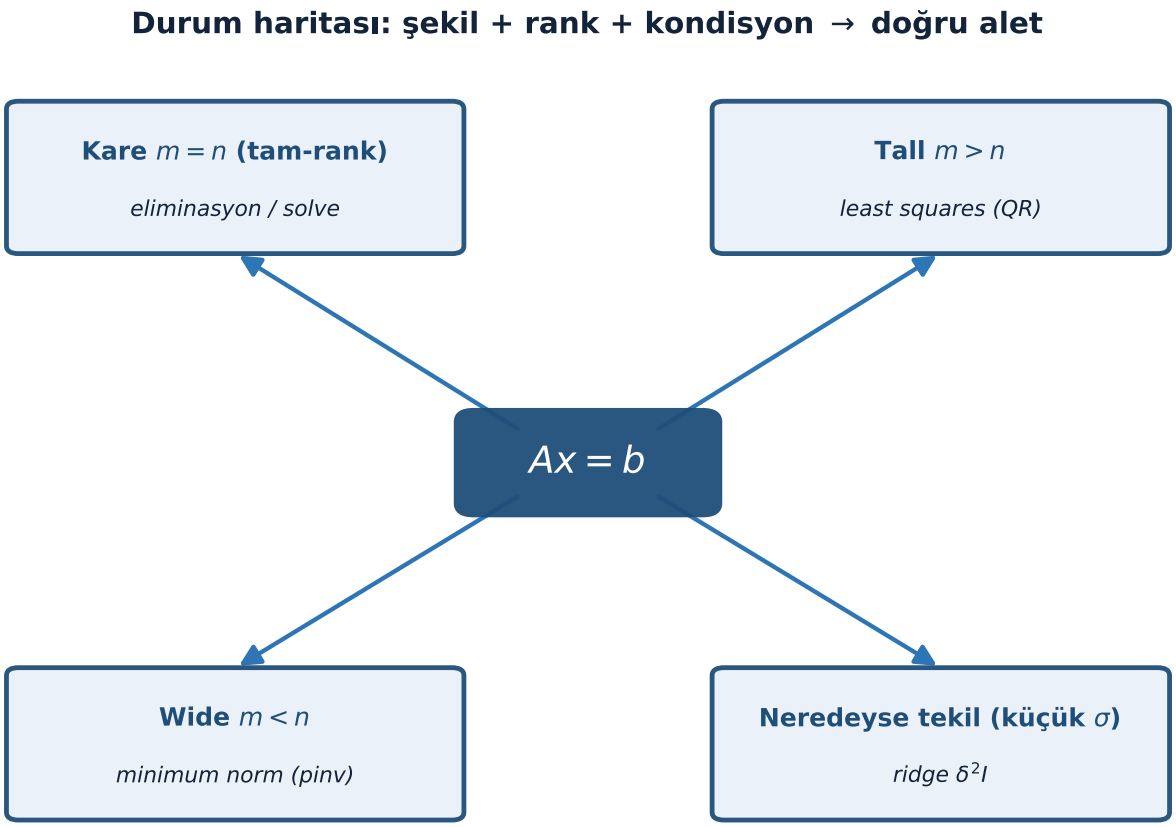
17.3 2. Durum 0-1: Pseudoinverse ve İyi Koşullu Kare

Durum 0: Pseudoinverse (Ders 9) her durumda bir cevap verir — sıfır tekil değerlerin tersini sıfır alır. Her zaman çalışır ama bazen "kolaycılık".

Durum 1 (iyi durum): A kare, makul boyutlu ve iyi koşullu. Sıradan eliminasyon (Matlab'de backslash) çalışır. "İyi koşullu" demek kondisyon sayısı makul demek:

$$\kappa = \frac{\sigma_1}{\sigma_n} \quad (\sigma_{\text{max}}/\sigma_{\text{min}})$$

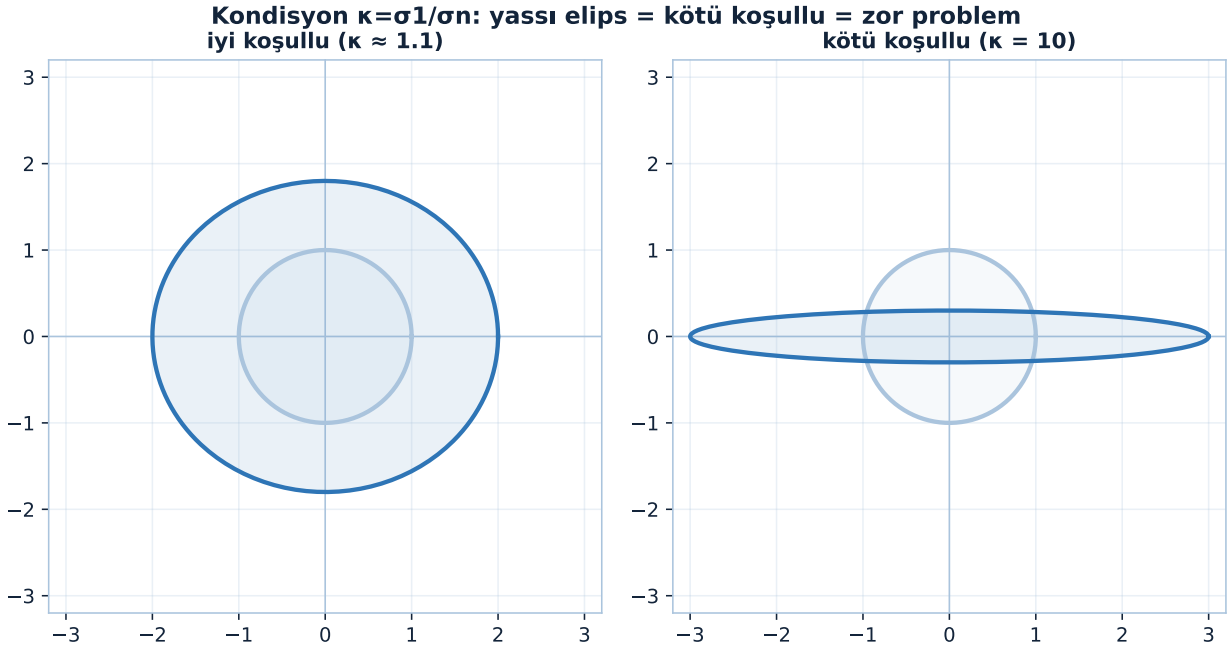
> *"…the condition number… sigma 1 over sigma n."* — Strang, 2:27



Şekil 17.2: Durum haritası: $Ax = b$ probleminin çözümü matrisin şekline, rankına ve kondisyonuna göre doğru alete yönlendirir. Kare $m = n$ tam-rank → eliminasyon / solve; tall $m > n$ (fazla denklem) → least squares (QR); wide $m < n$ (eksik denklem) → minimum norm (pseudoinverse); neredeyse tekil (küçük σ) → ridge $\delta^2 I$ ile düzenlileştirme. Merkezdeki $Ax = b$ düğümünden her duruma ait alete giden oklar bu tanı akışını gösterir.

κ büyükse (örneğin > 1000) matris zordur; küçükse rahat çözülür.

Birim çemberin bir matris altında nasıl bir elipse dönüştüğünü Şekil 17.3 gösteriyor: iyi koşullu matris çemberi neredeyse koruyor, kötü koşullu matris onu yassı bir elipse eziyor.



Şekil 17.3: Kondisyon sayısı κ=σ1/σn. Birim çember (açık gri) bir matrisle çarpılınca elipse dönüşür. İyi koşullu matris (sol, κ≈1.1) çemberi neredeyse aynı tutar; kötü koşullu matris (sağ, κ=10) onu çok yassı bir elipse ezer — küçük bir hata yönünde devasa büyür.

> Builder Notu — Zorluk Göstergesi
>
> Kondisyon sayısı $\kappa = \sigma_1/\sigma_n$, sayısal her şeyin "zorluk göstergesi"dir: büyük $\kappa \rightarrow$ yuvarlama hatası patlar, gradient descent yavaşlar (Ders 5 dar vadi). Normalizasyon, ön-koşullama ve batch norm hep κ'yı küçük tutmak içindir.

17.4 3. Durum 2: Fazla Denklem (Least Squares)

$m > n$: denklem sayısı bilinmeyenden çok (overdetermined). İstatistikte sürekli karşılaşılır — çözüm yok, en iyiyi bul (least squares):

$$A^T A\,\hat{x} = A^T b$$

A makul boyutluysa $A^T A$ tersinirdir ve `backslash` dikdörtgen A için bile bu çözümü verir (kare matris şart değil). Ders 9'da gördüğümüz normal denklemin ta kendisi.

> **Builder Notu — Klasik Regresyon**
>
> Bu, gözetimli öğrenmenin en yaygın hâli: çok örnek (m), az parametre (n) → least squares regresyon. Tahmin modellerinin, baseline'ların ve son-katman lineer çözümlerin standart durumu; aşırı-belirlenmiş ve iyi davranışlı.

17.5 4. Durum 3: Eksik Denklem (Derin Öğrenme)

$m < n$: yeterli denklem yok (underdetermined). Sonsuz çözüm var; birini seçmek gerekir. Bu, derin öğrenmenin tipik durumudur — ağda örnek sayısından çok daha fazla ağırlık vardır.

> *"…that's typical of deep learning."* — Strang, 4:34

Hangi çözümü seçersin? Minimum-norm (ℓ^2, en kısa çözüm) ya da ℓ^1 (seyrek). Strang'in heyecanlandığı açık soru: stokastik gradient descent (Ders 25) hangi çözüme gider — minimum ℓ^1'e mi? Bu, derin öğrenmenin neden çalıştığının kalbindeki sorudur.

Sonsuz çözüm doğrusunun üzerinde minimum-norm çözümün nasıl orijine en yakın nokta olduğunu Şekil 17.4 gösteriyor: gradyan inişi örtük yanlılıkla aynı $(1, 1)$ noktasını seçer.

> **Builder Notu — Derin Öğrenmenin Hâli**
>
> Eksik-belirlenmiş = fazla-parametreli (overparametrized) modern ağlar. Sonsuz sıfır-kayıp çözümü vardır; algoritmanın hangisini seçtiği (implicit bias) genellemeyi belirler. "Parametre > veri ama yine de çalışıyor" paradoksunun matematiksel zemini budur.

17.6 5. Durum 4: Kötü Kolonlar (Gram-Schmidt / QR)

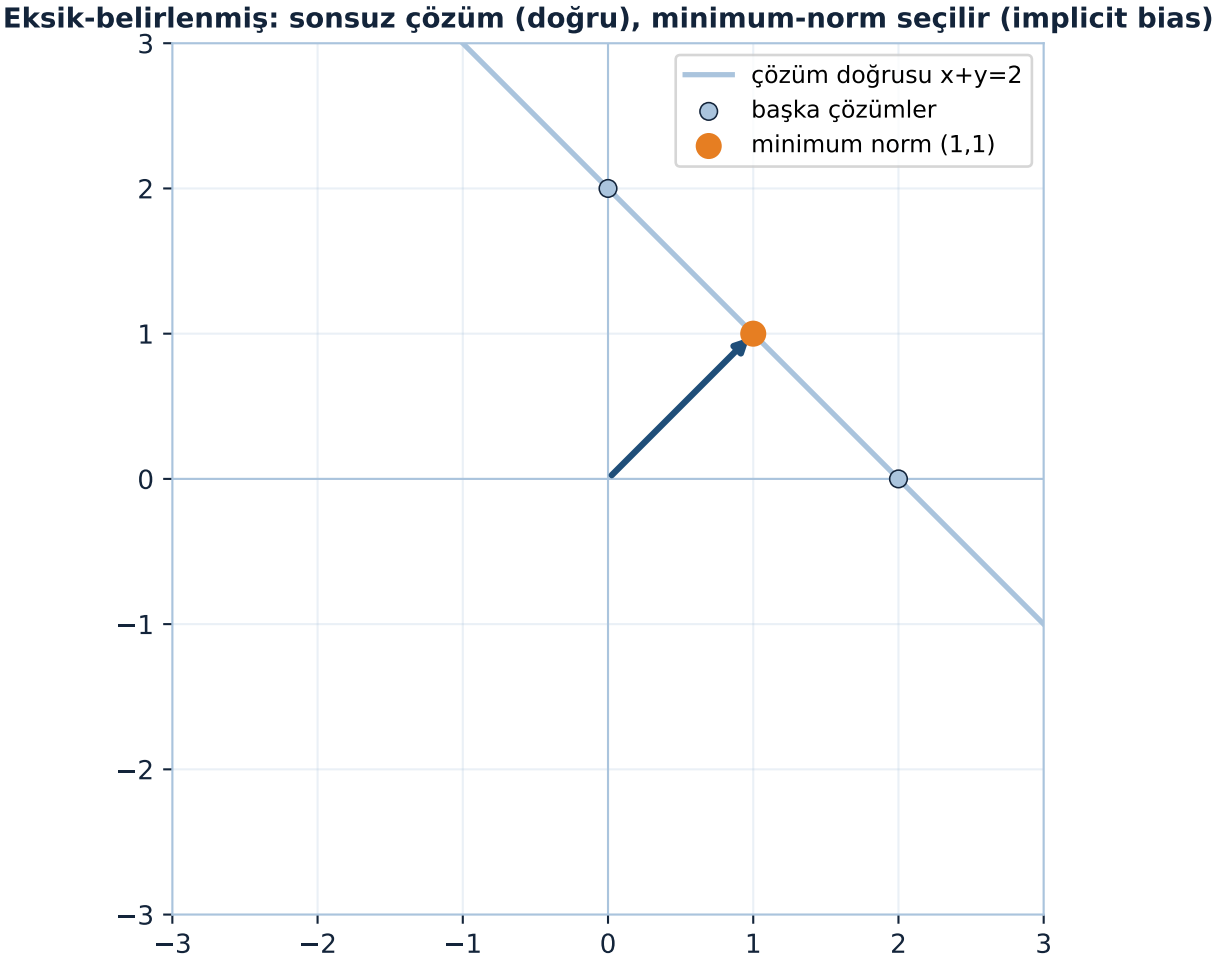
Kolonlar neredeyse bağımlıysa (kötü koşullu taban) matris hâlâ tersinirdir ama tersi devasadır. Çözüm: kolonları ortogonalleştir.

> *"…You orthogonalize columns."* — Strang, 7:06

Verilen kolonlar yerine kolon uzayında ortonormal bir taban ($Q_1 \dots Q_n$) bul; ikisi üst üçgensel R ile bağlıdır — bu $A = QR$'dir (Gram-Schmidt). Sayısal kararlılık için **kolon pivotlama** eklenir (kolonları daha iyi sırada al, tıpkı eliminasyondaki satır değişimi gibi).

$$A = QR \quad (Q: \text{ ortonormal},\ R: \text{ upper triangular})$$

> **Builder Notu — Kötü Tabanı Düzelt**
>
> QR + kolon pivotlama, rank-açığa-çıkaran faktorizasyonun temelidir: hangi kolonların gerçekten bağımsız olduğunu bulur. Özellik seçimi, kötü-koşullu tasarım matrislerini düzeltme ve sayısal kararlılıkta kritik (LAPACK'in geqp3).



Şekil 17.4: Eksik-belirlenmiş Ax=b: çözüm doğrusu x+y=2 üzerindeki sonsuz çözümden orijine en yakın olan (1,1) minimum-norm çözümüdür. Gradyan inişi bu örtük yanlılıkla aynı noktayı seçer.

17.7 6. Durum 5: Neredeyse Tekil → Düzenlileştirme

En ilginç durum: matris neredeyse tekil (sıfıra çok yakın tekil değerler). Tersi mantıksız büyüklükte; yuvarlama hatası her şeyi yok eder. Klasik kaynak: **ters problemler** (bir sistemin çıktısından parametrelerini geri bulmak — örneğin devre elemanlarını ölçümlerden kestirmek). Çare: bir **ceza terimi** ekle.

> *"…adding a penalty term."* — Strang, 14:49

Pseudoinverse "kolaycılık" sayılır; gerçek çözüm probleme düzenlileştirme katmaktır. $\delta^2\|x\|^2$ cezası matrisi tersinir ve iyi koşullu yapar.

> **Builder Notu — Gürültüye Karşı**
>
> Neredeyse tekil = küçük σ'lar = κ devasa. Ters problemler (tomografi, deconvolution, sistem kimliklendirme) bu sınıftadır; ham pseudoinverse gürültüyü patlatır. Düzenlileştirme (sonraki bölüm) bilgiyle gürültü arasında denge kurar.

17.8 7. Ceza Terimi: $(A^TA + \delta^2 I)x = A^Tb$

Düzenlileştirme, least squares'e bir ceza ekler: çözümün hem veriye uyması hem küçük kalması istenir.

$$\min_x \ \|Ax - b\|^2 + \delta^2\|x\|^2$$

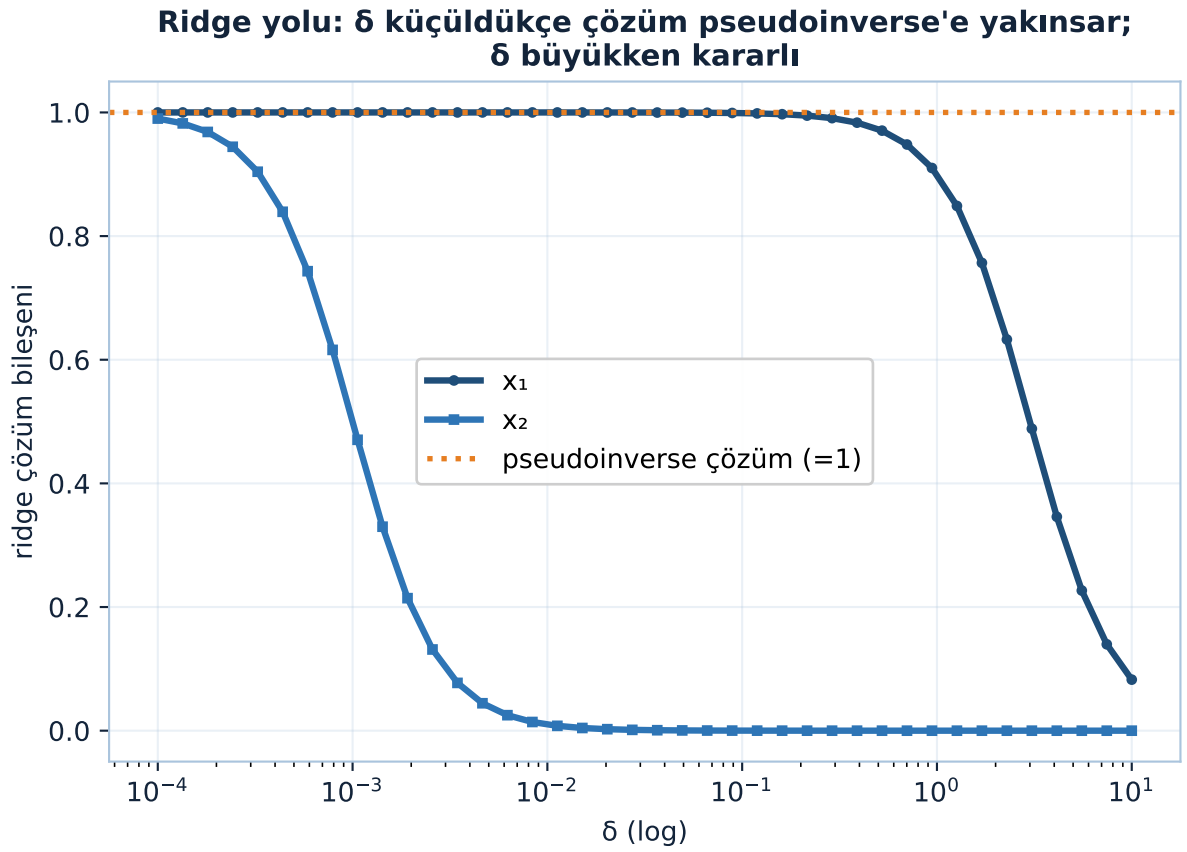
Bunu minimize edince düzeltilmiş normal denklem çıkar — köşegene δ^2 eklenir:

$$(A^TA + \delta^2 I)\,\hat{x} = A^Tb$$

$\delta > 0$ olduğu sürece $A^TA + \delta^2 I$ pozitif tanımlıdır (Ders 5) → her zaman tersinir, A ne kadar kötü olursa olsun. δ büyüdükçe problem daha iyi koşullu olur ama veriden uzaklaşır. Bu, istatistikte **ridge regresyon**, mühendislikte **Tikhonov düzenlileştirmesi**dir.

Kötü-koşullu bir matriste ridge çözümünün δ'ya göre nasıl yol aldığını Şekil 17.5 gösteriyor: δ küçüldükçe çözüm pseudoinverse'e yakınsar, δ büyükken kararlı kalır.

> **Builder Notu — Ağırlık Zayıflatma**
>
> $\delta^2\|x\|^2$ cezası = ℓ^2 ağırlık zayıflatma (weight decay): derin öğrenmede aşırı-uyumu önleyen en yaygın düzenlileştirici. Köşegene δ^2 eklemek hem sayısal kararlılık (tersinirlik) hem genelleme (küçük ağırlık) verir — tek hamlede iki kazanç.



Şekil 17.5: Ridge yolu: δ küçüldükçe çözüm pseudoinverse'e yakınsar; δ büyükken kararlı (kötü-koşullu A).

17.9 8. δ → 0 Limiti = Pseudoinverse

δ'yı sıfıra götürünce ne olur? 1×1 örnek ($A = \sigma$) her şeyi gösterir. Çözüm:

$$x = \frac{\sigma}{\sigma^2 + \delta^2} b \xrightarrow[\delta \to 0]{} \begin{cases} 1/\sigma & \sigma > 0 \\ 0 & \sigma = 0 \end{cases}$$

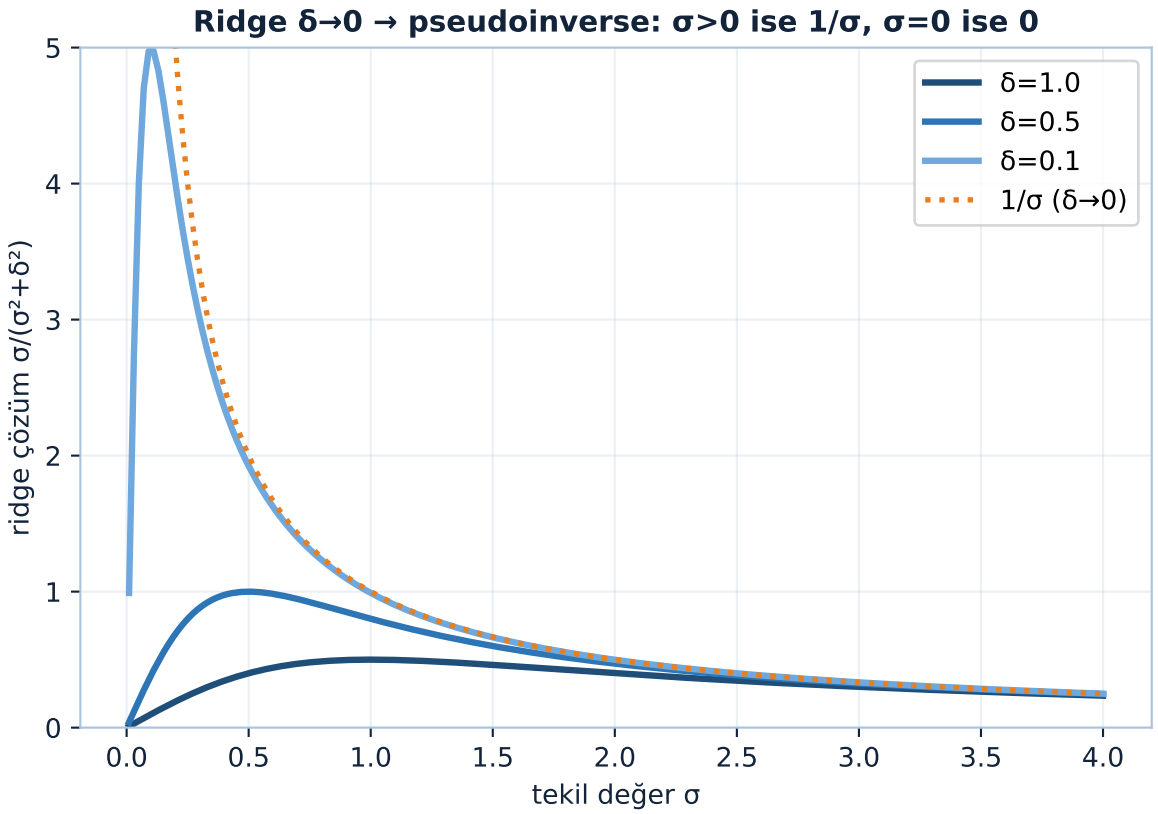
$\sigma > 0$ ise $1/\sigma$'ya (gerçek ters), $\sigma = 0$ ise 0'a gider — tam olarak pseudoinverse'in yaptığı! Genel matriste de aynı:

$$(A^T A + \delta^2 I)^{-1} A^T \xrightarrow[\delta \to 0]{} A^+$$

> *"…approaches the pseudo inverse, sigma plus."* — Strang, 46:41

Bu süreksiz, hassas bir limittir: σ büyüdükçe $1/\sigma$ patlar ama $\sigma = 0$'da birden 0'a düşer. İstatistikçiler pseudoinverse'i bu yoldan (ceza ekleyerek) bağımsız keşfetti.

Her tekil değer σ için ridge çözümünün $\delta \to 0$ limitinde nasıl $1/\sigma$ pseudoinverse eğrisine yakınsadığını Şekil 17.6 gösteriyor: $\sigma > 0$ yönlerinde $1/\sigma$'ya gider, $\sigma = 0$ yönünde 0 kalır.



Şekil 17.6: Ridge δ→0 limiti: her tekil değer σ için ridge çözümü $\sigma/(\sigma^2+\delta^2)$, δ küçüldükçe σ>0 olan yönlerde 1/σ pseudoinverse eğrisine yakınsar; σ=0 yönünde 0 kalır.

> Builder Notu — Ridge'in Sırrı
>
> "Ridge → pseudoinverse" köprüsü, düzenlileştirmenin neden işe yaradığını açıklar: δ küçük tekil değerlerin patlamasını engeller, yalnızca güvenilir (büyük σ) yönleri kullanır. Truncated SVD ve erken durdurma (early stopping) aynı sezginin akrabalarıdır.

17.10 9. Durum 6-7: İteratif ve Rastlantısal

Durum 6 (çok büyük): Matris büyükse doğrudan çözüm ($A^T A$, QR) pahalıdır. **İteratif yöntemler** çözüme adım adım yaklaşır — kahramanı conjugate gradient.

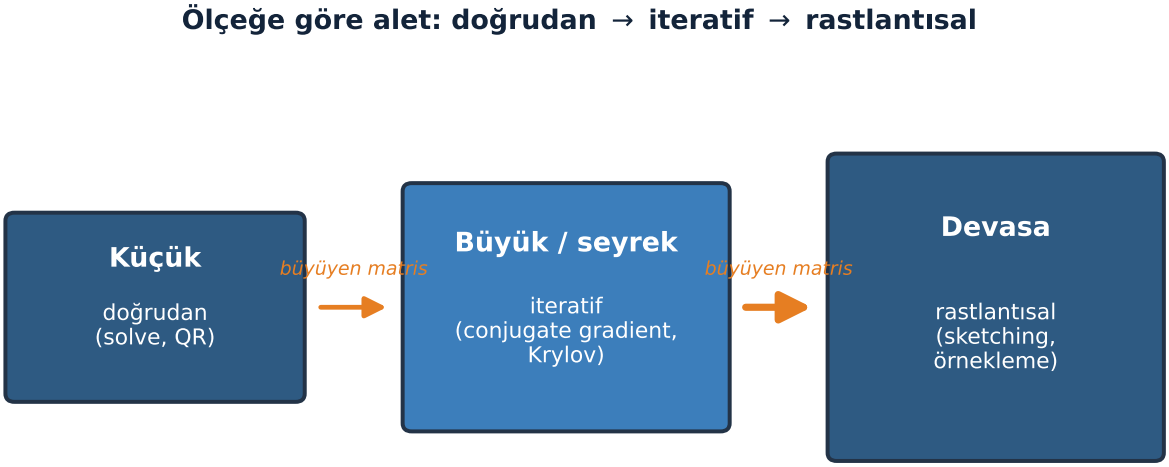
> *"…like conjugate gradients, you get pretty close, pretty fast."* — Strang, 18:05 *"…that name I erased is Krylov…"* — Strang, 17:37

Bu yöntemler (Krylov ailesi) matrisi açıkça kurmaz, yalnızca A ile çarpım kullanır. **Durum 7 (devasa):** Matris belleğe bile sığmıyorsa **rastlantısal lineer cebir** devreye girer — matrisi olasılıkla örnekle, örnekten cevap çıkar.

> *"So randomized linear algebra has popped up…"* — Strang, 18:46

Rastgele x'lerle Ax almak kolon uzayında rastgele vektörler verir; az sayıda örnek çoğu zaman kolon uzayını yeterince yakalar (Ders 13).

Ölçek büyüdükçe aletin nasıl değiştiğini Şekil 17.7 gösteriyor: küçük → doğrudan, büyük/seyrek → iteratif, devasa → rastlantısal.



Şekil 17.7: Ölçeğe göre alet seçimi. Küçük sistemler doğrudan yöntemlerle çözülür (solve, QR). Büyük ya da seyrek matrislerde iteratif yöntemler (conjugate gradient, Krylov) tercih edilir. Devasa boyutlarda ise rastlantısal yöntemler (sketching, örnekleme) devreye girer. Matris büyüdükçe (turuncu oklar) alet de değişir.

> Builder Notu — Ölçeğe Göre Alet
>
> Ölçek arttıkça alet değişir: küçük → doğrudan, büyük/seyrek → iteratif (conjugate gradient, Krylov), devasa → rastlantısal (sketching). Büyük-ölçek ML ve bilimsel hesap bu üçünün üstünde durur; gradient descent (Ders 21-23) iteratif ailenin optimizasyon akrabasıdır.

17.11 10. Derin Öğrenmede Implicit Bias

Eksik-belirlenmiş derin öğrenmede sonsuz çözüm vardır — hepsi eğitim verisine tam uyar. Soru: hangisi **test** verisinde de iyi çalışır (genelleşir)?

> *"…It's called implicit bias. How does that algorithm bias a solution toward a solution that generalizes…"* — Strang, 23:02

Kritik gözlem: bir algoritma (örneğin gradient descent) çözümleri sessizce belli bir yöne yatırır — buna **implicit bias** denir. İyi algoritmalar genelleşen çözümlere; kötüleri test verisinde çuvallayan çözümlere gider. Bu, 2018'in açık matematik sorusudur ve kursun varış noktasıdır.

> Builder Notu — Neden Genelleşir
>
> Implicit bias, modern derin öğrenme teorisinin merkezidir: açık düzenlileştirme olmadan bile GD küçük-norm/basit çözümlere yönelir (Srebro varsayımı, Ders 8). Fazla-parametreli ağların neden ezberlemeyip genelleştiğinin cevabı buradadır — algoritmanın seçtiği çözüm, çözüm uzayından daha önemlidir.

17.12 Bu Dersin Özeti

1. **Ax = b sözlüğü** — boyut/rank/koşula göre durum haritası, her birine reçete.
2. **Durum 0-1** — pseudoinverse (her zaman); iyi koşullu kare → eliminasyon (backslash).
3. **Kondisyon sayısı** $\kappa = \sigma_1/\sigma_n$ — zorluk göstergesi; büyükse tehlike.
4. **Durum 2** — fazla denklem → least squares ($A^TA\hat{x} = A^Tb$).
5. **Durum 3** — eksik denklem → minimum norm / ℓ^1; derin öğrenme.
6. **Durum 4** — kötü kolonlar → Gram-Schmidt/QR + pivotlama.
7. **Durum 5** — neredeyse tekil → düzenlileştirme ($\delta^2 I$); ters problemler.
8. **Ceza terimi** — $(A^TA + \delta^2 I)\hat{x} = A^Tb$; ridge/Tikhonov.
9. $\delta \to 0$ **= pseudoinverse** — ceza, $\sigma > 0$'da $1/\sigma$'ya, $\sigma = 0$'da 0'a gider.
10. **Durum 6-7** — çok büyük → iteratif (Krylov/CG); devasa → rastlantısal LA.
11. **Implicit bias** — hangi çözümün genelleştiği; GD'nin gizli tercihi.

> Tek Bir Cümle
>
> $Ax = b$'nin zorluğu boyut, rank ve kondisyon sayısı $\kappa = \sigma_1/\sigma_n$ ile belirlenir; her durumun kendi aleti vardır (eliminasyon, least squares, minimum norm, QR, iteratif, rastlantısal) ve $\delta^2 I$ düzenlileştirmesi neredeyse tekil problemleri pseudoinverse'e doğru yumuşatır.

17.13 Kontrol Soruları

i Soru 1: Aşağıdaki üç sistem hangi durumdadır? Hangi yöntemi kullanırsın?

$$A_1 = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$

Cevap: A_1: kare, tersinir (det = 3), iyi koşullu → doğrudan çözüm (backslash/eliminasyon). A_2: 3×1, fazla denklem (overdetermined) → least squares ($A^T A\hat{x} = A^T b$), tek bilinmeyen. A_3: 1×3, eksik denklem (underdetermined) → sonsuz çözüm, minimum-norm çözümü seç (pseudoinverse).

i Soru 2: A = diag(100, 1) matrisinin kondisyon sayısını bul. Neden bu matris terslemede risklidir?

Cevap: Tekil değerler 100 ve 1 → $\kappa = \sigma_1/\sigma_n = 100/1 = 100$. Ters $A^{-1} = \text{diag}(0.01, 1)$; küçük yön ($\sigma = 1$) terste 1'e, büyük yön 0.01'e gider. $\kappa = 100$ orta düzeyde kötü; girdideki küçük hata, çözümde 100 kata kadar büyüyebilir. $\kappa \to \infty$ ($\sigma_n \to 0$) olunca matris terslenemez hâle gelir.

i Soru 3: Ridge çözümü $(A^TA + \delta^2 I)^{-1}A^Tb$, $\delta \to 0$'da $\sigma = 0$ ve $\sigma > 0$ için neye gider?

Cevap: 1×1'de çözüm $\sigma/(\sigma^2 + \delta^2)$. $\sigma > 0$ ise $\delta \to 0$'da $1/\sigma$'ya (gerçek ters) gider. $\sigma = 0$ ise her δ için 0, limitte de 0. Bu süreksiz davranış tam olarak pseudoinverse'tir: sıfır olmayan tekil değerleri tersler, sıfır olanları sıfır bırakır. Ceza terimi, $\sigma = 0$ olsa bile son ana kadar problemi çözülebilir tutar.

i Soru 4: Fazla-parametreli (underdetermined) bir ağ sonsuz çözüme sahipken neden genelleşebilir?

Cevap: Çözüm uzayı sonsuz olsa da, eğitim algoritması (gradient descent) hepsini eşit seçmez — **implicit bias** ile belli çözümlere (genelde minimum-norm/basit) yönelir. Bu çözümler test verisinde de iyi çalışır. Yani genelleme, çözüm uzayının kendisinden değil, algoritmanın o uzaydan **hangisini seçtiğinden** gelir. Bu, modern derin öğrenme teorisinin merkezî sorusudur (Srebro varsayımı ile bağlantılı).

17.14 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. Aşağıdaki sistemleri durum haritasında sınıfla (tersinir / least squares / minimum-norm) ve uygun yöntemi söyle: (a) 4×4 tersinir A; (b) 1000×3 A; (c) 3×1000 A; (d) σ değerleri 1, 0.001 olan 2×2 A.

Egzersiz 2. $A = \begin{pmatrix} 3 & 0 \\ 0 & 0.01 \end{pmatrix}$ için kondisyon sayısını hesapla. A^{-1}'i yaz; küçük yöndeki büyütmeyi yorumla.

Egzersiz 3. $A = (2)$ (1×1) ve $b = 6$ için ridge çözümü $x = \sigma/(\sigma^2 + \delta^2) \cdot b$'yi $\delta = 1, 0.1, 0.01$ için hesapla; $\delta \to 0$'da $1/\sigma \cdot b = 3$'e yaklaştığını göster.

Egzersiz 4. Python ile düzenlileştirmeyi keşfet:

```python
import numpy as np

A = np.array([[3.0, 0.0], [0.0, 0.001]])
print("kondisyon:", np.linalg.cond(A))     # σ1/σn = 3000

b = np.array([3.0, 0.001])
for delta in [1.0, 0.1, 0.01, 0.0]:
    if delta > 0:
        x = np.linalg.solve(A.T @ A + delta**2 * np.eye(2), A.T @ b)
    else:
        x = np.linalg.pinv(A) @ b           # δ → 0 limiti
    print(f"δ={delta}: x={x}")
```

Egzersiz 5. (Ders 11 habercisi.) Ders 11, eksik-belirlenmiş sistemde ($m < n$) $\|x\|$'i minimize eden çözümü işler. $A = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$, $b = 3$ için $Ax = b$'yi sağlayan **minimum ℓ^2 norm** çözümünü bul (pseudoinverse veya Lagrange ile). Cevabın $(1, 1, 1)$ olmasını bekle — neden?

17.15 Sonraki Ders İçin Hazırlık

Ders 11: $Ax = b$ Koşuluyla $\|x\|$'i Minimize Etmek

Ders 10'da durum haritasını çıkardık; eksik-belirlenmiş durumda "hangi çözüm?" sorusunu açtık. Ders 11 bunu derinleştirir: kısıt altında minimum-norm çözümü.

- Minimum ℓ^2 norm çözümü (pseudoinverse'in verdiği)
- Minimum ℓ^1 norm (seyrek çözüm)
- Lagrange çarpanları ile kısıtlı minimizasyon
- Derin öğrenmede çözüm seçimiyle bağ

> ⚠ Ders 11 Öncesi Yapılacak
>
> - Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (minimum-norm).
> - Python'da `np.linalg.pinv` ile eksik-belirlenmiş sistemlerin minimum-norm çözümünü incele.
> - Ana cümleyi tekrar oku: "$Ax = b$'nin zorluğu $\kappa = \sigma_1/\sigma_n$; düzenlileştirme pseudoinverse'e yumuşatır."

17.16 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
Ax = b sözlüğü	Durum haritası; her duruma ayrı reçete	0m25
Kondisyon sayısı	$\kappa = \sigma_1/\sigma_n$; zorluk göstergesi	2m27

Kavram	Tanım	Strang'de
Durum 3: eksik	$m < n$; minimum norm / ℓ^1; derin öğrenme	4m34
Gram-Schmidt / QR	Kötü kolonlar → ortonormal taban ($A = QR$)	7m06
Ceza / düzenlileştirme	$\delta^2\|x\|^2$ ekle; $(A^TA + \delta^2 I)\hat{x} = A^Tb$	14m49
$\delta \to 0$ **= pseudoinverse**	Ridge/Tikhonov limiti; $\sigma > 0 \to 1/\sigma$, $\sigma = 0 \to 0$	46m41
İteratif (conjugate gradient)	Çok büyük; Krylov ailesi, yalnız Ax çarpımı	18m05
Krylov	İteratif yöntemlerin adı (detay Ders 12)	17m37
Rastlantısal LA	Devasa matris; örnekleme (Ders 13)	18m46
Implicit bias	Algoritmanın hangi çözümü seçtiği; genelleme	23m02

17.17 ML Bağlantıları Özeti

1. **Kondisyon sayısı** κ → eğitim zorluğu; normalizasyon/ön-koşullama κ'yı küçültür.
2. **Eksik-belirlenmiş = derin öğrenme** → fazla-parametre, sonsuz çözüm; seçim önemli.
3. **Minimum norm /** ℓ^1 → düzenlileştirme; ℓ^1 seyrek, ℓ^2 düzgün.
4. **Gram-Schmidt/QR** → kötü-koşullu tasarım matrislerini düzeltme; kararlı least squares.
5. **Ridge (**$\delta^2 I$**) → pseudoinverse** → weight decay, Tikhonov; sayısal kararlılık + genelleme.
6. **İteratif/rastlantısal** → büyük-ölçek ML; conjugate gradient, randomized SVD, sketching.
7. **Implicit bias** → fazla-parametreli ağların neden genelleştiği; GD'nin gizli düzenlileştirmesi.

> **!** Eğer bu dersten tek bir şey alıp gidersen
>
> $Ax = b$ tek bir problem değil, bir problemler ailesidir — boyut, rank ve kondisyon sayısı $\kappa = \sigma_1/\sigma_n$ hangisi olduğunu söyler. İyi koşulluda eliminasyon, fazla denklemde least squares, eksikte minimum norm, kötü kolonda QR, neredeyse tekilde düzenlileştirme ($\delta^2 I \to$ pseudoinverse), devasada iteratif/rastlantısal. Doğru tanı, doğru aleti seçtirir — ve eksik-belirlenmiş derin öğrenmede asıl soru, algoritmanın sonsuz çözümden hangisini (implicit bias) seçtiğidir.

ETAP 2 kapanışı (Ders 2-10): Bu on ders, lineer cebirin "highlights" turunu (faktorizasyonlar, ortogonallik, özdeğer/SVD, normlar) tamamlayıp veri bilimine (Eckart-Young, PCA, least squares, düzenlileştirme) ve derin öğrenmenin kapısına (gradient descent, implicit bias) bağladı. Sırada optimizasyon ve sinir ağları var.

18 Ax = b'nin Koşuluyla ‖x‖'i Minimize Etmek

Gram-Schmidt A = QR ve Krylov uzayı

> Bölüm bilgisi
>
> **Video:** Minimizing ‖x‖ Subject to Ax = b · **OCW:** MIT 18.065 Lecture 11 · **Okuma süresi:** ~38 dk · **Eğitmen:** Gilbert Strang · **Önkoşul:** Ders 10 (Ax = b durum haritası, kondisyon sayısı, kötü kolonlar).

18.1 Bu Derste Ne Var?

Bu ders üç parçadan oluşur. Başlık kısa açılışı ($\|x\|$ minimizasyonu) yansıtır ama dersin gövdesi **Gram-Schmidt** ve **Krylov**'dur. (Not: docx ders11 = OCW Lec 11; OCW başlığı açılış recap'ini vurgular, video asıl olarak Gram-Schmidt/QR + iteratif yöntemleri işler.)

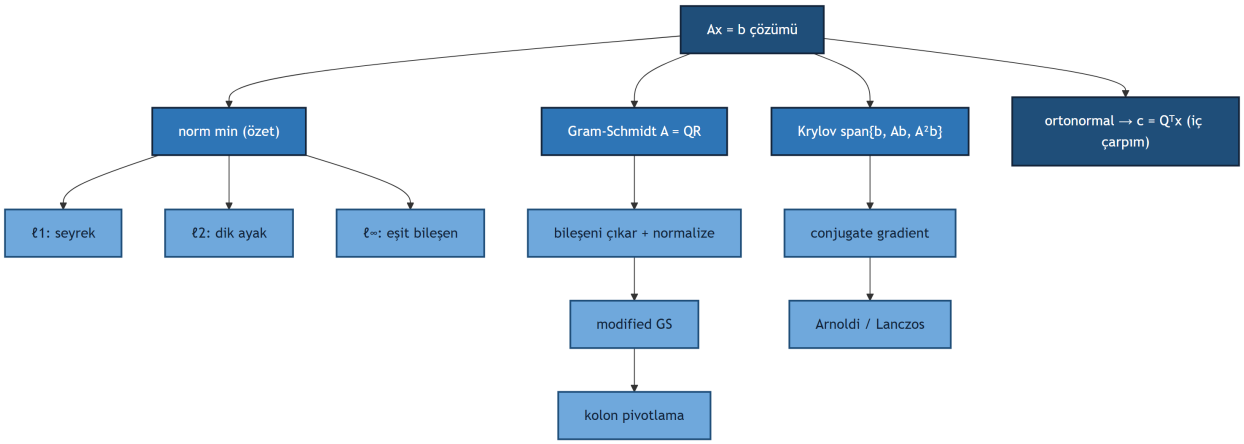
Üç temel fikir:

1. $\|x\|$ **minimizasyonu (recap)** — $Ax = b$ kısıtı altında en küçük normlu x: ℓ^1 (elmas) seyrek, ℓ^2 (çember) dik ayak, ℓ^∞ (kare) eşit bileşen.
2. **Gram-Schmidt:** $A = QR$ — bağımsız (belki kötü koşullu) kolonları ortonormal Q'ya çevir; $R = Q^T A$; kolon pivotlama sayısal kararlılık için.
3. **Krylov + Arnoldi/Lanczos** — büyük seyrek A için iteratif çözüm; Krylov uzayı $\{b, Ab, A^2b, ...\}$, conjugate gradient, ve bazı ortonormalleştirme.

> *"Now, I'm coming to the topic of the day, which is Gram-Schmidt."* — Strang, 9:20

Dersin üç dalını ve aralarındaki bağı Şekil 18.1 özetliyor: merkezdeki $Ax = b$ çözümünden norm minimizasyonu, Gram-Schmidt $A = QR$ ve Krylov uzayına uzanan üç kol, hepsi ortonormal kolonların sağladığı iç çarpım kolaylığında ($c = Q^T x$) buluşur.

> Builder Notu — Sayısal Cebrin Omurgası
>
> - **Gram-Schmidt / QR** — kötü koşullu tasarım matrislerini ortonormal tabana çevirir; least squares'in kararlı çözümü, özdeğer algoritmasının (Ders 12) temeli.
> - **Kolon pivotlama** — rank-açığa-çıkaran QR; en bilgilendirici kolonu seçer (öznitelik seçimi, sayısal kararlılık).
> - **Krylov + conjugate gradient** — büyük seyrek sistemlerin belkemiği; yalnız Ax çarpımı kullanır, matrisi açık kurmaz (büyük-ölçek ML/bilimsel hesap).
> - **Ortonormal baz →** $c = Q^T x$ — katsayılar tek bir transpozla; projeksiyon ve çözümün ucuz



Şekil 18.1: Ders 11 kavram haritası: Ax = b çözümü merkezinden üç ana dal — norm minimizasyonu özeti (ℓ1 seyrek / ℓ2 dik ayak / ℓ∞ eşit bileşen), Gram-Schmidt A = QR (bileşeni çıkar + normalize → modified GS → kolon pivotlama) ve Krylov uzayı span{b, Ab, A^2b} (conjugate gradient → Arnoldi/Lanczos); ayrıca ortonormal kolonlardan iç çarpımla katsayılar c = Q^Tx düğümü.

olduğu yer.

Tek cümle: kötü koşullu kolonları Gram-Schmidt ile ortonormalleştirmek ($A = QR$) sayısal lineer cebirin omurgasıdır; büyük sistemlerde aynı ortonormalleştirme Krylov/Arnoldi/Lanczos olarak iteratif çözümü mümkün kılar.

18.2 1. Önce Hatırlatma: ‖x‖ Minimizasyonu

Strang dersi, Ders 8'deki "kısıt altında norm minimizasyonu" resmini sayılarla somutlaştırarak açıyor. Kısıt tek bir doğru: $3x_1 + 4x_2 = 1$. Bu doğru üzerinde hangi nokta en küçük normludur — norma göre değişir.

> *"Do you remember the day that we minimized different norms?"* — Strang, 2:27

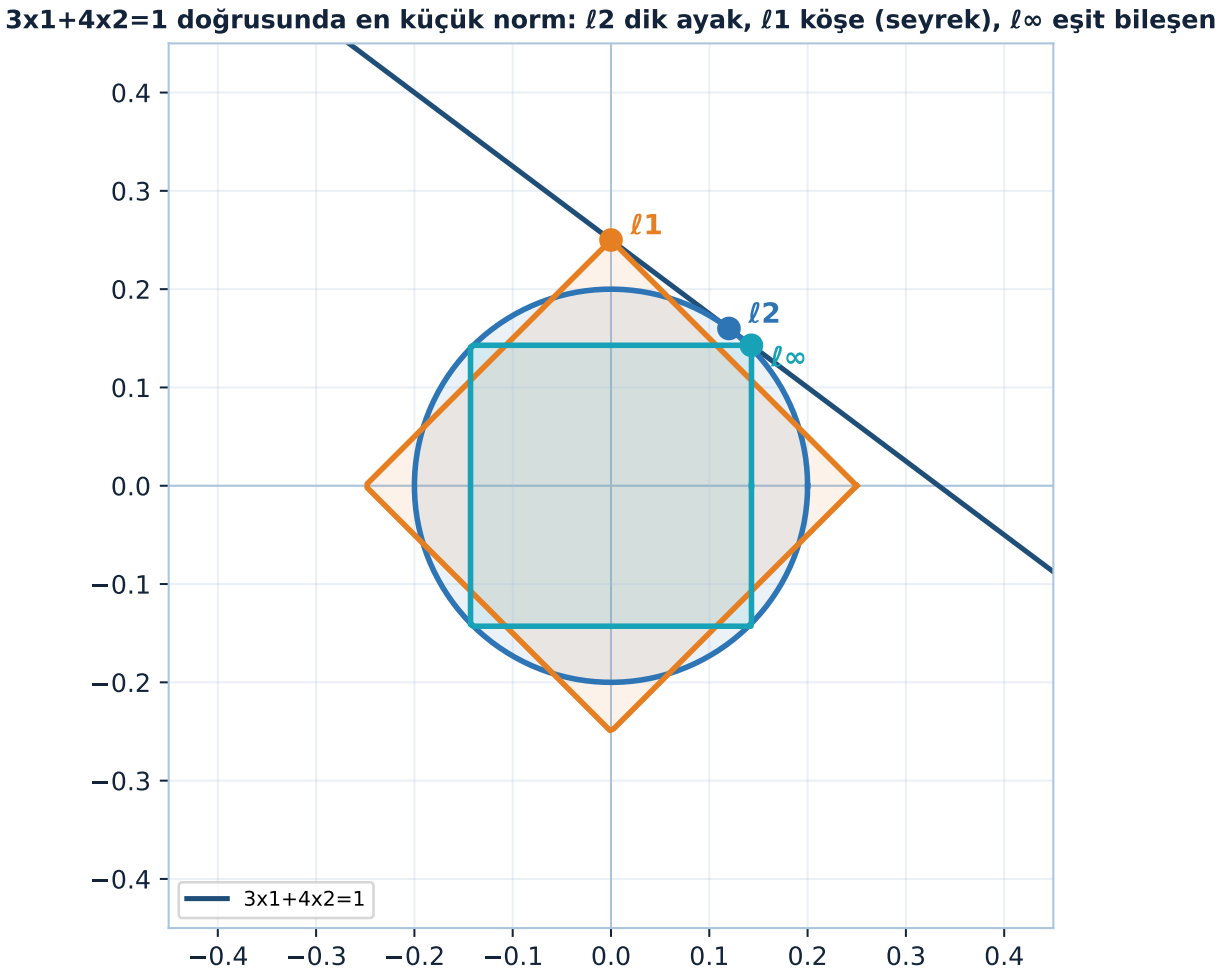
$$3x_1 + 4x_2 = 1 \quad \text{(constraint)}$$

- ℓ^2**:** çember doğruya **dik ayakta** değer → kazanan $(3/25, 4/25)$.
- ℓ^1**:** elmas → kazanan $(0, 1/4)$.
- ℓ^∞**:** kare → kazanan $(1/7, 1/7)$.

Aynı doğru üzerinde üç norm topunun farklı kazananlara değdiğini Şekil 18.2 gösteriyor: ℓ^2 çemberi dik ayakta, ℓ^1 elması köşesiyle eksen üstünde (seyrek), ℓ^∞ karesi eşit bileşende değer.

Builder Notu — Norm Seçimi = Çözüm Seçimi

Bu üç kazanan, aynı kısıtta üç farklı düzenlileştirmenin (ridge, Lasso, max-norm) seçeceği üç farklı çözümdür. Norm seçimi = çözüm seçimi: aynı veri, farklı geometri, farklı sonuç.



Şekil 18.2: $3x_1 + 4x_2 = 1$ doğrusu üzerinde aynı kısıtı sağlayan sonsuz çözümden hangisini seçeceğimizi *norm* belirler. Üç norm topunu, doğruya değecek şekilde büyütürüz: ilk değdiği nokta o normun kazananıdır. ℓ_2 çemberi doğruya teğet olduğu yerde değer — bu **dik ayak** $(0.12,\ 0.16)$, iki bileşen de sıfırdan farklı. ℓ_1 elması ise sivri **köşesiyle** değer — köşe eksenin üstünde olduğu için $x_1 = 0$ çıkar, yani **seyrek** çözüm (Lasso'nun nedeni). ℓ_∞ karesi köşesiyle değer ama köşeleri köşegende olduğundan iki bileşen **eşit** olur $(0.143,\ 0.143)$. Aynı doğru, üç farklı kazanan.

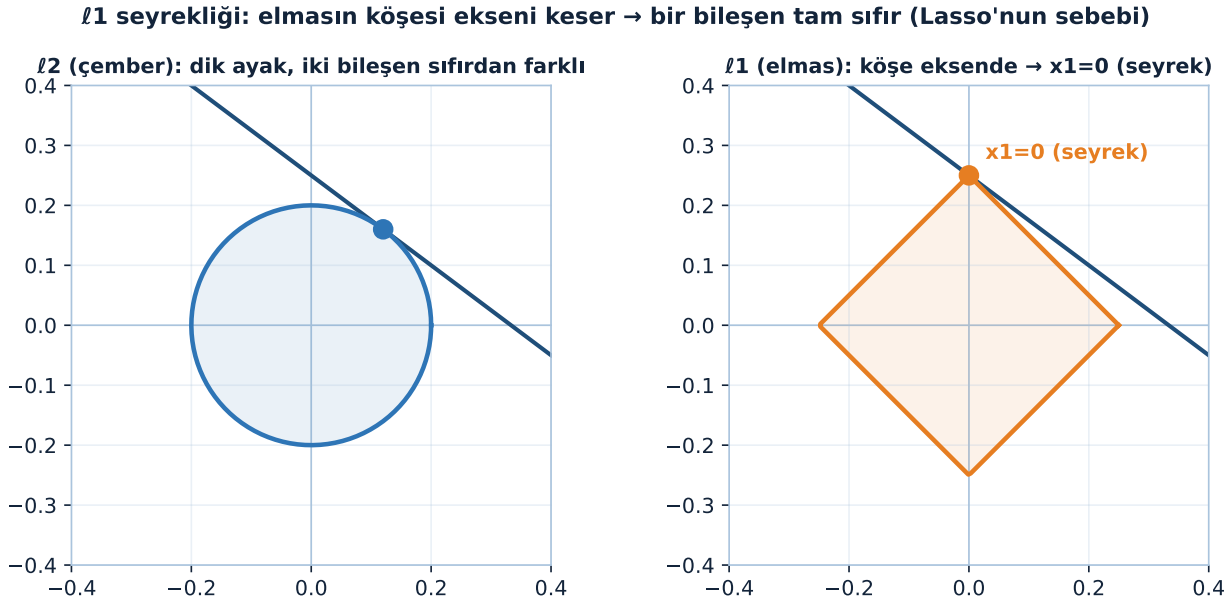
18.3 2. ℓ^1 Seyrekliği

ℓ^1 kazananı neden özel? Çünkü elmasın köşesi eksende oturur — bir bileşeni tam sıfır olur.

> *"A diamond, right. A diamond. So the diamond that first touches the line is here."* — Strang, 5:02

$3x_1 + 4x_2 = 1$ doğrusunda ℓ^1 kazananı $(0, 1/4)$: $x_1 = 0$ (seyrek!). p arttıkça $(1 \to 2 \to \infty)$ kazanan köşeden (seyrek) köşeye doğru kayar; çember ve karede seyreklik kaybolur. Yüksek boyutta bu, ℓ^1'in neden seyrek çözüm verdiğinin kaynağıdır.

İki normun aynı doğrudaki kazananını yan yana koyan Şekil 18.3, elmasın köşeli geometrisinin neden sıfır bileşen ürettiğini gösteriyor: ℓ^2 çemberi iki bileşeni de sıfırdan farklı bırakır, ℓ^1 elması ise tam $x_1 = 0$ verir.



Şekil 18.3: $\ell 1$ seyrekliği: aynı $3x_1 + 4x_2 = 1$ doğrusunda iki normun en küçük çözümü. Solda $\ell 2$ çemberi doğruya dik ayağında (0.12, 0.16) değer — iki bileşen de sıfırdan farklı (yoğun). Sağda $\ell 1$ elması doğruya bir köşesinde, tam y ekseninde (0, 0.25) değer — $x_1 = 0$ olur (seyrek). Elmasın köşeli geometrisi seyrekliği üretir; Lasso'nun çalışma sebebi budur.

> **Builder Notu — Sıfırın Geometrisi**
>
> "Elmas köşesi = sıfır bileşen" geometrisi Lasso'nun, compressed sensing'in ve seyrek modellerin matematiksel sebebidir. Kaç sıfır geleceği boyuta ve kısıt sayısına bağlıdır — Strang'in "ilginç proje" dediği açık soru.

18.4 3. Gram-Schmidt: A = QR

Dersin asıl konusu: kötü koşullu (neredeyse bağımlı) kolonları ortonormal bir tabana çevirmek.

"Now, I'm coming to the topic of the day, which is Gram-Schmidt." — Strang, 9:20

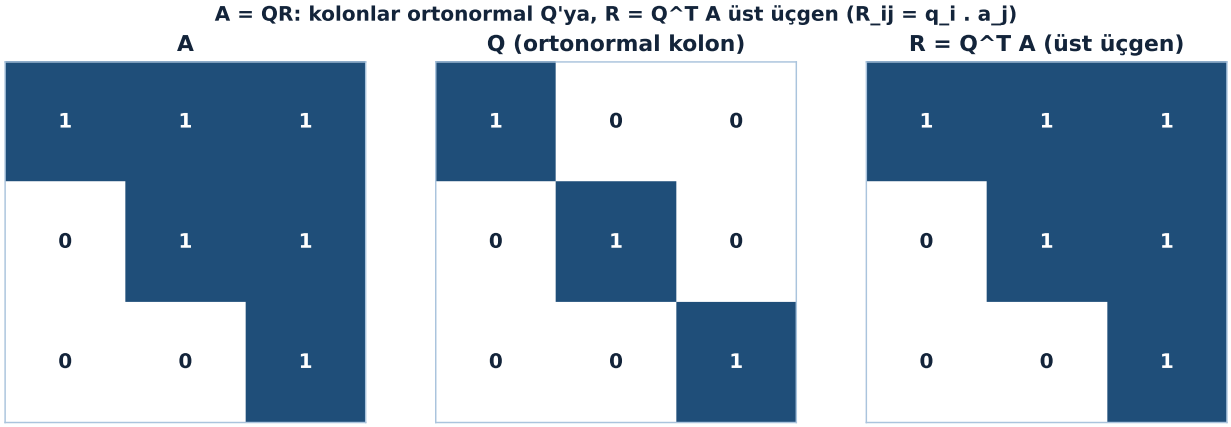
Bağımsız ama belki barely-bağımsız kolonlu A'dan, ortonormal kolonlu Q üretilir; ikisi üst üçgensel R ile bağlıdır:

$$A = QR$$

R, Q'nun kolonlarından A'nın kolonlarına geçiş reçetesidir. LU'da ileri gidip U'ya varırız; burada ileri gidip Q'ya varırız, sonra A ile Q'yu bağlayan R'yi yazarız.

"…the Matlab command is exactly QR." — Strang, 12:15

$A = QR$ ayrışımının üç matrisini Şekil 18.4 ısı haritasıyla yan yana koyuyor: soldaki A Gram-Schmidt'le ortada ortonormal kolonlu Q'ya geçer, sağda $R = Q^T A$'nın köşegen-altı tam sıfır (üst üçgen).



Şekil 18.4: A = QR ayrışımının üçlü ısı haritası. Soldaki A matrisinin kolonları Gram-Schmidt ile ortonormalleştirilerek ortadaki Q (her kolonu birim boyda ve birbirine dik) elde edilir; sağdaki R = Q^TA üst üçgendir ve köşegen-altı girdileri tam sıfırdır (R_ij = q_i^T a_j, i > j için sıfır). Üst üçgen yapı, j. kolonun yalnızca kendinden önceki q vektörlerine bileşeni olduğunu gösterir.

> Builder Notu — Günün Konusu
>
> $A = QR$, least squares'in kararlı çözüm yoludur (Ders 9-10) ve özdeğer algoritmasının (Ders 12 QR iterasyonu) çekirdeğidir. `numpy.linalg.qr` bunu verir; LU yerine QR, ortonormallik sayesinde kondisyonu karelemez.

18.5 4. $R = Q^T A$ (İç Çarpımlar)

R gizemli görünür ama basittir. $A = QR$'de Q ortogonal olduğundan $Q^{-1} = Q^T$; iki tarafı soldan Q^T ile çarp:

$$R = Q^T A$$

> *"...this mysterious R is Q transpose A."* — Strang, 13:13

R'nin (i, j) girdisi, q_i satırı çarpı a_j kolonu — yani basitçe bir iç çarpım:

$$R_{ij} = q_i^T a_j$$

Demek ki R, Q'ların A'larla nokta çarpımlarından ibarettir. Gizem yok: ortonormallik R'yi bu temiz forma indirir. (Yukarıdaki Şekil 18.4 aynı zamanda bu iç-çarpım yapısını görselleştirir: sağdaki $R = Q^T A$ paneli.)

> **Builder Notu — İç Çarpım Yeter**
>
> $R = Q^T A$ olması, ortonormal bir tabanda koordinat bulmanın iç çarpımdan ibaret olduğunu gösterir (Bölüm 11). Bu, Fourier katsayıları, projeksiyon ve dikkat (attention) skorlarının da temelidir: dik tabanlarda her şey iç çarpımla okunur.

18.6 5. Gram-Schmidt'in Çift Adımı

Tüm Gram-Schmidt iki adımın tekrarıdır. İlk kolon kolaydır — sadece normalize et: $q_1 = a_1/\|a_1\|$. Asıl fikir q_2'de:

> *"The whole idea of Gram-Schmidt is in q2."* — Strang, 16:05

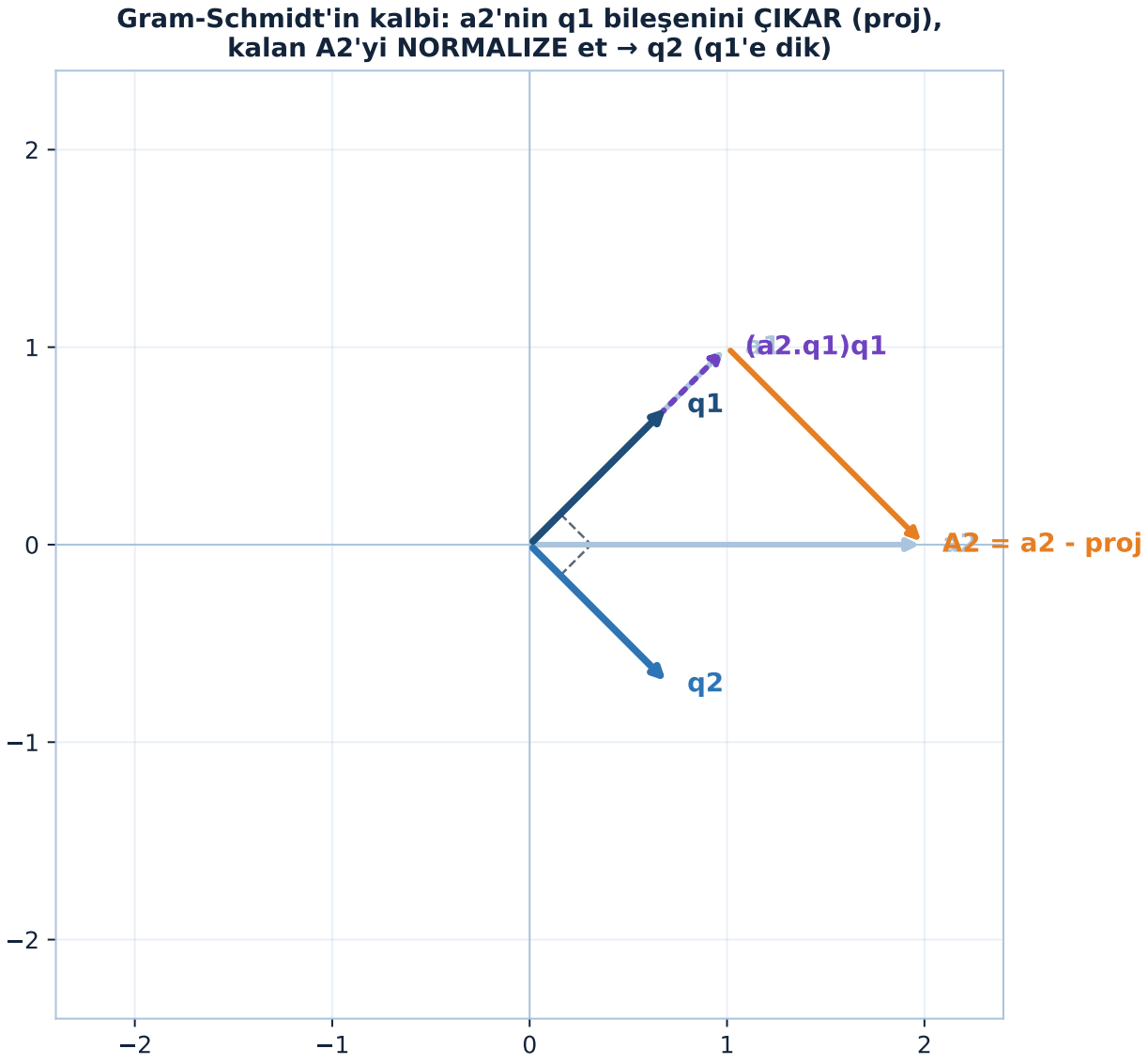
a_2'nin q_1 yönündeki bileşenini **çıkar** (ortogonal yap), sonra **normalize et**:

$$A_2 = a_2 - (a_2^T q_1)\, q_1, \qquad q_2 = \frac{A_2}{\|A_2\|}$$

Çıkarılan parça $(a_2^T q_1)q_1$; geriye kalan A_2, q_1'e diktir (kontrol: $q_1^T A_2 = a_2^T q_1 - (a_2^T q_1)(q_1^T q_1) = 0$). Bu "çıkar + normalize" çifti tüm algoritmanın kalbidir.

Bu iki adımı somut sayılarla ($a_1 = (1, 1)$, $a_2 = (2, 0)$) Şekil 18.5 gösteriyor: a_2'nin q_1 üzerindeki projeksiyonu çıkarılır, dik kalan A_2 normalize edilince q_2 elde edilir ve kökteki kesik kare $q_1 \perp q_2$ dikliğini işaretler.

> **Builder Notu — Çıkar ve Normalize**
>
> "Önceki yönlerin bileşenini çıkar, kalanı normalize et" — bu, dik bir taban kurmanın evrensel reçetesidir. Ortonormal RNN ağırlıkları, dik gömüler ve PCA sonrası dik eksenler aynı işlemi kullanır.



Şekil 18.5: Gram-Schmidt'in iki adımı, $a_1 = (1,1)$ ve $a_2 = (2,0)$ üzerinde. Açık çelik renkli a_1, a_2 başlangıç (dik olmayan) kolonları. İlk kolon doğrudan normalize edilir: $q_1 = a_1/\|a_1\|$ (koyu lacivert). İkinci adımın kalbi: a_2'nin q_1 yönündeki bileşeni $(a_2^T q_1)q_1 = (1,1)$ hesaplanır (mor kesik ok) ve ÇIKARILIR; geriye q_1'e dik kalan $A_2 = a_2 - (a_2^T q_1)q_1 = (1,-1)$ kalır (turuncu, projeksiyon ucundan çizili). Bu kalanı normalize edince $q_2 = A_2/\|A_2\|$ (mavi) elde edilir. Kökteki kesik kare $q_1 \perp q_2$ dikliğini gösterir: çıkar-ve-normalize çevrimi ortonormal tabanı üretir.

18.7 6. Modified Gram-Schmidt

Üçüncü vektörde iki bileşen çıkarılır (q_1 ve q_2):

$$A_3 = a_3 - (a_3^T q_1)\, q_1 - (a_3^T q_2)\, q_2, \qquad q_3 = \frac{A_3}{\|A_3\|}$$

İki çıkarmayı **ayrı ayrı** (önce q_1, sonra q_2) yapmak sayısal olarak daha kararlıdır — buna **modified Gram-Schmidt** denir.

> *"It's called modified Gram-Schmidt."* — Strang, 20:43

Klasik (hepsini bir kerede çıkar) ile modified (sırayla çıkar) matematiksel olarak aynı sonucu verir ama yuvarlama hatasına karşı modified daha dayanıklıdır.

> Builder Notu — Kütüphanenin Tercihi
>
> Modified Gram-Schmidt, kütüphanelerin tercih ettiği sürümdür: aynı işlem sayısı, daha iyi sayısal kararlılık. Householder QR (Ders 3) ise daha da kararlıdır; büyük problemlerde LAPACK Householder kullanır.

18.8 7. Kolon Pivotlama (Daha İyi Gram-Schmidt)

Standart Gram-Schmidt kolonları geldiği sırada alır — risklidir. Eğer a_2 neredeyse a_1 yönündeyse, çıkardıktan sonra geriye minik bir parça kalır; ona bölmek yuvarlama hatasını patlatır (tıpkı eliminasyonda küçük pivot gibi). Çözüm: her adımda kalan kolonların hepsinden q_1 (ve sonraki q'lar) bileşenini çıkar, **en büyüğünü** seç.

> *"…I'm going to take the largest."* — Strang, 30:49

Bu, eliminasyondaki satır değişiminin (en büyük pivot) kolon karşılığıdır — **kolon pivotlama**. Strang'in vurgusu: ekstra iş yok, çünkü bu çıkarmaları zaten yapacaktın; sadece sırayı en büyük-kalan-kolon olarak seçiyorsun.

> Builder Notu — En Büyüğü Seç
>
> Kolon pivotlu QR = rank-açığa-çıkaran faktorizasyon: hangi kolonların gerçekten bağımsız/bilgilendirici olduğunu bulur. Özellik seçimi, kötü-koşullu matris düzeltme ve sayısal kararlılıkta standart (LAPACK geqp3).

18.9 8. Krylov Uzayı

Dersin son parçası: A çok büyük ve seyrekse $Ax = b$'yi nasıl çözeriz? Büyük seyrek matrisle ucuz yapabileceğin tek şey **matris-vektör çarpımı** (Ax). O zaman b'den başlayıp ardışık çarpımlar üret:

$$\mathcal{K}_j = \text{span}\{\, b,\ Ab,\ A^2b,\ \ldots,\ A^{j-1}b \,\}$$

> *"…it's called the Krylov space."* — Strang, 37:35

Bu vektörlerin kombinasyonları **Krylov uzayı**dır (boyut j). Önemli: A^2 hiçbir zaman açıkça kurulmaz — $A \cdot (A \cdot b)$ olarak ardışık çarpımla hesaplanır. Seyrek A için her çarpım ucuzdur.

Krylov vektörlerinin neden "kötü bir taban" olduğunu Şekil 18.6 gösteriyor: ardışık $A^k b$ vektörleri giderek baskın özvektör yönüne döner, birbirine neredeyse bağımlı hale gelir — Arnoldi/Lanczos'un ortogonalleştirme ihtiyacının görsel sebebi.

> **Builder Notu — Yalnız Ax Çarpımı**
>
> Krylov uzayı, büyük-ölçek sayısal cebrin kalbidir: matrisi belleğe açmadan, yalnız Ax çarpımıyla çalışır. Conjugate gradient, GMRES, Lanczos hepsi Krylov tabanlıdır — devasa seyrek sistemler (PDE, ağ Laplacian'ları, öneri sistemleri) için tek pratik yol.

18.10 9. Conjugate Gradient

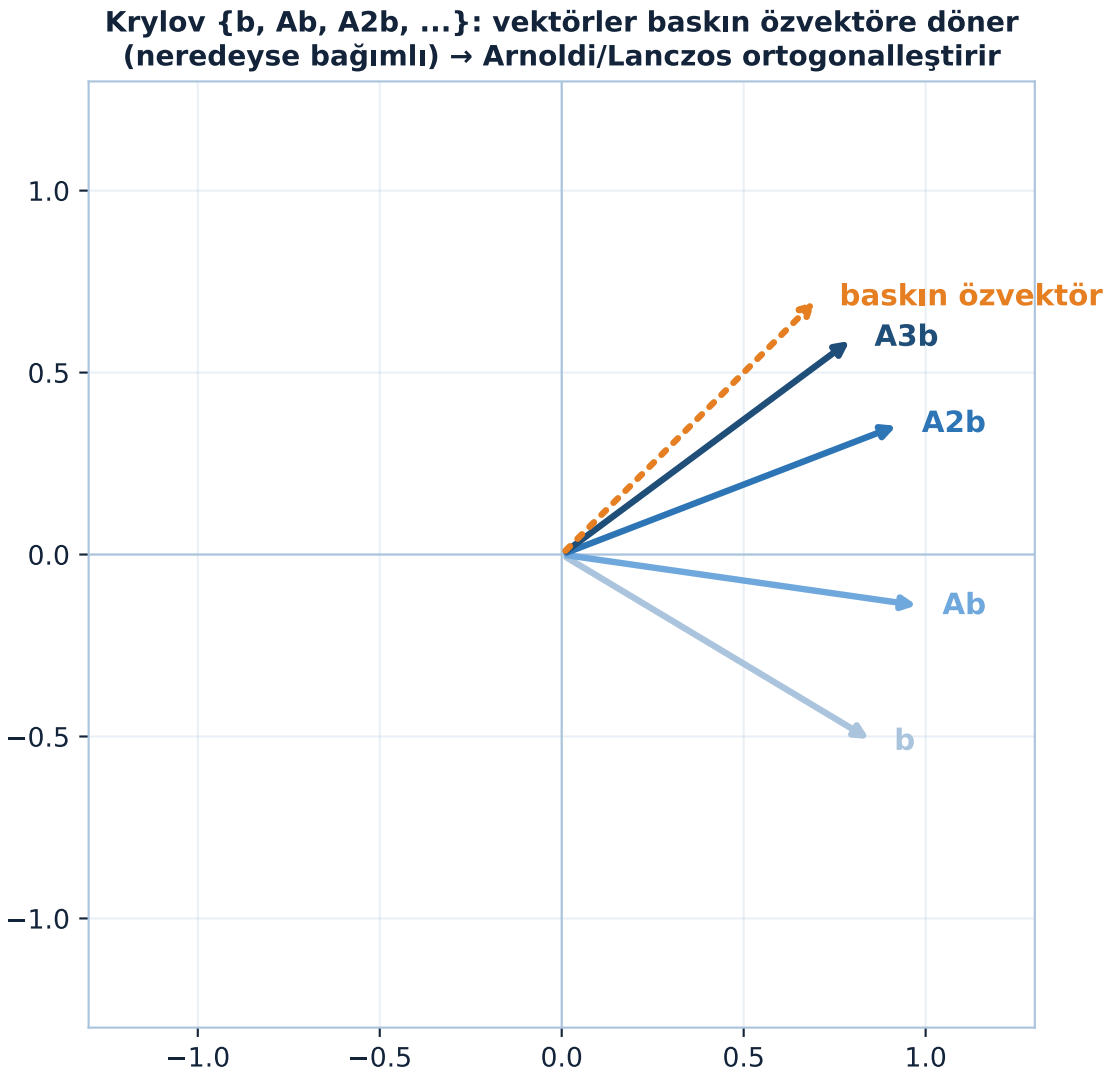
Krylov uzayında $Ax = b$'yi tam çözmeyiz; o uzaydaki **en iyi (en yakın)** çözümü buluruz.

> *"…x_j will be the best vector, or the closest vector [in the Krylov space]."* — Strang, 39:19

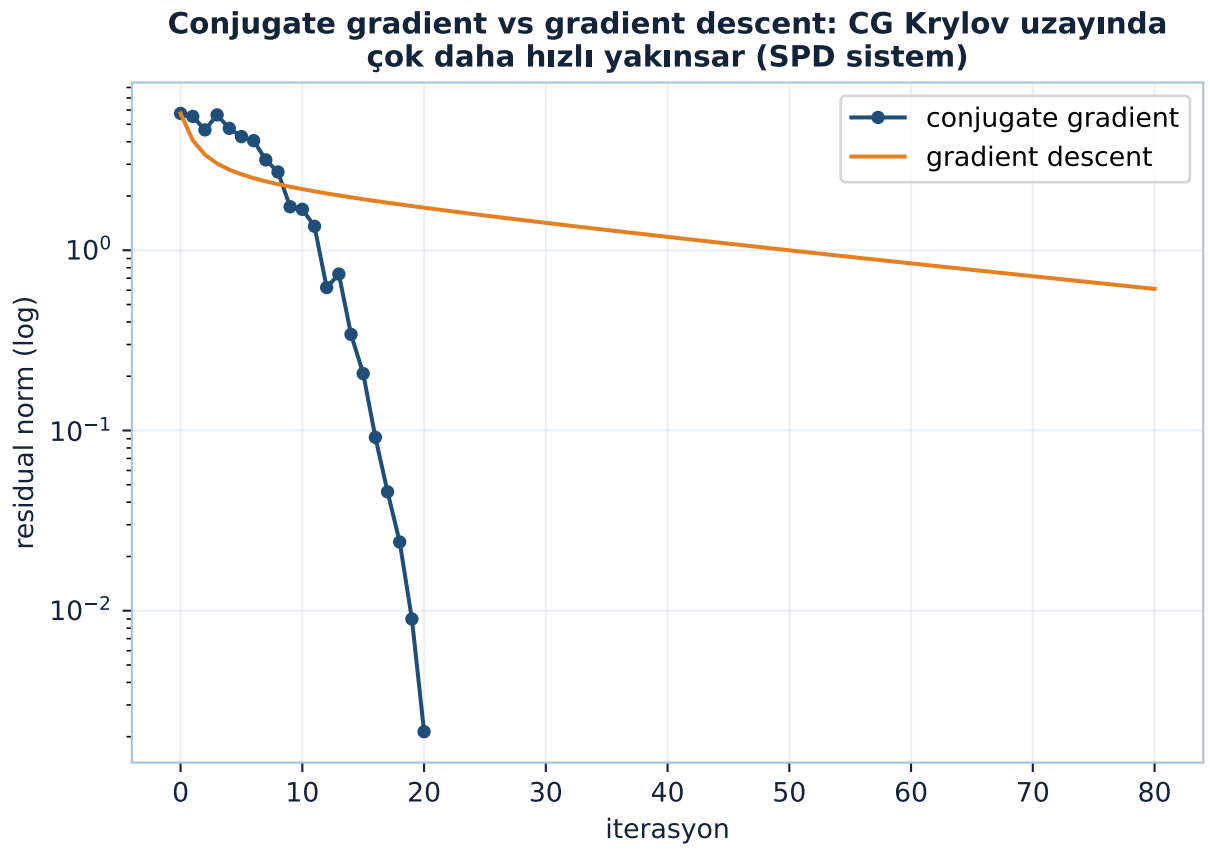
j büyüdükçe Krylov uzayı genişler ve çözüm gerçek cevaba yakınsar. Bu yaklaşımın kahramanı **conjugate gradient**'tir: simetrik pozitif tanımlı sistemlerde Krylov uzayında hızla en iyi çözüme iner. Tam çözüm yerine "yeterince yakın, yeterince hızlı".

Conjugate gradient'in gradient descent'e kıyasla ne kadar hızlı yakınsadığını Şekil 18.7 gösteriyor: aynı SPD sistemde CG, 20 adımda residual normunu ~10^{-3} düzeyine (sonlu-aritmetikte makine sıfırına ~27 adımda) indirirken, sabit-adım gradient descent çok daha yavaş ilerler ve aynı düzeye yaklaşamaz.

> **Builder Notu — Akıllı Akraba**
>
> Conjugate gradient, gradient descent'in (Ders 21-23) akıllı akrabasıdır: SPD sistemlerde gradient descent'ten çok daha hızlı yakınsar (kondisyon sayısının karekökü kadar adım). Büyük lineer sistemleri ve ikinci-derece optimizasyonu çözmenin standart aracı; yalnız Ax gerektirir.



Şekil 18.6: Krylov vektörleri $\{b, Ab, A^2b, A^3b\}$ baskın özvektöre döner. $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ için baskın özvektör $(1,1)/\sqrt{2}$ ($\lambda = 3$); her yeni $A^k b$ bu yöne daha çok yaslanır, dolayısıyla vektörler neredeyse doğrusal bağımlı (kötü bir taban) hale gelir. Arnoldi/Lanczos bu vektörleri adım adım ortogonalleştirerek iyi bir taban kurar.



Şekil 18.7: Conjugate gradient vs gradient descent: aynı SPD sistemde (20×20, $S = BB^T + 0.5I$, kondisyon ≈ 129) artık (residual) normunun logaritmik düşüşü. Conjugate gradient Krylov uzayında çok daha hızlı yakınsar — 20 adımda residual'ı ~10^{-3} düzeyine, ~27 adımda makine sıfırına indirir; gradient descent ise sabit adımla 80 iterasyonda hâlâ bu düzeye ulaşamaz. Aradaki fark kondisyon sayısının karekökünden gelir.

18.11 10. Arnoldi ve Lanczos

Krylov uzayının doğal bazı $\{b, Ab, A^2b, \ldots\}$ büyük bir sorun taşır: bu vektörler genelde neredeyse **bağımlıdır** (hepsi baskın özvektör yönüne döner). Kötü bir tabandır. Çözüm: bu bazı ortogonalleştir.

> *"…that's where Arnoldi comes in. And there's also a Hungarian guy named Lanczos."* — Strang, 41:15

Arnoldi genel matrisler için, **Lanczos** simetrik matrisler için Krylov bazını ortonormalleştirir (Gram-Schmidt'in Krylov uzayına uygulanması). Ortonormal baz elde edilince, o uzaya projeksiyon (en iyi çözümü bulma) kolaylaşır.

> Builder Notu — Üç-Terimli Rekürans
>
> Arnoldi/Lanczos, büyük seyrek matrislerin birkaç baskın özdeğerini/özvektörünü bulmanın standart yoludur (tam SVD yerine) — PCA, spektral kümeleme (NYU DL'deki spektral GCN), ve dev sistemlerin çözümünde kullanılır. Lanczos = simetrik durumda Arnoldi'nin ucuz hâli (üç-terimli rekürans).

18.12 11. Ortonormal Bazın Gücü

Neden ortonormal baz bu kadar değerli? Bir x vektörünü baz vektörleriyle yazmak istersen ($x = Qc$), katsayıları bulmak normalde $Q^{-1}x$ çözmek demektir. Ama Q ortonormalse $Q^{-1} = Q^T$, yani katsayılar bedava:

> *"…Q inverse is Q transpose. That's the payoff."* — Strang, 43:32

$$x = Qc \;\Rightarrow\; c = Q^T x, \qquad c_i = q_i^T x$$

Her katsayı tek bir iç çarpımdır ($c_i = q_i^T x$) — çünkü $q_i^T q_j = 0$ ($i \neq j$) diğer terimleri siler. Denklem çözmek yok; sadece nokta çarpımları. Tüm Gram-Schmidt/Krylov çabası bu kolaylık içindir.

> Builder Notu — Bedava Katsayılar
>
> "$c = Q^T x$" — ortonormal tabanda projeksiyon = iç çarpım. Fourier dönüşümü, dalgacık katsayıları, PCA bileşenleri ve dikkat skorları hep bu ucuzluktan yararlanır. Ortonormallik, hesabı $O(n^2)$ çözmeden $O(n)$ iç çarpıma indirir.

18.13 Bu Dersin Özeti

1. $\|x\|$ **minimizasyonu (recap)** — $Ax = b$ kısıtında ℓ^1 (elmas, seyrek), ℓ^2 (çember, dik ayak), ℓ^∞ (kare).
2. ℓ^1 **seyrekliği** — elmas köşesi eksende → bileşen sıfır.
3. $A = QR$ — Gram-Schmidt: kötü koşullu kolonları ortonormal Q'ya çevir.
4. $R = Q^T A$ — $R_{ij} = q_i^T a_j$; iç çarpımlar.
5. **Çift adım** — bileşeni çıkar, normalize et (Gram-Schmidt'in kalbi).

6. **Modified GS** — çıkarmaları sırayla yap; sayısal kararlılık.
7. **Kolon pivotlama** — en büyük kalan kolonu seç; rank-açığa-çıkaran.
8. **Krylov uzayı** — span$\{b, Ab, \dots, A^{j-1}b\}$; yalnız Ax çarpımı.
9. **Conjugate gradient** — Krylov uzayında en iyi/en yakın çözüm.
10. **Arnoldi/Lanczos** — Krylov bazını ortonormalleştir; $c = Q^T x$ ucuzluğu.

> **!** Tek Bir Cümle
>
> Kötü koşullu kolonları Gram-Schmidt ile ortonormalleştirmek ($A = QR$) sayısal lineer cebirin omurgasıdır; büyük seyrek sistemlerde aynı ortonormalleştirme (Krylov + Arnoldi/Lanczos) yalnız Ax çarpımıyla iteratif çözümü mümkün kılar — ve ortonormal tabanda her katsayı tek bir iç çarpımdır ($c = Q^T x$).

18.14 Kontrol Soruları

> **i** Soru 1: $a_1 = (1, 1)$, $a_2 = (2, 0)$ kolonlarına Gram-Schmidt uygula; q_1, q_2'yi bul.

Cevap: $q_1 = a_1/\|a_1\| = (1,1)/\sqrt{2}$. Çıkar: $a_2^T q_1 = 2/\sqrt{2} = \sqrt{2}$, $A_2 = a_2 - \sqrt{2} \cdot q_1 = (2,0) - (1,1) = (1,-1)$. Normalize: $q_2 = (1,-1)/\sqrt{2}$.

$$q_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}, \quad q_2 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$

$q_1^T q_2 = 0$ ✓ (ortonormal).

> **i** Soru 2: Soru 1'deki $A = [a_1\ a_2]$ için $R = Q^T A$ matrisini bul. Üst üçgensel mi?

Cevap: $R_{11} = \|a_1\| = \sqrt{2}$, $R_{12} = q_1^T a_2 = \sqrt{2}$, $R_{21} = q_2^T a_1 = 0$, $R_{22} = \|A_2\| = \sqrt{2}$:

$$R = \begin{pmatrix}\sqrt{2} & \sqrt{2}\\ 0 & \sqrt{2}\end{pmatrix}$$

Üst üçgensel ✓ ($R_{21} = 0$, çünkü q_2 sonradan kuruldu ve a_1'e dik). $A = QR$ doğrular.

> **i** Soru 3: Standart Gram-Schmidt kolonları sırayla alır. a_2 neredeyse a_1 yönündeyse risk nedir? Kolon pivotlama nasıl çözer?

Cevap: $a_2 \approx a_1$ ise, q_1 bileşenini çıkardıktan sonra A_2 çok küçük olur; $q_2 = A_2/\|A_2\|$'de minik sayıya bölmek yuvarlama hatasını patlatır (eliminasyondaki küçük pivot gibi). Kolon pivotlama: her adımda kalan tüm kolonlardan mevcut q'ların bileşenini çıkar, **en büyük** kalanı seç → minik sayıya bölmekten kaçınır, sayısal kararlılık.

> Soru 4: A $10^6 \times 10^6$ seyrek bir matris. Neden doğrudan QR/ters yerine Krylov + conjugate gradient kullanırsın?
>
> **Cevap:** Bu boyutta A'yı açıkça tersine çevirmek/QR yapmak imkânsızdır (bellek ve $O(n^3)$ maliyet). Seyrek A ile ucuz olan tek işlem matris-vektör çarpımıdır (Ax). Krylov uzayı $\{b, Ab, A^2b, \ldots\}$ sadece ardışık Ax çarpımlarıyla kurulur; conjugate gradient bu uzayda hızla en iyi çözüme yakınsar (matrisi hiç açmadan). Büyük seyrek sistemler (PDE, ağ Laplacian, öneri) için tek pratik yol.

18.15 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. $a_1 = (1,0,0)$, $a_2 = (1,1,0)$, $a_3 = (1,1,1)$ kolonlarına Gram-Schmidt uygula. q_1, q_2, q_3'ü bul ve ortonormal olduklarını doğrula.

Egzersiz 2. $3x_1 + 4x_2 = 1$ doğrusunda ℓ^2, ℓ^1, ℓ^∞ kazananlarını (sırasıyla $(3/25, 4/25)$, $(0, 1/4)$, $(1/7, 1/7)$) sağla — her birinin kısıtı gerçekten sağladığını ve neden o normda en küçük olduğunu açıkla.

Egzersiz 3. $A = \mathrm{diag}(2,3,0,0)$ ve $b = (1,1,1,1)$ için Krylov vektörlerini b, Ab, A^2b yaz. Bu Krylov uzayının boyutu kaçtır? (İpucu: sıfır özdeğerler hangi bileşenleri öldürür?)

Egzersiz 4. Python ile Gram-Schmidt ve QR:

```
import numpy as np

A = np.array([[1.0, 1.0, 1.0],
              [0.0, 1.0, 1.0],
              [0.0, 0.0, 1.0]])
Q, R = np.linalg.qr(A)
print("Q =\n", Q)
print("R =\n", R)                          # üst üçgensel
print("QᵀQ = I ?", np.allclose(Q.T @ Q, np.eye(3)))
print("QR = A ?", np.allclose(Q @ R, A))

# Krylov uzayı:
M = np.array([[2.0, 1.0], [1.0, 2.0]]); b = np.array([1.0, 0.0])
K = np.column_stack([b, M @ b, M @ (M @ b)])
print("Krylov rank:", np.linalg.matrix_rank(K))
```

Egzersiz 5. (Ders 12 habercisi.) Ders 12, QR'ı özdeğer hesabına uygular: $A_0 = QR$, sonra $A_1 = RQ$ (ters sırada), tekrarla. $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ için $A_1 = RQ$'yu hesapla; A_1'in özdeğerlerinin A'nınkiyle $(3, 1)$ aynı olduğunu doğrula (benzerlik).

18.16 Sonraki Ders İçin Hazırlık

Ders 12: Özdeğer ve Tekil Değerleri Hesaplamak

Ders 11'de Gram-Schmidt/QR ve Krylov'u kurduk. Ders 12 QR'ı beklenmedik bir yere uygular: özdeğer hesabı.

- QR iterasyonu: $A = QR \to A_1 = RQ \to$ tekrarla; köşegene yakınsar
- Neden çalışır: her adım benzerlik (özdeğerler değişmez)
- Hızlandırma (shift) ve simetrik durumda Lanczos
- Tekil değerlerin hesabı ($A^T A$ değil, doğrudan)

> ⚠ Ders 12 Öncesi Yapılacak
>
> - Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (QR iterasyonu).
> - Python'da `np.linalg.qr` ile birkaç matrise $A \to RQ$ adımını birkaç kez uygula; köşegenin özdeğerlere yakınsamasını gözlemle.
> - Ana cümleyi tekrar oku: "$A = QR$ ortonormalleştirir; Krylov + CG büyük sistemleri Ax çarpımıyla çözer."

18.17 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
$\|x\|$ **minimizasyonu**	Kısıt altında en küçük norm; $\ell^1/\ell^2/\ell^\infty$ farkı	2m27
ℓ^1 **seyreklik**	Elmas köşesi eksende → sıfır bileşen	5m02
Gram-Schmidt $A = QR$	Kötü koşullu kolonları ortonormal Q'ya çevir	9m20
$R = Q^T A$	$R_{ij} = q_i^T a_j$; iç çarpımlar	13m13
GS çift adım	Bileşeni çıkar, sonra normalize et	16m05
Modified GS	Çıkarmaları sırayla yap; sayısal kararlılık	20m43
Kolon pivotlama	En büyük kalan kolonu seç; rank-açığa-çıkaran	30m49
Krylov uzayı	span$\{b, Ab, A^2b, ...\}$; yalnız Ax çarpımı	37m35
Conjugate gradient	Krylov uzayında en iyi/en yakın çözüm	39m19
Arnoldi / Lanczos	Krylov bazını ortonormalleştir; $c = Q^T x$	41m15

18.18 ML Bağlantıları Özeti

1. $A = QR$ → least squares'in kararlı çözümü; ortonormal ağırlık başlatması; özdeğer algoritması (Ders 12).
2. $R = Q^T A$ / $c = Q^T x$ → ortonormal tabanda projeksiyon = iç çarpım; Fourier, dikkat skorları, PCA.
3. ℓ^1 **seyreklik** → Lasso, compressed sensing; elmas köşesi.
4. **Kolon pivotlama** → rank-açığa-çıkaran QR; öznitelik seçimi, kötü-koşullu düzeltme.
5. **Krylov + conjugate gradient** → büyük seyrek sistemler; yalnız Ax; PDE, ağ Laplacian, öneri.
6. **Arnoldi/Lanczos** → baskın özdeğer/özvektör (tam SVD yerine); PCA, spektral GCN, spektral kümeleme.
7. **Ortonormallik** → $O(n^2)$ çözmeden $O(n)$ iç çarpım; whitening, dik gömüler, hız.

> **!** Eğer bu dersten tek bir şey alıp gidersen
>
> Kötü koşullu kolonları Gram-Schmidt ile ortonormalleştirmek ($A = QR$) sayısal lineer cebirin omurgasıdır — least squares'i kararlı çözer, özdeğer algoritmasını besler, ve ortonormal tabanda her katsayı tek bir iç çarpıma ($c = Q^T x$) iner. Büyük seyrek sistemlerde aynı ortonormalleştirme Krylov uzayı + conjugate gradient + Arnoldi/Lanczos olarak yalnız Ax çarpımıyla iteratif çözümü mümkün kılar.

19 Özdeğer ve Tekil Değerleri Hesaplamak

QR iterasyonu, shift, Hessenberg ve bidiagonal

> **Bölüm bilgisi**
>
> **Video:** Computing Eigenvalues and Singular Values · **OCW:** MIT 18.065 Lecture 12 · **Okuma süresi:** ~36 dk · **Eğitmen:** Gilbert Strang · **Önkoşul:** Ders 11 (QR faktorizasyonu, Gram-Schmidt ve ortonormal tabanlar).

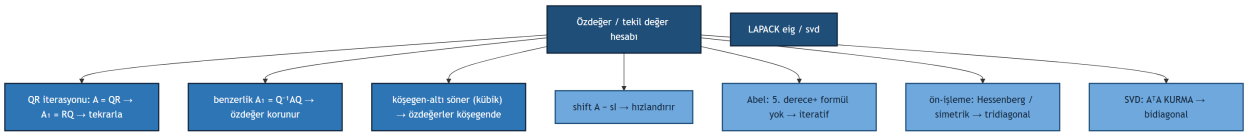
19.1 Bu Derste Ne Var?

`eig` ve `svd` perde arkasında ne yapar? Ders 11'in QR'ı beklenmedik bir yere uygulanır: **özdeğer hesabı**. Strang QR iterasyonunu, shift'i ve Hessenberg/tridiagonal ön-işlemeyi anlatır.

Üç temel fikir:

1. **QR iterasyonu** — $A = QR \to A_1 = RQ \to$ tekrarla; $A_1 = Q^{-1}AQ$ benzerdir (özdeğer korunur), köşegen-altı söner, özdeğerler köşegende belirir.
2. **Ön-işleme** — önce Hessenberg (simetrikte tridiagonal) forma indir; QR'ı çok hızlandırır. İteratif olmak zorunda (Abel: 5. derece denklemin formülü çözümü yok).
3. **Tekil değerler** — A^TA KURMA (kondisyon karelenir), determinant KULLANMA; iki-yönlü ortogonal (Q_1AQ_2) σ'ları korur → bidiagonal forma indir.

Bütün bu fikirlerin tek bir merkezden nasıl dallandığını Şekil 19.1 özetliyor: özdeğer/tekil değer hesabının kalbinde QR iterasyonu, çevresinde benzerlik, shift, Abel sınırı ve ön-işleme vardır.



Şekil 19.1: Ders 12 kavram haritası: özdeğer / tekil değer hesabı merkezinden ana fikirlere dallar — QR iterasyonu (A = QR → A_1 = RQ → tekrarla), benzerlik ($A_1 = Q^{-1}AQ$ özdeğeri korur), köşegen-altının kübik sönmesi (özdeğerler köşegende), shift (A − sI yakınsamayı hızlandırır), Abel sınırı (5. derece+ kapalı formül yok → iteratif), ön-işleme (Hessenberg, simetrikte tridiagonal) ve SVD (A^TA KURMA → bidiagonal); ayrıca her şeyi çalıştıran LAPACK eig/svd düğümü.

"It's called the QR method because you start by factoring your matrix into QR." — Strang, 2:12

Builder Notu — eig'in Motoru

- **QR iterasyonu = eig'in motoru:** `numpy.linalg.eig/eigh` arka planda Hessenberg/tridiagonal + shift'li QR (LAPACK) çalıştırır; doğrudan çağırırsın, kütüphane optimize eder.
- **Benzerlik = özdeğer korur** — her QR adımı $A_1 = Q^{-1}AQ$; spektral yöntemlerin (PCA, GCN) hesap temeli.
- **$\mathbf{A^TA}$'dan kaçın** — tekil değer için A^TA kurmak kondisyonu kareler (Ders 6); SVD doğrudan A üzerinde bidiagonalleştirme + QR ile bulunur.
- **Abel sınırı** — özdeğer/tekil değer kesin formülü yok (5. derece+); sadece iteratif yaklaşım, ε-yakın cevap.

Tek cümle: özdeğer/tekil değer hesabının motoru QR iterasyonudur — benzerlik dönüşümleriyle köşegen-altını söndürür; Hessenberg/tridiagonal/bidiagonal ön-işleme ve shift onu pratik kılar.

19.2 1. QR Yöntemi: A = QR → RQ

Özdeğerleri hesaplamak için QR faktorizasyonu (Ders 11) beklenmedik biçimde anahtar olur. Matrisi QR'a ayır, sonra çarpanları **ters sırada** çarp — yeni matris bu:

$$A_0 = Q_0R_0, \qquad A_1 = R_0Q_0$$

"It's called the QR method because you start by factoring your matrix into QR." — Strang, 2:12

Sonra A_1'i tekrar QR'a ayır, ters sırada çarp, A_2 üret — tekrar tekrar. Umut: özdeğerler değişmesin ve matris köşegene yakınsasın.

QR iterasyonunun bir simetrik matriste köşegeni adım adım özdeğerlere nasıl taşıdığını Şekil 19.2 gösteriyor: köşegen-altı sönerken köşegen değerleri kesik çizgilerle işaretli gerçek özdeğerlere oturuyor.

Builder Notu — Ters Sırada Çarp

QR iterasyonu, `numpy.linalg.eig/eigh`'in (LAPACK) motorudur. "A = QR sonra RQ" döngüsü, 1960'larda özdeğer hesabını kökten değiştirdi — önceki tüm yöntemleri silip süpürdü. Sen `eig` çağırırsın, arka planda bu döner.

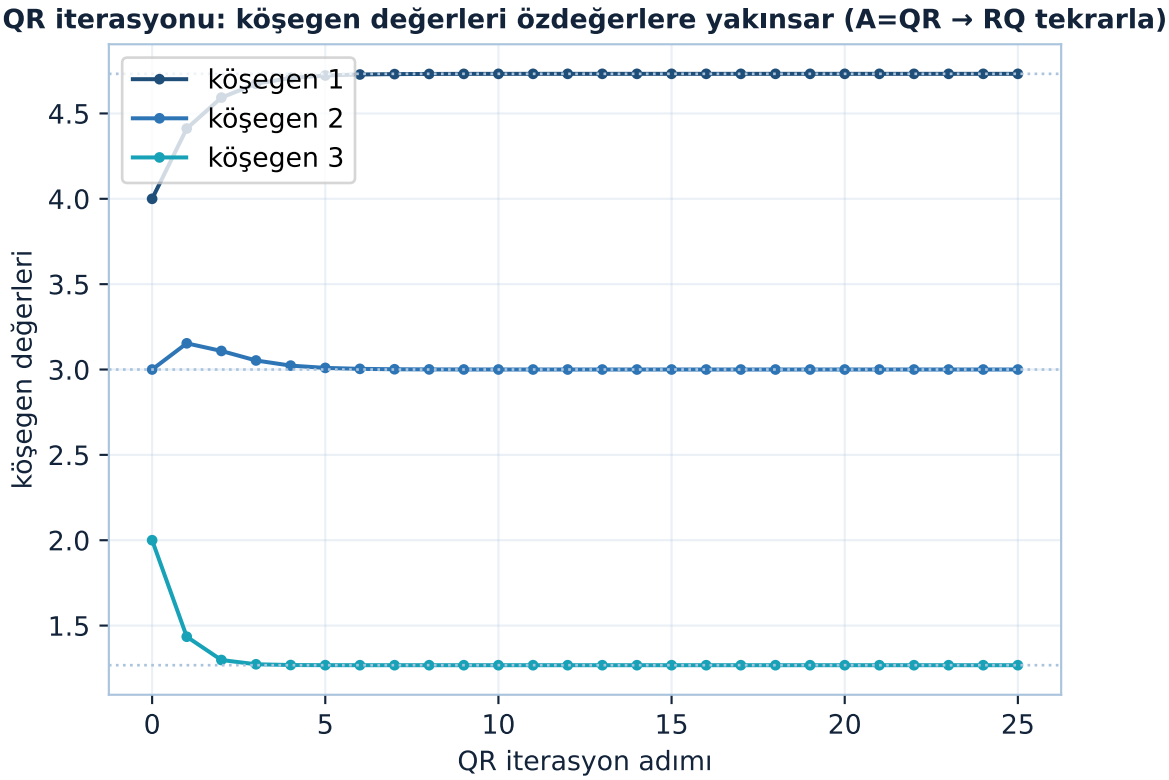
19.3 2. Neden Özdeğerler Korunur (Benzerlik)

A_1'in özdeğerleri A_0'la aynı mı? İki matrisin aynı özdeğere sahip olduğunu göstermenin en iyi yolu: **benzer** olduklarını göstermek (Ders 4).

"They're similar, that's right." — Strang, 4:29

$A_1 = R_0Q_0$. $Q_0R_0 = A_0$ olduğundan $R_0 = Q_0^{-1}A_0$; yerine koy:

$$A_1 = R_0Q_0 = Q_0^{-1}(Q_0R_0)Q_0 = Q_0^{-1}A_0Q_0$$



Şekil 19.2: QR iterasyonu ($A = QR \rightarrow A_1 = RQ$ tekrarla): bir simetrik A'nın köşegen girdileri adım adım özdeğerlere (kesik çizgiler) yakınsar — köşegen-altı sıfıra sönerken köşegende özdeğerler beliriyor.

Bu tam bir benzerlik dönüşümüdür → A_1 ve A_0 aynı özdeğerlere sahiptir. Her QR adımı özdeğerleri korur; yalnızca temsili değiştirir.

> Builder Notu — Benzerlik Güvencesi
>
> "Her adım $Q^{-1}AQ$ benzerliği" güvencesi, iterasyonun özdeğerleri bozmadan matrisi sadeleştirmesini sağlar. Aynı benzerlik fikri (taban değişimi özdeğeri korur) PCA, spektral kümeleme ve ağ reparametrizasyonunun ortak ilkesidir.

19.4 3. Köşegen-altı Söner → Özdeğerler

Mucize: çoğu matriste QR adımları köşegen-altı girdileri giderek küçültür. $A_0 \to A_1 \to A_2$ ... köşegen-altı epsilonlara iner ve köşegende özdeğerler belirir.

> *"…the eigenvalues pop up on the diagonal."* — Strang, 17:36

2×2 örnekte köşegen-dışı girdi her adımda **küpü** alınır ($\sin\theta \to \sin^3\theta \to \sin^9\theta \to \ldots$), yani çok hızlı 0'a gider:

> *"…cubic convergence…"* — Strang, 9:29

Köşegen-altı küçük epsilonlar özdeğerleri pek değiştirmez (benzerlik korur), o yüzden köşegen değerleri özdeğerlere yakınsar — en alttaki önce.

Köşegen-altı girdinin log eksende ne kadar dik düştüğünü Şekil 19.3 gösteriyor: birkaç adımda sıfıra çöken bu eğri, kübik yakınsamanın imzasıdır.

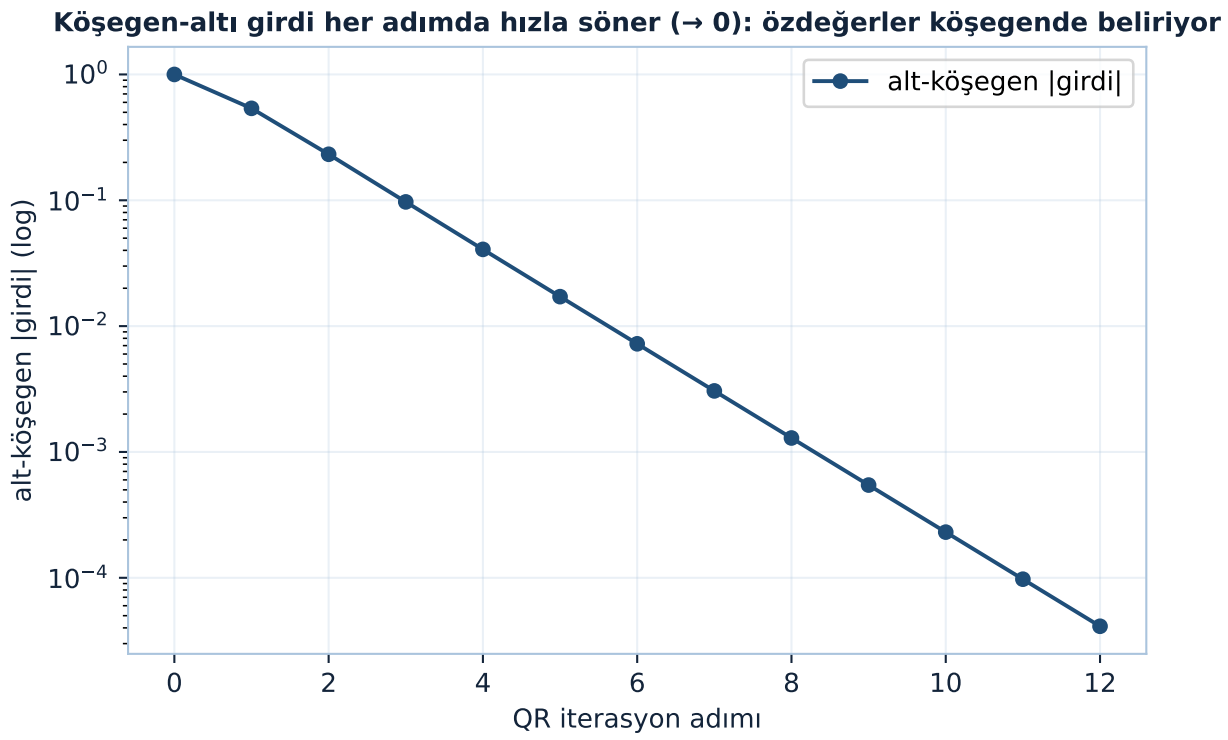
> Builder Notu — Kübik Hız
>
> Kübik yakınsama, QR iterasyonunu pratik kılan şeydir: birkaç adımda makine hassasiyetine ulaşır. Bu hız, `eig`'in milisaniyelerde çalışmasının sebebi; büyük matrislerde ise yalnız baskın özdeğerleri arayan yöntemler (Lanczos, Ders 11) devreye girer.

19.5 4. Shift (Kaydırma) ile Hızlandırma

Numerik analistler daha hızlısını istedi ve **shift** (kaydırma) bulundu. Birim matrisin bir katını çıkar, QR yap, ters sırada çarp, kaydırmayı geri ekle:

$$A_0 - sI = QR, \qquad A_1 = RQ + sI$$

> *"…the idea of introducing a shift…"* — Strang, 10:27



Şekil 19.3: Köşegen-altı girdi her QR adımında hızla söner (→ 0): A=QR → RQ tekrarlandıkça özdeğerler köşegende beliriyor. Log eksende dik düşüş, kübik yakınsamanın imzasıdır.

sI ile kaydırmak özvektörleri **değiştirmez**, özdeğerleri s kadar **kaydırır** ($\lambda \to \lambda - s$). İyi seçilmiş bir shift yakınsamayı çok hızlandırır. Ve benzerlik korunur: $R(Q)$ hesabında R_0 ile R_0^{-1} sadeleşir, çıkarılan sI ile eklenen sI birbirini götürür → A_1 hâlâ A_0'a benzerdir.

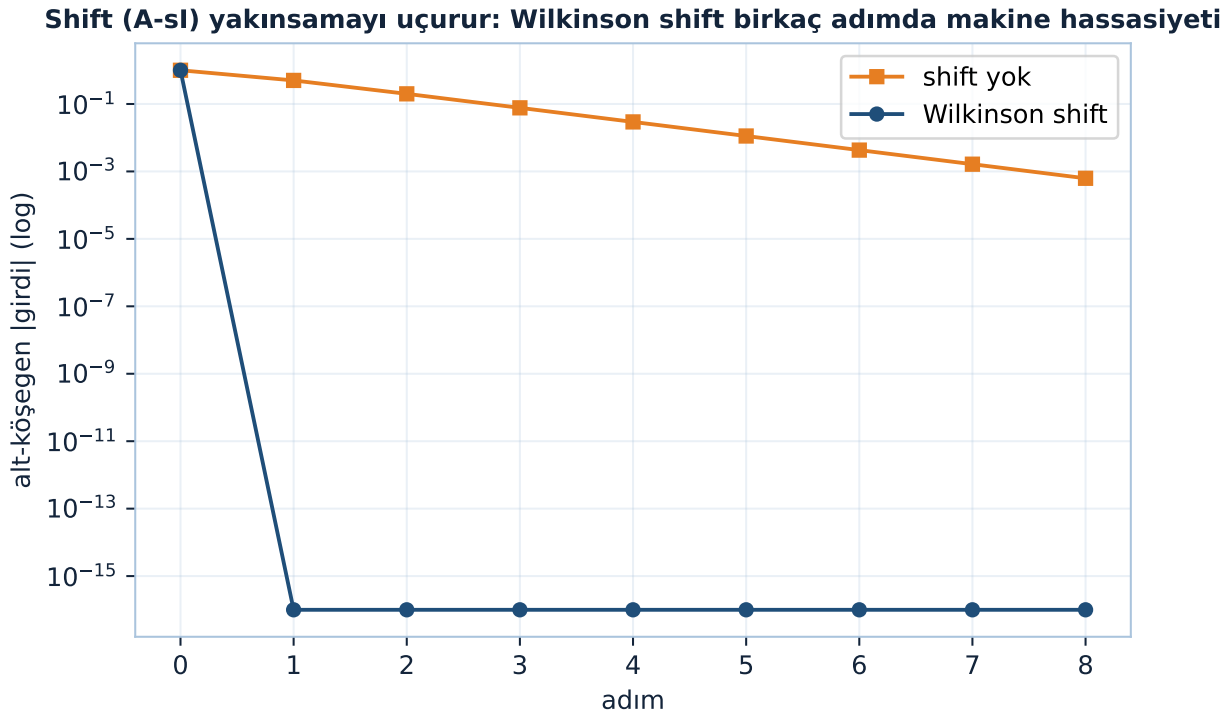
Shift'in yakınsamayı ne kadar dramatik hızlandırdığını Şekil 19.4 gösteriyor: saf QR sekiz adımda ancak ilerlerken Wilkinson shift'li QR tek adımda makine hassasiyetine çöker.

> Builder Notu — Akıllı Kaydırma
>
> Shift, QR iterasyonunu pratikte kullanılabilir kılan hızlandırmadır (özellikle son özdeğere yakın shift seçmek). Aynı "kaydır-çöz-geri kaydır" fikri ters iterasyon (inverse iteration) ve önkoşullamada da görülür; bilinen bir özdeğer tahminine yakınsamayı hızlandırır.

19.6 5. Tam Çözüm İmkânsız (Abel)

Neden tam çözüm yok, neden iterasyon? Eliminasyonda $Ax = b$'yi sonlu adımda tam çözeriz. Ama özdeğerler n. dereceden bir denklemin kökleridir ve Abel bir asır önce kanıtladı: 5. derece ve üstü denklemin kökleri için **kapalı formül yoktur**.

> *"5th degree, yeah."* — Strang, 21:36



Şekil 19.4: Shift (A − sI) yakınsamayı uçurur. Aynı 2×2 simetrik A için alt-köşegen girdisinin büyüklüğü (log ölçek): turuncu kareler saf QR iterasyonu (shift yok) — her adımda yalnızca geometrik olarak küçülür ve 8 adımda hâlâ 10^{-3} düzeyinde. Navy daireler Wilkinson shift'li QR — tek adımda makine hassasiyetine ($\approx 10^{-16}$) çöker. Shift, özdeğeri köşegene fırlatarak yakınsamayı dramatik biçimde hızlandırır.

Yani özdeğerleri (ve tekil değerleri) sonlu, kesin adımlarla bulmak matematiksel olarak imkânsızdır — yalnızca QR iterasyonuyla istediğimiz kadar (ε) yaklaşabiliriz. Özdeğer problemi, $Ax = b$'den bir zorluk seviyesi yukarıdadır.

> Builder Notu — Abel'in Duvarı
>
> Abel sınırı, neden `eig`'in "kesin" değil "ε-yakın" olduğunu açıklar: ~n^3 adımda makine hassasiyetine ulaşır ama prensipte sonsuz adım gerekir. Bu, sayısal lineer cebir ile cebirin temel ayrımı — ML'de özdeğer/SVD hep yaklaşık (ama pratikte yeterince kesin).

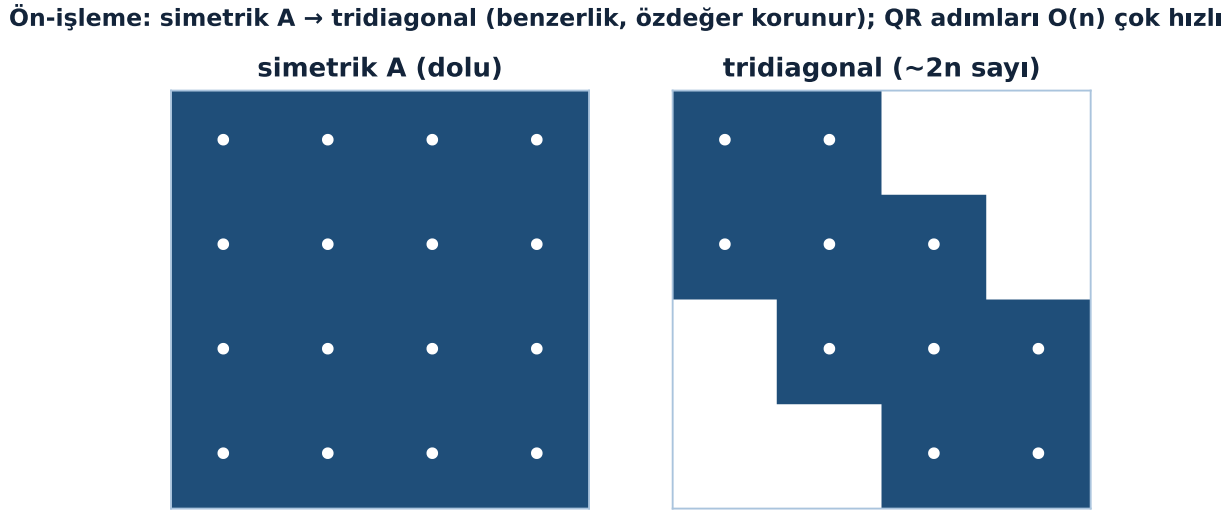
19.7 6. Hessenberg Formu (Ön-İşleme)

İş nerede? QR faktorizasyonunda. Onu hızlandırmak için matrisi önce çok sıfırlı bir forma indir. Tam üçgensel yapamayız (özdeğerleri bulmuş olurduk, Abel yasak), ama köşegenin bir altına kadar sıfırlayabiliriz — **Hessenberg** formu.

> *"Upper Hessenberg."* — Strang, 22:56

Hessenberg matris = üst üçgensel + bir alt köşegen, gerisi sıfır (≈yarı n^2 sıfır). Bu sıfırlar QR adımlarında sıfır kalır, böylece her QR çok daha ucuz. Tam yöntem: **(1)** A'yı benzerlikle Hessenberg'e indir, **(2)** shift'li QR uygula.

Dolu bir simetrik matrisin ön-işlemeyle nasıl tridiagonal yapıya indiğini Şekil 19.5 gösteriyor: 16 dolu girdi yalnız ana köşegen + bir alt/üst köşegene (~2n sayı) iniyor.



Şekil 19.5: Ön-işleme: simetrik bir matris A, benzerlik dönüşümleriyle tridiagonal forma indirilir. Sol panelde dolu simetrik A'nın 16 girdisi, sağ panelde tridiagonal yapının yalnızca ana köşegen ile bir alt/üst köşegeni (~2n sayı) doludur. Özdeğerler korunur, sonraki QR adımları O(n) çok hızlı çalışır.

> Builder Notu — Yarı Yol Sıfır
>
> Hessenberg ön-işleme, `eig`'in iki aşamalı yapısının ilk yarısıdır (LAPACK). Benzerlik dönüşümüyle (Householder) yapıldığından özdeğerleri korur ve sonraki QR adımlarının maliyetini $O(n^2)$'den çok aşağı çeker — pratik özdeğer hesabının sırrı.

19.8 7. Simetrik → Tridiagonal

Matris simetrikse daha da iyi. QR adımları simetriyi korur (A_0 simetrik → A_1 simetrik). Simetrik + Hessenberg = **tridiagonal**: yalnız ana köşegen + bir üst + bir alt köşegen.

> *"…tridiagonal matrix."* — Strang, 27:09

Tridiagonal matriste yalnızca ~2n sayı vardır (üst ve alt köşegen simetriden eşit). QR adımları tridiagonalliği korur, böylece her adım $O(n^2)$ yerine $O(n)$ — "bomba gibi" hızlı. Simetrik `eigh` bu yüzden çok hızlıdır.

> Builder Notu — İki Köşegen Yeter
>
> Simetrik → tridiagonal indirgeme, kovaryans/Gram/Hessian gibi ML matrislerinin özdeğerlerini hesaplamanın hızlı yoludur (`numpy.linalg.eigh`). PCA, kernel yöntemleri ve spektral kümeleme bu rutinin hızına dayanır.

19.9 8. eig(A): Hessenberg + Shift'li QR

Tam algoritma iki adımdır: **(1)** A'yı Hessenberg (simetrikse tridiagonal) forma indir, **(2)** shift'li QR uygula. `eig(A)` perde arkasında bunu yapar — ama Matlab/NumPy kendi kodunu değil, profesyonel **LAPACK** rutinlerini çağırır.

LAPACK, on yazarlı, onyıllarca emek verilmiş sayısal lineer cebir "İncili"dir; özdeğer kodları indirilebilir ve her ciddi sistem ona dayanır. Yani sen `eig`/`eigh`/`svd` çağırırsın, dünyanın en iyi sayısal analistlerinin kodu çalışır.

> Builder Notu — LAPACK'i Çağır
>
> "Kendin yazma, LAPACK çağır" — sayısal ML'nin altın kuralı. Özdeğer/SVD rutinleri (geev, syev, gesdd) onlarca yıllık kararlılık çalışmasının ürünü; elle QR yazmak hem yavaş hem riskli. NumPy/PyTorch/SciPy hepsi LAPACK'e köprüdür.

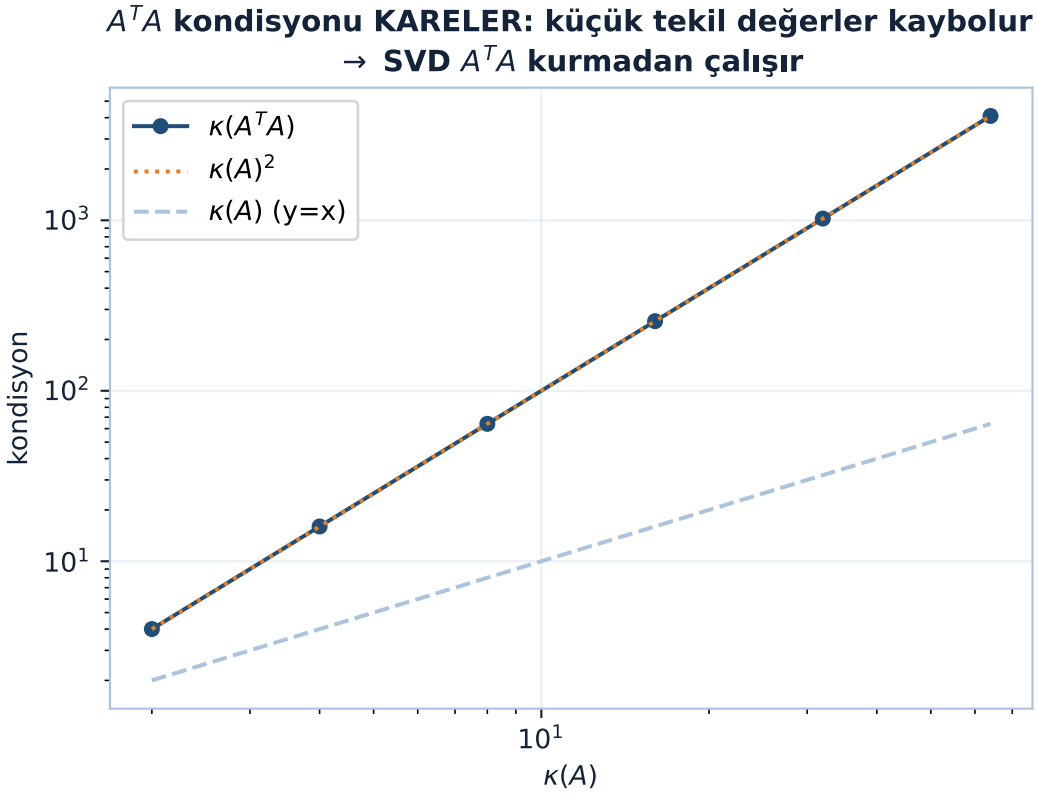
19.10 9. Tekil Değerler: A^TA ve Determinant'tan KAÇIN

Tekil değerler A^TA'nın özdeğerlerinin karekökleridir — ama bu yolla **hesaplanmaz**.

> *"…You would never form A transpose A."* — Strang, 28:43

$A^T A$ kurmak kondisyon sayısını **kareler** (Ders 6): küçük tekil değerlerin bilgisi yuvarlama hatasında kaybolur. İki "asla yapma": (1) $A^T A$ kurma; (2) özdeğerleri determinant denkleminden ($\det(A - \lambda I) = 0$) bulma — hem çok yavaş hem umutsuzca kötü-koşullu (n^2 bilgi n katsayıya sıkışır). Bunun yerine SVD doğrudan A üzerinde çalışır.

Kondisyon sayısının $A^T A$ kurmakla nasıl karelendiğini Şekil 19.6 gösteriyor: $\kappa(A^T A) = \kappa(A)^2$ eğrisi $\kappa(A)$'nın çok üstüne fırlar, küçük tekil değerleri yutar.



Şekil 19.6: A^TA kondisyonu KARELER: $\kappa(A^TA) = \kappa(A)^2$ (navy noktalar turuncu $\kappa(A)^2$ eğrisiyle çakışır), oysa $\kappa(A)$ (steel kesikli, y=x) yarı eğimde kalır. Küçük tekil değerler A^TA'da kaybolur — bu yüzden SVD, A^TA'yı kurmadan A üzerinden bidiagonal yolla çalışır.

> Builder Notu — A^TA'yı Asla Kurma
>
> "A^TA kurma, determinant kullanma" — iki yaygın hatayı yasaklar. `np.linalg.svd` doğrudan A'yı bidiagonalleştirir ($A^T A$'sız); en küçük tekil değerlerin doğruluğunu korur. Ters problemler, düşük-rank yaklaşım ve PCA bu kararlılığa bağlıdır.

19.11 10. Değişmezlik: Benzerlik vs İki-Yönlü Ortogonal

Özdeğer ve tekil değer için "neyi serbestçe değiştirebilirim" soruları farklı cevap verir. Özdeğer, **benzerlik** altında değişmez (her iki yan aynı matris ve tersi):

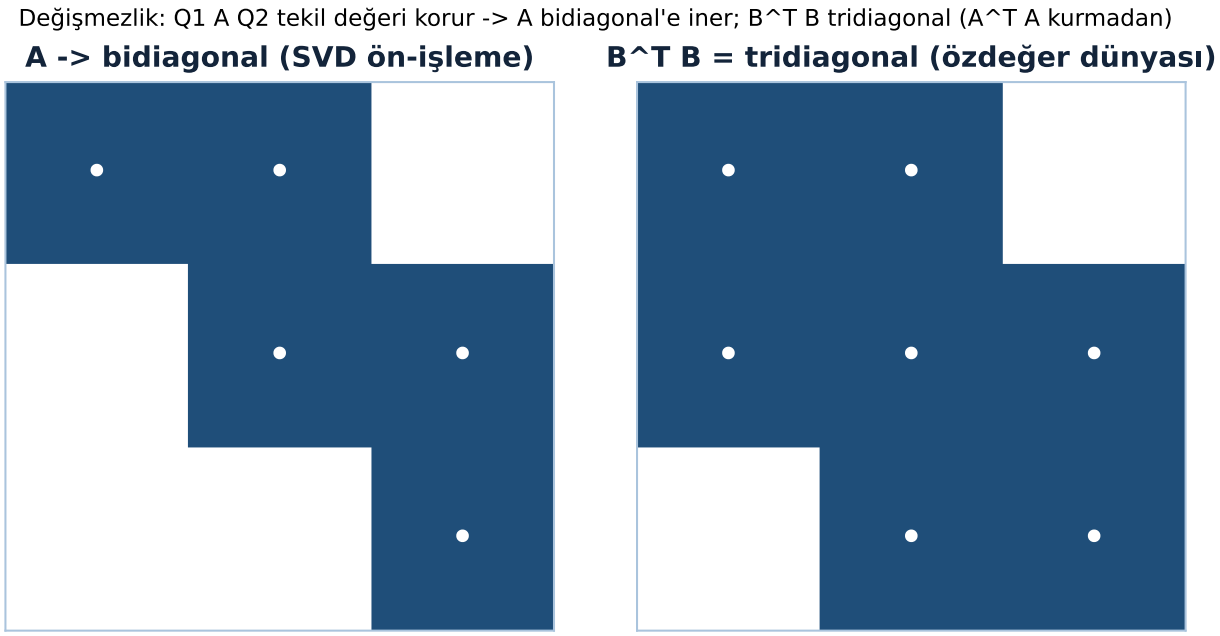
$$Q^{-1}AQ \;\Rightarrow\; \text{same eigenvalues}$$

Tekil değer ise **iki ayrı** ortogonal matris altında değişmez (SVD'de $A = U\Sigma V^T$; her iki yana farklı ortogonal çarpabilirim):

$$Q_1 A Q_2 \;\Rightarrow\; \text{same singular values}$$

Çünkü $Q_1 U$ ve $V^T Q_2$ yine ortogonaldir, ortadaki Σ değişmez. Eigenvalue'da tek bir Q ile sınırlıyken (tridiagonal'a iner), singular value'da iki serbestlik var — daha çok sadeleştirme.

İki ortogonal serbestliğin neye yol açtığını ve $B^T B$'nin tridiagonal çıkışını Şekil 19.7 gösteriyor: $Q_1 A Q_2$ tekil değeri koruduğundan A bidiagonal'e iner, $B^T B$ ise eigenvalue dünyasının tridiagonal'ıdır.



Şekil 19.7: Değişmezlik ve bidiagonal indirgeme: ortogonal $Q_1 A Q_2$ tekil değerleri korur, böylece A üst-bidiagonal forma iner (SVD ön-işleme). $B^\mathsf{T} B$ ise tridiagonaldir — özdeğer dünyasına $A^\mathsf{T} A$'yı hiç kurmadan geçilir.

> **Builder Notu — Hangi Dönüşüm Neyi Korur**
>
> "Hangi dönüşüm neyi korur" ayrımı temeldir: benzerlik (taban değişimi) özdeğeri, iki-yönlü ortogonal tekil değeri korur. Bu, PCA'nın (kovaryans benzerliği) ve SVD-tabanlı sıkıştırmanın (ortogonal değişmezlik) neden taban seçiminden bağımsız olduğunu açıklar.

19.12 11. Bidiagonal Form (SVD İçin)

İki ortogonal serbestlik sayesinde SVD ön-işlemesi tridiagonal'dan daha ileri gider: **bidiagonal** form (ana köşegen + bir üst köşegen, gerisi sıfır).

> *"…reduce it even further from tridiagonal to bidiagonal."* — Strang, 38:11

Algoritma yine iki aşamalı: **(1)** A'yı iki-yönlü ortogonal ile bidiagonal'e indir (σ'lar korunur), **(2)** QR-tipi yöntemle σ'ları bul. Bidiagonal A için A^TA tridiagonaldir — eigenvalue dünyasıyla tam örtüşür. Böylece SVD, A^TA'yı hiç açıkça kurmadan hesaplanır.

> Builder Notu — Bidiagonal'e İn
>
> Bidiagonalleştirme + örtük QR, `np.linalg.svd`'nin (LAPACK gesdd/gesvd) yaptığıdır: A^TA'sız, sayısal kararlı SVD. Düşük-rank yaklaşım, PCA, pseudoinverse ve gürültü gidermenin güvenilir temeli.

19.13 Bu Dersin Özeti

1. **QR iterasyonu** — $A = QR \to A_1 = RQ \to$ tekrarla; özdeğerler köşegende belirir.
2. **Benzerlik** — $A_1 = Q^{-1}AQ$; özdeğerler korunur.
3. **Köşegen-altı söner** — kübik yakınsama; köşegen → özdeğerler.
4. **Shift** — $A - sI$ ile hızlandır; benzerlik korunur, $\lambda \to \lambda - s$.
5. **Abel sınırı** — 5. derece+ kapalı formülü yok; iterasyon zorunlu (ε-yakın).
6. **Hessenberg** — bir alt köşegen + sıfırlar; ön-işleme QR'ı hızlandırır.
7. **Simetrik → tridiagonal** — ~2n sayı; $O(n)$ adım, çok hızlı.
8. **eig(A)** — Hessenberg + shift'li QR; LAPACK.
9. **A^TA ve determinant'tan KAÇIN** — kondisyon karelenir / kötü-koşullu.
10. **Değişmezlik** — $Q^{-1}AQ$ özdeğeri, Q_1AQ_2 tekil değeri korur; SVD → bidiagonal.

> Tek Bir Cümle
>
> Özdeğer/tekil değer hesabının motoru QR iterasyonudur: $A = QR \to RQ$ döngüsü her adımda $Q^{-1}AQ$ benzerliğiyle özdeğerleri korur ve köşegen-altını söndürür; Hessenberg/tridiagonal/bidiagonal ön-işleme ve shift bunu pratik kılar — ama Abel yüzünden hep iteratif (ε-yakın), asla kesin.

19.14 Kontrol Soruları

> Soru 1: QR iterasyonunda $A_1 = RQ$'nun $A_0 = QR$ ile aynı özdeğerlere sahip olduğunu göster.

Cevap: $A_0 = QR$ olduğundan $R = Q^{-1}A_0$. Yerine koy:

$$A_1 = RQ = Q^{-1}A_0Q$$

Bu bir benzerlik dönüşümü ($M = Q$ ile $M^{-1}A_0M$). Benzer matrisler aynı özdeğerlere sahiptir (Ders

4). Q ortogonal olduğundan $Q^{-1} = Q^T$, hesap kararlı. Demek ki tüm $A_0, A_1, A_2, \ldots$ aynı özdeğerlere sahiptir; köşegende beliren değerler gerçek özdeğerlerdir.

i Soru 2: A − sI kaydırması özdeğerleri ve özvektörleri nasıl etkiler? Neden yakınsamayı hızlandırır?

Cevap: $A_0 v = \lambda v$ ise $(A_0 - sI)v = (\lambda - s)v$: özvektörler **aynı**, özdeğerler s kadar **kayar** ($\lambda \to \lambda - s$). Eğer s bir özdeğere (örneğin en küçüğüne) yakınsa, o özdeğer 0'a yaklaşır ve QR adımında köşegen-altı o satırda çok hızlı söner. İyi shift seçimi kübik yakınsamayı daha da hızlandırır. Kaydırma geri eklenince ($A_1 = RQ + sI$) benzerlik korunur.

i Soru 3: Neden A'yı basit adımlarla tam üst üçgensel (özdeğerler köşegende) yapamayız, sadece Hessenberg'e indirebiliriz?

Cevap: Tam üçgensel yapabilseydik özdeğerleri köşegende **kesin** okurduk — sonlu adımda. Ama özdeğerler n. dereceden bir denklemin kökleridir ve Abel kanıtladı: 5. derece+ için kapalı formül yoktur. Sonlu kesin adımlarla çözmek bu teoremi çiğnerdi. Bu yüzden en fazla Hessenberg'e (bir alt köşegen kalır) inebiliriz; gerisi iteratif QR ile ε-yakın bulunur.

i Soru 4: Tekil değerleri bulmak için neden $A^T A$'yı açıkça kurmazsın?

Cevap: $\sigma = \sqrt{\lambda_i(A^T A)}$ (yani σ, $A^T A$'nın bir özdeğerinin kareköküdür) matematiksel olarak doğru ama $A^T A$ kurmak **kondisyon sayısını kareler** ($\kappa \to \kappa^2$). Küçük tekil değerlerin bilgisi yuvarlama hatasında kaybolur (örneğin $\sigma = 10^{-4}$ ise $\sigma^2 = 10^{-8}$ makine epsilon'una gömülür). Bunun yerine SVD doğrudan A'yı bidiagonalleştirir ($A^T A$'sız) ve en küçük σ'ların doğruluğunu korur. Ters problemler ve düşük-rank yaklaşımda bu kritiktir.

19.15 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ için bir QR iterasyon adımı yap: $A = QR$ bul, sonra $A_1 = RQ$ hesapla. A_1'in köşegeninin özdeğerlere (3, 1) yaklaştığını gözlemle.

Egzersiz 2. $A = \begin{pmatrix} 5 & 1 \\ 0 & 2 \end{pmatrix}$ için $s = 2$ shift uygula: $(A - 2I) = QR$, $A_1 = RQ + 2I$. A_1'in özdeğerlerinin hâlâ 5 ve 2 olduğunu doğrula.

Egzersiz 3. Aşağıdaki simetrik matris zaten tridiagonal mi? QR iterasyonunda tridiagonal kalır mı? Kaç bağımsız sayı içerir ($n = 4$)?

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

Egzersiz 4. Python ile QR iterasyonunu elle yap:

```
import numpy as np

A = np.array([[2.0, 1.0], [1.0, 2.0]])
for k in range(5):
    Q, R = np.linalg.qr(A)
    A = R @ Q                       # QR iterasyon adımı
    print(f"adım {k+1}, köşegen:", np.diag(A))   # 3, 1'e yakınsar
print("eig(A0):", np.linalg.eigvalsh([[2.0, 1.0], [1.0, 2.0]]))
```

Egzersiz 5. (Ders 13 habercisi.) Ders 13 rastlantısal matris çarpımına geçer: dev AB çarpımını tüm kolonlar yerine **örneklenmiş** kolon/satır çiftleriyle yaklaşık hesaplamak. $AB = \sum$ (A'nın kolonu)(B'nin satırı) ifadesini hatırla (Ders 1); neden bazı terimleri olasılıkla örneklemek tüm toplamı yaklaşık verebilir? Bir terim seçme olasılığını ne ile orantılı alırdın?

19.16 Sonraki Ders İçin Hazırlık

Ders 13: Rastlantısal Matris Çarpımı

Ders 12'de kesin özdeğer/SVD'yi gördük. Ders 13 dev matrisler için olasılığa döner: örnekleme ile yaklaşık hesap.

- Rastlantısal matris çarpımı: kolon/satır çiftlerini örnekle
- Norm-kare örnekleme olasılığı (önemli terimleri seç)
- Beklenen değer ve varyans (Stat 110 köprüsü)
- Randomized SVD'ye giriş; büyük-ölçek ML

> ⚠ Ders 13 Öncesi Yapılacak
>
> - Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (örnekleme sezgisi).
> - Python'da bir AB çarpımını az sayıda kolon/satır çiftiyle yaklaşık hesaplamayı dene; hatayı gözlemle.
> - Ana cümleyi tekrar oku: "QR iterasyonu özdeğerleri benzerlikle köşegene taşır; Abel yüzünden hep iteratif."

19.17 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
QR iterasyonu	$A = QR \to A_1 = RQ \to$ tekrarla	2m12
Benzerlik	$A_1 = Q^{-1}AQ$; özdeğerler korunur	4m29
Kübik yakınsama	Köşegen-dışı her adımda küpü alınır	9m29

Kavram	Tanım	Strang'de
Özdeğerler köşegende	İterasyon limiti; köşegen → λ	17m36
Shift	$A - sI$; $\lambda \to \lambda - s$, hızlandırır	10m27
Abel sınırı	5. derece+ kapalı formül yok; iterasyon zorunlu	21m36
Hessenberg	Bir alt köşegen + sıfırlar; ön-işleme	22m56
Simetrik → tridiagonal	~2n sayı; $O(n)$ adım, çok hızlı	27m09
A^TA'dan kaçın	Kondisyon karelenir; SVD doğrudan A'da	28m43
Bidiagonal (SVD)	İki-yönlü ortogonal; σ'lar için ön-işleme	38m11

19.18 ML Bağlantıları Özeti

1. **QR iterasyonu** → `eig`/`eigh`/`svd`'nin motoru (LAPACK); PCA, spektral yöntemler.
2. **Benzerlik** ($Q^{-1}AQ$) → özdeğer korur; taban değişiminin ML'deki güvencesi.
3. **Shift** → ters iterasyon, önkoşullama; bilinen tahmine hızlı yakınsama.
4. **Simetrik → tridiagonal** → kovaryans/Gram/Hessian özdeğerlerini hızlı hesaplama.
5. **A^TA'dan kaçın** → SVD'nin sayısal kararlılığı; küçük σ'ların korunması.
6. **Abel sınırı** → özdeğer/SVD hep yaklaşık (ε-yakın); ML'de "yeterince kesin".
7. **LAPACK çağır** → elle yazma; onyıllarca kararlılık çalışmasının ürünü.

> **!** Eğer bu dersten tek bir şey alıp gidersen
>
> `eig` ve `svd`'nin motoru QR iterasyonudur: $A = QR \to RQ$ döngüsü her adımda $Q^{-1}AQ$ benzerliğiyle özdeğerleri korur ve köşegen-altını kübik hızla söndürür. Hessenberg (simetrikte tridiagonal) ön-işleme ve shift bunu pratik kılar; SVD ise A^TA'yı hiç kurmadan bidiagonal forma indirger. Abel yüzünden hep iteratif (ε-yakın) — ama LAPACK bunu milisaniyelerde, güvenilir biçimde yapar.

20 Rastlantısal Matris Çarpımı

Mean, variance, norm-kare örnekleme — SGD'nin atası

> **Bölüm bilgisi**
>
> **Video:** MIT 18.065 — Randomized Matrix Multiplication · **OCW:** Lecture 13 · **Okuma süresi:** ≈38 dk · **Eğitmen:** Gilbert Strang · **Önkoşul:** Ders 12 (büyük matrisler, iteratif yöntemler).

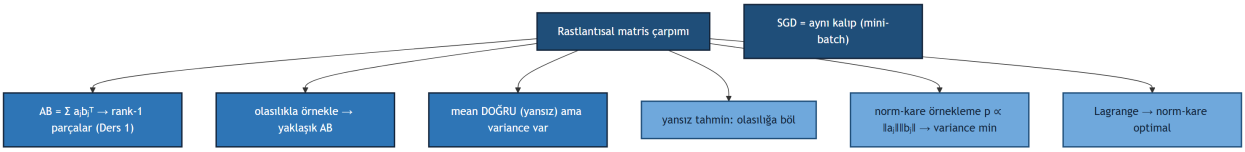
20.1 Bu Derste Ne Var?

Dev matrisler için AB'yi tüm terimlerle değil, **örnekleyerek** yaklaşık hesaplamak. Bu, kursun istatistik/olasılıkla ilk buluşması — mean, variance ve **norm-kare örnekleme**.

Üç temel fikir:

1. $AB = \sum$ **dış çarpımı örnekle** — $AB = \sum_j a_j b_j^T$ (Ders 1); tüm terimler yerine olasılıkla seçilen birkaçını al, ortalaması AB'ye yakın.
2. **Mean doğru, variance sıfır değil** — tahmin edici yansızdır (ortalama $= AB$) ama her örnek yanlış; iş, variance'ı küçültmek.
3. **Norm-kare örnekleme** — olasılığı $p_j \propto \|a_j\| \, \|b_j\|$ seçmek variance'ı minimize eder (Lagrange ile); büyük kolonlar daha sık.

Aşağıdaki kavram haritası bu dersin merkezindeki rastlantısal çarpımdan ana fikirlere uzanan dalları, ve aynı kalıbı tekrarlayan SGD düğümünü gösterir (Şekil 20.1).



Şekil 20.1: Ders 13 kavram haritası: rastlantısal matris çarpımı merkezinden ana fikirlere dallar — AB = Σ $a_j b_j^T$ rank-1 parçalarının toplamı (Ders 1), parçaları olasılıkla örnekle → yaklaşık AB, mean DOĞRU (yansız) ama variance var, yansız tahmin için her terimi olasılığına böl, norm-kare örnekleme p ∝ $\|a_j\| \|b_j\|$ variance'ı en aza indirir (önemli terimleri sık seç) ve Lagrange çarpanı bu norm-kare olasılığını optimal kılar; ayrıca aynı kalıbı tekrarlayan SGD = mini-batch düğümü.

> *"…the mean value of the random AB is correct AB. But there will be a variance."* — Strang, 2:28

Builder Notu — SGD'nin Atası

- **Yansız tahmin + variance** — stokastik gradient descent'in (Ders 25) tam mantığı: mini-batch gradyanı gerçeğin yansız tahminidir, variance batch boyutuyla düşer.
- **Norm-kare örnekleme** — önemli (büyük) terimleri daha sık seçmek = importance sampling; randomized SVD ve büyük-ölçek çarpımın temeli.
- $AB = \sum$ **rank-1** — Ders 1'in dış-çarpım görüşü; her örnek bir rank-1 parça, toplamı AB.
- **Mean/variance** — Stat 110 köprüsü; bu kursun ilk olasılık dersi, ileride olasılık bölümünde derinleşir.

Tek cümle: dev AB'yi rank-1 parçalarını olasılıkla örnekleyerek yaklaşık hesaplarız; doğru tahmin edici (yansız) mean'i AB yapar, norm-kare örnekleme ise variance'ı minimize eder — SGD'nin atası.

20.2 Rastlantısal LA: Neden

Matris çok çok büyükse (Ders 10'daki "devasa" durum) tam hesap imkânsızdır. Çözüm: olasılıkla **örnekle**.

> *"…randomized matrix multiplication. It's a pretty cool idea…"* — Strang, 0:33

Plan: AB çarpımında A'nın kolonlarını ve B'nin karşılık gelen satırlarını örnekle.

> *"…sample the columns of A and sample [the rows of B]…"* — Strang, 0:53

Hepsini değil, olasılıkla seçilen birkaçını al, topla — sonuç AB'ye yakın olsun.

Builder Notu — Devasaya Tek Çare

Rastlantısal LA, dev veri matrislerinin (10^6+) tek pratik yoludur: tam çarpım/SVD imkânsızken örnekleme makul bir yaklaşım verir. Randomized SVD, sketching ve stokastik gradient descent (Ders 25) hep bu sezgiye dayanır.

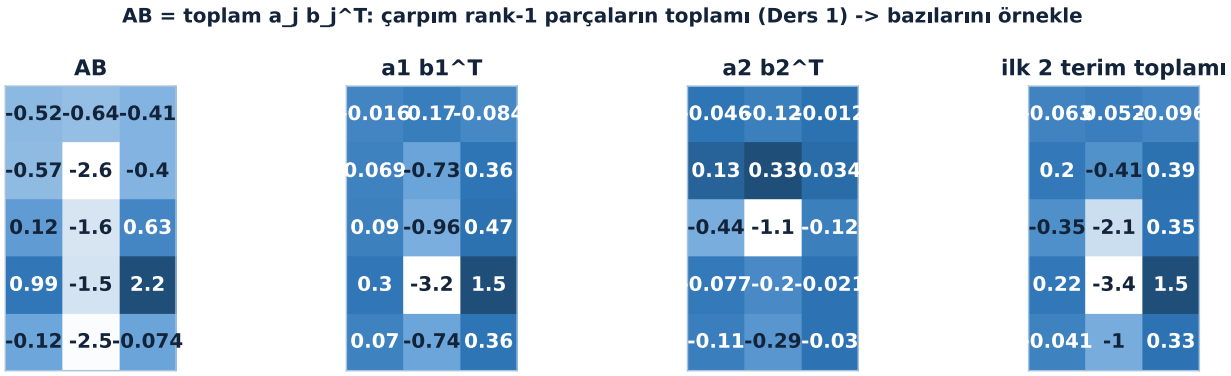
20.3 $AB = \sum$ Dış Çarpımı Örnekle

Ders 1'in anahtarı: matris çarpımı, dış çarpımların (rank-1 parçaların) toplamıdır.

$$AB = \sum_j a_j b_j^T$$

Burada $a_j = A$'nın j. kolonu, $b_j^T = B$'nin j. satırı. Tüm r terimi toplamak yerine, terimleri olasılıkla **örnekleriz** — her örnek bir rank-1 parça. Tek örnek AB'den çok uzaktır (sadece rank-1), ama doğru olasılıklarla ortalama AB'ye varır.

Aşağıdaki figür çarpımı rank-1 parçalarına ayırır: ilk iki terim ve toplamları AB'nin yapısını görünür kılar (Şekil 20.2).



Şekil 20.2: AB çarpımı rank-1 parçaların ($a_j b_j^T$) toplamıdır; rastlantısal çarpım bu parçalardan bazılarını örnekler.

> **Builder Notu — Rank-1 Parçalar**
>
> "Çarpım = rank-1 parçaların toplamı" görüşü, örneklemeyi mümkün kılar: her terim bağımsız bir katkıdır, bazılarını seçip ölçekleyerek toplamı kestirebilirsin. Aynı yapı SGD'de (gradyan = örnek gradyanların toplamı, mini-batch örnekle) ve Monte Carlo'da görülür.

20.4 Mean ve Variance (İstatistik Girişi)

Bu, kursun istatistikle ilk ciddi buluşması. İki kavram: **mean** (ortalama, beklenen değer) ve **variance** (yayılım). Olasılıklar 1'e toplanır; rastgele AB'nin mean'inin doğru AB olmasını umarız.

> *"…the mean value of the random AB is correct AB. But there will be a variance."* — Strang, 2:28

Ama variance sıfır değildir: hiçbir tek örnek doğru değil, sadece ortalamada doğru. İş, **olasılıkları variance'ı minimize edecek** şekilde seçmek (sonunda Lagrange çarpanlarıyla).

> *"…choose the probabilities that minimize the variance."* — Strang, 3:38

> *"…Lagrange multipliers will come in near the end."* — Strang, 4:32

> **Builder Notu — Mean ve Variance**
>
> Mean (yansızlık) + variance (gürültü) ayrımı, stokastik optimizasyonun kalbidir: SGD'de gradyan tahmini yansızdır (mean = gerçek gradyan) ama variance vardır; öğrenme hızı ve batch boyutu bu variance'ı yönetir. Bu ders, Stat 110'un mean/variance'ını matrise taşır.

20.5 Pratik Örnek: 1×2 Matris, Mean Doğru

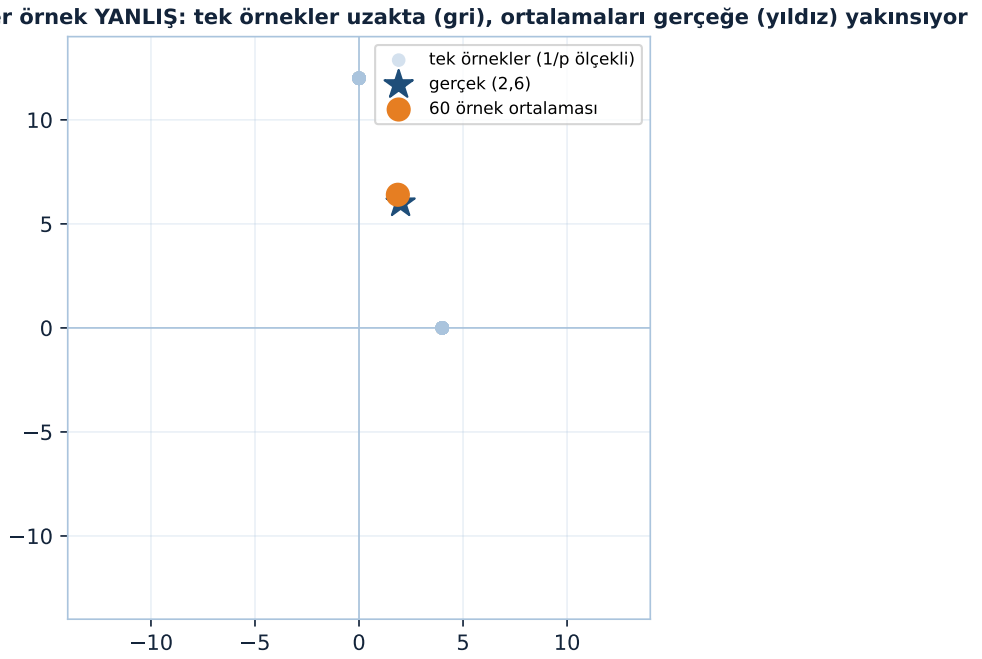
Strang basit bir örnekle ısınıyor: matris $[a \;\; b]$ (1×2). İki kolon var; her örnekte birini 1/2-1/2 olasılıkla seç, $s = 2$ örnek al, ortalamasını çıkar. Örnekler $(a, 0)$ veya $(0, b)$. Mean'i hesapla:

$$\mu = 2 \cdot \tfrac{1}{2}\,(a, b) = (a, b)$$

Her örneğin mean'i $(1/2)(a, b)$; iki örnekle çarpınca (a, b) — tam doğru.

> *"Mean is correct."* — Strang, 8:24

Süreç düzgün kurulduğu için (olasılık × ölçekleme) mean kesinlikle doğru çıkar. Aşağıdaki figür tek-örnek tahminlerin gerçekten uzakta olduğunu ama ortalamalarının hedefe yakınsadığını gösterir (Şekil 20.3).



Şekil 20.3: Mean DOĞRU ama her örnek YANLIŞ: tek örnekler (4,0) ve (0,12) uzakta, 60 örneğin ortalaması gerçeğe (2,6) yakınsıyor (yansız tahmin).

Builder Notu — Yansızlığın Güvencesi

Yansız tahmin edici (mean = gerçek değer), Monte Carlo ve SGD'nin temel güvencesidir: yeterince örnek alırsan ortalama doğruya gider. Ama "mean doğru" tek başına yetmez — variance da küçük olmalı (sonraki bölümler).

20.6 Variance ≠ 0 (Her Örnek Yanlış)

Mean doğru ama her örnek yanlıştır. $(a, 0)$ örneği gerçek (a, b)'den uzaktır; $(0, b)$ de öyle. Hiçbir tek örnek doğru değil — sadece ortalamada doğru.

Strang'in vurgusu: mean'in doğru olması, iyi cevap aldığın anlamına gelmez. -100 ile $+100$'ün ortalaması 0 olabilir (doğru mean) ama her örnek 100 birim uzakta. Variance bu uzaklığı ölçer: doğru mean + büyük variance = işe yaramaz tahminler. O yüzden asıl iş variance'ı küçültmektir.

> **Builder Notu — Doğru Mean Yetmez**
>
> "Doğru mean, büyük variance" tuzağı ML'de her yerde: yansız ama yüksek-varyanslı gradyan tahmini (küçük batch) eğitimi gürültülü yapar. Bias-variance dengesi, batch boyutu, momentum — hepsi variance'ı yönetmek için. Yansızlık gerekli ama yeterli değil.

20.7 İki Variance Formülü

Variance'ın iki eşdeğer formülü vardır. Birincisi mean'den uzaklığın karesi:

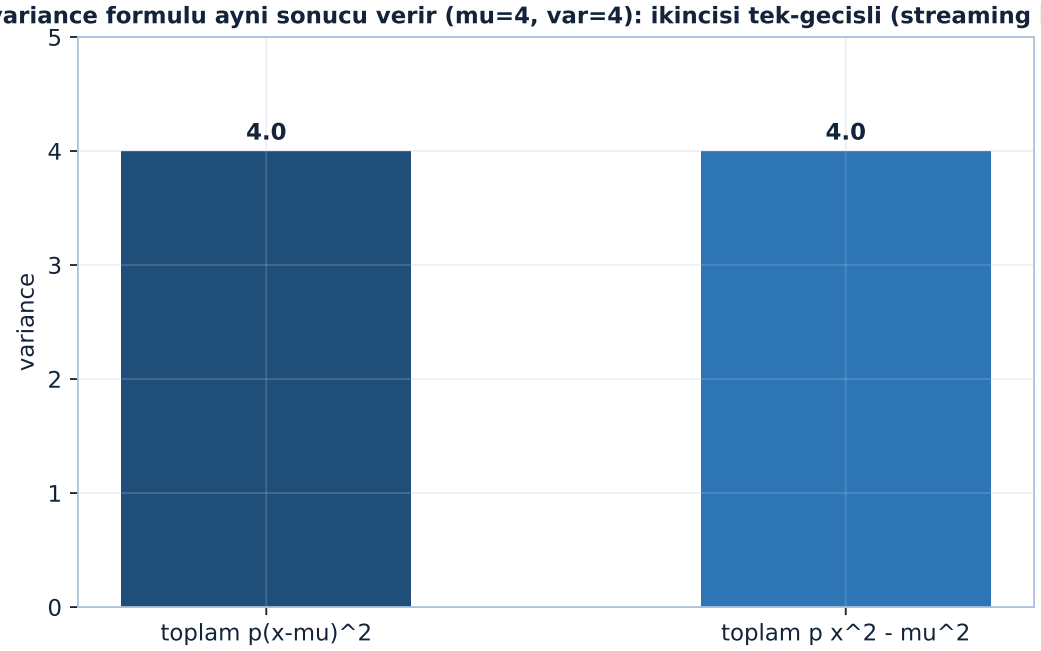
$$\sigma^2 = \sum_i p_i \, (x_i - \mu)^2$$

İkincisi mean'i sona bırakır (cebirle aynı):

$$\sigma^2 = \sum_i p_i \, x_i^2 - \mu^2$$

> *"…the second formula for the same quantity…"* — Strang, 19:34

İlk formül her çıktıdan mean'i çıkarır; ikincisi tüm karelerin ortalamasından μ^2'yi çıkarır. Hangisi kolaysa onu kullan — sonuç aynı. Aşağıdaki figür her iki formülün özdeş variance verdiğini doğrular (Şekil 20.4).



Şekil 20.4: Variance'ın iki formülü aynı sonucu verir: $\sigma^2 = \sum_i p_i (x_i - \mu)^2$ ile $\sigma^2 = \sum_i p_i x_i^2 - \mu^2$. İkincisi tek geçişlidir (streaming): mean ve variance'ı aynı taramada hesaplar.

> **Builder Notu — Tek Geçişte Variance**
>
> İkinci formül ($E[x^2] - (E[x])^2$) pratikte daha kullanışlıdır: tek geçişte hem $\sum x$ hem $\sum x^2$ biriktirip variance'ı hesaplarsın (online/streaming istatistik). Batch normalizasyonu ve running statistics bu formülü kullanır.

20.8 Norm-Kare Örnekleme

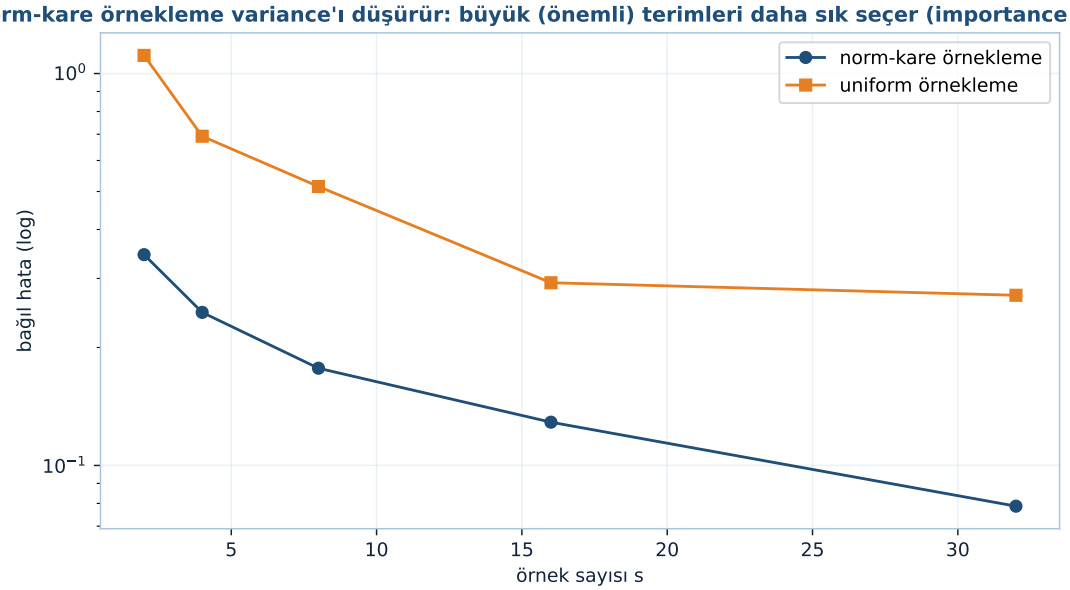
Eşit olasılık (1/2-1/2) en iyisi değil. Eğer bir kolon diğerinden çok büyükse, onu daha sık seçmek variance'ı düşürür. Optimal seçim **norm-kare** olasılıktır: bir terimin olasılığı, kolon ve satırının büyüklüğüyle orantılı.

> *"…norm squared probability."* — Strang, 18:02

$$p_j \propto \|a_j\|\,\|b_j\|, \qquad p_j = \frac{\|a_j\|\,\|b_j\|}{c}, \quad c = \sum_k \|a_k\|\,\|b_k\|$$

> *"…proportional to norm squared."* — Strang, 23:31

Bölü ile bölmek (c = toplam) olasılıkları 1'e toplar. Bir terimin $\|a_j\|\,\|b_j\|$ büyüklüğü diğerinin iki katıysa, onu orantılı olarak (iki kat) daha sık seçersin. Büyük (önemli) terimler daha sık örneklenir. ("Norm-kare" adı Strang'in sözlü ifadesinden gelir; optimal olasılık ise norm çarpımının birinci kuvvetiyle orantılıdır — Drineas-Kannan-Mahoney.) Aşağıdaki figür (FLAGSHIP) norm-kare örneklemenin uniform örneklemeye kıyasla bağıl hatayı her örnek sayısında düşürdüğünü gösterir (Şekil 20.5).



Şekil 20.5: Norm-kare örnekleme variance'ı düşürür: olasılıkları $p_j \propto \|a_j\|\,\|b_j\|$ alarak büyük (önemli) terimleri daha sık seçer (importance sampling). Bir kolon 10 kat büyük ölçeklendiğinde, norm-kare örnekleme (navi) her örnek sayısı s için uniform örneklemeden (turuncu) belirgin daha düşük bağıl hata verir.

> **Builder Notu — Önemliyi Sık Seç**
>
> Norm-kare örnekleme = importance sampling: önemli (büyük katkılı) terimleri daha sık seç. Randomized SVD, Monte Carlo integrasyonu ve önemli-örnekleme tabanlı SGD aynı ilkeyi kullanır — variance'ı, önemli olanı vurgulayarak düşür.

20.9 Yansız Tahmin Edici

Her örneği, seçim olasılığına bölerek ölçekleriz — bu, tahmini yansız (unbiased) yapar. s örnekle:

$$\widehat{AB} = \frac{1}{s}\sum_{i=1}^{s}\frac{a_{j_i}b_{j_i}^T}{p_{j_i}}$$

Mean'i hesapla: bir örnek j, p_j olasılıkla seçilir ve $(a_j b_j^T)/(s\,p_j)$ katkı verir; p_j'ler sadeleşir:

$$\text{mean} = s\cdot\sum_j p_j\cdot\frac{a_j b_j^T}{s\,p_j} = \sum_j a_j b_j^T = AB$$

p_j'lerin sadeleşmesi tam da yansızlığı verir — hangi olasılığı seçersen seç, mean daima AB. Olasılık seçimi yalnızca variance'ı etkiler, mean'i değil. Aşağıdaki figür örnek sayısı arttıkça koşan ortalamanın AB'ye yakınsadığını ($\sim 1/\sqrt{s}$) gösterir (Şekil 20.6).

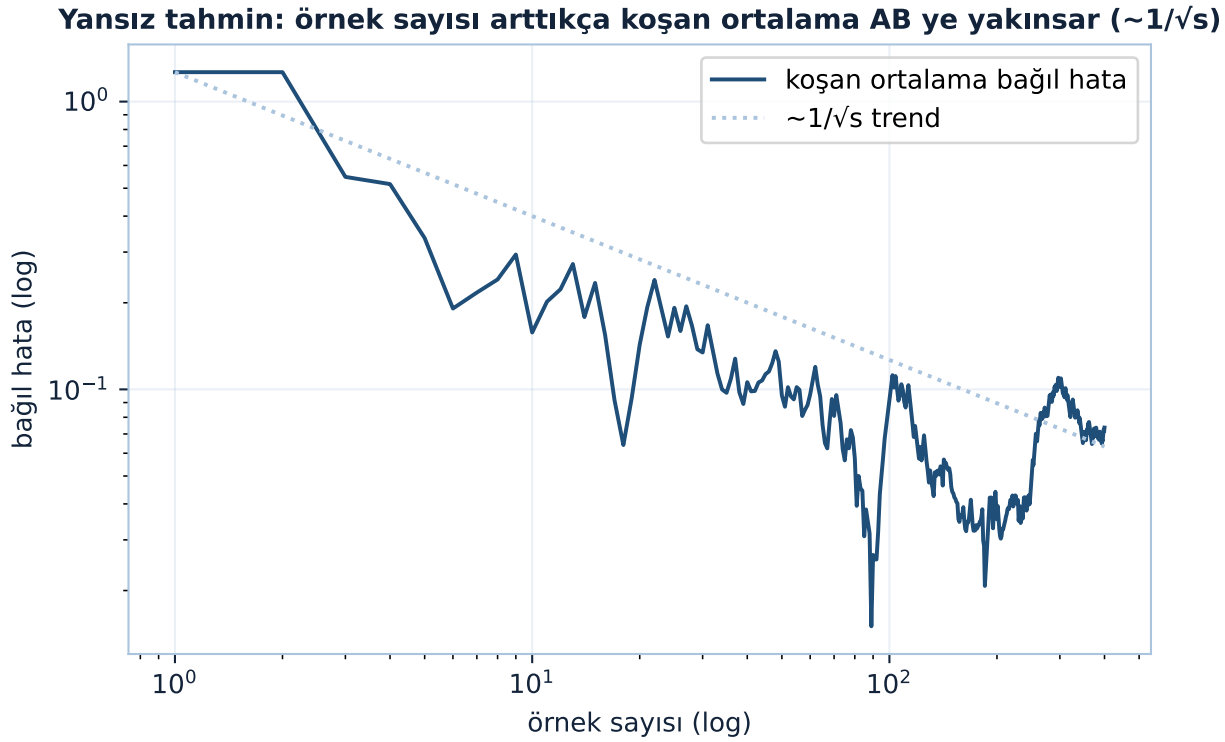
> **Builder Notu — Olasılığa Böl**
>
> "Olasılığa böl → yansız" hilesi (importance weighting), off-policy RL, önem-örnekleme ve düzeltilmiş Monte Carlo'nun temelidir: nadir seçilen terimleri büyük ağırlıkla telafi et, böylece beklenen değer doğru kalır. SGD'de mini-batch ortalaması aynı yansızlığı sağlar.

20.10 Variance'ı Minimize Et (Lagrange → Norm-Kare)

Mean her olasılıkla doğru olduğundan, asıl optimizasyon variance'ı küçültmektir. Variance formülü p_j'lere bağlıdır; "olasılıklar 1'e toplanır" kısıtı altında variance'ı minimize etmek bir **Lagrange çarpanı** problemidir. Çözüm: $p_j \propto \|a_j\|\,\|b_j\|$ — yani norm-kare örnekleme **optimaldir**.

$$\min_p\ \sigma^2(p)\quad \text{s.t.}\ \sum_j p_j = 1\quad\Rightarrow\quad p_j \propto \|a_j\|\,\|b_j\|$$

Variance, ancak AB rank-1 ise sıfır olur (tek terim, hep doğru); genelde pozitiftir ama norm-kare seçimle en küçüğe iner.



Şekil 20.6: Yansız tahmin: örnek sayısı arttıkça koşan ortalama AB'ye yakınsar (~1/√s); tek-örnek tahminler yanlış olsa da koşan ortalamaları gerçek çarpıma yaklaşır.

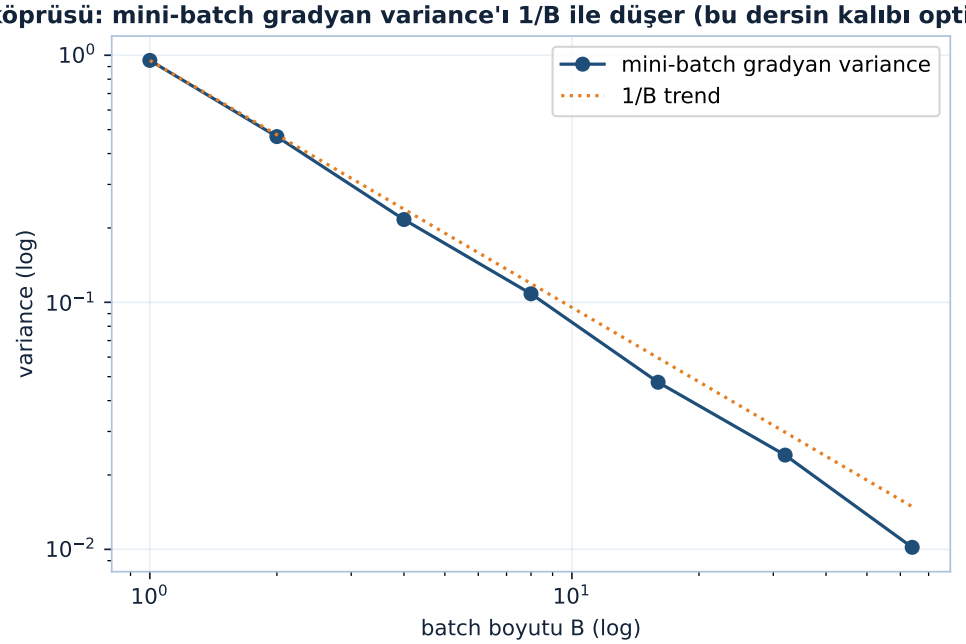
Builder Notu — Lagrange Optimali

Kısıt altında minimizasyon (Lagrange) → optimal örnekleme dağılımı: bu kalıp, varyans-azaltma tekniklerinin (control variates, stratified sampling) ve optimal importance sampling'in çekirdeğidir. "Önemliyi daha sık örnekle" sezgisinin matematiksel kanıtı.

20.11 SGD'ye Köprü ve Büyük Resim

Bu dersin kalıbı — **yansız tahmin + variance azaltma** — derin öğrenmenin merkezî algoritmasının (Ders 25, stokastik gradient descent) tam mantığıdır. SGD'de gerçek gradyan tüm verinin toplamıdır; biz bunu bir **mini-batch** örnekleyerek yaklaşık hesaplarız. Mini-batch gradyanı yansızdır (mean = gerçek gradyan) ama variance taşır; batch boyutu büyüdükçe variance düşer.

Aynı fikir randomized SVD'de (matrisi rastgele projeksiyonla küçült), Monte Carlo'da ve önem-örneklemede çalışır. Bu ders, "dev hesabı örnekleyerek yaklaşık yap, mean'i koru, variance'ı yönet" felsefesinin lineer cebir versiyonudur. Aşağıdaki figür (FLAGSHIP) mini-batch gradyan variance'ının batch boyutuyla $1/B$ düştüğünü doğrular (Şekil 20.7).



Şekil 20.7: SGD köprüsü: mini-batch gradyan tahmininin variance'ı batch boyutu B ile 1/B oranında düşer; bu dersin rastlantısal örnekleme kalıbı doğrudan optimizasyondaki mini-batch SGD'ye karşılık gelir.

Builder Notu — Mini-Batch'in Kalbi

SGD = bu dersin matris çarpımına uyguladığı örnekleme fikrinin optimizasyona taşınmış hâli: tam gradyan yerine örnek gradyan (yansız), variance'ı batch/öğrenme hızıyla yönet. Norm-kare örnekleme

bile SGD'de karşılığını bulur (importance sampling SGD — büyük gradyanlı örnekleri daha sık seç).

20.12 Bu Dersin Özeti

1. **Rastlantısal LA** — dev matrisler için örnekleme tabanlı yaklaşık hesap.
2. $AB = \sum a_j b_j^T$ **örnekle** — rank-1 parçaları olasılıkla seç, topla.
3. **Mean ve variance** — kursun ilk istatistik kavramları.
4. **Pratik örnek** — 1×2'de mean $= (a, b)$ doğru çıkar.
5. **Variance ≠ 0** — her örnek yanlış; doğru mean yetmez, variance da küçük olmalı.
6. **İki variance formülü** — $\sum_i p_i (x_i - \mu)^2 = \sum_i p_i x_i^2 - \mu^2$.
7. **Norm-kare örnekleme** — $p_j \propto \|a_j\| \, \|b_j\|$; büyük terimler daha sık.
8. **Yansız tahmin edici** — $(a_j b_j^T)/(s\, p_j)$; p_j sadeleşir → mean $= AB$.
9. **Variance'ı minimize** — Lagrange → norm-kare optimal.
10. **SGD köprüsü** — yansız tahmin + variance azaltma = stokastik optimizasyon.

! Tek Bir Cümle

Dev AB'yi, rank-1 parçalarını olasılıkla örnekleyerek yaklaşık hesaplarız; olasılığa bölen yansız tahmin edici mean'i daima AB yapar (seçimden bağımsız), norm-kare örnekleme ise variance'ı minimize eder — ve bu "yansız tahmin + variance azaltma" kalıbı stokastik gradient descent'in ta kendisidir.

20.13 Kontrol Soruları

i Soru 1: 1×2 matriste mean

$[2 \;\; 6]$ matrisini kolonları 1/2-1/2 olasılıkla, 2 örnekle örnekle. Tahmin edicinin mean'ini hesapla.
Cevap: Örnekler $(2, 0)$ veya $(0, 6)$, her biri 1/2. Yansız tahmin edici örneği olasılığa böler: $(2, 0)/(s \cdot \frac{1}{2})$ gibi. Bir örneğin mean'i $(\frac{1}{2})(2, 6)$; $s = 2$ örnekle çarp → mean $= (2, 6) =$ orijinal matris. Yansız: mean tam doğru, seçimden bağımsız.

i Soru 2: İki formülle variance

$[2 \;\; 6]$ örneğinde (eşit olasılık) variance'ı iki formülle de hesapla; aynı çıktığını gör.
Cevap: Çıktılar (her bileşeni mean'den uzaklık): $(2, 0)$ ve $(0, 6)$, mean $(1, 3)$. Formül 1: $\sigma^2 = \frac{1}{2}\|(2, 0) - (1, 3)\|^2 + \frac{1}{2}\|(0, 6) - (1, 3)\|^2 = \frac{1}{2}(1 + 9) + \frac{1}{2}(1 + 9) = 10$. Formül 2: $\sigma^2 = \frac{1}{2}\|(2, 0)\|^2 + \frac{1}{2}\|(0, 6)\|^2 - \|(1, 3)\|^2 = \frac{1}{2} \cdot 4 + \frac{1}{2} \cdot 36 - 10 = 20 - 10 = 10$. İkisi de 10 ✓.

i Soru 3: Variance ne zaman sıfır

Rastlantısal matris çarpımında variance ne zaman tam sıfır olur?
Cevap: AB rank-1 olduğunda. O zaman seçilecek tek bir $a_j b_j^T$ terimi vardır; her örnek aynı (doğru) sonucu verir, sapma yok → variance $= 0$. Rank > 1 ise her örnek farklı bir rank-1 parça seçer (hepsi tek

başına yanlış), ortalamada doğru ama variance pozitif. Yani variance, AB'nin "kaç bağımsız parçadan oluştuğunu" yansıtır.

i Soru 4: SGD bağlantısı

Bu dersin "yansız tahmin + variance azaltma" kalıbı SGD ile nasıl bağlantılı?
Cevap: SGD'de gerçek gradyan = tüm örneklerin gradyan toplamı (dev). Mini-batch gradyanı bunun örneklemesidir: yansız (mean = gerçek gradyan) ama variance taşır. Batch boyutu ↑ → variance ↓ (tıpkı s örnek ↑ → variance ↓). Norm-kare örnekleme bile karşılık bulur: büyük-gradyanlı örnekleri daha sık seçmek (importance sampling SGD) variance'ı düşürür. Aynı matematik, optimizasyona taşınmış hâli.

20.14 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. $[3\ \ 9]$ (1×2) matrisini kolonları p_1, p_2 olasılıkla örnekle. Yansız tahmin edicinin mean'inin her zaman $(3, 9)$ olduğunu (olasılıklardan bağımsız) göster.

Egzersiz 2. A (m×2) ve B (2×p) için norm-kare olasılıkları yaz: $p_1 \propto \|a_1\|\,\|b_1\|, p_2 \propto \|a_2\|\,\|b_2\|$. $\|a_1\|\|b_1\| = 1$ ve $\|a_2\|\|b_2\| = 3$ ise normalleştirilmiş p_1, p_2 nedir?

Egzersiz 3. AB hangi durumda rank-1'dir ve bu neden variance $= 0$ verir? (İpucu: A'nın tek kolonu veya B'nin tek satırı sıfırdan farklıysa ne olur?)

Egzersiz 4. Python ile rastlantısal matris çarpımı:

```
import numpy as np
rng = np.random.default_rng(0)

A = rng.standard_normal((100, 50))
B = rng.standard_normal((50, 80))
AB = A @ B

# norm-kare olasılıklar
w = np.linalg.norm(A, axis=0) * np.linalg.norm(B, axis=1)
p = w / w.sum()

s = 20   # örnek sayısı
est = np.zeros_like(AB)
for _ in range(s):
    j = rng.choice(50, p=p)
    est += np.outer(A[:, j], B[j, :]) / (s * p[j])
print("bağıl hata:", np.linalg.norm(est - AB) / np.linalg.norm(AB))
```

Egzersiz 5. (Ders 14 habercisi.) Ders 14 düşük-rank değişimleri işler: A tersini biliyorsan, A'ya rank-1 bir değişim $(A + uv^T)$ eklediğinde yeni tersi baştan hesaplamadan bulabilir misin? Bu, Sherman-Morrison formülüdür. 2×2 bir A ve $u = v = (1, 0)^T$ için $(A + uv^T)$'yi yaz.

20.15 Sonraki Ders İçin Hazırlık

Ders 14: A ve Tersinde Düşük-Rank Değişimler

Ders 13'te örnekleme ve rank-1 parçaları gördük. Ders 14, A'ya rank-1 (veya düşük-rank) bir değişim eklendiğinde tersinin nasıl ucuzca güncellendiğini işler.

- Sherman-Morrison: $(A + uv^T)^{-1}$ formülü
- Sherman-Morrison-Woodbury: düşük-rank genelleme
- Neden kritik: Kalman filtresi, ardışık güncellemeler, online öğrenme
- Matris determinant lemması

> ⚠ Ders 14 Öncesi Yapılacak
>
> - Bu dersin egzersizlerini çöz, özellikle Egzersiz 5'i (rank-1 değişim).
> - Python'da bir matrise rank-1 ekleyip tersini doğrudan vs Sherman-Morrison ile hesapla, karşılaştır.
> - Ana cümleyi tekrar oku: "Yansız tahmin + variance azaltma; norm-kare örnekleme = importance sampling = SGD'nin atası."

20.16 Anahtar Kavramlar (Cheat Sheet)

Kavram	Tanım	Strang'de
Rastlantısal LA	Dev matris için örnekleme tabanlı yaklaşık hesap	0m33
$AB = \sum$ **dış çarpım örnekle**	Rank-1 parçaları (kolon×satır) olasılıkla seç	0m53
Mean ve variance	Mean doğru olmalı ama variance da var	2m28
Mean doğru (yansız)	Her olasılıkla mean $= AB$; seçimden bağımsız	8m24
İki variance formülü	$\sum p(x-\mu)^2 = \sum p\,x^2 - \mu^2$	19m34
Norm-kare olasılık	Önemli (büyük) terimleri daha sık seç	18m02
$p_j \propto \|a_j\|\,\|b_j\|$	Norm-kare örnekleme; optimal olasılık	23m31
Variance minimize	Lagrange kısıtlı min → norm-kare optimal	3m38
Lagrange çarpanı	$\sum_j p_j = 1$ kısıtı altında variance minimizasyonu	4m32

20.17 ML Bağlantıları Özeti

1. **Yansız tahmin + variance** → stokastik gradient descent (Ders 25); mini-batch gradyanı yansız, variance batch'le düşer.
2. $AB = \sum$ **rank-1 örnekle** → randomized SVD, sketching; dev çarpımı ucuzlatma.
3. **Norm-kare örnekleme** → importance sampling; önemli terimleri/örnekleri vurgula.
4. **İki variance formülü** ($E[x^2] - \mu^2$) → online/streaming istatistik, batch normalizasyon.
5. **Yansız tahmin edici (olasılığa böl)** → importance weighting, off-policy RL.
6. **Lagrange ile optimal örnekleme** → varyans-azaltma; control variates, stratified sampling.
7. **Mean** $\leftrightarrow$ **variance ayrımı** → bias-variance dengesi; eğitim gürültüsü yönetimi.

> **!** Tek Şey
>
> Eğer bu dersten tek bir şey alıp gidersen: **Dev bir AB çarpımını, rank-1 parçalarını ($a_j b_j^T$) olasılıkla örnekleyip ölçekleyerek yaklaşık hesaplarsın. Olasılığa bölen tahmin edici yansızdır — mean daima AB, seçilen olasılıktan bağımsız; olasılık seçimi yalnız variance'ı etkiler ve norm-kare örnekleme (büyük terimleri daha sık) onu minimize eder. Bu "yansız tahmin + variance azaltma" kalıbı, stokastik gradient descent ve tüm büyük-ölçek rastlantısal yöntemlerin temelidir.**

21 A ve Tersinde Düşük-Rank Değişimler

Sherman-Morrison-Woodbury, recursive least squares, Kalman filtresi

> **Bölüm bilgisi**
>
> **Video:** MIT 18.065 — Low Rank Changes in A and Its Inverse · **OCW:** Lecture 14 · **Okuma süresi:** ≈35 dk · **Eğitmen:** Gilbert Strang · **Önkoşul:** Ders 13 (rastlantısal çarpım, rank-1 parçalar).

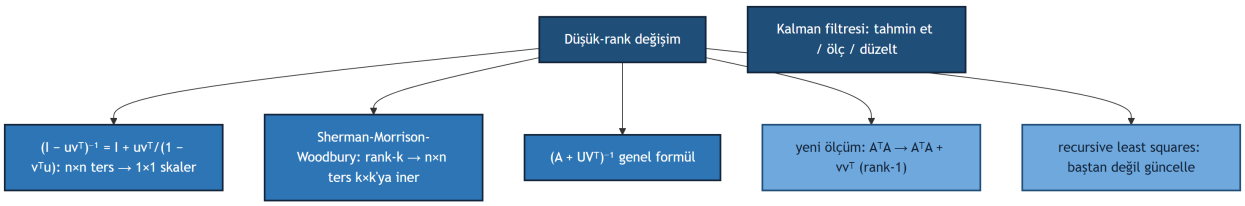
21.1 Bu Derste Ne Var?

Düşük-rank matrisler bölümü başlıyor. Bir matrise rank-1 (veya rank-k) bir değişim eklersen, tersini baştan hesaplamadan **ucuza güncelleyebilirsin** — Sherman-Morrison-Woodbury formülü.

Üç temel fikir:

1. **Rank-1 değişim → rank-1 ters değişimi:** $(I - uv^T)^{-1} = I + \frac{uv^T}{1-v^Tu}$; $n \times n$ ters, 1×1 tersle hesaplanır.
2. **Sherman-Morrison-Woodbury** — rank-k genelleme: $n \times n$ ters, $k \times k$ tersle; $A + UV^T$ için genel formül.
3. **Recursive least squares + Kalman filtresi** — yeni ölçüm geldiğinde A^TA rank-1 değişir; her şeyi yeniden hesaplama, güncelle.

Aşağıdaki kavram haritası bu dersin merkezindeki düşük-rank değişimden ana fikirlere uzanan dalları, ve ayrı bir düğüm olarak Kalman filtresinin tahmin et / ölç / düzelt döngüsünü gösterir (Şekil 21.1).



Şekil 21.1: Ders 14 kavram haritası: düşük-rank değişim merkezinden ana fikirlere dallar — $(I - uv^T)^{-1} = I + uv^T/(1 - v^Tu)$ ile n×n ters 1×1 skalere iner, Sherman-Morrison-Woodbury bunu rank-k'ya genelleyip n×n tersi k×k terse indirir, $(A + UV^T)^{-1}$ genel formülü A'nın küçük değişimini A^{-1}'in küçük değişimine bağlar, yeni ölçüm $A^TA \to A^TA + vv^T$ (rank-1) büyütür ve recursive least squares baştan hesaplamak yerine çözümü günceller; ayrı düğüm Kalman filtresi tahmin et / ölç / düzelt döngüsünü gösterir.

> *"today starts a new chapter about low-rank matrices."* — Strang, 0:27

> **Builder Notu — Bir Kez Kur Sonra Güncelle**
>
> - **SMW = online güncelleme:** yeni veri geldikçe modeli sıfırdan eğitme; ters/çözümü rank-1 güncelle — recursive least squares, online öğrenme.
> - **Kalman filtresi** — dinamik least squares; kovaryans matrisi + durum denklemi; navigasyon, takip, sensör füzyonu.
> - $n \times n \to k \times k$ **indirgeme** — büyük tersi küçük tersle hesapla; Gauss süreçlerinde ve düşük-rank kovaryans güncellemelerinde kritik.
> - **Rank-1 =** uv^T **(kolon×satır)** — Ders 1'in dış-çarpım görüşü; "küçük" demek sayılar küçük değil, rank küçük demek.
>
> Tek cümle: A'ya düşük-rank bir değişim eklemek tersini de düşük-rank değiştirir; Sherman-Morrison-Woodbury bu güncellemeyi büyük tersi yeniden hesaplamadan verir — recursive least squares ve Kalman filtresinin temeli.

21.2 Düşük-Rank Matrisler (Yeni Bölüm)

Yeni bir bölüm: düşük-rank matrisler. İki tür: **gerçekten** düşük-rank (uv^T gibi, rank 1) ve **yaklaşık** düşük-rank (tekil değerleri hızla düşen — Ders 17).

> *"today starts a new chapter about low-rank matrices."* — Strang, 0:27

Dikkat: "küçük" demek girdiler küçük demek değil — uv^T tümü-milyonlar matrisi bile olabilir. Önemli olan **rankın** küçük olmasıdır. Bu derste soru: A'ya rank-1 bir değişim eklersem tersi nasıl değişir?

Aşağıdaki figür bu ayrımı görselleştirir: girdileri büyük olan bir uv^T matrisinin yine de rankı 1'dir, çünkü tekil değerlerden sadece biri sıfırdan farklıdır (Şekil 21.2).

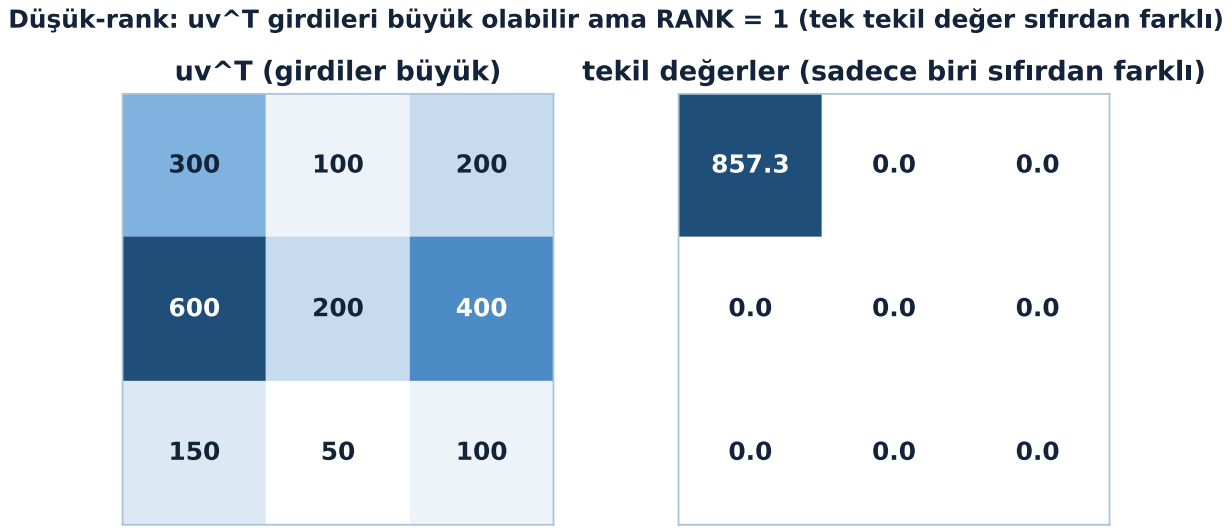
> **Builder Notu — Rank Küçük Girdiler Değil**
>
> "Rank küçük, girdiler değil" ayrımı ML'de kritik: bir ağırlık matrisi büyük değerler içerse de düşük-rank olabilir (LoRA tam bunu kullanır — ΔW düşük-rank). Düşük-rank yapı, az parametreyle çok bilgi taşır.

21.3 Identity'nin Rank-1 Perturbasyonu

En basit durum: birim matrise rank-1 ekle, tersini bul. Ünlü formül (sinyal işlemede "matris ters çevirme formülü" de denir):

> *"I do a rank 1 perturbation. And I want to know, what's the inverse?"* — Strang, 3:00

$$(I - uv^T)^{-1} = I + \frac{uv^T}{1 - v^T u}$$



Şekil 21.2: Düşük-rank yapı: dış çarpım uv^{T} girdileri büyük olabilir, ama matrisin rankı 1'dir — tekil değerlerden sadece biri sıfırdan farklıdır.

u, v kolon vektörler; uv^T bir rank-1 matris (kolon×satır). Sağ tarafta $v^T u$ bir **sayıdır** (1×1). Yani bu formül, $n \times n$ bir matrisin tersini 1×1 bir tersle (sayıyla bölme) verir.

> *"It's computing an n by n inverse in terms of a 1 by 1 inverse."* — Strang, 6:12

> **Builder Notu — n×n'i Bölmeye İndir**
>
> $n \times n$ tersi 1×1'e indirgemek devasa tasarruf: $O(n^3)$ yerine $O(n^2)$. Gauss süreçleri, online regresyon ve Kalman filtreleri her yeni veri için bu formülle güncellenir — baştan ters almazlar.

21.4 Rank-1 Değişim → Rank-1 Ters

Formülün güzelliği: hem giriş ($I - uv^T$) hem çıkış ($I + uv^T$/sayı) rank-1 değişimdir. Genel kural:

> *"I change [a matrix] by rank 1, I change its inverse by rank 1."* — Strang, 5:20

Bir matrisi rank-1 değiştirirsen, tersi de tam olarak rank-1 değişir — hiç bariz değil ama formül bunu kanıtlar. Bu kural identity için doğru; birazdan herhangi bir A için de geçerli olacak (Sherman-Morrison-Woodbury).

> **Builder Notu — Küçük Değişim Küçük Ters**
>
> "Rank-1 değişim → rank-1 ters değişimi" güvencesi, artımlı (incremental) algoritmaların temelidir: her güncelleme küçük (rank-1) olduğundan tersi de küçük güncellenir. Bu, milyonlarca güncelleme yapan online sistemlerde (öneri, takip) hesabı uygulanabilir kılar.

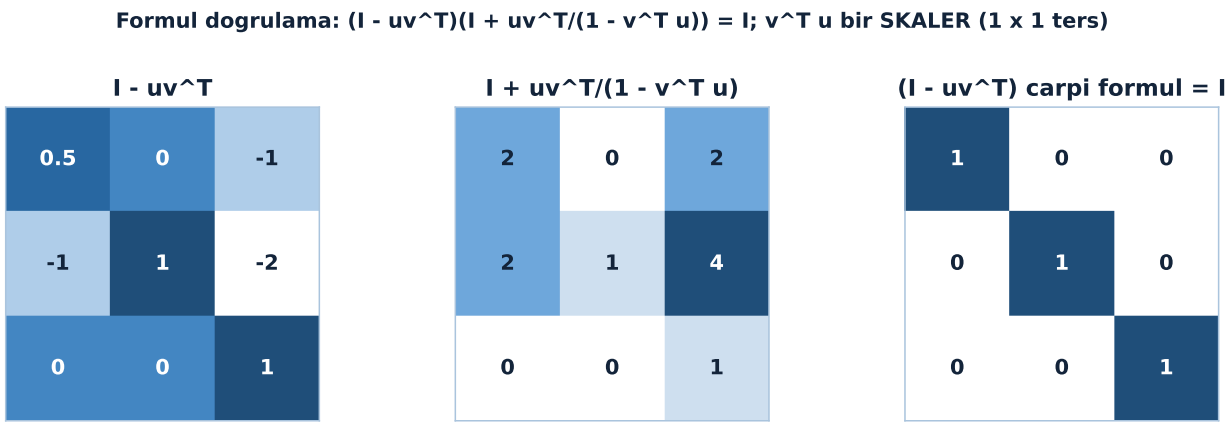
21.5 Formülü Doğrulama

İnanmıyorsan çarp: $(I - uv^T)$ ile $(I + uv^T/(1 - v^T u))$ çarpımı I vermeli.

$$(I - uv^T)\left(I + \frac{uv^T}{1 - v^T u}\right) = I + \frac{uv^T}{1 - v^T u} - uv^T - \frac{uv^T uv^T}{1 - v^T u}$$

Kilit nokta: $v^T u$ bir skaler, yani $uv^T uv^T = u(v^T u)v^T = (v^T u) \cdot uv^T$. Terimleri toplayınca uv^T payı $(1 - v^T u)$ ile sadeleşir ve geriye I kalır. Skalerleri vektörlerin arasından çekip çıkarabilmek tüm ispatın anahtarı.

Aşağıdaki figür formülü sayısal olarak doğrular: $I - uv^T$ ile kapalı-form ters çarpılınca tam olarak birim matrisi çıkar — ortadaki $v^T u$ bir skaler olduğundan ters 1×1'e iner (Şekil 21.3).



Şekil 21.3: $(I - uv^T)^{-1} = I + uv^T/(1 - v^T u)$ formülünün doğrulanması: soldaki $I - uv^T$ matrisi, ortadaki kapalı-form ters ile çarpılınca (sağda) tam olarak birim matrisi verir. Burada $v^T u$ bir SKALER olduğundan $n \times n$ ters problemi 1×1 tersine iner.

> **Builder Notu — Skaleri Dışarı Çek**
>
> "$v^T u$ skalerdir, dışarı çekebilirsin" hamlesi, Ders 1'in kolon×satır (uv^T) vs satır×kolon ($v^T u$) ayrımının doğrudan meyvesi. Dış-çarpım matris, iç-çarpım sayı — bu ayrımı içselleştirmek bu dersin tüm cebirini şeffaflaştırır.

21.6 Rank-k Genelleme: Sherman-Morrison-Woodbury

Şimdi u, v yerine $n \times k$ matrisler U, V. $(I - UV^T)$ bir rank-k değişim. Ters:

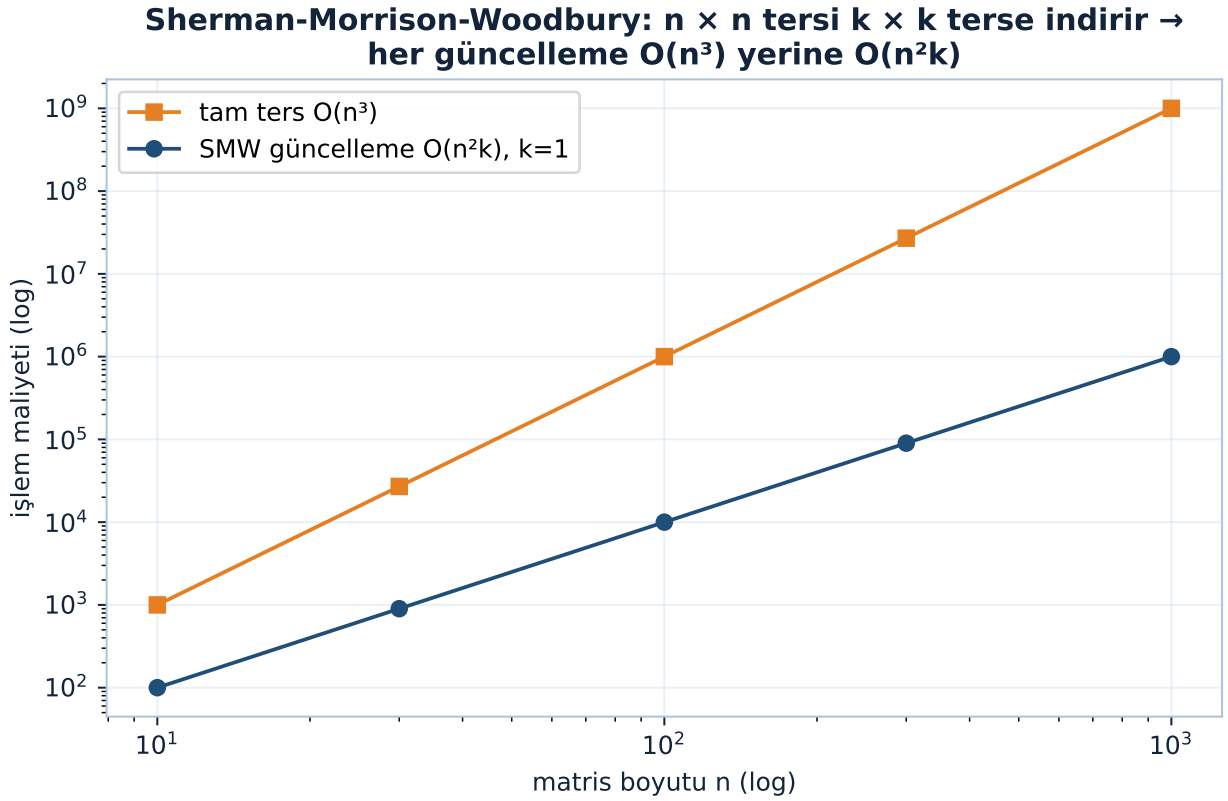
$$(I - UV^T)^{-1} = I + U(I_k - V^T U)^{-1} V^T$$

Büyük fark: ortadaki $(I_k - V^T U)$ artık $k \times k$ bir matris (sayı değil). Yani $n \times n$ bir tersi, $k \times k$ bir tersle hesaplıyoruz. k küçükse (örn. $k = 1, 2, 10$) bu devasa tasarruf. Bu, ünlü üç ismin formülü:

"…Sherman, Morrison, Woodbury…" — Strang, 14:03

Sherman ve Morrison rank-1 durumunu, Woodbury rank-k genellemesini verdi; literatürde üçü birlikte anılır.

Aşağıdaki flagship figür neden önemli olduğunu gösterir: tam ters $O(n^3)$ ölçeklenirken SMW güncellemesi $O(n^2k)$ kalır; n büyüdükçe iki maliyet eğrisi arasındaki uçurum açılır (Şekil 21.4).



Şekil 21.4: Sherman-Morrison-Woodbury maliyet karşılaştırması (log-log): tam matris tersi $O(n^3)$ ölçeklenir (turuncu, eğim 3), düşük-rank SMW güncellemesi ise yalnızca $O(n^2k)$'dır (lacivert, $k = 1$, eğim 2). n büyüdükçe iki eğri arasındaki uçurum açılır — $n \times n$ matrisin tersini baştan hesaplamak yerine SMW güncellemenin maliyeti küçük $k \times k$ tersine iner. Bu, recursive least squares ve Kalman filtresinin neden her yeni ölçümde sıfırdan çözmeden güncelleyebildiğini açıklar.

> **Builder Notu — k×k'ya İndirge**
>
> SMW'nin ML'deki en büyük müşterisi Gauss süreçleri ve kernel yöntemleri: n veri noktası için $n \times n$ kernel tersini, düşük-rank (k bileşenli) bir yaklaşımla $k \times k$ terse indirger. Nyström yöntemi tam bu mantık. Büyük kovaryans matrislerini tersini almadan yönetmenin standart yolu.

21.7 Genel A için: $(A + UV^T)^{-1}$

Identity yerine herhangi bir tersinir A: A'ya rank-k bir değişim eklenince ters şöyle güncellenir:

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1}$$

A^{-1}'i bir kez hesaplamışsan (veya A'yla çözmek ucuzsa), her rank-k güncelleme için baştan ters almazsın: sadece $k \times k$ bir ters ekstra. İşte tüm pratik gücün kaynağı bu.

> Builder Notu — Genel A'ya Geçiş
>
> Bu formül online sistemlerin kalbi: A^{-1} bir kez kurulur, sonra her yeni veri (UV^T) geldiğinde ters $O(n^2k)$ ile güncellenir — $O(n^3)$ değil. Recommendation sistemleri, adaptive filtreler, recursive least squares ve Kalman filtresinin hepsi bu güncellemeyi döngü içinde milyonlarca kez çağırır.

21.8 Kullanım 1: Değişmiş Sistemi Çözmek

İlk pratik kullanım: $(A - uv^T)x = b$ denklemini, A^{-1}'i (veya A ile çözmeyi) zaten biliyorken çöz. Yeni sistemi baştan kurmazsın:

$$x = (A - uv^T)^{-1}b = A^{-1}b + \frac{A^{-1}u\,(v^T A^{-1}b)}{1 - v^T A^{-1}u}$$

İki "eski" çözüm yeter: A ile b'yi çöz ($A^{-1}b$) ve A ile u'yu çöz ($A^{-1}u$). Gerisi skalerle çarpma. Yani bir LU ayrıştırması elindeyse, rank-1 değişen sistemi neredeyse bedavaya çözersin.

> Builder Notu — Eski Çözümü Kullan
>
> "A'yı bir kez ayrıştır, sonra rank-1 değişimleri ucuza çöz" Phase 1 18.06 Ders 4'ün ($A = LU$) doğrudan devamı: LU pahalı ($O(n^3)$) ama bir kez yapılır; her yeni sağ-taraf veya rank-1 perturbasyon $O(n^2)$. Bu, simülasyon ve optimizasyonda matris tekrar tekrar küçük değişirken kullanılır.

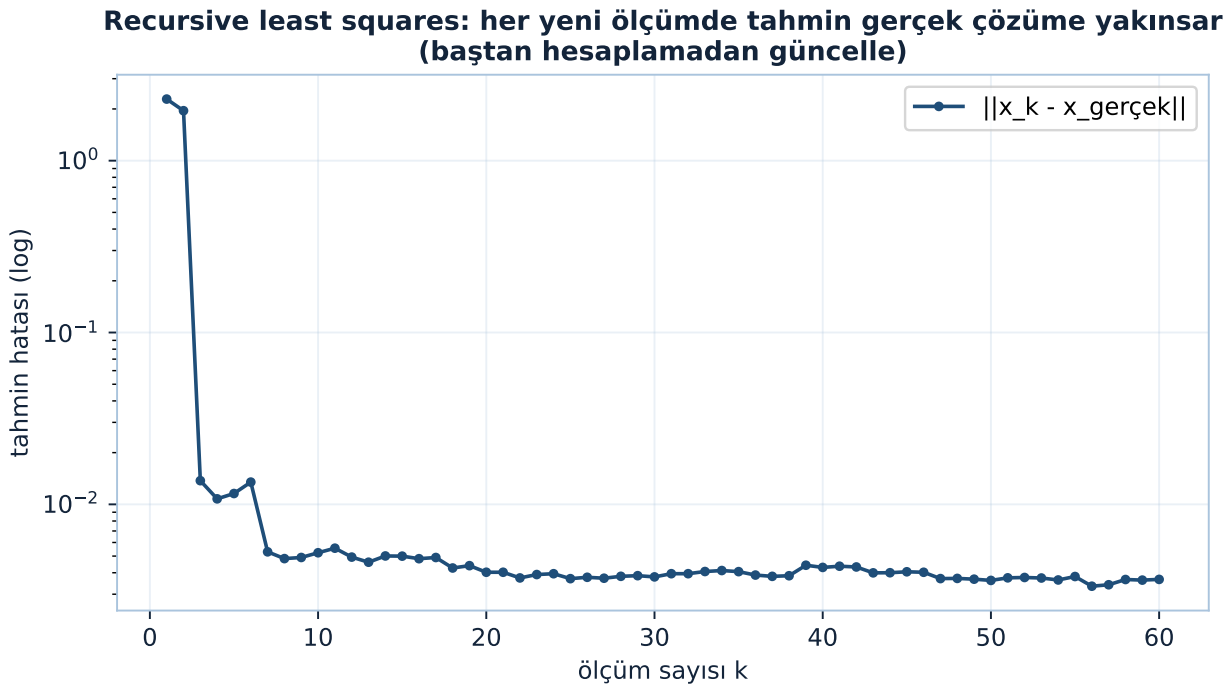
21.9 Kullanım 2: Recursive Least Squares (Yeni Ölçüm)

En güzel kullanım: least squares yaparken **yeni bir veri noktası** gelir. Tüm $A^T A$'yı baştan kurup tersini almak yerine, çözümü güncelle:

> *"…a new measurement…"* — Strang, 19:31

Yeni ölçüm = A'ya yeni bir satır eklemek. Bu, $A^T A$'yı rank-1 değiştirir (birazdan §9). SMW devreye girer ve eski çözümü düzeltme terimi ile güncelleriz. Buna **recursive least squares** denir:

> *"This is really recursive least squares."* — Strang, 28:22



Şekil 21.5: Recursive least squares (RLS): satır satır gelen ölçümlerle tahmin, baştan hesaplamadan rank-1 güncellemeyle gerçek çözüme yakınsar. Hata logaritmik eksende hızla düşer.

Her yeni veride sıfırdan başlamazsın; mevcut tahmini yeni ölçümle düzeltirsin. Online öğrenmenin tam tanımı.

Aşağıdaki figür bunu canlandırır: satır satır gelen ölçümlerle RLS tahmini, baştan hesaplamadan rank-1 güncellemeyle gerçek çözüme yakınsar; hata logaritmik eksende hızla düşer (Şekil 21.5).

> Builder Notu — Yeni Ölçüm Geldi
>
> Recursive least squares = SGD'nin atası. Modern ML'de her mini-batch ağırlığı günceller (eski ağırlık + düzeltme); RLS de her ölçümde tahmini günceller. Fark: RLS kapalı-form optimal düzeltme verir (ikinci dereceden bilgiyle), SGD birinci-derece adım atar. Adaptive filtreler (gürültü engelleme, eko iptali) hâlâ RLS kullanır.

21.10 A^TA'da Rank-1 Değişim

Neden least squares'e yeni satır eklemek "rank-1 değişim"? A'nın altına yeni bir satır v^T eklersen, normal denklemin matrisi A^TA şöyle değişir:

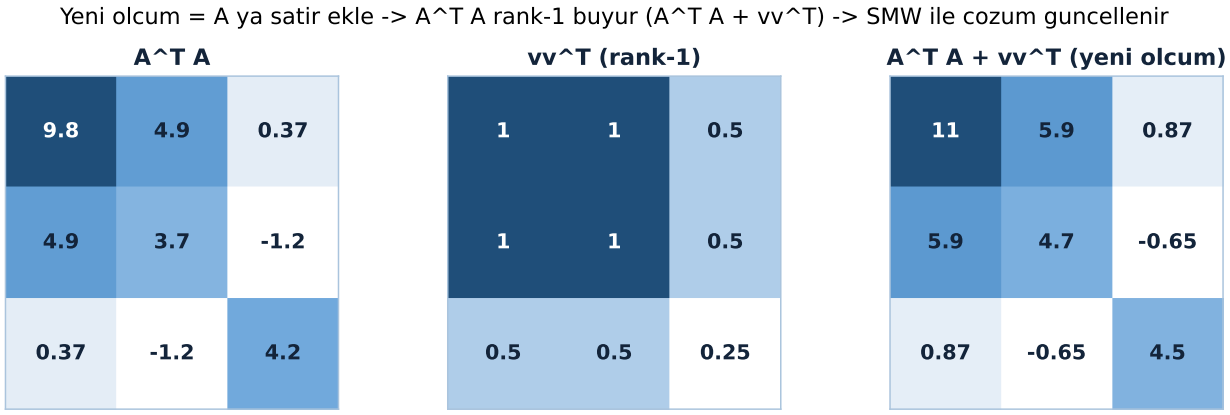
$$A_{\text{new}}^T A_{\text{new}} = A^TA + vv^T$$

vv^T bir rank-1 matris (kolon×satır, Ders 1). Yani yeni ölçüm, A^TA'yı tam olarak rank-1 büyütür:

> *"It's a rank 1 change in A transpose A."* — Strang, 26:06

İşte SMW'nin neden least squares'in doğal ortağı olduğu buradan: A^TA rank-1 değişiyor → tersi rank-1 güncelleniyor → çözüm ucuza düzeltiliyor.

Aşağıdaki figür güncellemeyi parçalarına ayırır: A^TA, eklenen vv^T (rank-1) ve toplamları olan yeni $A^TA + vv^T$ (Şekil 21.6).



Şekil 21.6: Yeni ölçüm = A'ya satır ekle → A^TA rank-1 büyür ($A^TA + vv^T$) → SMW ile çözüm güncellenir.

> **Builder Notu — A^TA Rank-1 Büyür**
>
> A^TA+vv^T güncellemesi Phase 1 18.06 Ders 16'nın (en küçük kareler, normal denklem $A^TA\hat{x} = A^Tb$) dinamik versiyonu. Statik least squares tüm veriyi bir kerede görür; recursive versiyon veriyi akış (stream) olarak işler. Büyük veri / gerçek zamanlı sistemlerde veri belleğe sığmadığında tek seçenek budur.

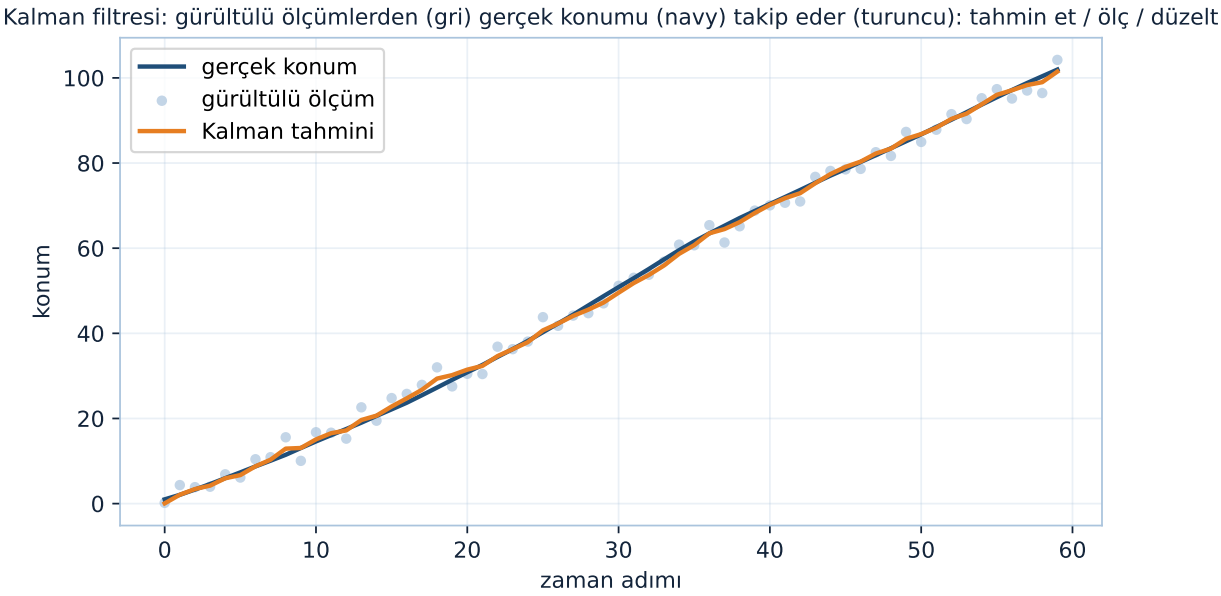
21.11 Kalman Filtresi: Dinamik Least Squares

Recursive least squares'i bir adım ileri taşı: ölçülen şey de zamanla **hareket ediyorsa** (uçak uçuyor, fiyat değişiyor). İşte Kalman filtresi:

> *"…the Kalman filter…"* — Strang, 27:58

Kalman filtresi iki şeyi birden yönetir: (1) yeni ölçümle tahmini düzeltmek (recursive least squares), (2) bir **durum denklemi** ile sistemin nasıl ilerlediğini öngörmek. İki ek malzeme: ölçümlerin güvenilirliğini tartan bir **kovaryans matrisi** ve durumun nasıl evrildiğini söyleyen dinamik denklem. Çekirdekte yine SMW: kovaryans matrisi her adımda düşük-rank güncellenir.

Aşağıdaki flagship figür çalışan bir Kalman filtresini gösterir: gürültülü ölçümlerden (gri) gerçek konumu (navy) takip eder (turuncu) — tahmin et / ölç / düzelt döngüsü (Şekil 21.7).



Şekil 21.7: Kalman filtresi: gürültülü ölçümlerden (gri) gerçek konumu (navy) takip eder (turuncu) — tahmin et / ölç / düzelt.

> **Builder Notu — Tahmin Et Ölç Düzelt**
>
> Kalman filtresi GPS, otopilot, robot navigasyonu ve sensör füzyonunun temelidir — "tahmin et, ölç, düzelt" döngüsü. ML köprüsü: Kalman = Gauss varsayımı altında optimal Bayesçi güncelleme; derin öğrenmedeki belirsizlik tahmini (uncertainty estimation) ve durum-uzay modelleri (S4, Mamba) bu fikrin doğrudan torunları. SMW olmadan kovaryans güncellemesi her adımda $O(n^3)$ olur — Kalman'ı gerçek zamanlı yapan tam da bu rank-k indirgemesidir.

21.12 Bu Dersin Özeti

1. **Düşük-rank değişim → düşük-rank ters değişimi.** A'yı rank-1 (rank-k) değiştirirsen tersi de rank-1 (rank-k) değişir.
2. **Rank-1 formülü:** $(I - uv^T)^{-1} = I + uv^T/(1 - v^T u)$; $n \times n$ ters, 1×1 (skaler) tersle.
3. **Sherman-Morrison-Woodbury:** $(I - UV^T)^{-1} = I + U(I_k - V^T U)^{-1} V^T$; $n \times n$ ters, $k \times k$ tersle. Genel A: $(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1} U)^{-1} V^T A^{-1}$.
4. **İspatın anahtarı:** $v^T u$ skalerdir, vektörlerin arasından dışarı çekilir (iç-çarpım sayı, dış-çarpım matris — Ders 1).
5. **Kullanım 1:** rank-1 değişmiş sistemi $(A - uv^T)x = b$, A çözümü elindeyken ucuza çöz.
6. **Kullanım 2 (recursive least squares):** yeni ölçüm $A^T A$'yı vv^T ile rank-1 büyütür; çözümü baştan değil, düzeltme ile güncelle.
7. **Kalman filtresi:** dinamik least squares; durum denklemi + kovaryans matrisi; her adımda SMW ile düşük-rank güncelleme.

! Tek Bir Cümle

A'ya düşük-rank bir değişim eklemek tersini de düşük-rank değiştirir; Sherman-Morrison-Woodbury bu güncellemeyi büyük tersi yeniden hesaplamadan verir ve recursive least squares ile Kalman filtresini gerçek zamanlı kılar.

21.13 Kontrol Soruları

i Soru 1: Paydadaki 1 – $v^T u$

$(I - uv^T)^{-1}$ formülünde paydadaki $1 - v^T u$ neyi temsil eder ve neden bir sayıdır?
Cevap: $v^T u$ bir iç-çarpımdır (satır vektör × kolon vektör = 1×1), yani bir **skalerdir**. Payda $1 - v^T u$ bu skalerin 1'den farkı. Eğer $v^T u = 1$ olursa payda sıfır olur — bu durumda $I - uv^T$ tersinir değildir (tekildir). Formülün skaler payda ile çalışması, $n \times n$ tersi tek bir bölmeye indirgemenin tüm gücüdür.

i Soru 2: SMW neye indirger

Sherman-Morrison-Woodbury $n \times n$ bir tersi neye indirger ve bu neden önemli?
Cevap: $n \times n$ tersi $k \times k$ bir terse indirger (k = değişimin rankı). Önemli, çünkü k genelde çok küçüktür (1, 2, birkaç on); $k \times k$ ters almak $O(k^3)$, $n \times n$ ters almak $O(n^3)$. $n = 10^6$, $k = 1$ ise fark astronomik. Online sistemler bu sayede her güncellemede baştan ters almaz.

i Soru 3: Yeni ölçüm neden rank-1 değişim

Least squares'e yeni bir ölçüm eklemek neden $A^T A$'da rank-1 değişimdir?
Cevap: Yeni ölçüm = A'ya yeni bir satır v^T eklemek. $A_{\text{new}} = [A; v^T]$ olunca $A_{\text{new}}^T A_{\text{new}} = A^T A + vv^T$ olur. vv^T bir dış-çarpım (kolon × satır) olduğundan rank tam olarak 1'dir. Dolayısıyla normal denklem matrisi rank-1 büyür ve SMW ile çözüm ucuza güncellenir — recursive least squares.

i Soru 4: Kalman'ın iki ek bileşeni

Kalman filtresini sıradan recursive least squares'ten ayıran iki ek bileşen nedir?
Cevap: (1) Bir **durum denklemi** — ölçülen sistemin zamanla nasıl evrildiğini (örn. hareket modeli) tanımlar; statik RLS'te durum sabittir, Kalman'da hareket eder. (2) Bir **kovaryans matrisi** — her ölçümün ve tahminin belirsizliğini/güvenilirliğini tartar, böylece güncelleme gürültülü ölçüme az, güvenilir ölçüme çok ağırlık verir. Çekirdek matematik yine SMW'dir.

21.14 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. Sayısal doğrulama. $u = [1, 2]^T$, $v = [1, 0]^T$ al. $(I - uv^T)$ matrisini yaz, doğrudan tersini hesapla. Sonra formülle $(I + uv^T/(1 - v^T u))$ hesapla ve aynı sonucu bulduğunu göster. $v^T u$ kaç çıktı?

Egzersiz 2. Tekillik sınırı. Hangi v seçiminde $1 - v^T u = 0$ olur (u sabit)? O durumda $I - uv^T$ matrisinin neden tersinir olmadığını, bir vektörü sıfıra götürdüğünü (çekirdek) göstererek açıkla.

Egzersiz 3. Çözümü güncelleme. $A = I$, $b = [3, 1]^T$, $u = [1, 0]^T$, $v = [0, 1]^T$. §7 formülüyle $(A - uv^T)x = b$ çözümünü, $A^{-1}b$'den başlayarak adım adım bul.

Egzersiz 4. A^TA güncellemesi. A 3×2 bir matris. Altına yeni satır $v^T = [1, 1]$ ekliyorsun. $A^TA + vv^T$'deki vv^T matrisini açıkça yaz; rankının 1 olduğunu (tüm satırları orantılı) göster.

Egzersiz 5. (Ders 15 habercisi) Bu derste matris **sabit** kalıp rank-1 değişti. Ya matris bir t parametresine bağlıysa, $A(t)$? Türevi dA/dt ne anlama gelir, özdeğerleri t değişince nasıl kayar? Bir tahmin yaz — Ders 15 bunu yanıtlıyor.

21.15 Sonraki Ders İçin Hazırlık

Ders 15: A(t) Matrisleri — Türev dA/dt ve Özdeğer Değişimi

Bu derste değişim ayrıktı (rank-1 sıçrama); Ders 15'te değişim **sürekli** olacak: matris bir parametreyle pürüzsüz değişirken özdeğerleri, tekil değerleri ve tersi nasıl türevlenir? Karpathy'nin backprop'unun matris versiyonuna ilk adım.

⚠ Ders 15 Öncesi Yapılacak

- Bu dersin egzersizlerini çöz, özellikle Egzersiz 1'i (sayısal doğrulama) ve Egzersiz 4'ü ($A^TA + vv^T$).
- Python'da bir matrise rank-1 ekleyip tersini doğrudan vs Sherman-Morrison ile hesapla, karşılaştır.
- Ana cümleyi tekrar oku: "Düşük-rank değişim → düşük-rank ters değişimi; SMW $n \times n$ tersi $k \times k$ terse indirir."

21.16 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Yeni bölüm	Düşük-rank matrisler: rank küçük, girdiler değil	0m27
Rank-1 perturbasyon	$(I - uv^T)^{-1} = I + uv^T/(1 - v^Tu)$	3m00
Rank-1 → rank-1 ters	matris rank-1 değişir → tersi rank-1 değişir	5m20
n×n → 1×1	v^Tu skalerdir; büyük ters, bölmeye iner	6m12
Sherman-Morrison-Woodbury	$(I - UV^T)^{-1} = I + U(I_k - V^TU)^{-1}V^T$; $n \times n \to k \times k$	14m03
Yeni ölçüm	least squares'e satır ekle → A^TA rank-1 büyür	19m31
A^TA rank-1 değişim	$A^TA + vv^T$ (kolon×satır dış-çarpım)	26m06
Kalman filtresi	dinamik least squares: durum denklemi + kovaryans	27m58

Kavram	Formül / Fikir	Strang (dk)
Recursive least squares	yeni veride sıfırdan değil, düzeltme ile güncelle	28m22

21.17 ML Bağlantıları Özeti

1. **Online öğrenme / recursive least squares** → veri akış halinde gelir, model her örnekte güncellenir; SGD'nin kapalı-form atası. Adaptive filtreler (eko/gürültü iptali) hâlâ RLS kullanır.
2. **Kalman filtresi** → GPS, otopilot, robot navigasyonu, sensör füzyonu; ML köprüsü = durum-uzay modelleri (S4, Mamba) ve Bayesçi belirsizlik tahmini.
3. **Gauss süreçleri & kernel yöntemleri** → $n \times n$ kernel tersini düşük-rank (Nyström) yaklaşımla $k \times k$ terse indirgemek tam SMW'dir.
4. **Düşük-rank yapı** → LoRA'nın ΔW'si, düşük-rank kovaryans güncellemeleri; "rank küçük, girdiler değil" fikrinin üretim karşılığı.
5. $n \times n \to k \times k$ **indirgeme** → büyük tersi/çözümü küçük tersle güncelle; gerçek zamanlı sistemler.
6. $A^TA + vv^T$ → normal denklemin akış (stream) versiyonu; bellek sığmayan veride tek yol.
7. **Geriye köprü** → Ders 1 (uv^T dış-çarpım), Phase 1 18.06 Ders 4 ($A = LU$ bir kez ayrıştır), normal denklem $A^TA\hat{x} = A^Tb$.

> **!** Tek Şey
>
> Eğer bu dersten tek bir şey alıp gidersen: A**'ya düşük-rank bir değişim eklemek tersini de düşük-rank değiştirir —** $(I - uv^T)^{-1} = I + uv^T/(1 - v^Tu)$ **bir** $n \times n$ **tersi tek bir skaler bölmeye indirger, Sherman-Morrison-Woodbury ise bunu rank-k'ya (**$n \times n$ **ters →** $k \times k$ **ters) genelleştirir. Bu, yeni ölçüm geldiğinde** A^TA**'nın rank-1 büyümesini (**$A^TA + vv^T$**) ucuza güncelleyen recursive least squares ile her adımda kovaryansı düşük-rank güncelleyen Kalman filtresinin matematiksel temelidir — büyük matrisleri "bir kez kur, sonra ucuza güncelle" diliyle yönetmek.**

22 t'ye Bağlı Matrisler A(t) — Türev dA/dt

Tersin türevi, özdeğer türevi ve interlacing eşitsizlikleri

> **Bölüm bilgisi**
>
> **Video:** MIT 18.065 — Matrices A(t) Depending on t, Derivative = dA/dt · **OCW:** Lecture 15 · **Okuma süresi:** ≈35 dk · **Eğitmen:** Gilbert Strang · **Önkoşul:** Ders 14 (tersin sonlu/düşük-rank değişimi).

22.1 Bu Derste Ne Var?

Geçen ders (Ders 14) A değişince **tersinin** nasıl değiştiğini sordu — sonlu, düşük-rank değişim. Bu ders aynı soruyu **özdeğerler ve tekil değerler** için sorar, ve bu kez kalkülüs devreye girer: matris sürekli (t'ye bağlı) değişirse türevler ne olur?

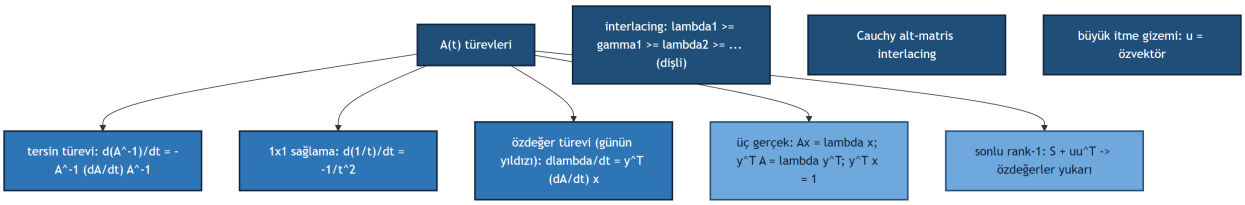
Üç sonuç:

1. **Tersin türevi:** $d(A^{-1})/dt = -A^{-1}\,(dA/dt)\,A^{-1}$ — Ders 14'ün kalkülüs tamamlaması; 1×1 hali $d(1/t)/dt = -1/t^2$.
2. **Özdeğer türevi (günün yıldızı):** $d\lambda/dt = y^T\,(dA/dt)\,x$ — sağ ve sol özvektör arasına değişim hızını koy. Özvektörün türevini içermez — güzelliği bu.
3. **Sonlu rank-1 değişim:** kesin formül YOK ama eşitsizlikler VAR. $S + uu^T$ özdeğerleri yukarı iter; **interlacing** (içiçe geçme) yeni özdeğerlerin eskileri geçmemesini garanti eder.

> *"This time is about changes in eigenvalues and changes in singular values when A change."* — Strang, 1:01

Aşağıdaki kavram haritası dersin merkezini (A(t) türevleri) ana fikirlere bağlar: tersin türevi, 1×1 sağlama, günün yıldızı özdeğer türevi, üç gerçek ve sonlu rank-1 değişimden interlacing zincirine, Cauchy alt-matrisine ve büyük itme gizemine uzanan dallar (Şekil 22.1).

> **Builder Notu — Hareket Eden Matrisin Kalkülüsü**
>
> - **Perturbasyon teorisi:** "A biraz değişirse λ ne kadar değişir?" sorusu ML'de her yerde — eğitim sırasında Hessian özdeğerleri, PCA'da veri güncellemesi, kararlılık analizi.
> - **$d\lambda/dt = y^T(dA/dt)x$** — backprop'un (Ders 27) özdeğer versiyonu: bir çıktının (λ) bir girdiye (A) duyarlılığı; çarpım kuralı + iptal.
> - **Interlacing** — rank-1 güncelleme özdeğerleri "kontrollü" oynatır; öneri sistemleri, online PCA, grafik Laplacian spektrumu (Ders 19, 35) buna dayanır.



Şekil 22.1: Ders 15 kavram haritası: A(t) türevleri merkezinden ana fikirlere dallar — tersin türevi $d(A^{-1})/dt = -A^{-1}\,(dA/dt)\,A^{-1}$, bunun 1×1 sağlaması lise kalkülüsünden $d(1/t)/dt = -1/t^2$, günün yıldızı özdeğer türevi $d\lambda/dt = y^T\,(dA/dt)\,x$ üç gerçeğe dayanır ($Ax = \lambda x$; $y^TA = \lambda y^T$; $y^Tx = 1$) ve sonlu rank-1 değişim $S + uu^T$ özdeğerleri yukarı iter; ayrı düğümler dişli gibi içiçe geçen interlacing zinciri $\lambda_1 \geq \gamma_1 \geq \lambda_2 \geq \cdots$, alt-matrise inen Cauchy interlacing ve u tam özvektörse yalnız onun özdeğerinin fırladığı büyük itme gizemini gösterir.

- **Geriye köprü:** Ders 14 (tersin değişimi), Ders 4 (özdeğer/özvektör), Calculus (türev = limit, çarpım kuralı).

Tek cümle: Matris t ile değişirken tersi, özdeğerleri ve tekil değerleri de değişir; sonsuz-küçük değişim için temiz türev formülleri ($d\lambda/dt = y^T(dA/dt)x$), sonlu değişim için ise interlacing eşitsizlikleri vardır.

22.2 Büyük Resim: Matrisler Hareket Eder

Matrisler sabit durmaz — bir t parametresine (zaman, adım, ayar) bağlı olabilirler: A(t). Matris hareket edince tersi, özdeğerleri ve tekil değerleri de hareket eder. Bu dersin sorusu: nasıl?

> *"This time is about changes in eigenvalues and changes in singular values when A change."* — Strang, 1:01

İki paralel soru var. Biri **sonsuz-küçük** değişim (dA → türev, kalkülüs); diğeri **sonlu** değişim (gerçek bir vektör eklemek, rank-1).

> *"What is the derivative when the change is infinitesimal?"* — Strang, 2:37

Türev tarafında **kesin formüller** elde edeceğiz. Sonlu tarafta özdeğerler için kesin formül yok — ama güçlü **eşitsizlikler** (interlacing) var.

> Builder Notu — İki Rejim Tek Soru
>
> "Sonsuz-küçük (türev) vs sonlu (gerçek adım)" ayrımı optimizasyonun kalbi: gradient descent sonsuz-küçük yön (gradyan) hesaplar ama sonlu adım atar (learning rate). Bu dersin iki yarısı, tam bu iki rejimin lineer cebir karşılığıdır.

22.3 Tersin Türevi: $d(A^{-1})/dt$

Ders 14 tersin sonlu değişimini verdi; şimdi sonsuz-küçük halini tamamlıyoruz. İşaret noktası bir özdeşlik:

"So here's a handy identity." — Strang, 5:43

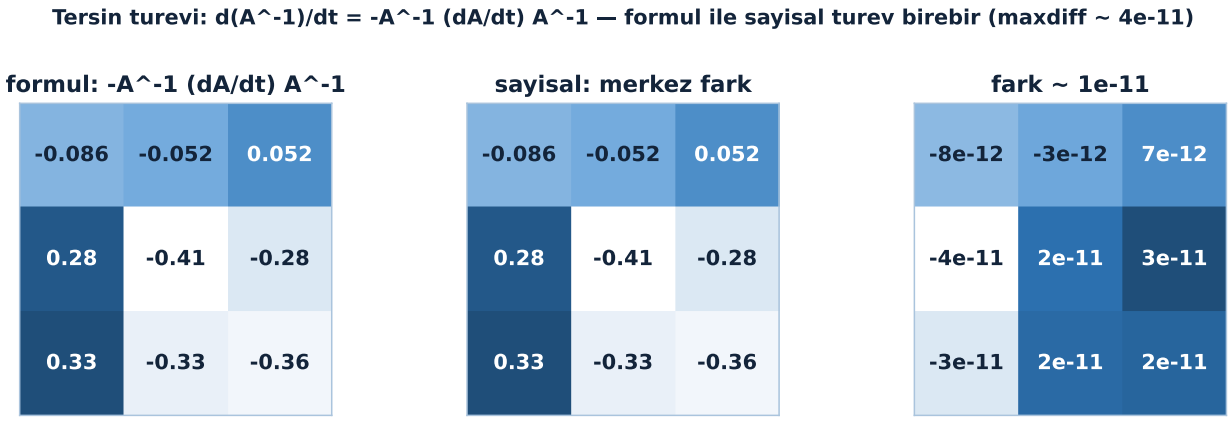
$$B^{-1} - A^{-1} = B^{-1}(A - B)A^{-1}$$

Doğrula: $B^{-1}(A - B)A^{-1} = B^{-1}AA^{-1} - B^{-1}BA^{-1} = B^{-1} - A^{-1}$. ✓ Şimdi $B = A + \Delta A$ al, yani $A - B = -\Delta A$. Her iki tarafı Δt'ye böl ve $\Delta t \to 0$ limitini al (kalkülüs): $\Delta A/\Delta t \to dA/dt$, $B \to A$. Sonuç:

$$\frac{d(A^{-1})}{dt} = -A^{-1}\,\frac{dA}{dt}\,A^{-1}$$

Eksi işaret kritik; iki tarafta da A^{-1} var. Bu, herkesin bilmesi gereken temiz bir formül.

Aşağıdaki figür formülü sayısal olarak doğrular: kapalı-form sandviç $-A^{-1}(dA/dt)A^{-1}$ ile merkez-fark sayısal türevi birebir örtüşür; aradaki fark yalnızca yuvarlama gürültüsüdür (maxdiff $\approx 4\times10^{-11}$, Şekil 22.2).



Şekil 22.2: Tersin türevi $\frac{d(A^{-1})}{dt} = -A^{-1}\frac{dA}{dt}A^{-1}$: soldaki kapalı-form sandviç ile ortadaki merkez-fark sayısal türevi aynı matrisi verir; sağdaki fark paneli yalnızca yuvarlama gürültüsü taşır (maxdiff $\approx 4 \times 10^{-11}$). Burada $A(t) = P\,\mathrm{diag}(5 + \sin t,\ 2 + t^2,\ -1 + 0.5t)\,P^{-1}$ ailesi $t_0 = 0.3$ noktasında değerlendirilmiştir.

> **Builder Notu — SMW'nin Türev Kardeşi**
>
> $d(A^{-1})/dt = -A^{-1}(dA/dt)A^{-1}$ formülü Ders 14'ün SMW'sinin sonsuz-küçük kardeşidir: SMW sonlu rank-k değişimi, bu ise türev. ML'de Newton yöntemi ve ikinci-derece optimizasyonda Hessian'ın tersi sürekli güncellenir — bu türev o güncellemenin teorik temeli.

22.4 1×1 Sağlaması: Lise Kalkülüsü

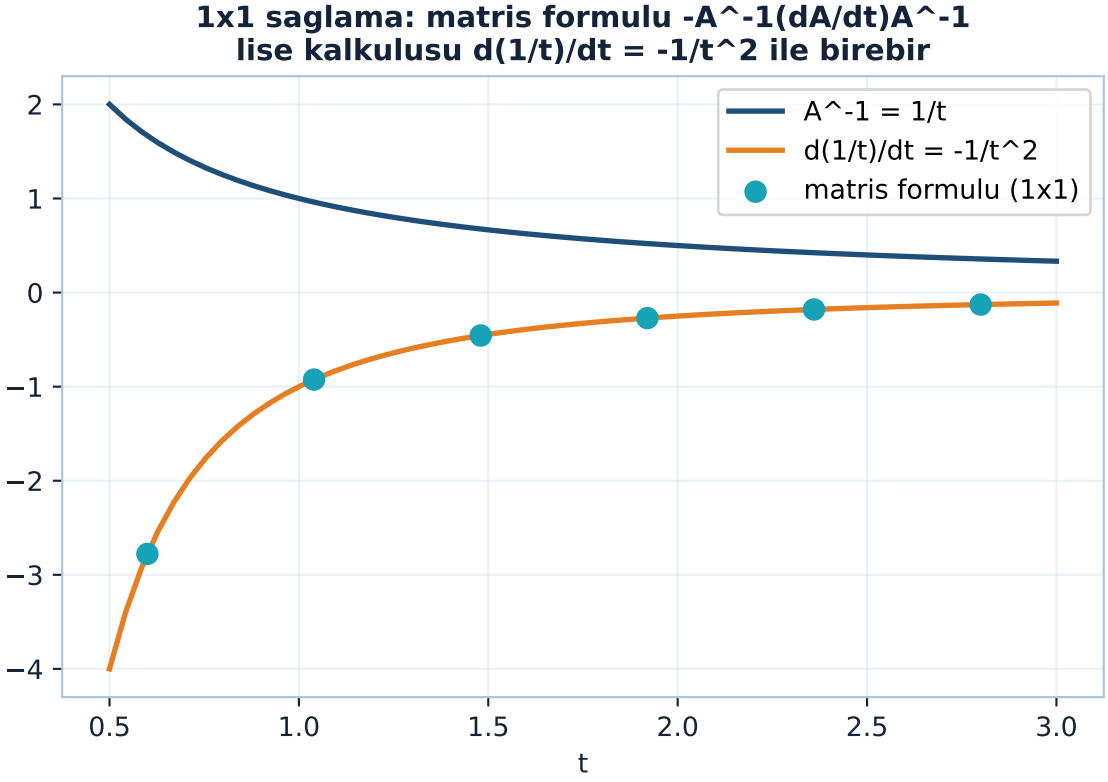
Formülün doğru olduğunu görmenin en hızlı yolu: A'yı 1×1 al, yani sadece bir sayı t. O zaman $A^{-1} = 1/t$ ve formül $d(1/t)/dt$ olmalı:

"If A is just t, then the derivative of 1 over t with respect to t is minus 1 over t squared." — Strang, 12:00

$$\frac{d}{dt}\left(\frac{1}{t}\right) = -\frac{1}{t^2}$$

Matris formülüyle karşılaştır: $-A^{-1}(dA/dt)A^{-1} \rightarrow$ 1×1'de $-(1/t)(1)(1/t) = -1/t^2$. Birebir aynı. Kalkülüsün 1×1 hâlini biliyorduk; ders bunu n×n'e taşıdı.

Aşağıdaki figür sağlamayı görselleştirir: 1×1 matris formülünün altı noktadaki değeri (teal), lise kalkülüsünün $-1/t^2$ eğrisinin (turuncu) tam üstüne oturur (Şekil 22.3).



Şekil 22.3: 1×1 sağlama: matris formülü $-A^{-1}(dA/dt)A^{-1}$ ile lise kalkülüsünün $d(1/t)/dt = -1/t^2$ türevi birebir örtüşür. Teal noktalar (matris formülü) turuncu eğrinin tam üstüne oturur.

> **Builder Notu — Skalere İndir Hatayı Yakala**
>
> "Formülü 1×1'e indir, bildiğin lise sonucuyla karşılaştır" Strang'ın tekrar tekrar kullandığı sağlama tekniği (Ders 14'te de $v^T u$ skaleriyle yaptı). Bir matris formülünden emin değilsen, skaler hâline indirgemek en hızlı hata yakalama yöntemidir — kod testinde de aynı: en küçük boyutta doğrula.

22.5 Özdeğer Türevine Hazırlık: Üç Gerçek

Asıl yıldız: özdeğer A ile değişirken $d\lambda/dt$ nedir? Önce üç bilineni topla. (1) Sağ özvektör denklemi, hepsi t'ye bağlı:

$$A(t)\,x(t) = \lambda(t)\,x(t)$$

(2) A^T'nin özdeğerleri A ile aynıdır ama özvektörleri farklı. A^T'nin özvektörünü y (satır vektör) yazıp denklemi sol-özvektör biçiminde tut:

$$y^T(t)\,A(t) = \lambda(t)\,y^T(t)$$

(3) Bir normalizasyon gerek — sağ ve sol özvektörü birbirine kenetle:

$$y^T(t)\,x(t) = 1$$

Simetrik durumda y = x = q (birim özvektör); o zaman $y^T x = 1$, $q^T q = 1$ demek. Bu üç gerçek tüm türetimin malzemesi.

> Builder Notu — Sağ ve Sol Kenetlenir
>
> Sağ özvektör (x) ve sol özvektör (y) ayrımı simetrik-olmayan matrislerde kritik: $A \neq A^T$ olduğunda iki ayrı özvektör ailesi vardır, ama özdeğerler ortaktır (Ders 4). $y^T x = 1$ normalizasyonu, birazdan iki terimin "1'in türevi = 0" diye iptal olmasını sağlayacak — formülün zarafetinin tohumu burada.

22.6 Formül 1: $\lambda = y^T Ax$

$Ax = \lambda x$ denklemini soldan y^T ile çarp: $y^T Ax = \lambda\, y^T x$. Ama $y^T x = 1$ (normalizasyon), yani:

$$\lambda(t) = y^T(t)\,A(t)\,x(t)$$

Özdeğeri, üç t-bağımlı parçanın çarpımı olarak yazdık. Bu, Ders 4'ün $\lambda = x^{-1}Ax$ köşegenleştirmesinin sol-sağ özvektör versiyonu. Artık türevini almaya hazırız.

> Builder Notu — Sandviç Yaz Türev Al
>
> $\lambda = y^T Ax$ bir "Rayleigh-tipi" ifadedir (Ders 19'da Rayleigh bölümü tam bunu genelleyecek). Bir skaleri (λ) vektör-matris-vektör sandviçi olarak yazmak, türevini almayı kolaylaştırır — çünkü çarpım kuralı uygulanabilir hâle gelir.

22.7 $d\lambda/dt = y^T(dA/dt)x$ — İptalin Zarafeti

Şimdi $\lambda = y^T Ax$'in türevini al. Üç t-bağımlı çarpan var:

> *"…That'll be the formula for the derivative of an eigenvalue."* — Strang, 21:38

> *"So I'm going to use the product rule."* — Strang, 22:56

$$\frac{d\lambda}{dt} = \frac{dy^T}{dt}Ax + y^T\frac{dA}{dt}x + y^TA\frac{dx}{dt}$$

Orta terim cevabımız. Diğer ikisi sıfıra gider: Ax = λx ve $y^TA = \lambda y^T$ kullan, λ skalerini dışarı çek:

$$\frac{dy^T}{dt}Ax + y^TA\frac{dx}{dt} = \lambda\left(\frac{dy^T}{dt}x + y^T\frac{dx}{dt}\right) = \lambda\frac{d(y^Tx)}{dt}$$

Parantez içi tam olarak y^Tx çarpımının türevi. Ama $y^Tx = 1$ (sabit), türevi 0:

> *"The derivative of the y transpose x."* — Strang, 28:34

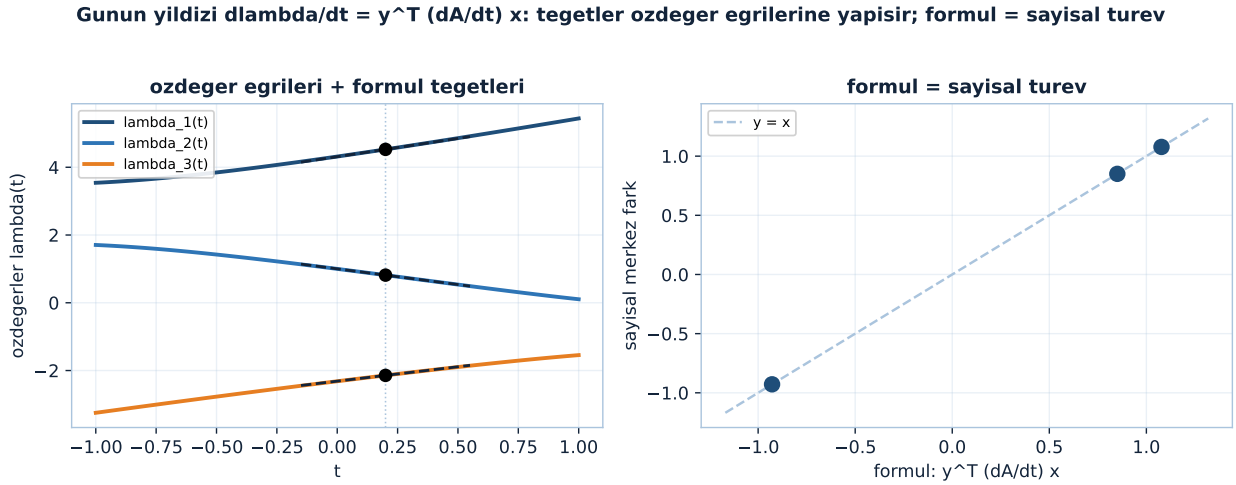
Geriye sade ve güçlü formül kalır:

$$\frac{d\lambda}{dt} = y^T\frac{dA}{dt}x$$

> *"The derivative of the eigenvalue is this formula. It's the rate at which the matrix is changing times the eigenvectors on right and left."* — Strang, 29:04

Dikkat: formül özvektörün türevini (dx/dt) **içermez** — sadece matrisin değişim hızını ve mevcut özvektörleri ister. İşte güzelliği bu.

Aşağıdaki flagship figür formülün görsel kanıtını verir: solda S(t) ailesinin üç özdeğer eğrisi, t_0=0.2'de formülün verdiği eğimle çizilen teğetlere birebir yapışır; sağda formül $y^T(dA/dt)x$ ile sayısal merkez fark karşılaştırması y=x doğrusunda oturur (Şekil 22.4).



Şekil 22.4: Günün yıldızı $d\lambda/dt = y^T(dA/dt)x$ görsel kanıtı. Sol: S(t) simetrik ailesinin üç özdeğer eğrisi $\lambda_i(t)$; t_0=0.2'de formülün verdiği eğimle çizilen teğet doğru parçaları (kesik) eğrilere birebir yapışır. Sağ: aynı noktada formül $y^T(dA/dt)x$ ile sayısal merkez fark karşılaştırması — noktalar y=x doğrusunda oturur (formül = sayısal türev).

> **Builder Notu — Tasarımla Gelen İptal**
>
> İki terimin "1'in türevi = 0" diye yok olması, normalizasyon $y^Tx = 1$ seçiminin meyvesi — tasarım gereği iptal. ML köprüsü: bu formül perturbasyon teorisinin birinci-derece sonucu; bir özdeğerin (örn. Hessian'ın en büyük özdeğeri = en dik yön) parametre değişimine duyarlılığını dx/dt hesaplamadan verir. Tam da backprop'un (Ders 27) "çıktının girdiye duyarlılığı" mantığı.

22.8 Sonlu Rank-1 Değişim: Özdeğerler Yukarı

Şimdi türevi bırak, **gerçek** bir değişime geç: simetrik S'ye rank-1 simetrik bir parça ekle, $S + uu^T$. Bu kalkülüs değil (değişim sonsuz-küçük değil), kesin formül de yok — ama bilinen güzel gerçekler var. Önce uu^T ne tür bir matris? Rank-1, simetrik ve **pozitif yarı-tanımlı** (tek sıfırdan farklı özdeğeri $u^Tu > 0$, özvektörü u). Yani S'ye "pozitif bir şey" ekliyoruz. Sonuç:

> *"They'll be more positive. They'll go up. This is a positive thing."* — Strang, 36:05

S'nin özdeğerlerine γ (gamma), $S + uu^T$'nin özdeğerlerine λ diyelim (ikisi de azalan sırada). Her λ karşılık gelen γ'dan büyük:

> *"Lambdas are bigger than gammas."* — Strang, 37:51

$$\lambda_j \geq \gamma_j \quad (\text{all } j)$$

Aşağıdaki figür bu akışı izler: $S + c \cdot uu^T$'deki dört özdeğer de monoton yukarı akar, ama $\lambda_2(c)$ eğrisi $\gamma_1 = 4.38$ yatayını asla geçemez — interlacing her eğriyi bir komşu γ bandına kilitler (Şekil 22.5).

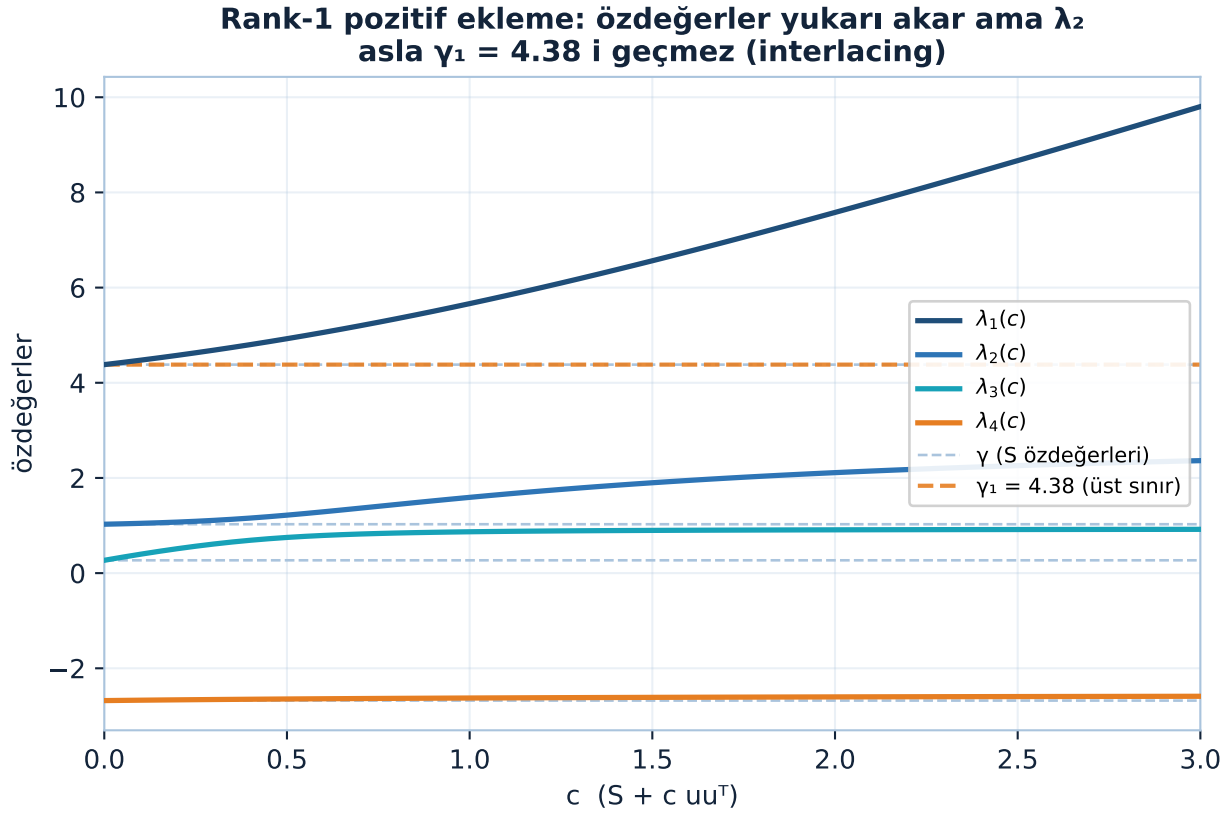
> **Builder Notu — Pozitif Ekleme Yukarı İter**
>
> Pozitif yarı-tanımlı bir parça eklemek özdeğerleri asla düşürmez — "17 eklemek gibi, yukarı taşır" (Strang). ML köprüsü: ridge regülarizasyonu (Ders 10) tam bunu yapar — $A^TA + \lambda I$, yani I yönünde pozitif ekleme, tüm özdeğerleri λ kadar yukarı kaydırıp matrisi iyi-koşullu kılar.

22.9 Interlacing (İçiçe Geçme) Teoremi

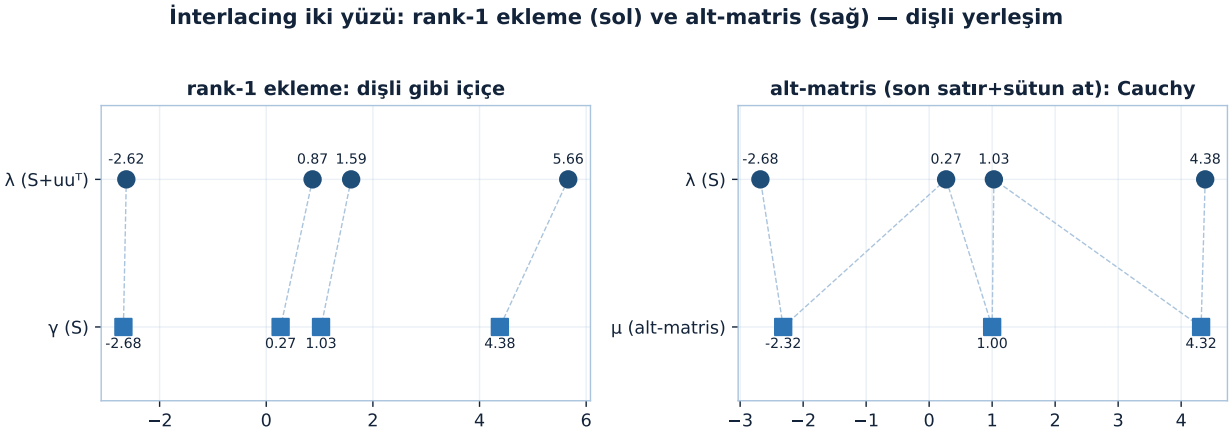
Özdeğerler yukarı gider — ama ne kadar? Sürpriz: kontrollü. Yeni λ_2, eski γ_1'i geçemez. Tam ifade:

$$\lambda_1 \geq \gamma_1 \geq \lambda_2 \geq \gamma_2 \geq \lambda_3 \geq \cdots$$

> *"And that's called interlacing."* — Strang, 39:13



Şekil 22.5: Rank-1 pozitif ekleme $S + c \cdot uu^{T}$: dört özdeğer de monoton yukarı akar, ama $\lambda_2(c)$ asla $\gamma_1 = 4.38$ yatayını geçemez. İnterlacing eşitsizliği $\lambda_1 \geq \gamma_1 \geq \lambda_2 \geq \gamma_2 \geq \cdots$ her c için her eğriyi bir komşu γ bandına kilitler (λ_2 maksimumu 2.36 < $\gamma_1 = 4.38$).



Şekil 22.6: İnterlacing'in iki yüzü. Sol: rank-1 ekleme $S + uu^{T}$ — yeni özdeğerler $\lambda = 5.66, 1.59, 0.87, -2.62$ (üst sıra, navy yuvarlak) eski $\gamma = 4.38, 1.03, 0.27, -2.68$ (alt sıra, çelik kare) değerlerinin arasına dişli gibi oturur ($\lambda_1 \geq \gamma_1 \geq \lambda_2 \geq \gamma_2 \geq \cdots$). Sağ: son satır + sütun atılarak küçültülen 3×3 alt-matrisin Cauchy özdeğerleri $\mu = 4.32, 1.00, -2.32$ (alt sıra) tam tam ardışık λ'ların arasında kalır ($\lambda_1 \geq \mu_1 \geq \lambda_2 \geq \mu_2 \geq \lambda_3 \geq \mu_3 \geq \lambda_4$). İki teorem de aynı dişli yerleşimin örnekleri.

Yani rank-1 pozitif değişimde her özdeğer yukarı çıkar, ama yeni değerler eski değerlerin arasına "dişli gibi" geçer — λ_2 eski birinci özdeğeri (γ_1) aşamaz, λ_3 ikinciyi (γ_2) aşamaz. Kalbi mutlu eden teoremlerden.

Aşağıdaki figürün sol paneli bu dişli yerleşimi sayı doğrusu üzerinde gösterir: yeni özdeğerler λ = 5.66, 1.59, 0.87, −2.62 (üst sıra) eski γ = 4.38, 1.03, 0.27, −2.68 (alt sıra) değerlerinin tam aralarına oturur (Şekil 22.6).

> Builder Notu — Dişli Gibi İçiçe
>
> Interlacing, online/artımlı PCA'nın garantisidir: yeni bir veri noktası (rank-1 güncelleme) ana bileşenleri yukarı oynatır ama sıralamayı çığırından çıkarmaz — birinci bileşen yine birinci kalır, sadece komşu sınırlar içinde kayar. Bu, spektral kümeleme (Ders 35) ve graf Laplacian özdeğer analizinde de (Ders 19) temel araçtır.

22.10 Rank-2 ve Alt-matris: Aynı Tema

İki uzantı. (1) Rank-2 değişim S + uu^T + ww^T: özdeğerler (α diyelim) daha çok yukarı çıkabilir — α_2 artık λ_1'i geçebilir, ama α_3 eski γ_1'i hâlâ geçemez. Genel kural: rank-r değişim, interlacing'i r adım "gevşetir".

> *"Let me give you another example of interlacing."* — Strang, 45:50

(2) Alt-matris interlacing (Cauchy): n×n simetrik S'den son satır+sütunu at, (n−1)×(n−1) matris S_{n−1} kalsın. Özdeğerleri interlace eder:

$$\lambda_1 \geq \mu_1 \geq \lambda_2 \geq \mu_2 \geq \cdots \geq \mu_{n-1} \geq \lambda_n$$

Sebep aynı: bir satır+sütun atmak $x_n = 0$ kısıtı koymak demek — bir serbestlik derecesi azalır, özdeğerler düşer ama interlace ederek. Yukarıdaki figürün sağ paneli tam bunu gösterir: 3×3 alt-matrisin μ özdeğerleri ardışık λ değerlerinin arasında kalır (Şekil 22.6).

> Builder Notu — Kısıt Ekle Serbestlik Azalt
>
> Alt-matris interlacing, özellik seçimi (feature selection) ve graf budamada (pruning) işe yarar: bir düğüm/özellik atıldığında spektrum nasıl değişir önceden sınırlanır. "Bir kısıt eklemek = bir serbestlik derecesi azaltmak" enerji görüşü (Ders 5'in x^TSx kâsesi) buraya doğrudan bağlanır.

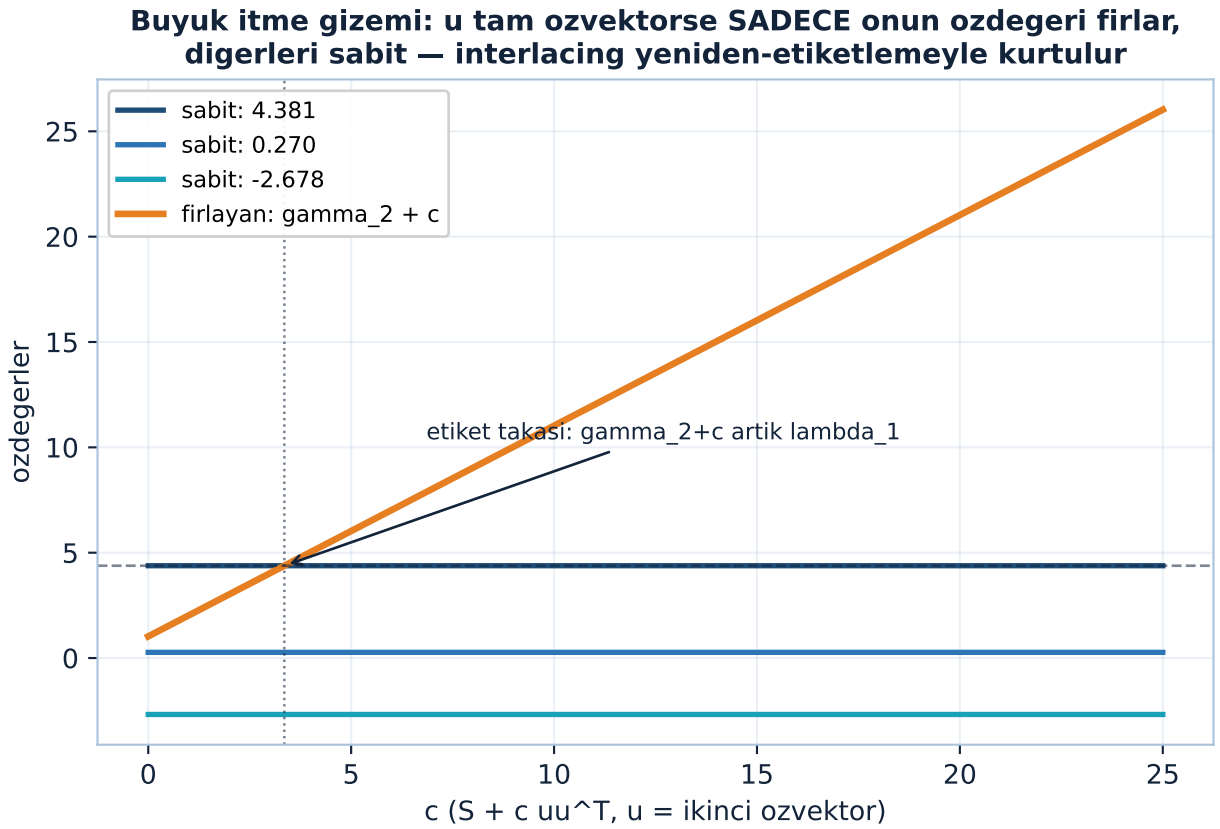
22.11 Son Gizem: Bir Özvektör Yönünde Büyük İtme

Strang dersi bir bilmeceyle bitirir. Diyelim u tam olarak S'nin **ikinci özvektörü**, yani Su = γ_2u. Şimdi büyük bir itme yap: S + $20uu^T$.

> *"…suppose it's actually the second eigenvector of S."* — Strang, 48:06

O zaman $(S + 20uu^T)u = \gamma_2 u + 20u = (\gamma_2 + 20)u$. Yani bu özvektörün özdeğeri γ_2'den $\gamma_2 + 20$'ye fırladı — çok yukarı. Görünüşte interlacing'i bozuyor (λ_2, γ_1'i geçmiş gibi). Strang bunu bilerek açık bırakır: çelişki gerçek mi, yoksa özdeğerlerin yeniden etiketlenmesi (γ_2+20 artık yeni λ_1 olabilir) bunu kurtarır mı? Cevap sonraki derste.

Aşağıdaki flagship figür gizemi canlandırır: u tam ikinci özvektör olunca dört özdeğerden sadece biri (γ_2+c) doğrusal fırlar, diğer üçü (γ_1=4.381, γ_3=0.270, γ_4=−2.678) hiç kıpırdamadan sabit kalır; fırlayan değer c≈3.35'te γ_1'i kesip λ_1 etiketini kapar (Şekil 22.7).



Şekil 22.7: Büyük itme gizemi (son gizem). u = S'nin ikinci özvektörü alınıp $S + c\,uu^T$ kurulduğunda, dört özdeğerden SADECE birinin değeri ($\gamma_2 + c$) doğrusal fırlar; diğer üçü ($\gamma_1 = 4.381$, $\gamma_3 = 0.270$, $\gamma_4 = -2.678$) hiç kıpırdamadan SABİT kalır. c = 20'de yeni spektrum $[21.028,\ 4.381,\ 0.270,\ -2.678]$ olur; fırlayan değer $\gamma_2 + 20 = 21.028$ başa geçip λ_1 eti-

Builder Notu — Sıra Korunur Kimlik Değil

22.12 Bu Dersin Özeti

- **Tersin türevi:** $d(A^{-1})/dt = -A^{-1}(dA/dt)A^{-1}$; 1×1'de $d(1/t)/dt = -1/t^2$. Ders 14'ün sonsuz-küçük tamamlaması.
- **Üç gerçek:** $Ax = \lambda x$ (sağ özvektör), $y^T A = \lambda y^T$ (sol özvektör), $y^T x = 1$ (normalizasyon).
- **Formül 1:** $\lambda = y^T Ax$ (Ax'i soldan y^T ile vur).
- **Özdeğer türevi (yıldız):** $d\lambda/dt = y^T(dA/dt)x$. Çarpım kuralı + iki terim "1'in türevi = 0" diye iptal. Özvektör türevi içermez.
- **Sonlu rank-1 değişim:** $S + uu^T$ (pozitif yarı-tanımlı) özdeğerleri yukarı iter; $\lambda_j \geq \gamma_j$.
- **Interlacing:** $\lambda_1 \geq \gamma_1 \geq \lambda_2 \geq \gamma_2 \geq \cdots$; yeni özdeğerler eskilerin arasına geçer. Rank-r değişim r adım gevşetir; alt-matris (Cauchy) interlacing'i de aynı.
- **Açık gizem:** u = ikinci özvektör + büyük itme → özdeğer fırlar; çelişki mi? (Sonraki ders.)

! Tek Bir Cümle

Matris t ile değişirken özdeğerin türevi $d\lambda/dt = y^T(dA/dt)x$'tir (sade, özvektör türevi gerektirmez); sonlu rank-1 değişimde ise kesin formül yerine interlacing eşitsizlikleri özdeğerlerin nasıl kayacağını sınırlar.

22.13 Kontrol Soruları

i Soru 1: 1×1'e indirgeme

$d(A^{-1})/dt = -A^{-1}(dA/dt)A^{-1}$ formülünü 1×1 durumda hangi lise sonucuna indirgersin?
Cevap: A'yı 1×1 al, yani bir sayı t. $A^{-1} = 1/t$, $dA/dt = 1$. Formül $-(1/t)(1)(1/t) = -1/t^2$ verir — bu tam olarak $d(1/t)/dt = -1/t^2$, lise kalkülüsünden bilinen sonuç. Matris formülünü skalere indirip bilinenle karşılaştırmak en hızlı sağlamadır.

i Soru 2: İki terim neden sıfır

$d\lambda/dt = y^T(dA/dt)x$ türetiminde çarpım kuralının iki terimi neden sıfır olur?
Cevap: $\lambda = y^T Ax$'in türevinde üç terim çıkar; baş ve son terim $(dy^T/dt)Ax + y^T A(dx/dt)$. $Ax = \lambda x$ ve $y^T A = \lambda y^T$ kullanıp λ skalerini dışarı çekince $\lambda\cdot((dy^T/dt)x + y^T(dx/dt)) = \lambda\cdot d(y^T x)/dt$ olur. Ama $y^T x = 1$ sabit, türevi 0. Dolayısıyla bu iki terim yok olur, geriye sadece orta terim $y^T(dA/dt)x$ kalır.

i Soru 3: $S + uu^T$ özdeğerleri

S simetrik, u bir vektör. $S + uu^T$'nin özdeğerleri S'ninkilere göre nasıl konumlanır?
Cevap: uu^T pozitif yarı-tanımlı (tek özdeğeri $u^T u > 0$) olduğundan tüm özdeğerler yukarı gider: $\lambda_j \geq \gamma_j$. Ama keyfî değil — interlacing geçerli: $\lambda_1 \geq \gamma_1 \geq \lambda_2 \geq \gamma_2 \geq \cdots$. Yani her yeni özdeğer eski karşılığından büyüktür, fakat bir sonraki eski özdeğeri aşamaz.

Soru 4: Son satır+sütun atınca

Simetrik n×n matristen son satır+sütunu atınca özdeğerler nasıl davranır?
Cevap: Cauchy interlacing: (n−1)×(n−1) alt-matrisin özdeğerleri μ, orijinalin özdeğerleri λ ile içiçe geçer: $\lambda_1 \geq \mu_1 \geq \lambda_2 \geq \mu_2 \geq \cdots \geq \lambda_n$. Sebep: bir satır+sütun atmak $x_n = 0$ kısıtı koymak, yani bir serbestlik derecesini kaldırmaktır; bu özdeğerleri düşürür ama interlace ederek (kontrollü).

22.14 Egzersizler

Cevapsız problemler. Çöz, sonra `numpy` ile kontrol et.

Egzersiz 1. Tersin türevi, somut. A(t) = [[t, 0], [0, t^2]]. $A^{-1}(t)$'yi yaz, doğrudan türevini al. Sonra $-A^{-1}(dA/dt)A^{-1}$ formülüyle hesapla; aynı sonucu bulduğunu göster.

Egzersiz 2. Özdeğer türevi. A(t) = [[2, t], [t, 2]] simetrik. t = 0'da özdeğerler 2, 2. dA/dt = [[0,1],[1,0]]. t = 0'daki bir özvektör x = $[1,1]^T/\sqrt{2}$ için $d\lambda/dt = y^T(dA/dt)x$'i hesapla (simetrik olduğundan y = x).

Egzersiz 3. İptali kanıtla. $y^Tx = 1$ yerine $y^Tx = c$ (sabit) olsaydı, çarpım kuralının iki terimi yine sıfır olur muydu? Neden? (İpucu: d(c)/dt.)

Egzersiz 4. Interlacing testi. S = diag(3, 1) (özdeğerler $\gamma = 3, 1$). u = $[1, 0]^T$. S + uu^T'nin özdeğerlerini bul (λ). $\lambda_1 \geq \gamma_1 \geq \lambda_2$ interlacing'ini doğrula.

Egzersiz 5. (Ders 16 habercisi) Bu derste özdeğer türevini bulduk: $d\lambda/dt = y^T(dA/dt)x$. Aynı fikri **tekil değer** için kurmaya çalış: σ değişirken dσ/dt ne olur? A^TA ve tekil vektörler nasıl girer? Bir tahmin yaz — Ders 16 bunu (ve tersin/tekil değerlerin türevlerini) işliyor.

22.15 Sonraki Ders İçin Hazırlık

Ders 16: Ters ve Tekil Değerlerin Türevleri. Bu dersin özdeğer türevi formülünü ($d\lambda/dt = y^T(dA/dt)x$) tekil değerlere taşıyacağız: $d\sigma/dt = u^T(dA/dt)v$ (sol/sağ tekil vektörler). Ayrıca interlacing'in tekil değer versiyonu ve perturbasyon sınırları gelir — PCA ve düşük-rank yaklaşımın kararlılığının teorik temeli.

Ders 16 Öncesi Yapılacak

- Bu dersin egzersizlerini çöz, özellikle Egzersiz 2'yi (özdeğer türevi) ve Egzersiz 4'ü (interlacing testi).
- Python'da bir A(t) ailesi kur, $d\lambda/dt = y^T(dA/dt)x$ formülünü sayısal merkez fark türeviyle karşılaştır.
- Ana cümleyi tekrar oku: "Özdeğer türevi $d\lambda/dt = y^T(dA/dt)x$ özvektör türevi gerektirmez; sonlu değişimde interlacing özdeğerleri sınırlar."

22.16 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Büyük resim	A değişince ters, özdeğer, tekil değer değişir	1m01
Tersin türevi	$d(A^{-1})/dt = -A^{-1}\,(dA/dt)\,A^{-1}$	5m43
1×1 sağlama	$d(1/t)/dt = -1/t^2$	12m00
Üç gerçek	$Ax = \lambda x$; $y^TA = \lambda y^T$; $y^Tx = 1$	14m26
Formül 1	$\lambda = y^TAx$	20m43
Özdeğer türevi	$d\lambda/dt = y^T\,(dA/dt)\,x$	29m04
İptal	iki terim = $\lambda \cdot d(y^Tx)/dt = \lambda \cdot 0$	28m34
Rank-1 değişim	$S + uu^T$ (psd) → özdeğerler ↑, $\lambda_j \geq \gamma_j$	36m05
Interlacing	$\lambda_1 \geq \gamma_1 \geq \lambda_2 \geq \gamma_2 \geq \cdots$	39m13
Alt-matris (Cauchy)	son satır+sütun at → özdeğerler interlace	45m50
Açık gizem	u = 2. özvektör, $S + 20uu^T$ → özdeğer fırlar	48m06

22.17 ML Bağlantıları Özeti

1. **Perturbasyon teorisi:** $d\lambda/dt = y^T(dA/dt)x$ — bir özdeğerin parametre değişimine birinci-derece duyarlılığı; Hessian'ın en büyük özdeğeri (en dik yön) eğitimde nasıl kayar sorusuna doğrudan cevap.
2. **İkinci-derece optimizasyon:** $d(A^{-1})/dt = -A^{-1}(dA/dt)A^{-1}$ — Newton yöntemi ve doğal gradyanın Hessian-tersi güncellemelerinin temeli.
3. **Online/artımlı PCA:** rank-1 güncelleme + interlacing → ana bileşenler kontrollü kayar, sıralama bozulmaz.
4. **Regülarizasyon:** $S + \lambda I$ (ridge) pozitif yarı-tanımlı ekleme → tüm özdeğerler yukarı, iyi-koşullu (Ders 10 bağı).
5. **Spektral yöntemler:** graf Laplacian özdeğer analizi (Ders 19 maxmin, Ders 35 kümeleme) interlacing'e dayanır.
6. **Geriye köprü:** Ders 14 (tersin sonlu değişimi → buradaki türev), Ders 4 (özdeğer/özvektör, sağ-sol ayrımı), Ders 5 (psd, enerji kâsesi), Calculus (türev = limit, çarpım kuralı).

> **!** Tek Şey
>
> *"This time is about changes in eigenvalues and changes in singular values when A change."* — Strang, 1:01 — Matrisler hareket eder; bu ders onların özdeğerlerinin nasıl hareket ettiğini, hem sonsuz-küçük (türev) hem sonlu (interlacing) rejimde okumayı öğretir.

23 Ters ve Tekil Değerlerin Türevleri

Tekil değer türevi, Weyl eşitsizliği ve nuclear norm ile matris tamamlama

> **Bölüm bilgisi**
>
> **Video:** MIT 18.065 — Derivatives of Inverse and Singular Values · **OCW:** Lecture 16 · **Okuma süresi:** ≈32 dk · **Eğitmen:** Gilbert Strang · **Önkoşul:** Ders 15 (özdeğer türevi ve interlacing).

23.1 Bu Derste Ne Var?

Ders 15'i tamamlıyoruz. Orada özdeğer türevini ($d\lambda/dt = y^T(dA/dt)x$) bulduk; bu ders aynı hikayeyi **tekil değer** için yazar ve sonlu değişim eşitsizliklerini (interlacing → Weyl) genelleştirir.

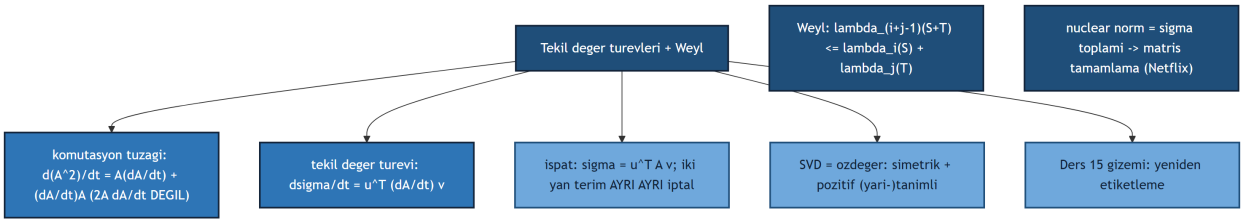
Dört sonuç:

1. **Komütasyon tuzağı:** $d(A^2)/dt = A(dA/dt) + (dA/dt)A$ — **$2A(dA/dt)$ DEĞİL**, çünkü matrisler değişmeli (commute) değil.
2. **Tekil değer türevi:** $d\sigma/dt = u^T (dA/dt)\, v$ — sol tekil vektör u, sağ tekil vektör v. Özdeğer formülünün tam ikizi.
3. **Ders 15 gizemi çözüldü:** büyük itmede μ_2 λ_1'i "geçince" artık μ_1 olarak yeniden etiketlenir — çelişki yok.
4. **Weyl eşitsizliği:** interlacing'in genel hali — herhangi rank değişim T için $\lambda_{i+j-1}(S+T) \le \lambda_i(S) + \lambda_j(T)$. Sonda: nuclear norm + matris tamamlama (Netflix).

   "Boy, you couldn't ask for a nicer formula than that, right?" — Strang, 7:36 (tekil değer türevi için)

Aşağıdaki kavram haritası dersin merkezini (tekil değer türevleri + Weyl) ana fikirlere bağlar: komütasyon tuzağı, günün formülü tekil değer türevi, $\sigma = u^T A v$ ispatı, SVD'nin özdeğere ne zaman eşit olduğu, Ders 15'in gizemi ve ayrı düğümlerde Weyl eşitsizliği ile nuclear norm → matris tamamlama (Netflix) dalları (Şekil 23.1).

> **Builder Notu — Türevden Netflix'e**
>
> - **$d\sigma/dt = u^T(dA/dt)v$** — PCA'nın kararlılığı: veri biraz değişirse ana bileşenlerin "önemi" (σ) ne kadar kayar; tekil vektör türevini gerektirmez.
> - **Komütasyon tuzağı** — matris kalkülüsünde sıra korunur; backprop'ta (Ders 27) Jacobian'ların sırası bu yüzden önemli.



Şekil 23.1: Ders 16 kavram haritası: tekil değer türevleri ve Weyl merkezinden ana fikirlere dallar — komütasyon tuzağı $d(A^2)/dt = A(dA/dt) + (dA/dt)A$ olur, $2A(dA/dt)$ DEĞİL çünkü matrisler genelde değişmeli (commute) değildir; günün formülü tekil değer türevi $d\sigma/dt = u^T(dA/dt)v$ özdeğer formülünün ikizidir; ispatı $\sigma = u^TAv$ yazıp türev alınca iki yan terim ($\dot{u}^TAv$ ve $u^TA\dot{v}$) AYRI AYRI sıfırlanır; SVD özdeğere ancak matris simetrik ve pozitif (yarı-)tanımlıysa eşittir; Ders 15'in büyük itme gizemi yeniden etiketleme ile çözülür; ayrı düğümler Weyl eşitsizliği $\lambda_{i+j-1}(S+T) \le \lambda_i(S) + \lambda_j(T)$ ile nuclear norm = σ toplamı → düşük-rank matris tamamlama (Netflix) sonucunu gösterir.

- **Nuclear norm + matris tamamlama** — Netflix öneri problemi; eksik matris hücrelerini doldurmak için $\sum\sigma_i$'yi minimize et (düşük-rank çözüm). ℓ^1'in seyreklik (sparsity) etkisinin matris versiyonu.
- **Geriye köprü:** Ders 15 (özdeğer türevi paraleli), Ders 6 (SVD, $Av=\sigma u$), Ders 7-8 (nuclear norm, Eckart-Young).

Tek cümle: Özdeğer türevi formülünün tam ikizi tekil değer için geçerlidir ($d\sigma/dt = u^T(dA/dt)v$), ve sonlu değişimlerde özdeğer/tekil değerleri Weyl eşitsizliği (interlacing'in genel hali) sınırlar.

23.2 Komütasyon Tuzağı: $d(A^2)/dt$

Isınma sorusu: A^2 nin türevi nedir? İçgüdü "$2A(dA/dt)$" der — ama YANLIŞ. Neden? Türevi tanımdan kur: limit al.

$$\frac{d(A^2)}{dt} = \lim_{\Delta t \to 0} \frac{(A+\Delta A)^2 - A^2}{\Delta t}$$

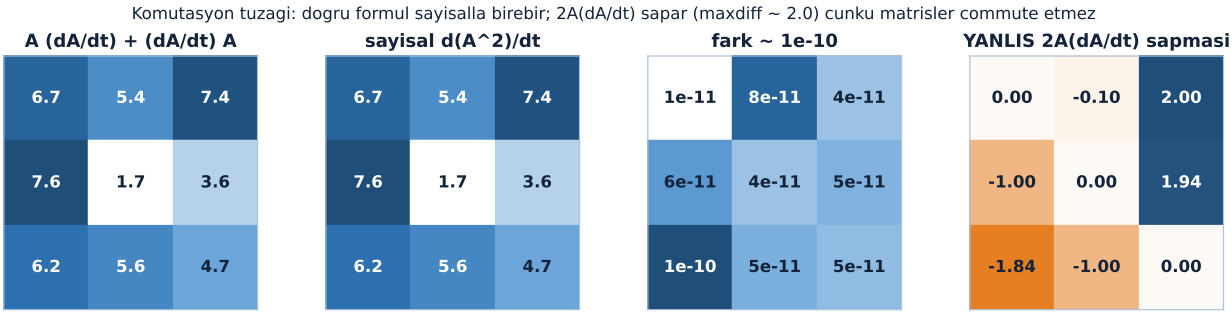
$(A + \Delta A)^2$ açılınca $A^2 + A\Delta A + \Delta A \cdot A + \Delta A^2$. A^2 sadeleşir, ΔA^2 ihmal edilir. İki orta terim kalır — ve bunlar **aynı değil**:

> *"…one term is A delta A, and another term is delta A A."* — Strang, 5:01

$$\frac{d(A^2)}{dt} = A\frac{dA}{dt} + \frac{dA}{dt}A$$

$2A(dA/dt)$ olmamasının sebebi: matrisler genelde **değişmeli (commute) değildir**, $A\cdot(dA/dt) \neq (dA/dt)\cdot A$. Skalerde iki terim birleşip $2A$ olurdu; matriste sıra korunur.

Aşağıdaki dört panel bunu sayısal olarak gösterir: doğru formül $A\,\frac{dA}{dt} + \frac{dA}{dt}\,A$ ile merkez-fark $d(A^2)/dt$ birebir örtüşür (fark ~ 1e-10), oysa yanlış içgüdü $2A\frac{dA}{dt}$ sapar (maxdiff ~ 2.0) — çünkü A(t) yolunda matrisler commute etmez (Şekil 23.2).



Şekil 23.2: Komütasyon tuzağı: doğru formül $A\,\dot{A} + \dot{A}\,A$ sayısal $d(A^2)/dt$ ile birebir (fark ~ 1e-10); yanlış içgüdü $2A\dot{A}$ sapar (maxdiff ~ 2.0) çünkü matrisler commute etmez.

> **Builder Notu — Sıra Bozulmaz Çünkü Commute Yok**
>
> "Matrisler commute etmez, sıra korunur" matris kalkülüsünün birinci kuralı. ML köprüsü: backprop'ta (Ders 27) zincir kuralı Jacobian'ları belirli sırada çarpar — sırayı bozmak yanlış gradyan verir. Çarpım kuralının skaler sezgisine körü körüne güvenmek en sık matris-kalkülüs hatasıdır.

23.3 Tekil Değer Türevi: dσ/dt = u^T(dA/dt)v

Ders 15 özdeğer türevini verdi: dλ/dt = y^T(dA/dt)x. Şimdi aynı soruyu tekil değer için sor: A değişirken σ nasıl değişir? Cevap, özdeğer formülünün tam ikizi:

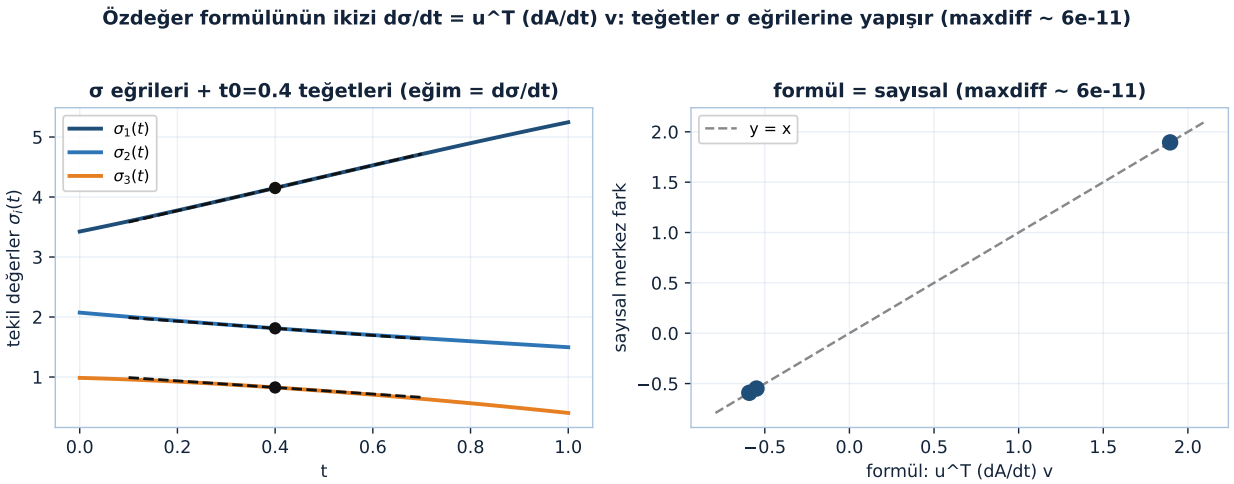
$$\frac{d\sigma}{dt} = u^T \frac{dA}{dt} v$$

Burada u sol tekil vektör, v sağ tekil vektör (Ders 6'nın A = UΣV^T ayrışımından). Özdeğer formülünde sol/sağ özvektör (y, x) vardı; burada sol/sağ tekil vektör (u, v).

> *"Boy, you couldn't ask for a nicer formula than that, right?"* — Strang, 7:36

Soldaki panel A(t) yolu boyunca üç $\sigma_i(t)$ eğrisini çizer; t0=0.4'te her eğriye çizilen teğetin eğimi formülün verdiği türevdir ve teğet eğriye yapışır. Sağdaki panel formül değeri ile sayısal merkez farkı y=x üzerinde örtüştürür (maxdiff ~ 6e-11) (Şekil 23.3).

> **Builder Notu — Özdeğer Formülünün İkizi**
>
> dσ/dt = u^T(dA/dt)v, PCA ve düşük-rank yaklaşımın kararlılık analizidir: veri matrisi biraz değişince ana bileşenlerin "ağırlığı" (σ) ne kadar kayar — tekil vektör türevini hesaplamadan. Öneri sistemleri ve gürültü gidermede, küçük veri güncellemelerinin sıralamayı bozup bozmadığını bu formül söyler.



Şekil 23.3: Özdeğer formülünün ikizi: dσ/dt = u^T(dA/dt)v. Solda üç $\sigma_i(t)$ eğrisine t0=0.4'te çizilen teğetlerin eğimi formülün verdiği türevdir (teğetler eğriye yapışır); sağda formül değeri ile sayısal merkez fark y=x üzerinde örtüşür (maxdiff ~ 6e-11).

23.4 İspat: σ = u^TAv ve Ayrı Ayrı İptal

Önce σ için bir formül. SVD'den Av = σu (Ders 6). Soldan u^T ile çarp:

> *"So I believe that sigma is u transpose times A times v."* — Strang, 8:25

$$\sigma(t) = u^T(t)\, A(t)\, v(t)$$

Doğrula: u^TAv = u^T(σu) = σ(u^Tu) = σ·1 = σ ✓ (çünkü u^Tu = 1). Şimdi türevini al — üç t-bağımlı çarpan, çarpım kuralı:

$$\frac{d\sigma}{dt} = \frac{du^T}{dt}Av + u^T\frac{dA}{dt}v + u^TA\frac{dv}{dt}$$

Orta terim cevap. Diğer ikisi sıfır — ama Ders 15'ten farklı olarak burada **ayrı ayrı** yok olurlar (özdeğerde birbirini götürüyorlardı). Birinci terim: Av = σu, yani (du^T/dt)Av = σ(du^T/dt)u. Bu, u^Tu = 1'in türevinin yarısı → 0. Üçüncü terim: A^Tu = σv, yani u^TA(dv/dt) = σv^T(dv/dt). Bu da v^Tv = 1'in türevinden → 0.

> *"Now, they disappear separately, leaving the right answer."* — Strang, 16:19

Geriye sade formül kalır: dσ/dt = u^T(dA/dt)v.

> **Builder Notu — İki Kısıt İki Ayrı İptal**
>
> Özdeğer durumunda iki terim **birlikte** (d(y^Tx)/dt = 0 ile) iptal oluyordu; tekil değerde **ayrı ayrı** (u^Tu = 1 ve v^Tv = 1 iki ayrı kısıt). Sebep: SVD'nin u ve v'si ayrı ayrı birim uzunlukta, oysa özdeğerin y ve x'i sadece çiftleşik (y^Tx = 1). Bu fark, SVD'nin neden özdeğerden daha "kararlı" olduğunun ipucu.

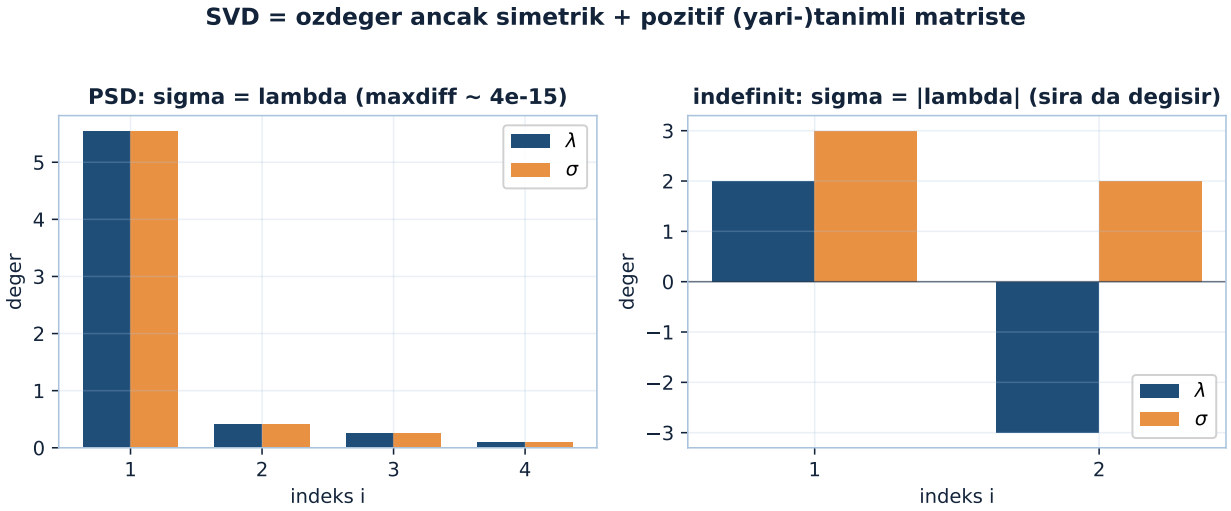
23.5 SVD ve Özdeğer Ne Zaman Aynı?

$\lambda = y^T Ax$ ve $\sigma = u^T Av$ neredeyse aynı görünüyor. Ne zaman birebir çakışırlar (tekil değerler = özdeğerler, tekil vektörler = özvektörler)? Cevap Strang'ın favori iki kelimesi:

> *"…write down positive definite in the answer to any question, because sigmas are by definition positive."* — Strang, 11:13

Simetrik **ve** pozitif (yarı-)tanımlı matrisler. Çünkü tekil değerler tanımı gereği ≥ 0; özdeğerlerle çakışmaları için özdeğerlerin de ≥ 0 (pozitif yarı-tanımlı) olması gerek. Simetrik ama işaretsiz bir matriste $\sigma = |\lambda|$ olur (negatif özdeğerler mutlak değere döner).

Sol panel pozitif yarı-tanımlı bir S'de tekil değerlerin özdeğerlere birebir oturduğunu (maxdiff ~ 4e-15) gösterir; sağ panel diag(2, −3) gibi belirsiz (indefinit) bir matriste λ=[2, −3] iken σ=[3, 2] olduğunu — negatif özdeğerin mutlağa katlandığını ve sıranın da değiştiğini — gösterir (Şekil 23.4).



Şekil 23.4: SVD özdeğer ayrıştırmasına ne zaman eşittir? Sol panel pozitif yarı-tanımlı bir matriste tekil değerlerin özdeğerlere birebir oturduğunu (maxdiff ~ 4e-15), sağ panel ise belirsiz (indefinit) bir matriste tekil değerlerin özdeğerlerin mutlak değeri olduğunu ve sıralamanın değiştiğini gösterir.

> **Builder Notu — Emin Değilsen Pozitif Tanımlı Yaz**
>
> "Soruda emin değilsen 'pozitif tanımlı' yaz" Strang'ın esprili kuralı ama derin: pozitif tanımlılık (Ders 5) SVD ile özdeğer ayrışımını birleştiren köprüdür. ML'de Gram matrisi $A^T A$, kovaryans matrisi ve kernel matrisleri hep pozitif yarı-tanımlıdır — bu yüzden onların SVD'si ve özdeğer ayrışımı aynıdır, hesap tek seferde iş görür.

23.6 Ders 15'in Gizemi Çözülüyor

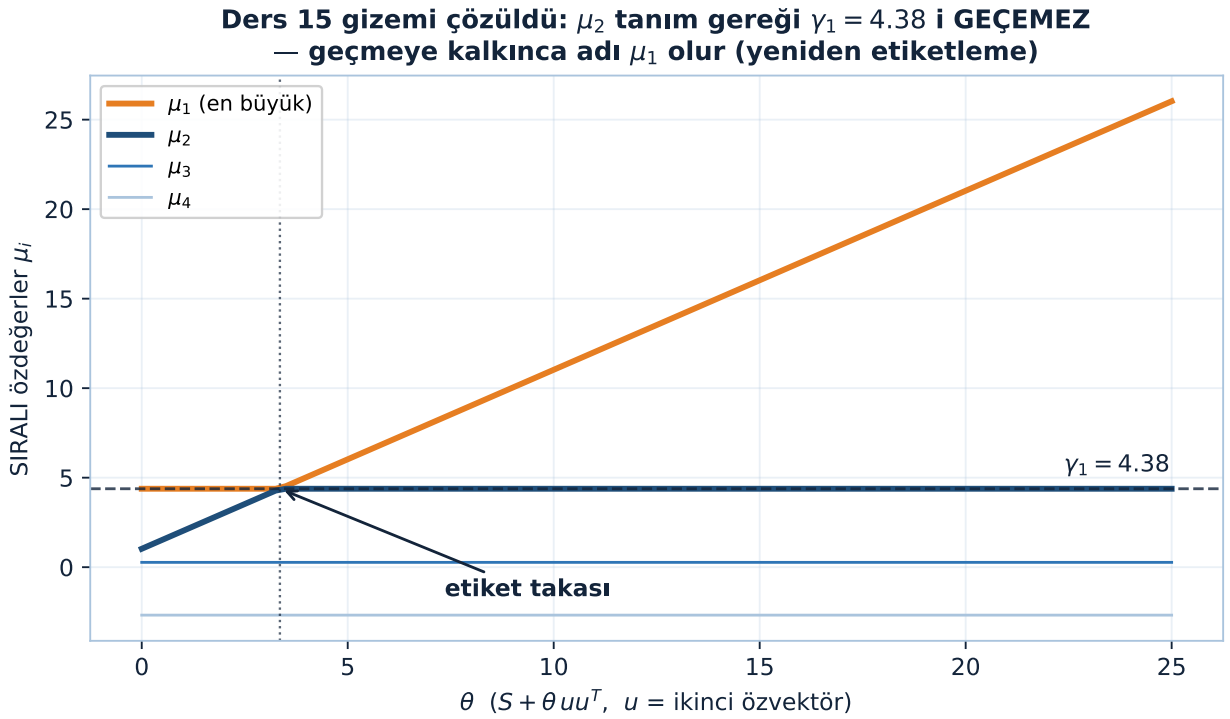
Ders 15 bir bilmeceyle bitmişti: u = S'nin ikinci özvektörü ise, $S + \theta uu^T$ o özvektörün özdeğerini $\lambda_2 + \theta$ yapar. θ büyükse (20, 200, 2000) bu değer λ_1'i geçer — interlacing'i bozar gibi görünür ($\mu_2 \le \lambda_1$ olmalıydı).

Çözüm:

> *"…if I increase theta beyond that, then this becomes not mu 2 any more, but mu 1."* — Strang, 25:20

İncelik şu: λ_2 + θ değeri λ_1'i geçtiği an, artık **ikinci** özdeğer (μ_2) değildir — **birinci** özdeğer (μ_1) olur. Özdeğerler azalan sırada etiketlenir; bir değer yukarı tırmanıp komşusunu geçince etiket de değişir. μ_2 tanım gereği λ_1'i asla geçmez, çünkü geçtiği anda adı μ_1 olur. Çelişki yok — sadece yeniden etiketleme.

Aşağıdaki figür bu çözümü **etiket perspektifi**nden gösterir: μ_2 tanım gereği γ_1=4.38'i geçemez (θ≈*3.35'te ona yapışır), θ* ardından fırlayan değer artık μ_1 olarak devam eder. Ders 15'in fig-buyuk-itme'si KİMLİK perspektifiydi (belirli özvektörün özdeğeri komşusunu geçer, eğriler kesişir); bu figür ETİKET perspektifidir (sıralı μ_i eğrileri kesişMEZ, yapışır) (Şekil 23.5).



Şekil 23.5: Ders 15 gizemi çözüldü — etiket perspektifi. $S + \theta\, uu^T$'nin SIRALI özdeğerleri: μ_2 tanım gereği $\gamma_1 = 4.38$ i geçemez ($\theta^\star \approx 3.35$'te ona yapışır), $\theta^\star$ ardından fırlayan değer artık μ_1 olarak devam eder. D15'in fig-buyuk-itme'si KİMLİK perspektifiydi (eğriler kesişir); bu ETİKET perspektifi — eğriler kesişmez, yapışır.

Builder Notu — Etiket Kayar Kimlik Kalır

"Sıralı eşitsizlik etiket korur, kimlik değil" — interlacing bir özvektörü değil, sıralı konumu izler. ML köprüsü: online PCA'da bir bileşen başka bir bileşeni geçince "birinci ana bileşen" rolü el değiştirir; algoritmanın izlediği şey bileşen kimliği değil, varyans sırasıdır. Bu, takip (tracking) problemlerinde etiket karışmasının (label switching) kaynağıdır.

23.7 Weyl Eşitsizliği: Interlacing'in Genel Hali

Interlacing rank-1 değişim içindi. Genel rank için (S simetrik, T herhangi simetrik değişim) Weyl'in büyük eşitsizliği:

> *"…after the discoverer, Weyl's inequality."* — Strang, 26:19

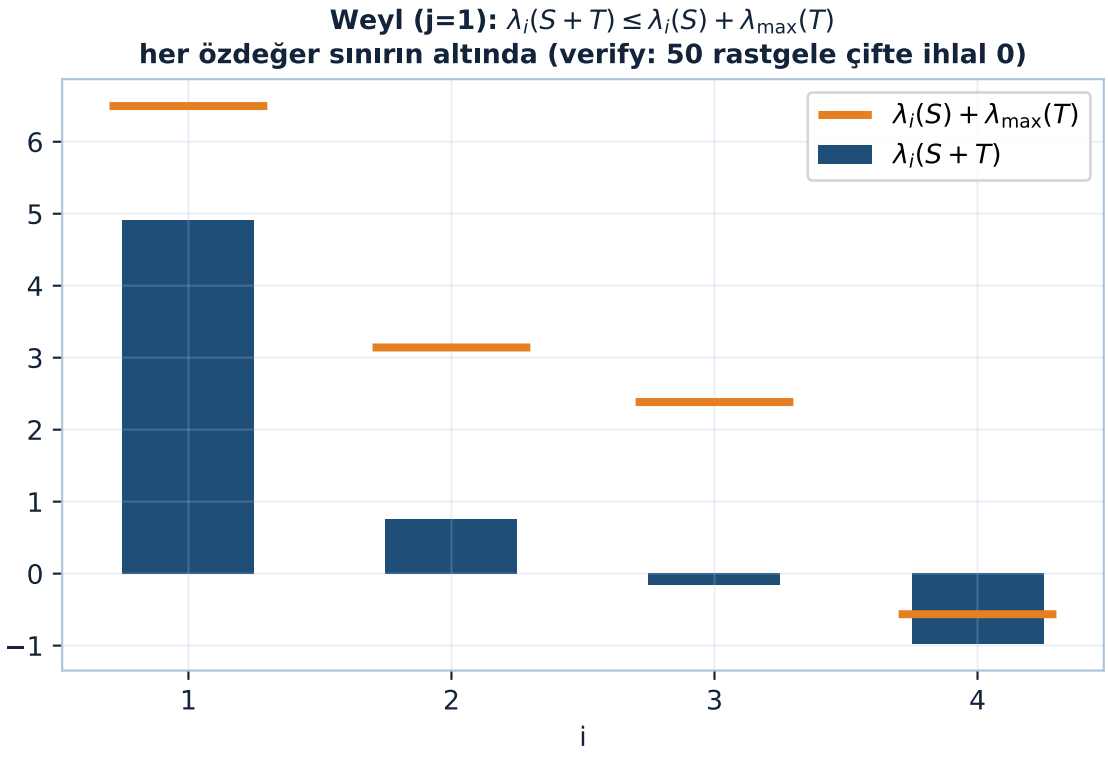
$$\lambda_{i+j-1}(S+T) \le \lambda_i(S) + \lambda_j(T)$$

Tüm interlacing buradan düşer. j = 1 al: $\lambda_i(S+T) \le \lambda_i(S) + \lambda\_\max(T)$ — T pozitifse özdeğerler yukarı çıkar ama en fazla T'nin en büyük özdeğeri kadar. Farklı i, j seçimleri komşu-olmayan özdeğerler hakkında da bilgi verir. Weyl aynı zamanda tekil değerleri keşfeden isimlerden; eşitsizliğin tekil değer versiyonu da geçerli.

$$\lambda_i(S+T) \le \lambda_i(S) + \lambda_{\max}(T) \quad (j=1)$$

Notlarda iki ispat var: standart Weyl yaklaşımı ve Prof. Rao'nun grafik tabanlı argümanı.

Aşağıdaki figür j=1 halini gösterir: navy çubuklar S+T'nin sıralı özdeğerleri, turuncu yatay çizgiler $\lambda_i(S)+\lambda\_\max(T)$ üst sınırı — her çubuk kendi sınırının altında kalır. Motor bunu 50 rastgele simetrik (S, T) çifti üzerinde de tarar ve ihlal 0 sayar (Şekil 23.6).



Şekil 23.6: Weyl eşitsizliği (j=1): S aynı motor matrisi, T rastgele simetrik. Navy çubuklar S+T'nin sıralı özdeğerleri, turuncu yatay çizgiler $\lambda_i(S)+\lambda\_\max(T)$ üst sınırı — her çubuk kendi sınırının altında. Motor 50 rastgele simetrik (S,T) çifti üzerinde ihlal 0 sayar.

> **Builder Notu — Gürültüye Karşı Spektral Sigorta**
>
> Weyl eşitsizliği perturbasyon teorisinin temel taşı: bir matrise gürültü/değişim T eklenince özdeğerlerin ne kadar kayabileceğini kesin sınırlar. ML köprüsü: gürültülü veride PCA'nın güvenilirliği, spektral kümelemenin (Ders 35) gürültüye dayanıklılığı ve düşük-rank yaklaşımın hata analizi hep Weyl-tipi sınırlara dayanır. "Küçük değişim → küçük özdeğer kayması" güvencesi olmadan spektral yöntemler kırılgan olurdu.

23.8 Nuclear Norm ve Matris Tamamlama (Önizleme)

Türevler bitti; ders yeni bir bölüme (compressed sensing) köprü kurar. Anahtar nesne nuclear norm:

> *"The nuclear norm a matrix is the sum of the singular values."* — Strang, 35:04

$$\|A\|_* = \sigma_1 + \sigma_2 + \cdots + \sigma_r$$

Bu, vektördeki ℓ^1 normun matris karşılığıdır (Ders 8). ℓ^1'in sihirli özelliği: bir kısıt altında minimize edilince **seyrek** (sparse) çözüm verir. Nuclear norm da matrisler için **düşük-rank** çözüm verir. Uygulaması matris tamamlama:

> *"And I'll remember the words Netflix, which made the problem famous."* — Strang, 36:26

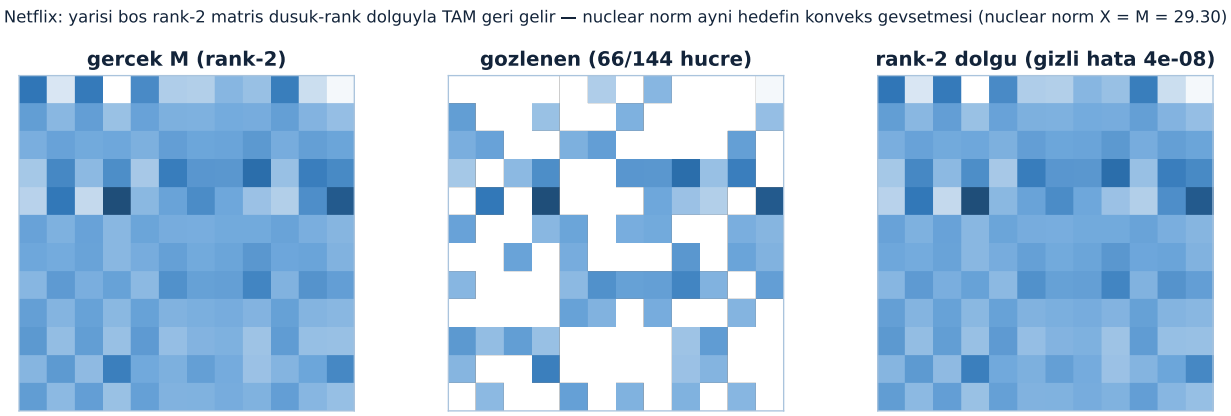
Netflix problemi: kullanıcı×film matrisinin çoğu hücresi boş (herkes her filmi izlemedi). Eksik hücreleri nasıl doldurursun? Fikir: nuclear normu minimize eden dolgu, en düşük-rank (en "açıklayıcı") matristir. Neden nuclear norm? Çünkü asıl istediğimiz **rank**'i minimize etmek — ama rank bir norm değil (matrisi 2 ile çarpınca rank değişmez). rank, vektördeki ℓ^0 normun (sıfır-olmayan eleman sayısı) matris eşi; ikisi de norm değil. Konveks gevşetme: $\ell^0 \to \ell^1$, rank → nuclear norm.

> *"…gradient descent finds the solution to the minimum problem in the nuclear norm."* — Strang, 42:02

Açık bir conjecture: derin öğrenmedeki gradient descent'in (Ders 22+) doğal olarak minimum-nuclear-norm (düşük-rank) çözümü bulduğu düşünülüyor — ama henüz kanıtlanmadı (implicit bias, Ders 10).

Aşağıdaki üç panel demoyu gösterir: gerçek rank-2 matris M, yarısı boş gözlem (66/144 hücre, %46) ve düşük-rank dolgu. Demo dolgusu **rank-2 hard impute** (dönüşümlü projeksiyon) kullanır — bu, nuclear-norm hedefinin "sert" halidir; küçük demoda kesin yöntemdir. Prose'da anlatılan nuclear norm minimizasyonu bu hedefin **konveks gevşetmesi**dir. İki yöntem aynı düşük-rank çözüme götürür: dolgu gizli hücreleri 4.4e-08 hatayla TAM geri getirir ve nuclear norm eşitliği ‖X‖∗ = ‖M‖∗ = 29.30 tutar (Şekil 23.7).

> **Builder Notu — Rankın Konveks Gölgesi**
>
> rank → nuclear norm gevşetmesi modern ML'in temel hilesi: kombinatoryal (NP-zor) bir hedefi (rank minimize) konveks (çözülebilir) bir hedefe çevirir. Netflix Prize bu yöntemi ünlü etti. Köprü: LoRA (düşük-rank adaptasyon) aynı düşük-rank önyargısını fine-tuning'e taşır; "gradient descent düşük-rank bulur" conjecture'ı derin öğrenmenin neden genelleştirdiğini açıklama çabalarının merkezinde.



Şekil 23.7: Netflix: yarısı boş rank-2 matris düşük-rank dolguyla TAM geri gelir. Ortadaki dolgu rank-2 hard impute (nuclear-norm hedefinin sert hali; küçük demoda kesin yöntem) ile hesaplanır — nuclear norm minimizasyonu aynı hedefin konveks gevşetmesidir ($\|X\|* = \|M\|* = 29.30$, gizli hata 4e-08).

23.9 Bu Dersin Özeti

- **Komütasyon tuzağı:** $d(A^2)/dt = A(dA/dt) + (dA/dt)A$, **$2A(dA/dt)$ değil** — matrisler değişmeli değil.
- **Tekil değer türevi:** $d\sigma/dt = u^T(dA/dt)v$ (sol tekil vektör u, sağ tekil vektör v). Özdeğer formülünün ikizi.
- **İspat:** $\sigma = u^TAv$ ($Av = \sigma u$, $u^Tu = 1$). Çarpım kuralı; iki yan terim **ayrı ayrı** sıfır ($u^Tu = 1$ ve $v^Tv = 1$ — özdeğerdeki gibi birlikte değil).
- **SVD = özdeğer ne zaman:** simetrik + pozitif (yarı-)tanımlı.
- **Ders 15 gizemi:** μ_2 büyüyüp λ_1'i geçince artık μ_1 olarak yeniden etiketlenir — çelişki yok, sadece sıralama etiketi kayar.
- **Weyl eşitsizliği:** $\lambda_{i+j-1}(S+T) \le \lambda_i(S) + \lambda_j(T)$; interlacing'in genel hali (j=1 özel durum). Tekil değerler için de geçerli.
- **Nuclear norm:** $\|A\|* = \Sigma\sigma_i$ (matrisler için ℓ^1). rank $\to$ nuclear norm konveks gevşetmesi; matris tamamlama (Netflix), compressed sensing.

! Tek Bir Cümle

Özdeğer türevinin tam ikizi tekil değer için $d\sigma/dt = u^T(dA/dt)v$'dir (iki yan terim ayrı ayrı iptal); sonlu değişimleri Weyl eşitsizliği sınırlar, ve nuclear norm minimizasyonu düşük-rank matris tamamlamanın (Netflix) anahtarıdır.

23.10 Kontrol Soruları

> **Soru 1: d(A²)/dt neden 2A(dA/dt) değil**
>
> **d(A^2)/dt neden 2A(dA/dt) değildir? Doğru formül nedir?**
> Çünkü matrisler genelde değişmeli (commute) değildir: A·(dA/dt) ≠ (dA/dt)·A. $(A+\Delta A)^2$ açılımında iki orta terim AΔA ve ΔA·A ayrıdır, birleşip 2AΔA olmazlar. Doğru formül: d(A^2)/dt = A(dA/dt) + (dA/dt)A. Sıra korunmalıdır.

> **Soru 2: İki yan terim nasıl yok olur**
>
> **dσ/dt = u^T(dA/dt)v ispatında iki yan terim özdeğer durumundan farklı olarak nasıl yok olur?**
> σ = u^TAv'in türevinde üç terim çıkar. Birinci terim Av = σu kullanılınca σ(du^T/dt)u = (σ/2)·d(u^Tu)/dt; u^Tu = 1 sabit, türevi 0. Üçüncü terim A^Tu = σv ile σv^T(dv/dt); v^Tv = 1, türevi 0. Yani iki terim **ayrı ayrı** (iki bağımsız birim-uzunluk kısıtından) sıfır olur — özdeğerde ise tek bir y^Tx = 1 kısıtından **birlikte** götürüyorlardı.

> **Soru 3: Interlacing neden bozulmaz**
>
> **u = S'nin ikinci özvektörü ve S + 20uu^T ikinci özdeğeri çok yukarı itiyorsa, interlacing neden bozulmaz?**
> λ_2 + 20 değeri λ_1'i geçtiği anda, azalan-sıra etiketlemesi gereği artık ikinci özdeğer (μ_2) değil, **birinci** özdeğer (μ_1) olur. μ_2 tanım gereği λ_1'i geçemez — çünkü geçtiği an adı μ_1'e döner. Interlacing özvektör kimliğini değil, sıralı konumu izler; çelişki yoktur, sadece yeniden etiketleme vardır.

> **Soru 4: Neden rank yerine nuclear norm**
>
> **Matris tamamlamada (Netflix) neden rank yerine nuclear norm minimize edilir?**
> Asıl hedef en düşük-rank dolguyu bulmak, ama rank bir norm değildir (matrisi 2 ile çarpınca rank değişmez) ve minimize etmesi NP-zordur. Nuclear norm ($\|A\|_* = \Sigma\sigma_i$) rankın konveks gevşetmesidir — vektördeki $\ell^0 \to \ell^1$ geçişinin matris karşılığı. Konveks olduğu için verimli çözülür ve pratikte düşük-rank çözüm verir.

23.11 Egzersizler

1. **Komütasyon kontrolü.** A(t) = [[t, 1], [0, t]]. A^2'yi hesapla, türevini doğrudan al. Sonra A(dA/dt) + (dA/dt)A formülüyle hesapla; eşit olduğunu, ama 2A(dA/dt)'nin farklı çıktığını göster.

   [motor notu] Bu özel ailede dA/dt = I olduğundan ve I her matrisle değişir (commute), iki formül aynı çıkar — fark görünmez. Farkı görmek için dA/dt'nin A ile değişmediği bir aile gerekir (örn. motorun a16_path'i: sapma ~2.0).

2. **Tekil değer türevi.** A(t) = [[t, 0], [0, 2]] (t > 0). σ_1 = 2, σ_2 = t (t < 2 için). dA/dt = [[1,0],[0,0]]. σ_2 = t'nin türevi dσ_2/dt = u^T(dA/dt)v ile 1 çıkar mı? (İpucu: σ_2 tekil vektörleri e_1.)

3. **Üçüncü terimi göster.** dσ/dt ispatında üçüncü terim $u^T A(dv/dt)$'nin $A^T u = \sigma v$ kullanılarak $\sigma v^T(dv/dt)$ olduğunu ve $v^T v = 1$'den sıfırlandığını adım adım yaz.

4. **Weyl → interlacing.** Weyl eşitsizliğinde $j = 1$ ve $T = \theta uu^T$ (rank-1, pozitif) al. $\lambda_i(S+T) \le \lambda_i(S) + \theta\|u\|^2$ çıktığını ve bunun "özdeğerler en fazla $\theta\|u\|^2$ kadar yukarı" interlacing ifadesini verdiğini göster.

5. **(Ders 17 habercisi — konuk Townsend)** Bu derste tekil değerlerin türevini gördük. Peki tekil değerler **ne kadar hızlı** küçülür? Bazı matrislerde σ_k üstel hızla düşer (etkin rank çok küçük). Hangi matrisler? Neden önemli? Bir tahmin yaz — Ders 17'de konuk Prof. Alex Townsend "hızla azalan tekil değerler"i işliyor.

23.12 Sonraki Ders İçin Hazırlık

Ders 17: Hızla Azalan Tekil Değerler (konuk: Alex Townsend). Strang'ın bahsettiği gibi Prof. Townsend §4.3'ü (kendi araştırma alanı) anlatır: hangi matrislerin tekil değerleri üstel hızla düşer, neden bu "sayısal düşük-rank" demektir ve Sylvester denklemleri/sınır değer problemleriyle bağı. Quote'lar konuşmacıya atfedilir ("— Townsend, X:XX").

> ⚠ Ders 17 Öncesi Yapılacak
>
> Ders 17 Strang'ın değil, **konuk Prof. Alex Townsend**'in dersidir. Tekil değer kavramı (Ders 6) ve bu dersteki dσ/dt türevi tazeyken gir; "hızla azalan tekil değer = sayısal düşük-rank" fikrinin Eckart-Young (Ders 7) ile bağını düşün. Townsend'in tüm alıntıları ona atfedilecek ("— Townsend, X:XX"), Strang'a değil.

23.13 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Komütasyon tuzağı	$d(A^2)/dt = A(dA/dt) + (dA/dt)A$, $2A(dA/dt)$ değil	5m01
Tekil değer türevi	$d\sigma/dt = u^T (dA/dt)\, v$	7m36
σ formülü	$\sigma = u^T A v$ ($Av = \sigma u$, $u^T u = 1$)	8m25
SVD = özdeğer	simetrik + pozitif (yarı-)tanımlı	11m13
Ayrı ayrı iptal	$u^T u = 1$ ve $v^T v = 1 \to$ iki yan terim 0	16m19
Gizem çözümü	μ_2 büyüyüp λ_1'i geçince μ_1 olur (yeniden etiket)	25m20
Weyl eşitsizliği	$\lambda_{i+j-1}(S+T) \le \lambda_i(S) + \lambda_j(T)$	26m19
Nuclear norm	$\|A\|_* = \Sigma\sigma_i$ (matrisler için ℓ^1)	35m04
Matris tamamlama	Netflix; nuclear norm min → düşük-rank dolgu	36m26
GD conjecture	gradient descent düşük-rank çözüm bulur (kanıtsız)	42m02

23.14 ML Bağlantıları Özeti

- **PCA kararlılığı:** $d\sigma/dt = u^T(dA/dt)v$ — veri güncellemesinde ana bileşenlerin ağırlığı ne kadar kayar; tekil vektör türevi gerekmez.
- **Matris kalkülüsü:** komütasyon tuzağı ($d(A^2)/dt$) — backprop'ta (Ders 27) Jacobian sırasının neden önemli olduğunun temeli.
- **Matris tamamlama / öneri:** Netflix problemi; nuclear norm minimizasyonu eksik matrisleri düşük-rank ile doldurur.
- **Konveks gevşetme:** rank → nuclear norm ($\ell^0 \to \ell^1$ matris versiyonu); NP-zor hedefi çözülebilir kılar.
- **Implicit bias / LoRA:** gradient descent'in düşük-rank bulma conjecture'ı + düşük-rank fine-tuning (LoRA); derin öğrenmenin genelleme gizemiyle bağlı.
- **Perturbasyon sınırları:** Weyl eşitsizliği — gürültülü veride spektral yöntemlerin (PCA, kümeleme) güvenilirlik garantisi.
- **Geriye köprü:** Ders 15 (özdeğer türevi paraleli + açık gizem), Ders 6 (SVD $Av=\sigma u$), Ders 8 (nuclear/ℓ^1 norm), Ders 5 (pozitif tanımlı), Ders 10 (implicit bias).

> **!** Tek Şey
>
> *"Boy, you couldn't ask for a nicer formula than that, right?"* — Strang, 7:36 — Tekil değer türevi $d\sigma/dt = u^T(dA/dt)v$, özdeğer formülünün ikizi; sonlu değişimleri Weyl sınırlar, ve nuclear norm düşük-rank dünyasının kapısını açar.

24 Hızla Azalan Tekil Değerler

Konuk ders, Alex Townsend: düşük-rank dünyası, sayısal rank ve Sylvester denklemleri

> **i** Bölüm bilgisi
>
> **Video:** MIT 18.065 — Rapidly Decreasing Singular Values (konuk: Alex Townsend) · **OCW:** Lecture 17 · **Okuma süresi:** ≈35 dk · **Konuk:** Alex Townsend (Cornell) · **Takdim:** Gilbert Strang · **Önkoşul:** Ders 16 (tekil değer türevi).
> Bu bir **konuk dersidir**. Cornell'den Prof. Alex Townsend (eskiden MIT'de 18.06 hocası), kendi araştırma alanı olan §4.3'ü anlatır. Ana içerik **Townsend'e** atfedilir; yalnız açılış takdimi Strang'a.

24.1 Bu Derste Ne Var?

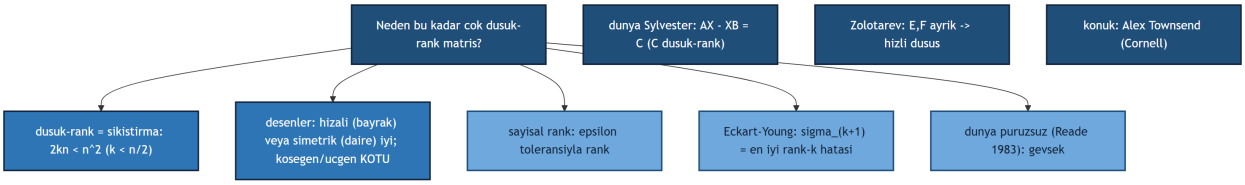
Büyük soru: **neden dünyada bu kadar çok düşük-rank matris var?** Townsend bu hikayeyi anlatır. Önce düşük-rank'ın ne demek olduğunu (sıkıştırma), sonra **sayısal rank** kavramını ve son olarak iki derin açıklamayı ("dünya pürüzsüz" → "dünya Sylvester") görürüz.

Beş sonuç:

1. **Düşük-rank = sıkıştırma:** rank-k matris n^2 yerine 2kn sayıyla gönderilir; $k < n/2$ ise düşük-rank.
2. **Hangi matrisler:** ızgaraya hizalı (bayrak) veya simetrik (daire) olanlar düşük-rank; köşegen/üçgensel desenler **kötü**.
3. **Sayısal rank (rank_ε):** ε toleransı içinde; tam-rank matrisler bile tekil değerleri hızla düşüyorsa düşük **sayısal** rank'a sahip (Hilbert, Vandermonde).
4. **"Dünya pürüzsüz":** pürüzsüz fonksiyon örneklemesi polinomla yaklaşılabilir → düşük sayısal rank (Reade 1983) — ama Hilbert için zayıf (719 vs gerçek 28).
5. **"Dünya Sylvester":** $AX - XB = C$ denklemini sağlayan matrisler; Zolotarev sayısı sınır verir (Hilbert: 34).

> *"Today I want to tell you a little about why there [are] so many matrices that are low rank in the world."* — Townsend, 1:15

Aşağıdaki kavram haritası dersin merkezini ("neden bu kadar çok düşük-rank matris?") ana fikirlere bağlar: düşük-rank = sıkıştırma, hizalı/simetrik desenler iyi (köşegen/üçgen kötü), sayısal rank, Eckart-Young, dünyanın pürüzsüzlüğü (Reade gevşek) ve ayrı düğümlerde dünyanın Sylvester yapısı ile Zolotarev düşüşü — bir düğüm de bunun bir konuk dersi olduğunu (Alex Townsend, Cornell) işaretler (Şekil 24.1).



Şekil 24.1: Ders 17 kavram haritası (konuk: Alex Townsend, Cornell): "Neden bu kadar çok düşük-rank matris?" sorusu merkezinden ana fikirlere dallar — düşük-rank demek sıkıştırma demektir, çünkü rank-k matris yalnızca 2kn sayıyla saklanır ve bu n^2'den küçük olduğunda (yani $k < n/2$) yer kazanılır; hangi matrislerin düşük-rank olduğunu desen belirler: hizalı desenler (bayraklar) veya simetrik yapılar (daireler) iyi sıkışır, ama köşegen ve üçgen desenler sıkışmaz; sayısal rank bir ϵ toleransının üstündeki tekil değerleri sayar, böylece tam rank büyük olsa bile etkin rank küçük olabilir; Eckart-Young teoremi en iyi rank-k yaklaşımının hatasını $\sigma_{(k+1)}$'e bağlar; Townsend'in gözlemi (Reade 1983) dünyanın çoğu matrisinin pürüzsüz fonksiyonlardan geldiği ve bu yüzden düşük-rank olduğu, ama bu sınırın gevşek kaldığıdır; ayrı düğümler Hilbert matrisinin bir Sylvester denklemi ($AX - XB = C$, C düşük-rank) sağladığını ve Zolotarev sayılarının E,F kümeleri ayrık olduğunda hızlı tekil-değer düşüşünü açıkladığını gösterir.

> **Builder Notu — Sıkıştırılabilir Dünyanın Haritası**
>
> - **Düşük-rank = sıkıştırma** — görüntü sıkıştırma (Ders 7 Eckart-Young), model sıkıştırma, LoRA: hepsi "az parametreyle çok bilgi" fikrini kullanır.
> - **Sayısal rank** — gerçek dünyada matrisler tam düşük-rank değil, **sayısal** düşük-rank; ML'de veri matrisleri ve ağırlıklar neredeyse hep böyle.
> - **Sylvester denklemi** — bir matrisin neden sıkıştırılabilir olduğunu önceden söyler; sayısal lineer cebirin derin aracı.
> - **Geriye köprü:** Ders 6 (SVD = $\sum \sigma_i u_i v_i^T$), Ders 7 (Eckart-Young, en iyi rank-k), Ders 16 (tekil değer türevi).

Tek cümle: Dünyada bu kadar çok (sayısal) düşük-rank matris vardır çünkü bunlar pürüzsüz fonksiyonlardan veya Sylvester denklemlerinden doğar; tekil değerlerin hızla düşmesi (Zolotarev sayısıyla sınırlı) onları sıkıştırılabilir kılar.

24.2 Strang'ın Takdimi

Strang konuğu tanıtır: Alex Townsend, MIT'de 18.06 dersini başarıyla vermiş, şimdi Cornell'de. §4.3 tamamen onun çalışması — yaratıcısından dinleme fırsatı.

> *"And now you get to hear from the creator himself."* — Strang, 1:00

Buradan sonra ana içerik Townsend'e aittir; quote'lar "— Townsend" diye atfedilir.

> **Builder Notu — Yaratıcısından Birinci Ağızdan**
>
> Konuk ders formatı, bir konunun yaratıcısından birinci ağızdan dinlemenin değerini gösterir. Townsend'in açısı pratik/hesaplamalı: "neden gerçek dünyada bu kadar çok düşük-rank matris var?" — teorem

değil, gözlem ve açıklama peşinde.

24.3 Neden Bu Kadar Çok Düşük-Rank Matris Var?

Townsend'in motivasyonu: hesaplamalı matematikçiler her yerde düşük-rank matrislerle karşılaşır. Neden? X bir n×n matris olsun. SVD'den (Ders 6) X'i rank-1 parçaların toplamı olarak yaz; X rank-k ise k tane parça:

> *"Today I want to tell you a little about why there [are] so many matrices that are low rank in the world."* — Townsend, 1:15

$$X = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

X rank-k demek: tekil değer dizisinde çok sayıda sıfır var ($\sigma_{k+1} = \cdots = 0$), kolon uzayı = satır uzayı boyutu = k. Soru: hangi X'ler bu sıfırları üretir?

> Builder Notu — Rank-1 Parçaların Ekonomisi
>
> "Rank-k = k rank-1 parçanın toplamı" Ders 6-7'nin tam tekrarı, ama Townsend bunu **veri sıkıştırma** merceğinden kullanıyor: az sayıda parça = az bilgi. Bu, görüntü/sinyal sıkıştırma ve model küçültmenin temel sezgisi.

24.4 Düşük-Rank = Sıkıştırma

Townsend'in düşük-rank tanımı somut: X'i bir resim gibi düşün (her eleman bir piksel). Arkadaşına iki türlü gönderebilirsin. Ya tüm elemanları — n^2 sayı. Ya da düşük-rank formda $u_1, v_1, \ldots, u_k, v_k$ vektörlerini — 2kn sayı (her vektör n uzunlukta, k çift).

> *"…a matrix is low rank if it's more efficient to send x to our friend [in low-rank form]."* — Townsend, 6:33

$$2kn < n^2 \quad \Longleftrightarrow \quad k < \frac{n}{2}$$

Katı tanım: rank, boyutun yarısından küçükse matris düşük-rank. Pratikte daha fazlasını isteriz — k, n'den **çok** küçük olsun ki sıkıştırma kayda değer olsun.

> Builder Notu — Gönder Bakalım Kaç Sayı
>
> "Göndermesi daha ucuzsa düşük-rank" tanımı bilgi-teorik: sıkıştırma = aynı bilgiyi daha az bitle taşımak. ML köprüsü: LoRA bir ağırlık güncellemesini $\Delta W = BA^T$ (düşük-rank) olarak saklar — milyonlarca parametre yerine 2kn. JPEG, PCA, otokodlayıcılar hep bu "$2kn < n^2$" pazarlığını yapar.

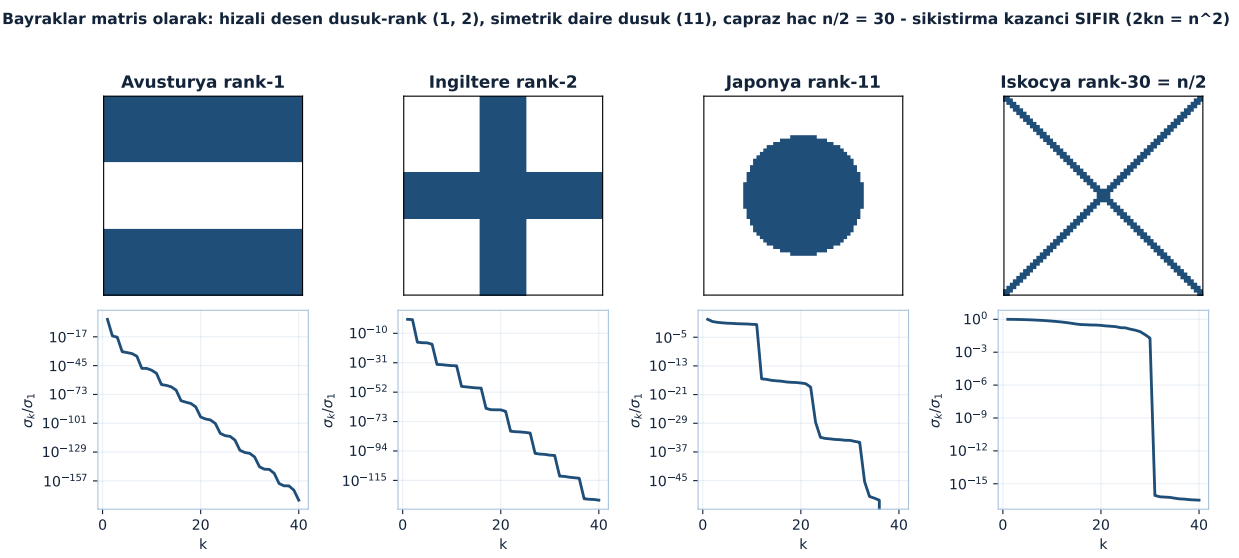
24.5 Bayraklar: Hangi Desenler Düşük-Rank?

Townsend dünya bayraklarını örnek alır (her bayrak bir matris). Avusturya bayrağı (üç yatay şerit) rank-1 — arkadaşına sadece iki vektör (bir kolon, bir satır) gönderirsin:

> *"For example, the Austria flag– if you want to send that to your friend, that matrix is of rank 1."*
> — Townsend, 8:04

İngiltere bayrağı rank-2 (u_1,v_1,u_2,v_2). Japonya bayrağı (daire) düşük-rank ama o kadar küçük değil — n=60 için rank-11. İskoçya bayrağı (çapraz haç) ise rank-30 = n/2: tam burada $2kn = 2 \cdot 30 \cdot 60 = 3600 = n^2$ olduğundan düşük-rank formda göndermenin sıkıştırma kazancı **sıfırdır**. Gözlem: matris satır/sütunlarla **hizalı** ise düşük-rank (Avusturya); köşegen/çapraz desenler kötü.

Aşağıdaki figür bayrakları matris olarak ve her birinin tekil değer profilini gösterir: Avusturya (1) ve İngiltere (2) neredeyse anında sayısal sıfıra düşer, Japonya k≈11'de uçurum yapar, İskoçya ise k≈30'a kadar ~1 kalıp orada düşer — köşegen desenin sıkışmaya direncini gözle gösterir (Şekil 24.2).



Şekil 24.2: Bayraklar matris olarak: hizalı (eksen-paralel) desenler en düşük rankı verir — Avusturya'nın üç yatay şeridi rank-1, İngiltere'nin haçı rank-2. Simetrik daire (Japonya) düşük ama daha yüksek: rank-11. Çapraz haç (İskoçya saltire) ise rank-30 = $n/2$ — burada $2kn = 2 \cdot 30 \cdot 60 = 3600 = n^2$ olduğundan düşük-rank formda göndermenin sıkıştırma kazancı SIFIR. Alt sıra her bayrağın σ_k/σ_1 profilini gösterir: Avusturya ve İngiltere neredeyse anında sayısal sıfıra düşer, Japonya k≈11'de uçurum yapar, İskoçya ise k≈30'a kadar ~1 kalıp orada düşer — köşegen desenin sıkışmaya direncini gözle gösterir.

> **Builder Notu — Bayrağına Bak Rankını Söyle**
>
> "Izgaraya hizalı → düşük-rank" sezgisi pratik bir tarama: bir veri matrisinin yapısı eksenlerle hizalıysa (bloklar, şeritler) sıkıştırılabilir. Ama Townsend birazdan bu sezginin yarı-doğru olduğunu gösterecek — daire (hizasız) da düşük-rank çıkar. Sezgiye körü körüne güvenme; tekil değerlere bak.

24.6 Üçgen Bayrak: Köşegen Neden Kötü

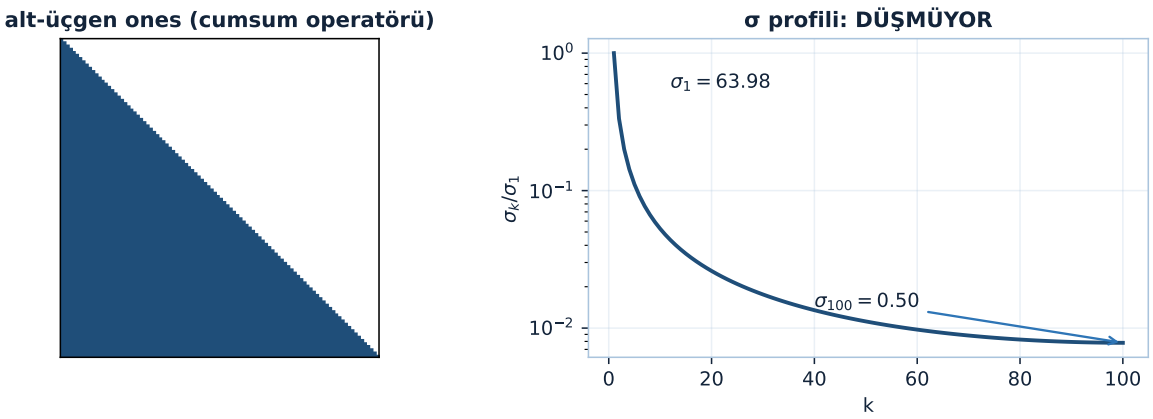
En kötü örneği incele: köşegen altı tamamen dolu (ones) üçgen matris L. Bu matrisin tersi, birinci-mertebe sonlu fark matrisine benzer; iki tersi çarpılınca Strang'ın favori −1, 2, −1 (ikinci fark) matrisi çıkar. Bu matrisin tekil değerleri çok iyi bilinir (Ders 4: A^TA'nın özdeğerleri X'in tekil değerlerini verir). Sonuç:

$$\sigma_1(L) \approx \frac{2n}{\pi}, \qquad \sigma_n(L) \approx \frac{1}{2}$$

n=100 için motor $\sigma_1 = 63.98$ ($\approx 2n/\pi = 63.66$) ve $\sigma_n = 0.5001$ ($\approx ½$) verir; en küçük oran $\sigma_n/\sigma_1 \approx 7.8\times10^{-3} \approx \pi/(4n)$, yani **düşmüyor**. Tüm tekil değerler büyük — sıfıra **yaklaşmıyorlar**. Köşegen desenler düşük-rank için felaket:

> *"So triangular patterns are extremely bad for low rank."* — Townsend, 14:27

Aşağıdaki figür solda alt-üçgen ones matrisini, sağda hiç düşmeyen σ profilini gösterir — hiçbir σ sıfıra yaklaşmadığından sayısal rank tam 100 kalır (Şekil 24.3).



Şekil 24.3: Üçgen desen düşük-rank için felaket: alt-üçgen ones (cumsum operatörü) matrisinin tekil değerleri düşmüyor. $\sigma_1 = 63.98 \approx 2n/\pi$ ve $\sigma_{100} = 0.50 \approx 1/2$ — hiçbir σ sıfıra yaklaşmıyor, tam sayısal rank 100.

> **Builder Notu — Geçmişi Unutmayan Sıkışmaz**
>
> Üçgen "ones" matrisinin tersinin sonlu farka, karesinin Strang'ın −1,2,−1 matrisine bağlanması, Phase 1 18.06'nın köşeyi tutan örneğidir — burada tekil değer analizini mümkün kılan köprü. ML köprüsü: kümülatif toplam (cumsum) operatörü tam bu üçgen matristir; tekil değerlerinin düşmemesi, "geçmişi unutmayan" işlemlerin neden sıkıştırılamadığını açıklar (uzun-bağlam dikkat mekanizmasının maliyeti).

24.7 Daire ve Simetri: Hizasız ama Düşük-Rank

Japonya bayrağı (ortada daire) ızgaraya hizalı değil — ama yine de düşük-rank. Neden? **Simetri**. Townsend daireyi parçalara böler: rank ≤ (ortadaki rank-1 kare) + (iç bölge). İç bölgeyi simetriyle tekrar tekrar yarıya böler — kolonların solu sağında, satırların üstü altında tekrarlandığından kolon/satır uzayı boyutu küçük kalır. Yarıçap r için sınır:

$$\text{rank} \lesssim \tfrac{1}{2}\, r + 1$$

Yani daire, çizgi-ızgara hizasının neredeyse zıttı olmasına rağmen düşük-rank. Ders: hizalama yeterli koşul değil; simetri de düşük-rank üretir.

> **Builder Notu — Simetri de Sıkıştırır**
>
> "Simetri → düşük-rank" gözlemi, sezginin (ızgara hizası) tek açıklama olmadığını gösterir. ML köprüsü: evrişimsel ağlar (CNN, Ders 32) tam da uzaysal simetriyi (öteleme değişmezliği) sömürerek parametre paylaşır — simetri, az parametreyle çok şey ifade etmenin bir başka yoludur. Yapı (hizalama veya simetri) her zaman sıkıştırılabilirliğe yol açar.

24.8 Sayısal Rank: ε Kadar Esneklik

Gerçek matrisler nadiren tam düşük-rank'tır — ama **sayısal** düşük-rank olurlar. Townsend sayısal rank'ı tanımlar: tam rank tanımına biraz "esneme payı" (ε toleransı) ekle.

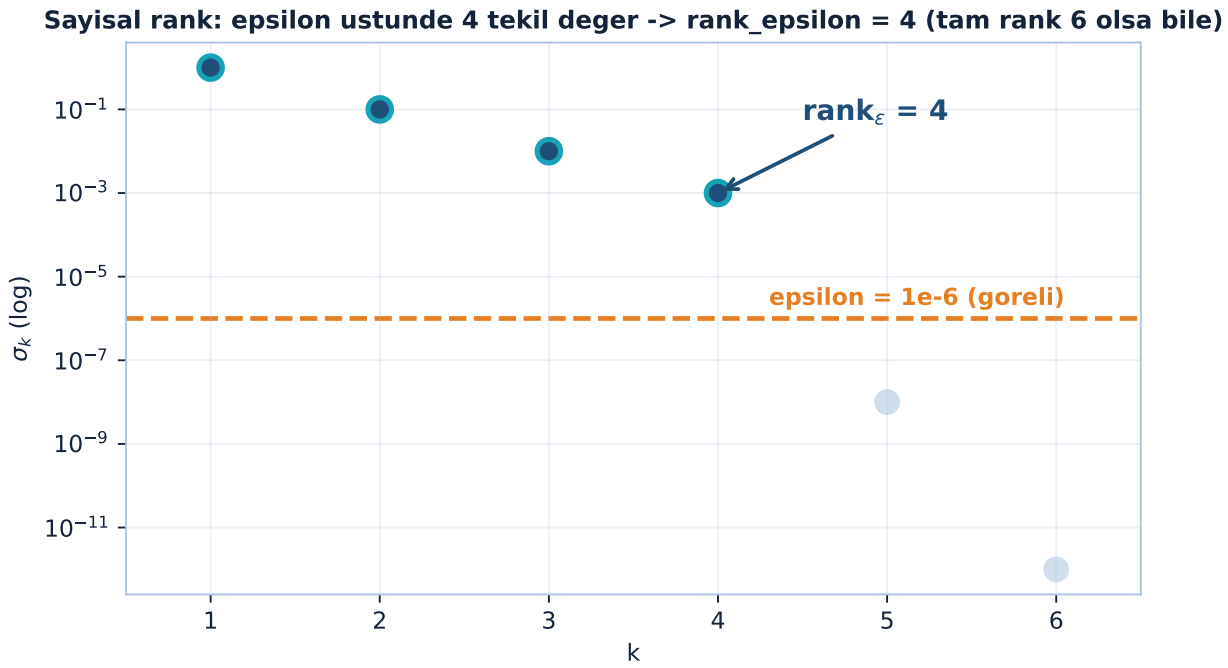
> *"…we allow ourselves a little bit of wiggle room when we define it…"* — Townsend, 19:58

$$\text{rank}_\varepsilon(X) = k \quad \Longleftrightarrow \quad \sigma_{k+1} < \varepsilon\, \sigma_1 \leq \sigma_k$$

Yani ε'dan (göreli) küçük tekil değerleri "sıfır say". Pratikte bu daha önemli: arkadaşına bayrağı 16 hane hassasiyetle göndermek yeter — gözü farkı göremez. Bilgisayar zaten her sayıyı 16 haneye yuvarlar; $\varepsilon = 10^{-15}$ ise X ile rank-k yaklaşımını ayırt edemez.

Aşağıdaki figür somut bir örnek verir: $\sigma = (1, 0.1, 0.01, 0.001, 10^{-8}, 10^{-12})$ ve $\varepsilon = 10^{-6}$ için ilk 4 tekil değer eşiğin üstünde kalır, son 2'si altına düşer — matris tam rank 6 olsa bile rank_ε = 4 (Şekil 24.4).

> **Builder Notu — Epsilon Kadar Esneklik**
>
> Sayısal rank, teorik rank'tan çok daha pratiktir: kayan-nokta aritmetiğinde "tam sıfır" diye bir şey yoktur, sadece "ε'dan küçük" vardır. ML köprüsü: bir sinir ağı ağırlık matrisinin etkin (effective) rank'ı tekil değerlerin nereye düştüğüyle ölçülür; budama (pruning) ve düşük-rank sıkıştırma tam bu ε-eşiğine dayanır. Modelin "gerçek boyutu" sayısal rank'tır, nominal boyut değil.



Şekil 24.4: Sayısal rank: ε toleransıyla, $\varepsilon \cdot \sigma_1$ eşiğinin üstünde kalan tekil değer sayısı. σ = (1, 0.1, 0.01, 0.001, 10^{-8}, 10^{-12}) ve ε = 10^{-6} için ilk 4 tekil değer eşiğin üstünde (navy + teal halka), son 2'si soluk steel ile altında — matris tam rank 6 olsa bile rank_ε = 4.

24.9 Eckart-Young Köprüsü: Hızlı Düşüş Yeter

Sayısal rank neden işe yarar? Çünkü Eckart-Young (Ders 7): (k+1)'inci tekil değer, X'in rank-k bir matrisle ne kadar iyi yaklaşılabileceğini söyler.

> *"…the singular values tell us how well we can approximate x by a low-rank matrix."* — Townsend, 21:58

$$\min_{\text{rank}(Y)\leq k} \|X - Y\| = \sigma_{k+1}$$

Sonuç çarpıcı: bir matris **tam rank** olabilir (hiçbir tekil değeri tam sıfır değil) ama tekil değerleri **hızla sıfıra düşüyorsa** düşük sayısal rank'a sahiptir. ε esnemesi sayesinde küçük tekil değerleri atarız ve devasa sıkıştırma kazanırız.

> Builder Notu — Tam Rank Ama Sayısal Küçük
>
> "Tam rank ama hızla düşen σ → düşük sayısal rank" bu dersin kalbidir. ML köprüsü: gerçek veri matrisleri (görüntü, metin gömme, kullanıcı-ürün) neredeyse hiç tam düşük-rank değildir — ama σ hızla düştüğü için düşük sayısal rank'tırlar. Bütün PCA/SVD sıkıştırma pratiği bu olguya dayanır; "intrinsic dimension" (içsel boyut) kavramı budur.

24.10 Hilbert ve Vandermonde: Klasik Örnekler

İki ünlü örnek. Hilbert matrisi H jk = 1/(j+k−1):

> *"…this is called the Hilbert matrix."* — Townsend, 24:38

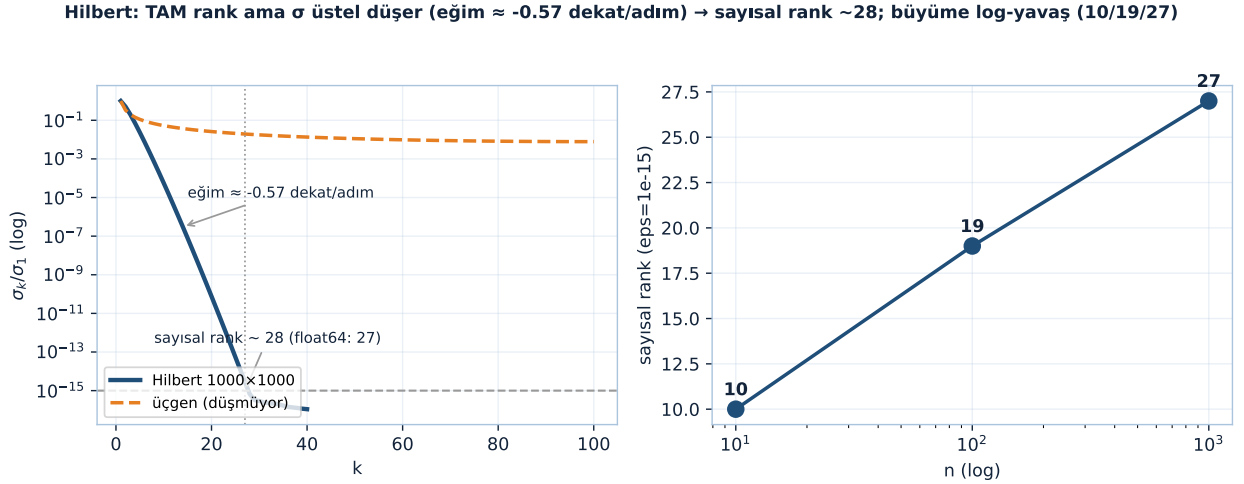
$$H_{jk} = \frac{1}{j + k - 1}$$

1000×1000 Hilbert matrisi tam rank'tır (tüm σ pozitif), ama ε = 10^{-15} ile sayısal rank'ı sadece **28** — yani 15 haneyi feda ederek rank-28 ile yaklaşılır.

> *"…by a rank 28 matrix and only give up [15 digits]."* — Townsend, 35:13

İkinci örnek Vandermonde matrisi V jk = x_j^(k−1) (polinom interpolasyonunda çıkar). O da düşük sayısal rank'tır — **ama bu kötü haber**: düşük sayısal rank, tersini almayı çok zorlaştırır (kondisyon sayısı Ders 10 patlar). n=20 equispaced [0,1] noktalarında motor sayısal rank 19 (<20) ve kondisyon ≈ 1.2×10^{16} verir — tersini almak felaket. Sayısal düşük-rank her zaman lehine değildir.

Aşağıdaki figür solda Hilbert(1000) σ profilini (üçgen "ones" ile kontrastlı) gösterir: float64 gürültü tabanına (ε=10^{-15}) k≈27'de çarpar — sağda ise sayısal rank n ile log-yavaş büyür (10 → 19 → 27) (Şekil 24.5).



Şekil 24.5: Hilbert matrisi 1000×1000 olarak TAM rank taşır, ama tekil değerleri üstel hızla düşer (gerçek-düşüş bölgesi k=1..27 üzerinde motor linfit eğimi ≈ −0.57 dekat/adım); float64 gürültü tabanına (ε = 1e-15) k ≈ 27'de çarpar — sayısal rank ~28. Üçgen 'ones' (turuncu kesik) ise hiç düşmez: kontrast belirgin. Sağda sayısal rank n ile log-yavaş büyür: 10 → 19 → 27.

> **Builder Notu — İyi Sıkışan Kötü Çözülür**
>
> Hilbert ve Vandermonde, "düşük sayısal rank iyi mi kötü mü?" ikiliğini gösterir: sıkıştırma için harika (Hilbert), ama denklem çözme için felaket (Vandermonde — neredeyse tekil, ters kararsız). ML köprüsü: polinom özellikleri (polynomial features) Vandermonde'dur — yüksek dereceli polinom regresyonun neden sayısal olarak kararsız olduğunun ve neden ortogonal polinom/spline tercih edildiğinin sebebi budur.

24.11 "Dünya Pürüzsüz" — Reade'in Açıklaması (1983)

Neden bu kadar çok düşük-rank matris var? Klasik cevap: dünya pürüzsüz.

> *"…so many low-rank matrices is because the world is smooth, as people say."* — Townsend, 27:16

Reade'in (1983) fikri: matris pürüzsüz bir fonksiyondan örnekleniyorsa, o fonksiyonu polinomla yaklaşabiliriz; iki değişkenli, her birinde derece m olan bir polinomu örneklemek **en fazla m^2 rank** verir (her terim bir rank-1 katkı). Pürüzsüz fonksiyon ≈ düşük dereceli polinom → düşük sayısal rank.

$$p(x,y) = \sum_{i,j} c_{ij}\, x^i y^j \;\Rightarrow\; \text{rank} \le m^2$$

Sorun: Hilbert matrisi için Reade'in sınırı **719** çıkar — ama gerçek sayısal rank 28. Açıklama "doğru ama zayıf"; 719, "düşük sayısal rank" demek için çok büyük. Tatmin edici değil. (719 bir literatür değeridir; Townsend bunu Reade'e atıfla aktarır, motor-tanıklı 27-28 gerçek değerin yanında.)

> **Builder Notu — Pürüzsüz Ama Gevşek**
>
> "Dünya pürüzsüz" sezgisel ve çoğu zaman doğru ama niceliksel olarak gevşek. ML köprüsü: aynı fikir kernel yöntemlerinde yaşar — pürüzsüz kernel'ler (RBF) düşük sayısal rank Gram matrisleri üretir, bu yüzden Nyström/rastlantısal özellikler (Ders 13) işe yarar. Ama "ne kadar düşük?" sorusuna pürüzsüzlük net cevap vermez — daha keskin bir araç gerek.

24.12 "Dünya Sylvester" — Daha Keskin Açıklama

Townsend'in tercih ettiği bakış: dünya Sylvester.

> *"…the world is Sylvester."* — Townsend, 36:50

Anlamı: birçok matris bir **Sylvester denklemini** sağlar.

> *"…the matrices satisfy a certain type of equation called the Sylvester equation…"* — Townsend, 37:10

$$AX - XB = C$$

İş şu: X için öyle A, B, C bul ki bu denklem sağlansın ve C **düşük-rank** olsun (örneklerde rank 1). Hem Hilbert hem Vandermonde matrisi bir Sylvester denklemi sağlar — uygun köşegen A, B ve rank-1 C ile. Denklemin yapısı, X'in tekil değerlerinin ne kadar hızlı düşeceğini belirler.

Hilbert için bunu somutlaştır: $A = \text{diag}(j - \frac{1}{2})$, $B = -\text{diag}(k - \frac{1}{2})$ alınırsa AH − HB tam olarak tüm-birler (rank-1) matrisine eşit çıkar — motor n=5 ve n=8 için maxdiff 1.1×10^{-16} doğrular (Şekil 24.6).

Hilbert bir Sylvester denklemi saglar: A = diag(j-1/2), B = -diag(k-1/2) -> AH - HB = rank-1 ones (maxdiff 1e-16)

AH - HB					C = tum-birler (rank-1)					fark ~ 1e-16				
1	1	1	1	1	1	1	1	1	1	0e+00	0e+00	-1e-16	0e+00	0e+00
1	1	1	1	1	1	1	1	1	1	0e+00	0e+00	0e+00	0e+00	0e+00
1	1	1	1	1	1	1	1	1	1	-1e-16	0e+00	0e+00	-1e-16	-1e-16
1	1	1	1	1	1	1	1	1	1	0e+00	0e+00	-1e-16	0e+00	0e+00
1	1	1	1	1	1	1	1	1	1	0e+00	0e+00	-1e-16	0e+00	0e+00

Şekil 24.6: Hilbert matrisi bir Sylvester denklemini sağlar: $A = \text{diag}(j - \frac{1}{2})$, $B = -\text{diag}(k - \frac{1}{2})$ alınırsa $AH - HB$ tüm-birler (rank-1) matrisine eşit çıkar (maxdiff $\approx 1.1 \times 10^{-16}$). C düşük-rank olduğu için H de hızlı azalan tekil değerlere sahiptir.

> **Builder Notu — Denklemi Bul Sınırı Al**
>
> Sylvester denklemi (AX − XB = C) kontrol teorisi, sinyal işleme ve sayısal lineer cebirin temel denklemidir. Townsend'in içgörüsü: "neden düşük sayısal rank?" sorusunu "hangi Sylvester denklemini sağlıyor?" sorusuna çevirir — soyut ama güçlü, çünkü Sylvester denklemi hakkında çok şey biliniyor. ML köprüsü: Lyapunov denklemleri (Sylvester'ın özel hali) durum-uzay modellerinde (S4/Mamba, Ders 15 bağı) kovaryans ve kararlılık analizinde çıkar.

24.13 Zolotarev Sayısı ve Sınır

Sylvester denklemini sağlayan X için (A, B normal; rank(C) = r), tekil değerler için bir sınır kanıtlanmıştır:

$$\frac{\sigma_{1+rk}(X)}{\sigma_1(X)} \leq Z_k(E, F)$$

Burada r = rank(C) (örneklerde 1), ve sağdaki "kötü görünümlü" sayı Zolotarev sayısıdır.

> *"This nasty guy here is called the Zolotarev number."* — Townsend, 44:55

E, A'nın özdeğerlerini; F, B'nin özdeğerlerini içeren kümeler. Bu sınırın değeri, Zolotarev sayısının 1870'lerden beri çok iyi çalışılmış olması. Anahtar nokta: E ve F **ayrık** (separated) ise Zolotarev sayısı k ile çok hızlı küçülür → tekil değerler hızla düşer → düşük sayısal rank.

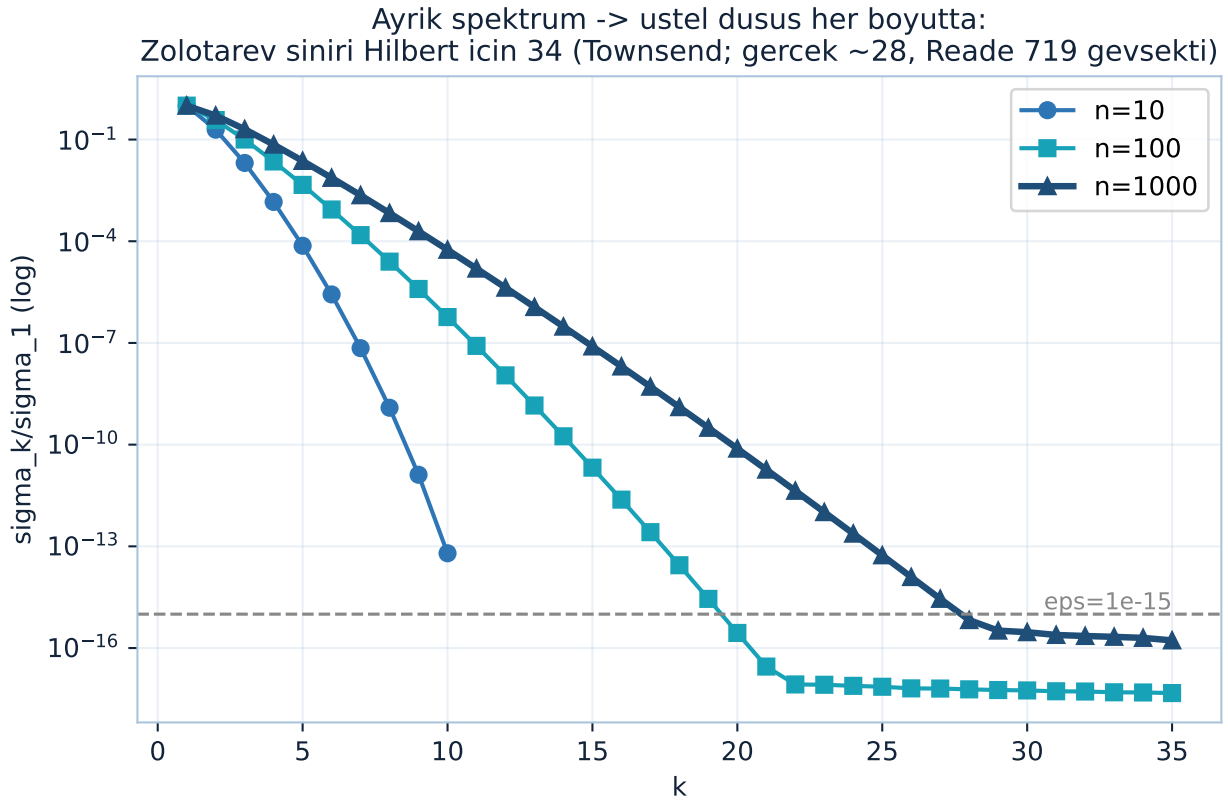
> *"…the sets E and F are separated."* — Townsend, 46:41

Hilbert matrisi için bu analiz sayısal rank sınırını **34** verir — 719 değil, gerçek 28'e çok daha yakın:

> *"…a world record bound. For the Hilbert matrix is 34…"* — Townsend, 48:43

Renkli bir kapanış: Zolotarev ve Penzl, bu sayı üzerinde çalışırken ikisi de 31 yaşında ölmüş (biri tren kazası, diğeri çığ) — Townsend kendi 31'ine yaklaştığını espriyle anar ("Zolotarev laneti").

Aşağıdaki figür ayrıklığın etkisini gösterir: n=10, n=100 ve n=1000 Hilbert profilleri aynı üstel koridorda iner — E,F ayrıklığı boyuttan bağımsız hızlı düşüş verir (Şekil 24.7). (34 ve 719 literatür değerleridir, Townsend anlatımı; motor-tanıklı gerçek sayısal rank ~28.)



Şekil 24.7: Ayrık spektrum, üstel düşüş demek — her boyutta. n=10, n=100 ve n=1000 Hilbert matrislerinin tekil değer profilleri aynı üstel koridorda iner: E ve F kümelerinin ayrıklığı boyuttan bağımsız hızlı düşüş verir. Zolotarev sınırı Hilbert için 34'ü işaret eder (Townsend); motor-tanıklı gerçek sayısal rank ~28, Reade'in eski 719 sınırı çok gevşekti.

Motor Notu — Sayısal Rank ε-Duyarlıdır

[motor notu] Townsend Hilbert(1000) için sayısal rank **28** raporlar. Motor float64 SVD ile rank_ε(10^{-15}) = **27** verir; çünkü $\sigma_{28}/\sigma_1 = 6.7\times10^{-16}$, eşiğin ($10^{-15}$) tam altında ama gürültü tabanının ~3 katı. Eşik biraz gevşetilirse ($\varepsilon = 5\times10^{-16}$) sonuç 28 olur. Bu bir sapma değil — sayısal rank kavramının kendisi ε-duyarlıdır, ve tam bu kenar durumda kavramın inceliği canlı görünür: 27 ile 28 arasındaki fark, "kaç tekil değer gürültü tabanının üstünde sayılır?" sorusunun ε seçimine bağlı olmasıdır. Bu dersin tam konusu.

Builder Notu — Yüz Elli Yıllık Sınır

Zolotarev sayısı, "soyut bir probleme çevir, sonra zengin literatürü kullan" stratejisinin güzel örneği: "tekil değerler ne kadar düşer?" sorusunu 150 yıllık bir matematik nesnesine bağlar. ML köprüsü: "ayrık spektrum → hızlı tekil değer düşüşü" ilkesi, iyi-koşullu vs kötü-koşullu problemleri ayırt etmenin derin sebebidir; düşük-rank yaklaşımın ne zaman işe yarayacağını önceden söyler.

24.14 Bu Dersin Özeti

- **Düşük-rank = sıkıştırma:** rank-k matris 2kn sayı (n^2 değil); $k < n/2$ ise düşük-rank, pratikte k n istenir.
- **Hangi desenler:** ızgaraya hizalı (bayrak, rank-1 Avusturya) veya simetrik (daire ≈ ½r+1); köşegen/üçgen **kötü** (σ düşmüyor: $\sigma_1 \approx 2n/\pi$, $\sigma_n \approx ½$).
- **Sayısal rank:** rank_ε(X) = ε'dan büyük tekil değer sayısı; tam-rank ama hızla düşen σ → düşük sayısal rank (Eckart-Young: σ k+1 = en iyi rank-k hatası).
- **Klasik örnekler:** Hilbert H jk =1/(j+k−1) (1000×1000 → rank 28); Vandermonde (düşük sayısal rank → ters almak zor).
- **"Dünya pürüzsüz" (Reade):** pürüzsüz fonksiyon → polinom yaklaşım → rank ≤ m^2; Hilbert için zayıf (719).
- **"Dünya Sylvester":** AX − XB = C, C düşük-rank; Zolotarev sayısı sınırı (E, F ayrık → hızlı düşüş; Hilbert → 34).

Tek Bir Cümle

Dünyada bu kadar çok sayısal düşük-rank matris vardır çünkü onlar pürüzsüz fonksiyonlardan veya düşük-rank C'li Sylvester denklemlerinden doğar; tekil değerlerin hızlı düşüşü (Zolotarev sayısıyla sınırlı) onları sıkıştırılabilir kılar.

24.15 Kontrol Soruları

Soru 1: Bir matris hangi koşulda düşük-rank sayılır ve kaç sayıyla gönderilir

rank-k bir n×n matris, k tane (u_i, v_i) çiftiyle, yani 2kn sayıyla gönderilir (tüm elemanlar için n^2 yerine). Katı tanım: $2kn < n^2$, yani $k < n/2$ ise düşük-rank — düşük-rank formda göndermek daha verimli olduğunda. Pratikte k'nin n'den çok daha küçük olması istenir.

Soru 2: Üçgen matris neden kötüyken daire iyidir

Üçgen matrisin tekil değerleri sıfıra düşmez ($\sigma_1 \approx 2n/\pi$, $\sigma_n \approx ½$ — hepsi büyük), çünkü köşegen deseni hiçbir simetri/hizalama sunmaz; sıkıştırılamaz. Daire ızgaraya hizalı olmasa da güçlü simetriye sahiptir; satır/sütun uzayları tekrar ettiğinden rank ≈ ½r+1 ile sınırlıdır — sıkıştırılabilir. Ders: hizalama veya simetri, ikisi de düşük-rank üretir.

> **Soru 3: Bir matris tam rank olduğu halde nasıl düşük sayısal rank olabilir**
>
> Tüm tekil değerleri pozitif (tam rank) olsa bile, σ değerleri **hızla** sıfıra düşüyorsa, ε toleransının altına düşenleri atarak düşük-rank bir matrisle çok iyi yaklaşılır. Eckart-Young'a göre σ k+1 , en iyi rank-k yaklaşım hatasıdır; σ hızla düştüğünde küçük bir k yeter. Hilbert matrisi örnek: 1000×1000 tam rank, ama $\varepsilon=10^{-15}$ ile sayısal rank 28.

> **Soru 4: Dünya Sylvester açıklaması dünya pürüzsüzden neden daha keskindir**
>
> "Dünya pürüzsüz" (Reade) doğru ama gevşek bir sınır verir (Hilbert için 719, gerçek 28). "Dünya Sylvester" matrisin AX − XB = C denklemini (C düşük-rank) sağladığını kullanır; bu, tekil değer düşüşünü Zolotarev sayısına bağlar. E (A'nın özdeğerleri) ve F (B'nin özdeğerleri) ayrıksa Zolotarev sayısı hızla küçülür ve Hilbert için 34 gibi çok daha keskin (gerçek 28'e yakın) bir sınır verir.

24.16 Egzersizler

1. **Sıkıştırma hesabı.** 1000×1000 bir matris rank-5 ise, tam formda kaç sayı, düşük-rank formda kaç sayı gönderirsin? Sıkıştırma oranı nedir? k < n/2 katı tanımını sağlıyor mu? (Motor: 10^6 vs 10^4, oran 100×; k=5 500 = n/2 — fazlasıyla sağlar.)

2. **Rank-1 bayrak.** Üç yatay şeritli bir bayrağı (Avusturya: üst kırmızı, orta beyaz, alt kırmızı) matris olarak yaz. Bunun rank-1 = uv^T olduğunu, u ve v'yi açıkça vererek göster.

3. **Sayısal rank.** Tekil değerleri σ = (1, 0.1, 0.01, 0.001, 10^{-8}, 10^{-12}) olan bir matrisin $\varepsilon = 10^{-6}$ (göreli) toleransındaki sayısal rank'ı kaçtır? (İpucu: $\sigma_k/\sigma_1 > \varepsilon$ koşulu. Cevap: 4.)

4. **Sylvester kontrolü.** Hilbert matrisi H jk = 1/(j+k−1) için, köşegen A = diag(j−½) ve B = diag(−(k−½)) seçimiyle AH − HB çarpımının her elemanının (j−½)+(k−½) = j+k−1 ile çarpıldığını, dolayısıyla paydanın sadeleşip C = (tüm-birler) rank-1 matrisi verdiğini göster. (Motor: n=5 ve n=8 için maxdiff 1.1×10^{-16}.)

5. **(Ders 18 habercisi)** Bu derste düşük-rank'ın "kaç sayı" gerektirdiğini gördük (2kn). Peki bir SVD, LU veya QR ayrışımı **toplam kaç bağımsız parametre** içerir? Serbestlik derecelerini saymak ne öğretir? Bir tahmin yaz — Ders 18 "SVD, LU, QR'da parametre sayımı"nı işliyor.

24.17 Sonraki Ders İçin Hazırlık

Ders 18: SVD, LU, QR ve Eyer Noktalarında Parametre Sayımı. Townsend "2kn sayı" diyerek parametre saymaya başladı; Strang bunu sistematikleştirir: her matris ayrışımı (SVD, LU, QR, kutupsal) kaç bağımsız parametre içerir? Serbestlik derecesi sayımı, ayrışımların "neden bu boyutta" olduğunu ve eyer noktalarının yapısını açıklar.

⚠ Ders 18 Öncesi Yapılacak

Bu dersin "2kn sayı" sayımını hatırla: rank-k bir n×n matris kaç bağımsız sayı içerir? Ders 18 aynı soruyu SVD ($U\Sigma V^T$), LU ve QR için soracak — $U\Sigma V^T$'de ortogonal U ve V kaç serbestlik derecesi taşır? Ortonormallik kısıtlarının (Ders 3) parametre sayısını nasıl düşürdüğünü düşün.

24.18 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Konuşmacı (dk)
Strang takdimi	konuk Townsend: §4.3'ün yaratıcısı	Strang 1m00
Neden düşük-rank	$X = \Sigma \sigma_i u_i v_i^T$ (k rank-1 parça)	Townsend 1m15
Sıkıştırma	$2kn < n^2 \iff k < n/2$	Townsend 6m33
Bayraklar	Avusturya rank-1; İskoçya tam rank	Townsend 8m04
Üçgen kötü	$\sigma_1 \approx 2n/\pi$, $\sigma_n \approx ½$ (düşmez)	Townsend 14m27
Sayısal rank	rank_ε = ε·σ_1 üstündeki σ sayısı	Townsend 19m58
Eckart-Young	σ k+1 = en iyi rank-k yaklaşım hatası	Townsend 21m58
Hilbert matrisi	H jk = 1/(j+k−1); 1000×1000 → rank 28	Townsend 24m38
Dünya pürüzsüz	Reade: polinom → rank ≤ m^2 (719, zayıf)	Townsend 27m16
Dünya Sylvester	AX − XB = C, C düşük-rank	Townsend 36m50
Zolotarev sayısı	E, F ayrık → hızlı düşüş; Hilbert 34	Townsend 44m55

24.19 ML Bağlantıları Özeti

- **Sıkıştırma / LoRA:** düşük-rank = 2kn parametre; LoRA ($\Delta W = BA^T$), görüntü sıkıştırma, otokodlayıcılar bu pazarlığı yapar.
- **Sayısal rank / etkin boyut:** gerçek veri matrisleri ve ağ ağırlıkları tam değil **sayısal** düşük-rank'tır; budama (pruning) ve SVD sıkıştırma buna dayanır.
- **Simetri → düşük-rank:** CNN'ler (Ders 32) uzaysal simetriyle parametre paylaşır — "az parametre, çok bilgi"nin başka yolu.
- **Pürüzsüzlük → kernel:** pürüzsüz kernel'ler düşük sayısal rank Gram matrisi verir; Nyström/rastlantısal özellikler (Ders 13) bu yüzden çalışır.
- **Kondisyon uyarısı:** Vandermonde gibi düşük sayısal rank matrisler tersini almayı zorlaştırır (Ders 10); polinom özellikleri kararsız, ortogonal polinom/spline tercih edilir.
- **Geriye köprü:** Ders 6 (SVD = $\Sigma \sigma_i u_i v_i^T$), Ders 7 (Eckart-Young, en iyi rank-k), Ders 16 (tekil değer türevi), Phase 1 18.06 (−1,2,−1 sonlu fark matrisi).

❗ Townsend'in Sorusu, Townsend'in Cevabı

> *"Today I want to tell you a little about why there [are] so many matrices that are low rank in the world."* — Townsend, 1:15

Cevap: pürüzsüzlük ve Sylvester yapısı tekil değerleri hızla düşürür; gerçek dünyanın matrisleri bu yüzden sıkıştırılabilir.

25 SVD, LU, QR ve Eyer Noktalarında Parametre Sayımı

Serbest parametre muhasebesi, KKT matrisleri ve Rayleigh bölümü

> **Bölüm bilgisi**
>
> **Video:** MIT 18.065 — Counting Parameters in SVD, LU, QR, Saddle Points · **OCW:** Lecture 18 · **Okuma süresi:** ≈33 dk · **Konuşmacı:** Gilbert Strang · **Önkoşul:** Ders 17 (hızla azalan tekil değerler). Bu ders **lineer cebir blokunun kapanışıdır**. İki konu birleşir: her matris ayrışımının kaç serbest parametre taşıdığı (muhasebe) ve eyer noktalarının iki kaynağı (KKT + Rayleigh). Buradan sonra ders 19 ile maxmin/Courant-Fischer dünyasına geçilir.

25.1 Bu Derste Ne Var?

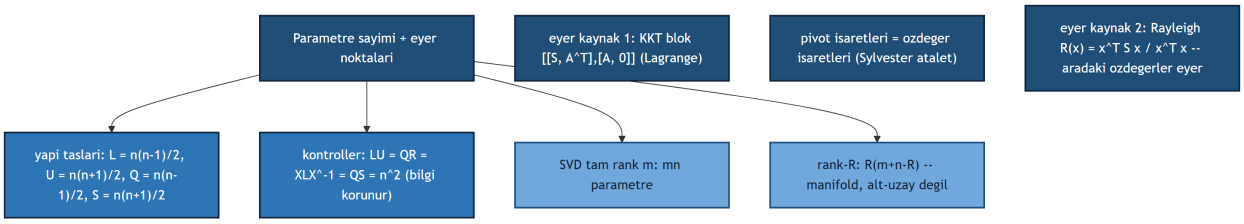
Lineer cebir bloku kapanıyor; iki konu. **İlk yarı:** her matris ayrışımı (LU, QR, $X\Lambda X^{-1}$, QS, $U\Sigma V^{T}$) kaç **serbest parametre** içerir? Hepsi A'nın parametre sayısına (n^2 veya mn) tam tamamlanır — ayrışım bilgi kaybetmez. **İkinci yarı: eyer noktaları** (saddle points) nereden gelir — iki kaynak.

Beş sonuç:

1. **Yapı taşları:** üçgensel L = ½n(n−1), U = ½n(n+1), köşegen = n, ortogonal Q = ½n(n−1), simetrik S = ½n(n+1).
2. **Kontroller:** LU = QR = $X\Lambda X^{-1}$ = QS = n^2; SVD (tam rank m) = mn; rank-R matris = R(m+n−R).
3. **Eyer kaynağı 1 — Lagrange/KKT:** kısıtlı optimizasyon → blok matris [[S, A^T],[A, 0]]; belirsiz (indefinite), eyer noktası.
4. **Pivot işaretleri = özdeğer işaretleri:** n pozitif + m negatif pivot → eyer.
5. **Eyer kaynağı 2 — Rayleigh bölümü:** $R(x) = x^TSx/x^Tx$; maks = λ_1, min = λ_n, **aradaki özdeğerler eyer** (Ders 19'a köprü).

> *"…the number of free parameters agrees with the number of parameters in A itself, like n squared."*
> — Strang, 0:59

Aşağıdaki kavram haritası dersin merkezini ("parametre sayımı + eyer noktaları") ana fikirlere bağlar: yapı taşlarının serbest parametre sayıları, her ayrışımın n^2'ye (SVD'nin mn'ye) oturması, rank-R matrisin R(m+n−R) parametreli bir manifold oluşturması ve eyer noktalarının iki kaynağı (KKT bloğu + Rayleigh bölümü) ile pivot/özdeğer işaret eşitliği (Şekil 25.1).



Şekil 25.1: Ders 18 kavram haritası: "Parametre sayımı + eyer noktaları" merkezinden ana fikirlere dallar — her faktörizasyonun yapı taşları kaç serbest parametre tuttuğunu sayar (alt-üçgen L = n(n−1)/2, üst-üçgen U = n(n+1)/2, ortogonal/skew Q = n(n−1)/2, simetrik S = n(n+1)/2), ve bu sayım her ayrışımda korunur çünkü LU = QR = $X\Lambda X^{-1}$ = QS = n^2 hepsi tam n^2'ye oturur (bilgi kaybolmaz); tam rank m olan bir m×n SVD mn parametreyle sayılır, oysa rank-R matrisler R(m+n−R) parametreyle bir manifold oluşturur — bir alt-uzay değil, eğri bir yüzey (LoRA'nın az parametresi buradan gelir); eyer noktalarının iki ayrı kaynağı vardır: birincisi Lagrange çarpanlı kısıtlı problemlerin KKT bloğu [[S, A^T],[A, 0]] belirsizdir, ikincisi Rayleigh bölümü R(x) = $x^T Sx$ / $x^T x$ birim kürede ararken en büyük ve en küçük özdeğerler arasındaki özdeğerler birer eyer noktası verir; ayrı bir düğüm KKT matrisinin pivot işaretlerinin özdeğer işaretleriyle aynı olduğunu (Sylvester atalet yasası) hatırlatır.

> **Builder Notu — Muhasebesi Tutan Cebir**
>
> - **Serbest parametre = serbestlik derecesi** — bir modelin "gerçek" boyutu; Ders 17'nin "2kn sayı" saymanının sistematik hali. Etkin parametre sayısı ML'de kapasite ve aşırı-öğrenme analizinin temeli.
> - **KKT matrisi** — kısıtlı optimizasyonun (SVM, eşitlik kısıtları) çekirdek yapısı; eyer noktası çözmek = denge bulmak.
> - **Eyer noktaları** — derin ağ kayıp yüzeylerinde minimumdan çok eyer vardır; SGD'nin eyerlerden kaçması eğitimin kalbi.
> - **Geriye köprü:** Ders 2 (5 ayrışım), Ders 3 (ortogonal Q), Ders 5 (pozitif tanımlı, pivot), Ders 6 (SVD), Ders 17 (parametre sayma).

Tek cümle: Her matris ayrışımı orijinal matrisle aynı sayıda serbest parametre taşır (bilgi korunur); eyer noktaları ise ya Lagrange kısıtlarının KKT blok matrisinden ya da Rayleigh bölümünün aradaki özdeğerlerinden doğar.

25.2 Fikir: Serbest Parametre Sayımı

Lineer cebir bölümünün kapanış jesti: Ders 2'nin beş ayrışımını (A = LU, QR, $X\Lambda X^{-1}$, $Q\Lambda Q^T$, $U\Sigma V^T$) bir "muhasebe" testinden geçir. A genelde n^2 sayı içerir. Bir ayrışım hesaplandıktan sonra A'yı atabilir miyiz? Evet — bilgi ayrışımda. O hâlde ayrışımın parçalarındaki **serbest parametre** sayısı da n^2'ye tamamlanmalı:

> *"…the number of free parameters agrees with the number of parameters in A itself, like n squared."*
> — Strang, 0:59

Bu, Strang'ın "hiçbir ders kitabında görmedim" dediği bir alıştırma — ama tüm lineer cebiri tek satıra sıkıştırır.

Builder Notu — Bilgi Ayrışımda Saklı

"Serbest parametre = serbestlik derecesi" Ders 17'nin "2kn sayı" sayımının sistematik hâli. ML köprüsü: bir modelin **etkin** parametre sayısı, nominal parametre sayısından az olabilir (kısıtlar, simetriler, düşük-rank yapı). Kapasite, aşırı-öğrenme ve genelleme analizinin tamamı "kaç gerçek serbestlik derecesi?" sorusuna dayanır.

25.3 Yapı Taşları: L, U, Köşegen, Q, S

Önce parçaları say. Üçgensel matrisler:

$$L: \tfrac{1}{2}n(n-1), \qquad U: \tfrac{1}{2}n(n+1)$$

L alt-üçgensel, köşegeni hep 1 (serbest değil) → 1+2+⋯+(n−1) = ½n(n−1). U üst-üçgensel, köşegende pivotlar serbest → 1+2+⋯+n = ½n(n+1). İkisi tam n kadar farklı. Köşegen matris: n. Ortogonal Q: ilk kolon birim (n−1 serbest), ikinci kolon birim **ve** birinciye dik (n−2), … toplam:

$$Q: (n-1) + (n-2) + \cdots + 0 = \tfrac{1}{2}n(n-1)$$

Simetrik S: üst üçgen + köşegen serbest, alt yarı belirli → ½n(n+1).

Aşağıdaki figür dört faktörizasyonun (LU, QR, $X\Lambda X^{-1}$, QS) serbest parametre muhasebesini n=6 için gösterir; her çubuk iki parçaya ayrılır ve dördü de tam n^2=36'da buluşur — sonraki bölümün kontrol sonucunu önceden görselleştirir (Şekil 25.2).

Builder Notu — Kısıt Parametre Yer

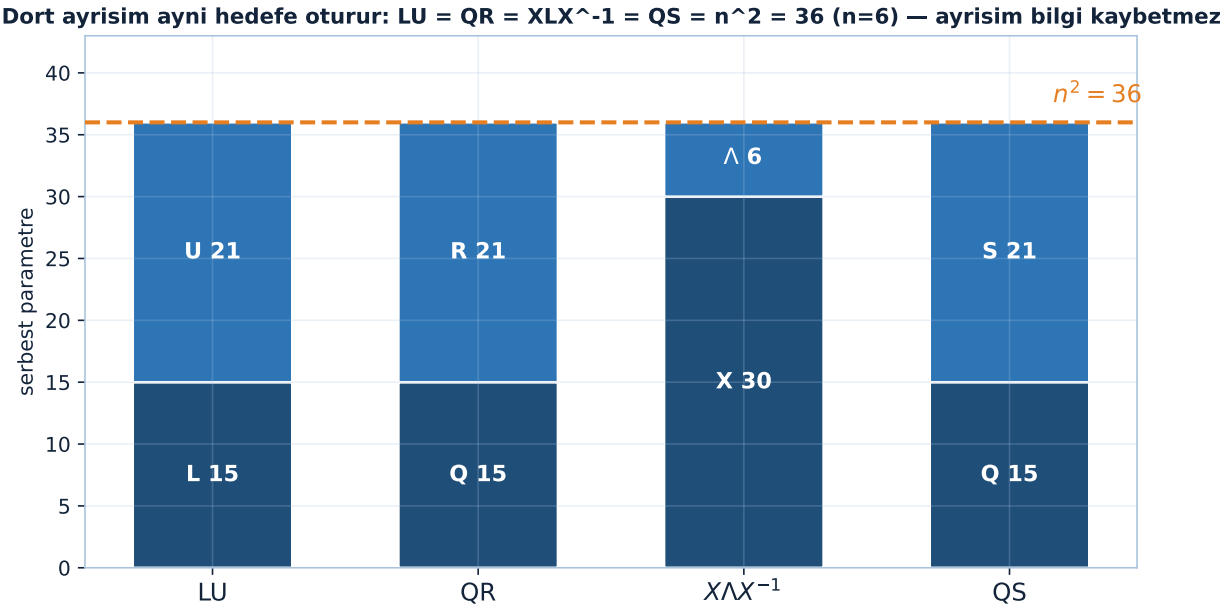
Ortogonal matrisin n^2 değil sadece ½n(n−1) serbestlik derecesi olması (Ders 3) önemli: ortogonallik kısıtları parametrelerin yarısını "yer". ML köprüsü: ortogonal/üniter ağırlık katmanları (gradyan patlamasını önler) bu yüzden daha az serbestlik taşır; Stiefel manifoldu üzerinde optimizasyon tam bu kısıtlı parametre uzayında çalışır.

25.4 Özvektör Matrisi: $n^2 - n$

Özvektör matrisi X kaç serbest parametre içerir? Her özvektör bir skalerle çarpılabilir (x özvektörse 2x de özvektör). Bir konvansiyonla her kolonun ilk bileşenini 1 yap — o zaman üst satırdaki n tane "1" sayılmaz:

$$X: n^2 - n, \qquad \Lambda: n, \qquad X^{-1}: 0$$

X^{-1} serbest değil — X tarafından belirlenir. Toplam: (n^2−n) + n + 0 = n^2. Köşegenleştirme $A = X\Lambda X^{-1}$ tam n^2'ye oturur.



Şekil 25.2: Dört faktörizasyon (LU, QR, $X\Lambda X^{-1}$, QS) için serbest parametre muhasebesi (n=6). Her çubuk iki parçaya ayrılır: navy alt parça birinci çarpan (L=15, Q=15, X=30, Q=15), steel üst parça ikinci çarpan (U=21, R=21, Λ=6, S=21). Dördü de turuncu kesik çizgide, tam n^2=36'da buluşur — ayrışım hangi biçimde olursa olsun aynı bilgiyi taşır, hiçbir parametre kaybolmaz.

> **Builder Notu — Yön Var Uzunluk Yok**
>
> "Özvektör ölçeklenebilir → bir serbestlik derecesi düşer" gözlemi, özvektörlerin **yön** taşıdığını, uzunluk taşımadığını hatırlatır. ML köprüsü: gömme (embedding) vektörlerinde de aynı belirsizlik var — ölçek/işaret çoğu zaman serbest; normalize etmek (birim uzunluk) bu fazla serbestliği sabitler.

25.5 Faktorizasyon Kontrolleri: Hepsi n^2

Yapı taşlarını birleştir, hepsi n^2'ye toplanmalı:

$$\underbrace{\tfrac{1}{2}n(n-1)}_{L} + \underbrace{\tfrac{1}{2}n(n+1)}_{U} = n^2$$

$$\underbrace{\tfrac{1}{2}n(n-1)}_{Q} + \underbrace{\tfrac{1}{2}n(n+1)}_{R} = n^2$$

LU'da eksi artıyı götürür; QR'da Q (ortogonal) + R (üst üçgensel) yine n^2. Köşegenleştirme $(n^2-n)+n = n^2$. Son olarak kutupsal ayrışım A = QS (ortogonal × simetrik):

> *"…That's called the polar decomposition."* — Strang, 9:33

$$\underbrace{\tfrac{1}{2}n(n-1)}_{Q} + \underbrace{\tfrac{1}{2}n(n+1)}_{S} = n^2$$

Kutupsal ayrışım SVD'ye çok yakındır (Ders 6); bazı mühendisler SVD yerine QS hesaplamayı tercih eder. Yukarıdaki (Şekil 25.2) tam bu dört kontrolün — LU, QR, $X\Lambda X^{-1}$, QS — hepsinin aynı n^2'ye oturduğunu görselleştirir.

> **Builder Notu — Farklı Kılık Aynı Bilgi**
>
> Her ayrışımın aynı n^2'ye toplanması "ayrışım bilgi yaratmaz/yok etmez" ilkesinin niceliksel kanıtıdır — A'daki bilgi, farklı kılıklarda (L+U, Q+R, Q+S) aynen korunur. ML köprüsü: bir veriyi farklı bazlarda temsil etmek (PCA, Fourier, dalgacık) boyut/bilgi miktarını değiştirmez, sadece **düzenini** değiştirir — sıkıştırma ancak gerçek düşük-rank/seyreklik varsa kazanılır (Ders 17).

25.6 SVD Parametre Sayımı: mn

Şimdi dikdörtgen A (m × n, m ≤ n, tam rank m). SVD: $A = U\Sigma V^T$, U (m×m ortogonal), Σ (m tekil değer), V^T (n×n ama sadece ilk m kolonu önemli). Say:

$$\underbrace{\tfrac{1}{2}m(m-1)}_{U} + \underbrace{m}_{\Sigma} + \underbrace{mn - \tfrac{1}{2}m(m+1)}_{V} = mn$$

V'nin katkısı incelik ister: ilk m kolonu (sıfırdan farklı tekil değerlere ait) önemli, geri kalan boş-uzaydan gelir ve sayılmaz. Önemli kolonlar (n−1)+(n−2)+⋯+(n−m) = mn − ½m(m+1). Hepsi toplanınca tam **mn** — A'nın eleman sayısı.

> **Builder Notu — Boş Uzay Sayılmaz**
>
> SVD'nin tam mn parametreye oturması, "tam rank SVD bilgi kaybetmez" anlamına gelir. ML köprüsü: bir m×n veri matrisinin gerçek bilgi içeriği mn'dir; ama tekil değerler hızla düşüyorsa (Ders 17), pratikte çok daha az parametre yeter — işte sıkıştırmanın kazancı buradan, parametre sayımı bu üst sınırı verir.

25.7 Rank-R Matris: Bir "Yüzey"

Daha ilginç soru: rank-R bir m × n matris kaç parametre içerir?

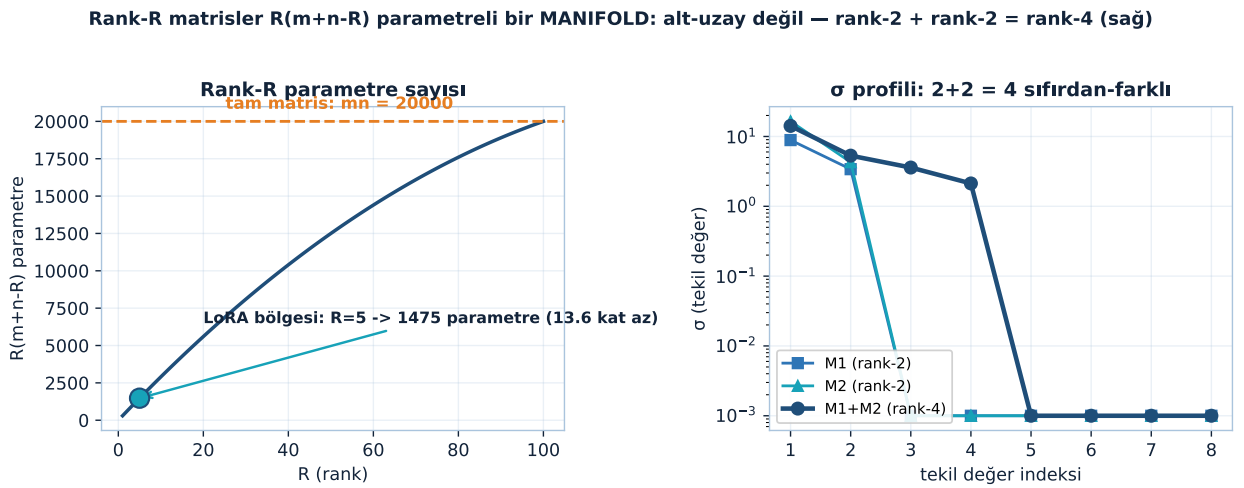
> *"…how many parameters are there in a rank R matrix?"* — Strang, 19:30

Sıkıştırılmış SVD: U (m×R), Σ (R), V^T (R×n). Say:

$$\underbrace{mR - \tfrac{1}{2}R(R+1)}_{U} + \underbrace{R}_{\Sigma} + \underbrace{nR - \tfrac{1}{2}R(R+1)}_{V} = R(m+n-R)$$

Bu sayı ilginç çünkü rank-R matrisler bir **alt-uzay değildir**: iki rank-R matrisin toplamı rank-2R olabilir. Matris uzayı (m×n matrisler mn-boyutlu vektör uzayı) içinde rank-R matrisler bir **yüzey/manifold** oluşturur — toplama altında kapalı değil, ama yerel boyutu R(m+n−R).

Aşağıdaki figür solda R(m+n−R) parametre eğrisini (m=100, n=200; R=5'te 1475 parametre = tam matrisin 13.6 kat altı, LoRA bölgesi), sağda rank-2 + rank-2 = rank-4 manifold tanığını gösterir (Şekil 25.3).



Şekil 25.3: Rank-R matrisler R(m+n-R) parametreli bir MANIFOLD oluşturur — alt-uzay değil. Sol: m=100, n=200 için parametre sayısı; R=5'te yalnızca 1475 parametre (tam matrisin 20000'inden 13.6 kat az — LoRA'nın çalışma bölgesi). Sağ: rank-2 + rank-2 = rank-4 tanığı; iki rank-2 matrisin toplamında 4 sıfırdan-farklı tekil değer belirir, yani toplam alt-uzayda kapalı değildir.

> **Builder Notu — LoRA'nın Sayısı**
>
> R(m+n−R) sayısı modern ML'in kalbinde: LoRA bir n×n ağırlık güncellemesini rank-R tutarak n^2 yerine ~2nR parametreyle eğitir (R n). "Rank-R matrisler bir manifold" görüşü, düşük-rank optimizasyonun neden Öklid uzayında değil bir manifold üzerinde yapıldığını (Riemann optimizasyonu) açıklar — matris tamamlama (Ders 16 Netflix) bu manifold üzerinde arama yapar.

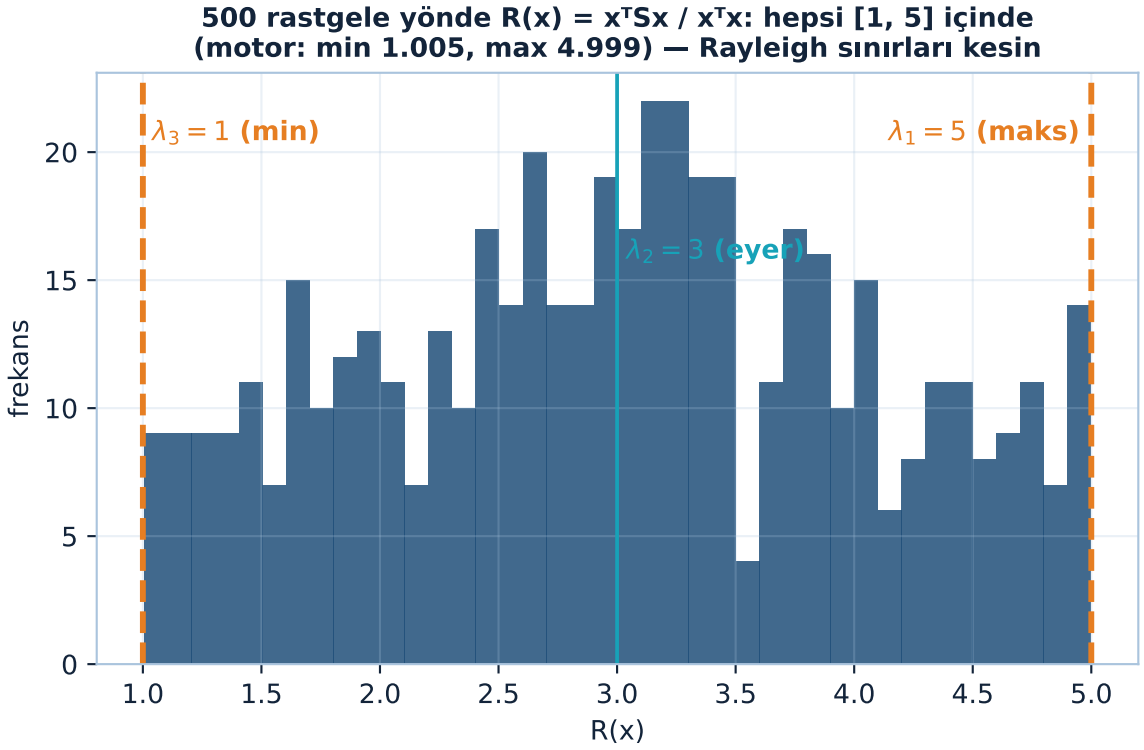
25.8 Eyer Noktaları: İki Kaynak

İkinci konu: eyer noktaları (saddle points). Bunlar ne maksimum ne minimumdur — ilgili matris pozitif tanımlı **değildir** (özdeğerleri karışık işaretli). Strang iki ana kaynak ayırt eder:

> *"…there are two main sources of saddle points."* — Strang, 27:08

Birincisi **kısıtlı optimizasyon** (Lagrange çarpanları), ikincisi **Rayleigh bölümü**nün aradaki özdeğerleri. İkisi de derin öğrenme ve optimizasyonda her yerde.

Aşağıdaki figür 500 rastgele yönde Rayleigh bölümünün hep $[\lambda_3, \lambda_1] = [1, 5]$ aralığında kaldığını gösterir — eyer kavramına girmeden önce sınırların kesinliğini görselleştirir; aradaki $\lambda_2 = 3$ eyeri sonraki bölümlere taşınır (Şekil 25.4).



Şekil 25.4: 500 rastgele yönde R(x): hepsi [λ_3, λ_1] = [1, 5] içinde — Rayleigh sınırları kesin (motor: min 1.005, max 4.999).

> **Builder Notu — Minimum Az Eyer Çok**
>
> Eyer noktası, yüksek boyutta minimumdan çok daha yaygındır: derin ağ kayıp yüzeyinde kritik noktaların ezici çoğunluğu eyerdir, gerçek minimum azdır. ML köprüsü: SGD'nin gürültüsü eyerlerden kaçmaya yarar (Ders 25); "eyerden kaçış" modern optimizasyon teorisinin merkezî sorusudur.

25.9 Kaynak 1: Lagrange ve KKT Matrisi

Klasik kısıtlı problem: ½x^TSx'i (S pozitif tanımlı) Ax = b kısıtı altında minimize et. Kısıt olmasa minimum sıfırdır; kısıt işi ilginçleştirir. Lagrange'ın reçetesi — m kısıt için m çarpan (λ vektörü) ekle:

$$L(x, \lambda) = \tfrac{1}{2} x^T S x + \lambda^T (Ax - b)$$

x ve λ'ya göre türev al, sıfıra eşitle:

$$\frac{\partial L}{\partial x} = Sx + A^T \lambda = 0, \qquad \frac{\partial L}{\partial \lambda} = Ax - b = 0$$

İki denklemi blok matris formunda yaz:

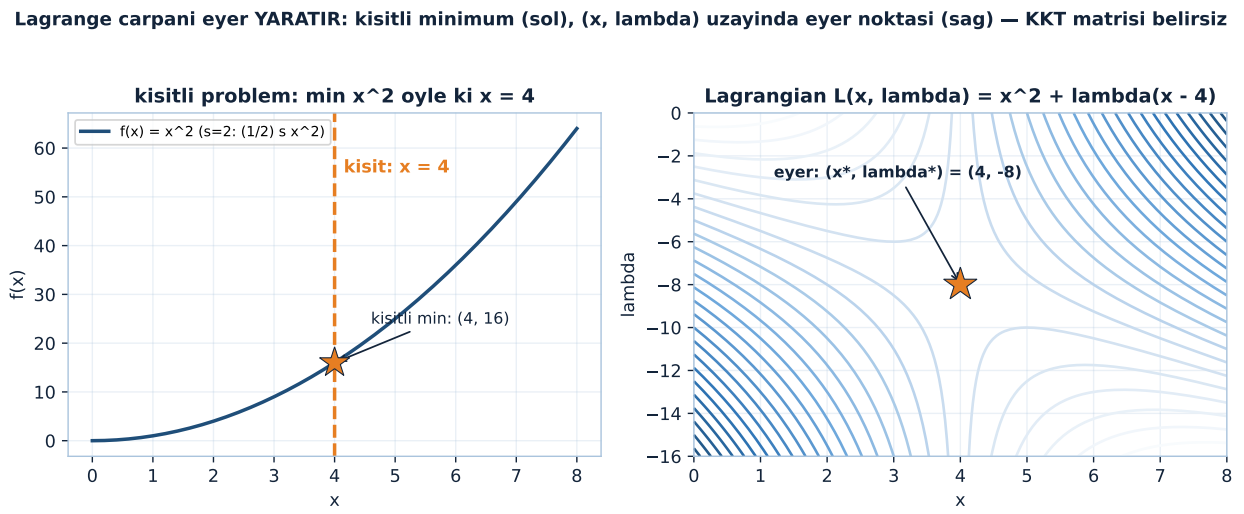
$$\begin{bmatrix} S & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}$$

λ derivatifi kısıtları geri getirir. Bu blok matris, optimizasyoncuların **KKT matrisi** dediği şey (Karush-Kuhn-Tucker).

> *"…these KKT matrices…"* — Strang, 36:13

Aşağıdaki figür 1D bir Lagrangian örneğini çizer: solda min x^2 öyle ki x = 4 kısıtlı problemi (kısıtlı min (4, 16)), sağda L(x,λ) = x^2 + λ(x−4) kontur haritası ve (x, λ) = (4, −8) eyer noktası — Lagrange çarpanının eyer yarattığını görselleştirir (Şekil 25.5).

> **Builder Notu — Denge Noktası Eyerdir**
>
> KKT matrisi kısıtlı optimizasyonun evrensel yapısıdır — Course 6'da (EECS) sürekli çıkar. ML köprüsü: SVM'nin dual problemi, eşitlik-kısıtlı en küçük kareler, fizik-bilgili ağlar (PINN) hep bir KKT sistemi çözer. Köşedeki sıfır blok (kısıt denklemlerinin λ'ya bağlı olmaması) bu matrisi eyer yapar — minimum değil, denge.



Şekil 25.5: Lagrange çarpanı eyer YARATIR: solda kısıtlı minimum $\min x^2$ öyle ki $x = 4$ (kısıtlı min noktası $(4, 16)$); sağda Lagrangian $L(x, \lambda) = x^2 + \lambda(x-4)$ kontur haritası, $(x^*, \lambda^*) = (4, -8)$ noktasında bir eyer. Hiperbolik konturlar eyeri gösterir — KKT matrisi belirsiz (bir özdeğer pozitif, bir negatif).

25.10 KKT Belirsizdir: Pivot İşaretleri = Özdeğer İşaretleri

Bu blok matris neden eyer? Köşegende **sıfır** blok var — bu, pozitif tanımlılığı anında bozar. Küçük örnek: [[3,1],[1,0]] determinantı negatif (−1), yani bir özdeğer artı bir özdeğer eksi. Genelde de blok eliminasyon yap: ilk n pivot S'den gelir (S pozitif tanımlı → hepsi pozitif). Sonra Schur tümleyeni:

$$\begin{bmatrix} S & A^T \\ A & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} S & A^T \\ 0 & -AS^{-1}A^T \end{bmatrix}$$

$-AS^{-1}A^T$ negatif tanımlı → sonraki m pivot negatif. Şimdi anahtar gerçek:

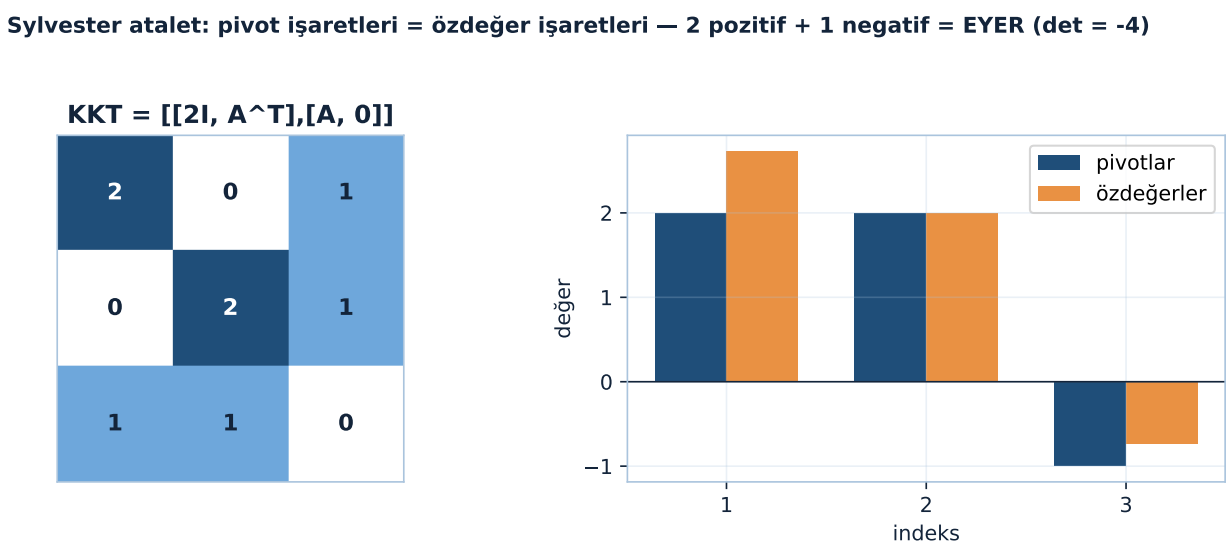
> *"Plus and minus signs of pivots give us the plus and minus signs of the eigenvalues."* — Strang, 42:02

Simetrik matriste pivot işaretleri özdeğer işaretlerini verir (Sylvester atalet yasası). Yani KKT matrisinde **n pozitif + m negatif** özdeğer → kesin bir eyer noktası.

Aşağıdaki figür Egzersiz 3'ün KKT matrisi için (S = 2I, A = [1,1]) pivot işaretleri ile özdeğer işaretlerinin birebir örtüştüğünü gösterir: değerler farklı (pivotlar [2, 2, −1]) ama işaretler aynı — 2 pozitif + 1 negatif, det = −4 < 0, bir eyer (Şekil 25.6).

Builder Notu — İşaretler Eliminasyondan

"Pivot işaretleri = özdeğer işaretleri" (atalet) bir matrisin tanımlılığını özdeğer hesaplamadan, sadece eliminasyonla söyler — çok ucuz bir test. ML köprüsü: bir kritik noktanın eyer mi minimum mu olduğu Hessian'ın özdeğer işaretlerine bakılarak anlaşılır; pivot/atalet testi bunu $O(n^3)$ tam özdeğer çözümü olmadan verir.



Şekil 25.6: Egzersiz 3'ün KKT matrisi $[[2I, A^T], [A, 0]]$ için pivot işaretleri ile özdeğer işaretleri birebir örtüşür. Değerler farklı olabilir (pivotlar $[2, 2, -1]$, özdeğerler $\approx [2.73, 2.00, -0.73]$) ama İŞARETLER aynı: 2 pozitif + 1 negatif. Bu belirsiz atalet (det $= -4 < 0$) noktanın bir eyer olduğunu söyler — Sylvester atalet yasası.

25.11 Kaynak 2: Rayleigh Bölümü

İkinci eyer kaynağı, ünlü bir orandan gelir. Simetrik S için Rayleigh bölümü:

$$R(x) = \frac{x^T S x}{x^T x}$$

"…It's Rayleigh quotient." — Strang, 43:32

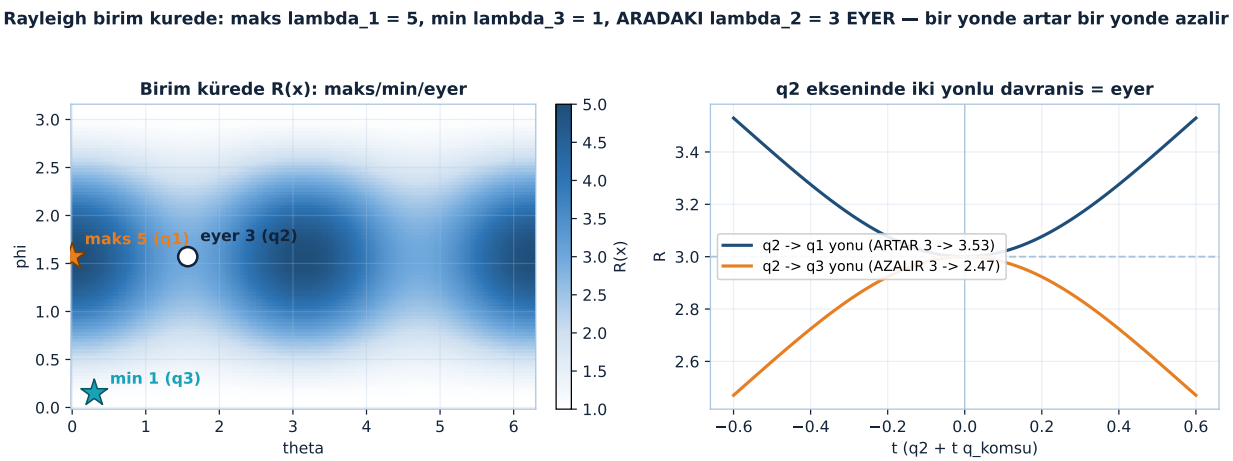
Maksimum değeri λ_1 (en büyük özdeğer), x = q_1'de (ilk özvektör); minimum değeri λ_n, x = q_n'de. Herhangi bir x için R(x) bu ikisinin arasındadır: $\lambda_n \le R(x) \le \lambda_1$. Peki **aradaki** özdeğerler (λ_2, …, λ_{n-1})?

"…those are saddle points." — Strang, 46:22

Aradaki özvektörler R(x)'in **eyer noktalarıdır**: türev orada sıfır, değer özdeğere eşit, ama bazı yönlerde yukarı bazı yönlerde aşağı gider. Maksimum/minimum kolay (bir taraf), eyer zor (iki taraflı). Bu, Ders 19'un maxmin (Courant-Fischer) ilkesine doğrudan köprü.

Aşağıdaki figür solda Rayleigh bölümünü birim küre üzerinde harita olarak gösterir (maks 5 = q_1, min 1 = q_3, aradaki eyer 3 = q_2), sağda q_2 ekseninde iki yönlü davranışı kanıtlar: q_1 yönünde R 3 → 3.53 artar, q_3 yönünde 3 → 2.47 azalır — tam bir eyer (Şekil 25.7).

> **Builder Notu — Spektrumun Motoru**
>
> Rayleigh bölümü spektral yöntemlerin motoru: en büyük/küçük özdeğeri bulmak = R(x)'i maks/min etmek (power iteration, Lanczos — Ders 11). ML köprüsü: PCA (en büyük varyans yönü = R'yi maksimize eden x), spektral kümeleme (Ders 35), ve graf Laplacian'ın Fiedler vektörü hep Rayleigh bölümü



Şekil 25.7: Rayleigh bölümü birim kürede: maks $\lambda_1 = 5$ (q_1), min $\lambda_3 = 1$ (q_3), aradaki $\lambda_2 = 3$ (q_2) bir EYER noktasıdır — q_1 yönünde $3 \to 3.53$ artar, q_3 yönünde $3 \to 2.47$ azalır.

optimizasyonudur. Aradaki eyer noktaları, ara özdeğer/özvektörleri bulmanın neden zor olduğunu açıklar.

25.12 Bu Dersin Özeti

- **Yapı taşları:** L = ½n(n−1), U = ½n(n+1), köşegen = n, ortogonal Q = ½n(n−1), simetrik S = ½n(n+1), özvektör X = n^2−n.
- **Kontroller (hepsi n^2):** LU, QR, $X\Lambda X^{-1}$, QS (kutupsal) $\to n^2$. Ayrışım bilgi kaybetmez.
- **SVD (tam rank m):** U + Σ + V = ½m(m−1) + m + (mn − ½m(m+1)) = mn.
- **Rank-R matris:** R(m+n−R) parametre; matris uzayında bir manifold (alt-uzay değil).
- **Eyer kaynağı 1 (KKT):** ½x^TSx, Ax=b kısıtı $\to$ Lagrangian $\to$ blok [[S, A^T],[A, 0]]; köşedeki 0 $\to$ belirsiz. Pivotlar: n pozitif (S'den) + m negatif (−$AS^{-1}A^T$'den) = eyer.
- **Pivot işaretleri = özdeğer işaretleri** (Sylvester atalet).
- **Eyer kaynağı 2 (Rayleigh):** R(x) = x^TSx/x^Tx; maks λ_1 (q_1), min λ_n (q_n), aradaki özdeğerler eyer noktası.

! Tek Bir Cümle

Her matris ayrışımı orijinal matrisle tam aynı sayıda serbest parametre taşır (LU/QR/QS = n^2, SVD = mn, rank-R = R(m+n−R)); eyer noktaları ise ya Lagrange kısıtlarının KKT blok matrisinden (n pozitif + m negatif özdeğer) ya da Rayleigh bölümünün aradaki özdeğerlerinden doğar.

25.13 Kontrol Soruları

i Soru 1: n×n ortogonal matris Q neden n^2 değil ½n(n−1) serbest parametre içerir

Kolonlar birim ve birbirine dik olmalı. İlk kolon birim → n yerine n−1 serbest. İkinci kolon birim **ve** birinciye dik (iki kısıt) → n−2. Üçüncü → n−3, … Toplam (n−1)+(n−2)+⋯+0 = ½n(n−1). Ortogonallik kısıtları parametrelerin yaklaşık yarısını "yer".

i Soru 2: $A = U\Sigma V^T$ (m×n, tam rank m) tam kaç parametreye toplanır ve neden

mn'ye toplanır (A'nın eleman sayısı). U (m×m ortogonal) = ½m(m−1), Σ = m, V'nin önemli ilk m kolonu = mn − ½m(m+1) (geri kalanı boş-uzaydan, sayılmaz). Toplam ½m(m−1) + m + mn − ½m(m+1) = mn. Tam rank SVD bilgi kaybetmez.

i Soru 3: KKT matrisi [[S, A^T],[A, 0]] neden pozitif tanımlı olamaz, ve özdeğer işaretleri nedir

Köşegende sıfır blok var; pozitif tanımlı bir matrisin köşegeni hep pozitif olmalı, sıfır bunu bozar. Blok eliminasyon: ilk n pivot S'den (pozitif tanımlı → pozitif), Schur tümleyeni $-AS^{-1}A^T$ negatif tanımlı → sonraki m pivot negatif. Pivot işaretleri = özdeğer işaretleri (Sylvester atalet), yani n pozitif + m negatif özdeğer → eyer noktası.

i Soru 4: Rayleigh bölümü $R(x) = x^TSx/x^Tx$'in maksimum, minimum ve eyer noktaları hangi özdeğer/özvektörlerdir

Maksimum = λ_1 (en büyük özdeğer), x = q_1 (ilk özvektör); minimum = λ_n, x = q_n. Herhangi x için $\lambda_n \le R(x) \le \lambda_1$. Aradaki özdeğerler λ_2, …, λ_{n-1} ise eyer noktalarıdır: o özvektörlerde türev sıfır ama bazı yönlerde yukarı, bazılarında aşağı gider. Bu yüzden uç (maks/min) özdeğerler kolay, ara özdeğerler zordur.

25.14 Egzersizler

1. **Sayım kontrolü.** n = 3 al. L, U, köşegen, ortogonal Q ve simetrik S için serbest parametre sayılarını hesapla. LU ve QR'ın 9 (= n^2) verdiğini doğrula. (Motor: L=3, U=6, köşegen=3, Q=3, S=6 → LU = 3+6 = 9, QR = 3+6 = 9 = n^2.)

2. **Rank-R formülü.** 100×200 bir matris rank-5 ise R(m+n−R) formülüyle kaç parametre içerir? Tam (mn = 20000) sayıyla karşılaştır; sıkıştırma oranı nedir? (Motor: R(m+n−R) = 5·(100+200−5) = 1475 vs 20000 — oran ≈ 13.6×.)

3. **KKT örneği.** S = [[2,0],[0,2]], A = [1, 1] (tek kısıt), b = [4]. KKT blok matrisini [[S, A^T],[A, 0]] yaz (3×3). Determinantının negatif olduğunu (eyer) göster. (Motor: 3×3 blok det = −4 < 0; çözüm x = [2, 2], λ = −4.)

4. **Rayleigh sınırları.** S = diag(5, 3, 1). x = $[1,1,1]^T$ için $R(x) = x^TSx/x^Tx$'i hesapla. Sonucun λ_n=1 ile λ_1=5 arasında olduğunu doğrula. (Motor: R([1,1,1]) = 3 ∈ [1, 5].)

5. **(Ders 19 habercisi)** Bu derste Rayleigh bölümünün maks = λ_1, min = λ_n verdiğini gördük; aradaki özdeğerler eyerdi. Peki λ_2'yi (ikinci özdeğer) bir **maks-min** olarak nasıl yazarsın? Bir tahmin yaz — Ders 19 "eyer noktaları (devam) + maxmin ilkesi"ni (Courant-Fischer) işliyor; Phase 2 paralel NYU dersinin spektral graf ağı (GCN) köprüsü de orada.

25.15 Sonraki Ders İçin Hazırlık

Ders 19: Eyer Noktaları (Devam), Maxmin İlkesi. Bu dersin Rayleigh bölümü eyer noktalarını sistematikleştiririz: Courant-Fischer maxmin/minimax ilkesi her özdeğeri λ_k'yı bir maks-min olarak tanımlar. Ders 15'in interlacing'i buradan kanıtlanır. NYU paralel dersinin spektral graf evrişim ağı (GCN) köprüsü de Ders 19'da.

> ⚠ Ders 19 Öncesi Yapılacak
>
> Bu dersin Rayleigh bölümü sonucunu hatırla: maks = λ_1 (q_1'de), min = λ_n (q_n'de), aradaki özdeğerler eyer. Ders 19 aradaki $\lambda_2, \ldots, \lambda_{n-1}$'i sistematik bir **maks-min** (Courant-Fischer) olarak yazacak. Soru: λ_2'yi bir alt-uzay üzerinden maksimum, alt-uzaylar üzerinden minimum olarak nasıl tanımlarsın? Eyerin "bir yönde artar, bir yönde azalır" davranışını (Şekil 25.7) düşün — maxmin tam bunu yakalar.

25.16 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Serbest parametre	ayrışım = A ile aynı sayı; bilgi korunur	0m59
Yapı taşları	L = ½n(n−1), U = ½n(n+1), Q = ½n(n−1)	2m05
Özvektör matrisi	X = n^2−n, Λ = n, $X^{-1} = 0 \to n^2$	4m09
Kutupsal QS	ortogonal × simetrik = n^2	9m33
Rank-R matris	R(m+n−R); matris uzayında manifold	19m30
Eyer: iki kaynak	KKT (Lagrange) + Rayleigh bölümü	27m08
KKT matrisi	[[S, A^T],[A, 0]]; belirsiz (eyer)	36m13
Pivot = özdeğer işareti	n pozitif + m negatif (Sylvester atalet)	42m02
Rayleigh bölümü	$R = x^TSx/x^Tx$; maks λ_1, min λ_n	43m32
Aradaki = eyer	$\lambda_2 \ldots \lambda_{n-1}$ eyer noktaları	46m22

25.17 ML Bağlantıları Özeti

- **Etkin parametre / serbestlik derecesi:** modelin gerçek kapasitesi; düşük-rank/ortogonal/simetri kısıtları nominal parametreyi azaltır (aşırı-öğrenme analizi).
- **LoRA / düşük-rank:** rank-R matris R(m+n−R) parametre; n^2 yerine ~2nR ile fine-tuning; rank-R manifoldu üzerinde optimizasyon.
- **KKT / kısıtlı optimizasyon:** SVM dual, eşitlik-kısıtlı en küçük kareler, PINN — hepsi KKT blok sistemi çözer (eyer = denge).

- **Eyer noktaları:** derin ağ kayıp yüzeyinde minimumdan çok eyer var; SGD gürültüsü eyerden kaçışı sağlar (Ders 25).
- **Rayleigh / spektral:** PCA (maks varyans), spektral kümeleme (Ders 35), Fiedler vektörü — hepsi Rayleigh bölümü optimizasyonu.
- **Pivot/atalet testi:** Hessian'ın eyer mi minimum mu olduğunu özdeğer çözmeden, eliminasyon işaretleriyle anla.
- **Geriye köprü:** Ders 2 (5 ayrışım), Ders 3 (ortogonal Q), Ders 5 (pozitif tanımlı, pivot), Ders 6 (SVD), Ders 16 (rank-R/matris tamamlama), Ders 17 (parametre sayma).

! Ayrışımlar Korur, Eyerler Armağandır

> *"…the number of free parameters agrees with the number of parameters in A itself, like n squared."* — Strang, 0:59

Ayrışımlar bilgiyi farklı kılıklarda korur; eyer noktaları ise kısıtların (KKT) ve ara özdeğerlerin (Rayleigh) kaçınılmaz armağanıdır.

26 Eyer Noktaları (Devam), Maxmin İlkesi

Courant-Fischer maxmin ilkesi, interlacing kanıtı ve covariance köprüsü

> **Bölüm bilgisi**
>
> **Video:** MIT 18.065 — Saddle Points Continued, Maxmin Principle · **OCW:** Lecture 19 · **Okuma süresi:** ≈32 dk · **Konuşmacı:** Gilbert Strang · **Önkoşul:** Ders 18 (parametre sayımı, Rayleigh, eyer). Ders 18'in eyer noktası fikrini tamamlıyoruz. Çekirdek: **maxmin ilkesi** (Courant-Fischer) — aradaki bir özdeğeri (eyer değeri) bir **maks-min** olarak yaz. Bu, Ders 15'in **interlacing** teoremini doğrudan kanıtlar. Dersin sonu istatistiğe (covariance) köprü kurar; lineer cebir bloğu kapanıyor, istatistik bloğu başlıyor.

26.1 Bu Derste Ne Var?

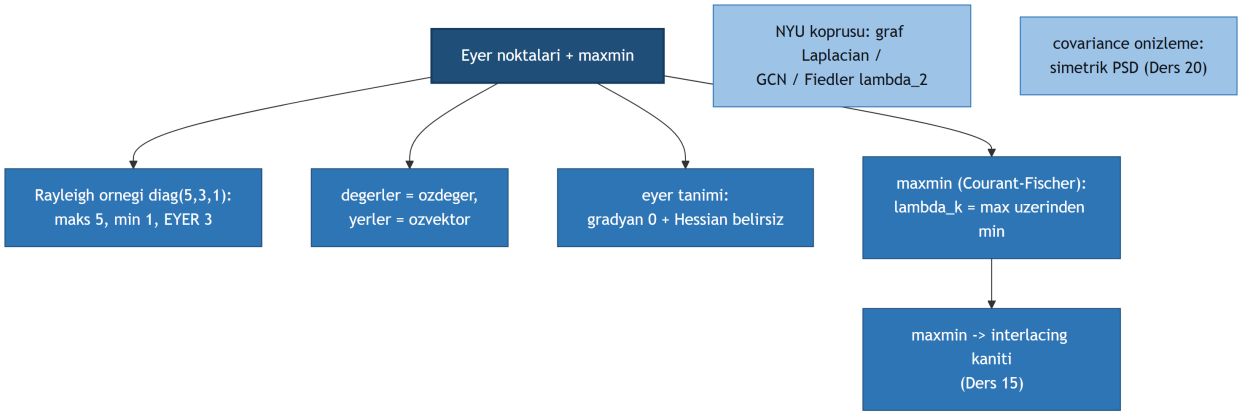
Bu ders Ders 18'in yarım kalan eyer tartışmasını kapatır ve dört sonuca yürür: önce somut bir Rayleigh örneğinde değerlerin nereden geldiğini görürüz, ardından her kritik noktanın bir özvektör (değeri bir özdeğer) olduğunu okuruz, sonra eyer değerini bir **maks-min** olarak yazan ilkeyi (Courant-Fischer) kurarız ve son olarak bu ilkeden **interlacing** kanıtının nasıl düştüğünü görürüz. Kapanışta covariance köprüsü açılır.

Dört sonuç:

1. **Rayleigh örneği:** $S = \text{diag}(5, 3, 1)$ için R'nin maks = 5 (özdeğer λ_1, özvektör e_1), min = 1 (λ_3, e_3), **eyer = 3** (λ_2, e_2).
2. **Değerler = özdeğer, yerler = özvektör:** Rayleigh'in tüm kritik noktaları özvektör, değerleri özdeğer.
3. **Maxmin ilkesi:** $\lambda_k = \max$ (boyut-k alt-uzaylar V) min ($x \in V$) $R(x)$. Eyeri maks-min'e çevirir.
4. **Interlacing kanıtı:** maxmin'den, S perturbe edilince veya bir satır+sütun atılınca özdeğerlerin neden içiçe geçtiği düşer.

> *"…what happens if you have a saddle point or a degenerate minimum?"* — Strang, 1:00 (derin öğrenme motivasyonu)

Kavram haritası (Şekil 26.1) dersin omurgasını gösterir: merkezde eyer noktaları ve maxmin ilkesi, dallarda Rayleigh örneği, özdeğer-özvektör okuması, eyer tanımı ve Courant-Fischer'dan interlacing'e giden kanıt; yan düğümlerde NYU GCN köprüsü ve covariance önizlemesi.



Şekil 26.1: Ders 19 kavram haritasi: eyer noktalari ve maxmin ilkesi merkezde; Rayleigh ornegi, ozdeger-ozvektor okumasi, eyer tanimi, Courant-Fischer maxmin ve interlacing kaniti dallarda; NYU GCN koprusu ve covariance onizlemesi ayri dugumlerde.

Builder Notu — Eyerden Maks-Min'e

- **Eyer noktaları = derin öğrenmenin merkezi:** yüksek boyutlu kayıp yüzeyinde minimumdan çok eyer var; gradient descent'in (Ders 22) eyerlerden kaçışı başarının anahtarı.
- **Maxmin (Courant-Fischer)** — ara özdeğerleri optimizasyon diliyle tanımlar; spektral graf teorisinin (NYU paralel ders, GCN) temeli.
- λ_2 **= Fiedler değeri** — graf bölmenin (spektral kümeleme, Ders 35) anahtarı; maxmin ile karakterize edilir.
- **Geriye köprü:** Ders 18 (Rayleigh, eyer, KKT), Ders 15 (interlacing), Ders 16 (Weyl), Ders 5 (pozitif tanımlı).

26.2 Eyer Noktaları Neden Önemli?

Strang neden eyer noktalarına bu kadar yer ayırdığını açıklar: derin öğrenmenin kalbi, toplam maliyet fonksiyonunun minimumunu gradient descent ile bulmaktır. Ama yüksek boyutta her şey minimum değildir.

> *"…what happens if you have a saddle point or a degenerate minimum?"* — Strang, 1:00

Maksimum ve minimumu biliyoruz; eyer noktaları daha bulanık. Derin öğrenmenin anlaşılması giderek "gradient descent ne üretiyor, eyerlerde ne oluyor?" sorusuna odaklanıyor. Bu yüzden eyeri netleştirmek şart.

Builder Notu — Kayıp Yüzeyinin Çoğu Eyer

Yüksek boyutlu kayıp yüzeylerinde kritik noktaların ezici çoğunluğu eyerdir — gerçek yerel minimum nadirdir. ML köprüsü: derin öğrenmenin "neden çalışıyor?" gizeminin büyük kısmı, SGD'nin (Ders 25) eyerlerden nasıl kaçtığıyla ilgili; eyer geometrisi optimizasyon teorisinin sıcak konusu.

26.3 Rayleigh Örneği: S = diag(5, 3, 1)

Somut örnek. Köşegen (dolayısıyla simetrik) bir matris al — her simetrik matris ortogonal değişkene dönüştürülebileceğinden genelliği kaybetmeyiz. $x = (u, v, w)$, 3 boyut:

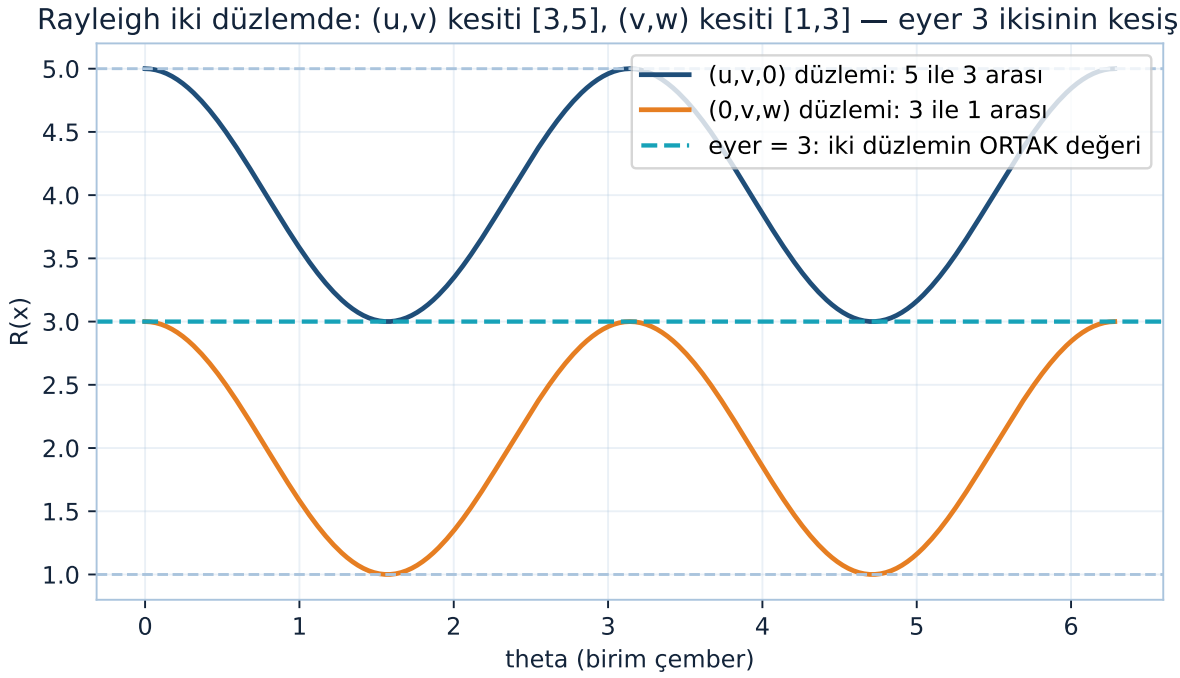
$$R(x) = \frac{x^T S x}{x^T x} = \frac{5u^2 + 3v^2 + w^2}{u^2 + v^2 + w^2}$$

Üç soru: maks, min ve eyer. **Maks** = 5: tüm ağırlığı u'ya yükle, $x = (1, 0, 0)$. **Min** = 1: her şeyi w'ye yükle, $x = (0, 0, 1)$. Birinci türevlerin sıfır olduğu **üçüncü** bir nokta daha var mı? Evet:

$$\max R = 5 \text{ at } (1, 0, 0), \quad \min R = 1 \text{ at } (0, 0, 1), \quad \text{saddle } R = 3 \text{ at } (0, 1, 0)$$

Eyer değeri 3, orta noktada $(0, 1, 0)$.

İki düzlem kesiti bunu görselleştirir (Şekil 26.2): birim çemberi $(u, v, 0)$ düzleminde gezdirirsen R değerleri $[3, 5]$ arasında salınır (5 ile 3'ün arası); $(0, v, w)$ düzleminde ise $[1, 3]$ arasında. Eyer değeri **3, iki düzlemin ORTAK değeridir** — bir düzlemde maks, diğerinde min. İşte eyerin geometrik imzası budur.



Şekil 26.2: Rayleigh iki düzlemde: (u,v) kesiti [3,5], (v,w) kesiti [1,3] — eyer 3 ikisinin kesişimi (maks bir düzlemde min diğerinde)

> **Builder Notu — Önce Köşegenleştir Sonra Oku**
>
> Rayleigh bölümünü köşegen S ile incelemek tüm genelliği korur (Ders 18): herhangi simetrik S, ortogonal bir değişken dönüşümüyle köşegen hâle gelir, R'nin maks/min/eyer yapısı değişmez. Bu "önce köşegenleştir, sonra oku" stratejisi spektral analizin standart ilk hamlesidir.

26.4 Değerler = Özdeğer, Yerler = Özvektör

Rayleigh bölümünün kritik noktalarının (birinci türevin sıfır olduğu yerler) tümü S'nin özvektörleridir; oradaki değerler özdeğerlerdir:

> *"The values there are the eigenvalues. And the places where you reach them are the eigenvectors."*
> — Strang, 7:38

Örnekte: maks 5 = λ_1 (e_1), min 1 = λ_3 (e_3), eyer 3 = λ_2 (e_2). Rayleigh karmaşık bir fonksiyon (türevi için bölüm kuralı veya Lagrange çarpanı gerek), ama sonucu kusursuz: değerler özdeğer, yerler özvektör. Pratik not: en büyük ve en küçük özvektörü hesaplamak, **aradaki** (eyer) özvektörleri hesaplamaktan genelde çok daha kolaydır.

İki düzlem kesiti (Şekil 26.2) bunu da gösterir: tepe değer 5 ile dip değer 1, eğrilerin uç noktalarında (özvektör doğrultularında) okunur; eyer değeri 3 ise iki düzlemin ortak noktasıdır.

> Builder Notu — Uçlar Kolay Ara Zor
>
> "Uç özdeğerler kolay, ara özdeğerler zor" gerçeği sayısal lineer cebirin temel asimetrisidir: power iteration en büyüğü bulur, ama λ_2'yi (Fiedler) bulmak deflation/Lanczos ister. ML köprüsü: spektral kümeleme (Ders 35) tam da λ_2 özvektörüne (Fiedler vektörü) ihtiyaç duyar — bu yüzden iyi kodlar ve dikkat gerektirir.

26.5 Eyer = Gradyan 0, Hessian Belirsiz

Eyer noktası iki koşulla tanımlanır. (1) Birinci türevler (gradyan vektörü) sıfır:

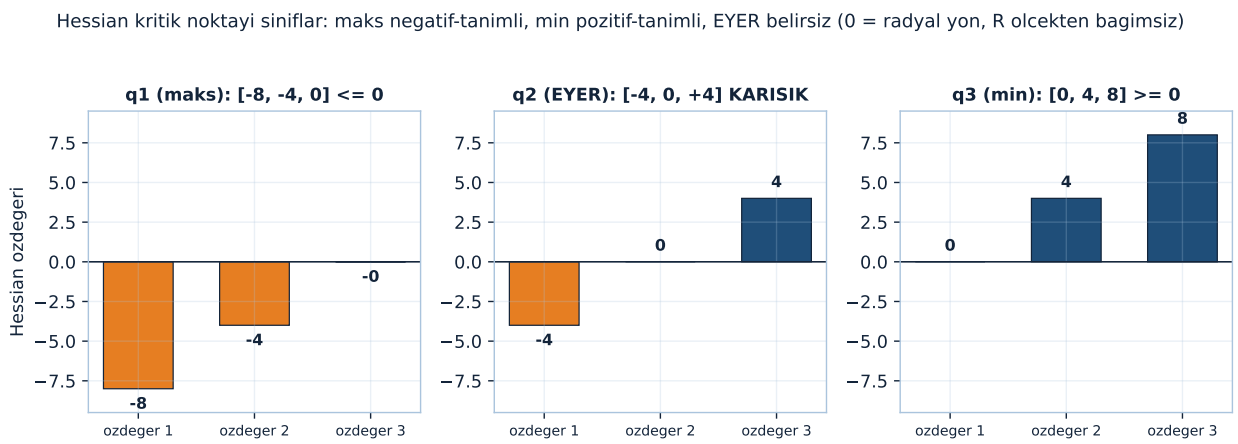
$$\nabla R = \left(\frac{\partial R}{\partial u}, \frac{\partial R}{\partial v}, \frac{\partial R}{\partial w} \right) = 0$$

(2) İkinci türevler — 3×3 bir matris (Hessian). Karışık türevler eşit olduğundan ($\partial^2 R/\partial u\,\partial v = \partial^2 R/\partial v\,\partial u$) bu matris **simetrik**:

$$H = \begin{bmatrix} R_{uu} & R_{uv} & R_{uw} \\ R_{vu} & R_{vv} & R_{vw} \\ R_{wu} & R_{wv} & R_{ww} \end{bmatrix}$$

Maksimumda H negatif tanımlı, minimumda pozitif tanımlı, **eyerde belirsiz** (karışık işaretli özdeğerler) — bazı yönlerde yukarı, bazılarında aşağı.

Üç kritik noktada sayısal Hessian'ı (merkez fark) hesaplayıp özdeğerlerine bakınca işaret deseni netleşir (Şekil 26.3): maks q_1'de $[-8, -4, 0] \leq 0$ (negatif-tanımlı), min q_3'te $[0, 4, 8] \geq 0$ (pozitif-tanımlı), eyer q_2'de ise $[-4, 0, +4]$ — işaretler **karışık**. Buradaki 0 özdeğer radyal yöndendir: R ölçekten bağımsız (0-homojen) olduğundan o yönde değişmez. Sınıflandırmayı yapan tam da bu işaret desenidir.



Şekil 26.3: Üç kritik noktanın sayısal Hessian özdeğerleri (merkez fark). Maksimum (q_1) negatif-tanımlı [−8, −4, 0], minimum (q_3) pozitif-tanımlı [0, 4, 8], eyer (q_2) ise işaretleri karışık [−4, 0, +4] — sınıflandırmayı yapan tam da bu işaret deseni.

> **Builder Notu — İşaretler Kritik Noktayı Sınıflar**
>
> Hessian'ın tanımlılığı bir kritik noktayı sınıflar: pozitif tanımlı = minimum, negatif tanımlı = maksimum, belirsiz = eyer (Ders 5 + Ders 18 pivot/atalet testi). ML köprüsü: ikinci-derece optimizasyon (Newton) ve eyer-kaçış yöntemleri Hessian'ın özdeğer işaretlerine bakar; derin ağ eğitiminde negatif eğrilik (negative curvature) yönleri eyerden çıkış kapısıdır.

26.6 Maxmin İlkesi (Courant-Fischer)

Eyer için en güzel fikir: onu bir **maksimum-minimum** olarak yaz, böylece maks/min'in kolaylığına geri dön.

> *"…write it as the maximum of a minimum."* — Strang, 10:38

İkinci özdeğer λ_2 için: tüm **2-boyutlu** alt-uzaylar V üzerinde, her V içinde Rayleigh'in minimumunu al, sonra bu minimumların maksimumunu al:

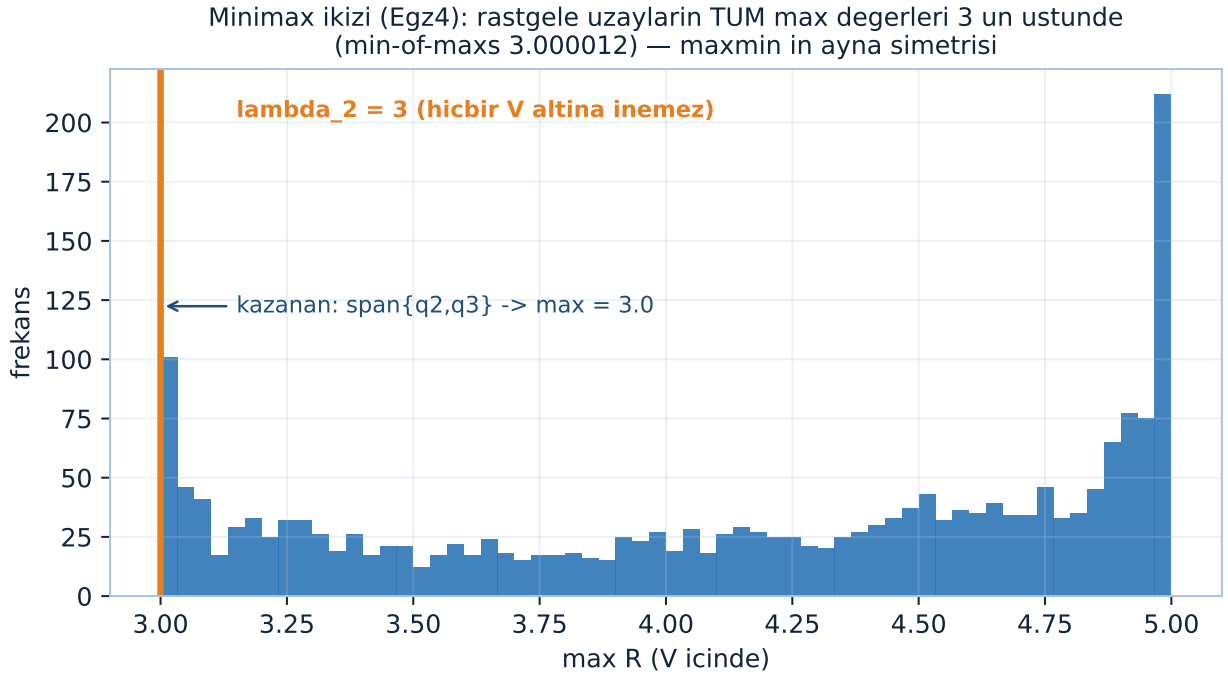
$$\lambda_2 = \max_{\dim V = 2} \; \min_{0 \neq x \in V} \; R(x)$$

Genel hâli (Courant-Fischer): k'inci özdeğer, boyut-k alt-uzaylar üzerinde maks-min:

$$\lambda_k = \max_{\dim V = k} \; \min_{0 \neq x \in V} \; \frac{x^T S x}{x^T x}$$

> *"…I'm going to take a maximum over two dimensional spaces…"* — Strang, 12:08

Bu, sayısal olarak doğrudan doğrulanabilir (Şekil 26.5): 2000 rastgele 2B alt-uzayda Rayleigh'in minimumu hesaplanır; **hepsi $\lambda_2 = 3$'ün altında** kalır (max-of-mins 2.999999) ve eşitlik yalnız özvektör uzayında (span$\{q_1, q_2\}$) sağlanır. Aynı ilkenin ayna ikizi minimax'tır (Şekil 26.4, Egz4): $n - k + 1$ boyutlu rastgele uzayların **TÜM max değerleri $\lambda_2 = 3$'ün üstünde** (min-of-maxs 3.000012); kazanan span$\{q_2, q_3\}$ tam 3.0 verir.



Şekil 26.4: Minimax ikizi (Egz4): n-k+1 boyutlu rastgele uzaylarin TUM max degerleri lambda_2 = 3 un ustunde (min-of-maxs 3.000012) — maxmin ilkesinin ayna simetrisi.

> **Builder Notu — Zoru İki Kolaya Böl**
>
> Maxmin, eyer noktasını (zor) iki kolay işlemin (min, sonra maks) bileşimine indirger — optimizasyonun klasik "minimax" yapısı (Ders 24 oyun teorisi/LP ile akraba). ML köprüsü: GAN'lar (üretici-ayırıcı) tam bir minimax problemidir; robust optimizasyon, adversarial eğitim hep maks-min/min-maks dengesi arar. Courant-Fischer bu yapının spektral kökenidir.

26.7 Örnek Hesap: $\lambda_2 = 3$

İlkeyi örnekte çalıştır ($S = \text{diag}(5, 3, 1)$, $\lambda_2 = 3$ bekliyoruz). $V = \text{span}\{e_1, e_2\}$ al, yani $(u, v, 0)$ vektörleri ($w = 0$). Bu 2B uzayda Rayleigh:

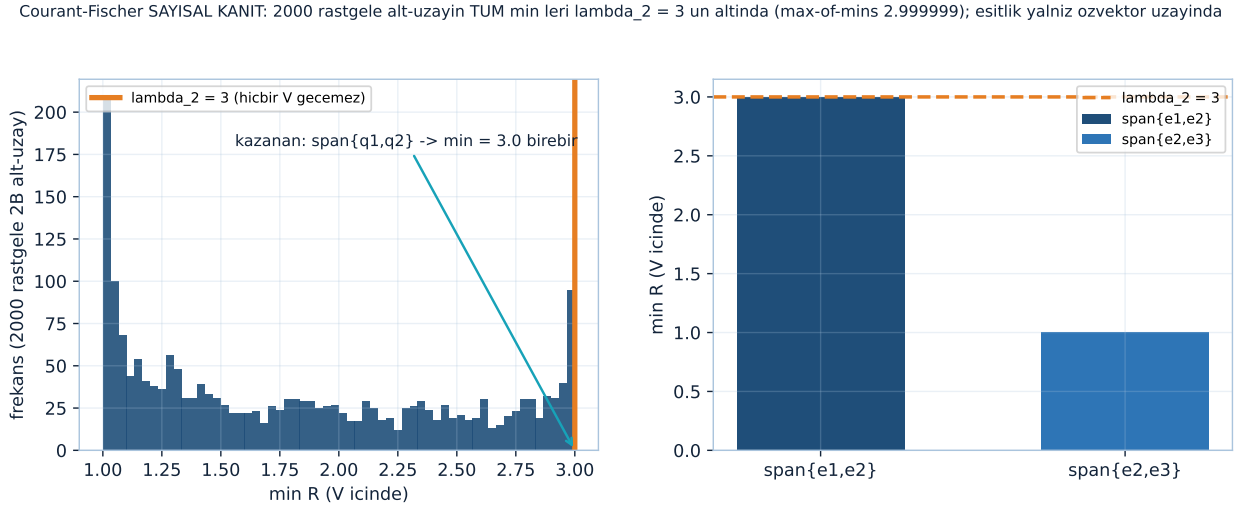
$$R = \frac{5u^2 + 3v^2}{u^2 + v^2} \;\Rightarrow\; \text{min} = 3 \text{ (tum agirlik } v)$$

Bu V için min = 3. Şimdi maks: başka bir V denersek, örn. span$\{e_2, e_3\} = (0, v, w)$ vektörleri:

$$R = \frac{3v^2 + w^2}{v^2 + w^2} \;\Rightarrow\; \min = 1$$

Bu V'nin minimumu 1 — daha küçük, kazanan değil. Tüm 2B uzaylar üzerinde minimumların maksimumu 3'tür (en iyi seçim span$\{e_1, e_2\}$). Yani $ _{2} = $ max-min $= 3$. ✓

Sayısal deney bunu tam onaylar (Şekil 26.5): solda 2000 rastgele 2B alt-uzayın min'leri histogramda hep $\lambda_2 = 3$'ün altında toplanır (hiçbir V geçemez); sağda iki örnek V — span$\{e_1, e_2\}$ min = 3 (kazanan) ile span$\{e_2, e_3\}$ min = 1 karşılaştırılır.



Şekil 26.5: Courant-Fischer maxmin SAYISAL KANIT: 2000 rastgele 2B alt-uzayın TÜM min değerleri λ_2 = 3'ün altında (max-of-mins 2.999999); eşitlik yalnız özvektör uzayında. Sağda iki örnek: span$\{e_1,e_2\}$ → min = 3 (kazanan), span$\{e_2,e_3\}$ → min = 1.

> **Builder Notu — Kazanan Alt-Uzay Özvektörlerin**
>
> "Doğru alt-uzayı seç, minimumu maksimize et" hesabı, maxmin'in neden tam λ_2'yi verdiğini somutlaştırır: en iyi 2B uzay, ilk iki özvektörün gerdiği uzaydır ve oradaki min tam λ_2'dir. ML köprüsü: bu, PCA'nın "en iyi k-boyutlu alt-uzay" karakterizasyonunun (Ders 7 Eckart-Young) özdeğer ikizidir — alt-uzay optimizasyonu ve spektral gömme aynı maxmin iskeletini paylaşır.

26.8 Maxmin → Interlacing Kanıtı

Maxmin ilkesi sadece şık bir tanım değil — Ders 15'in **interlacing** teoremini doğrudan üretir:

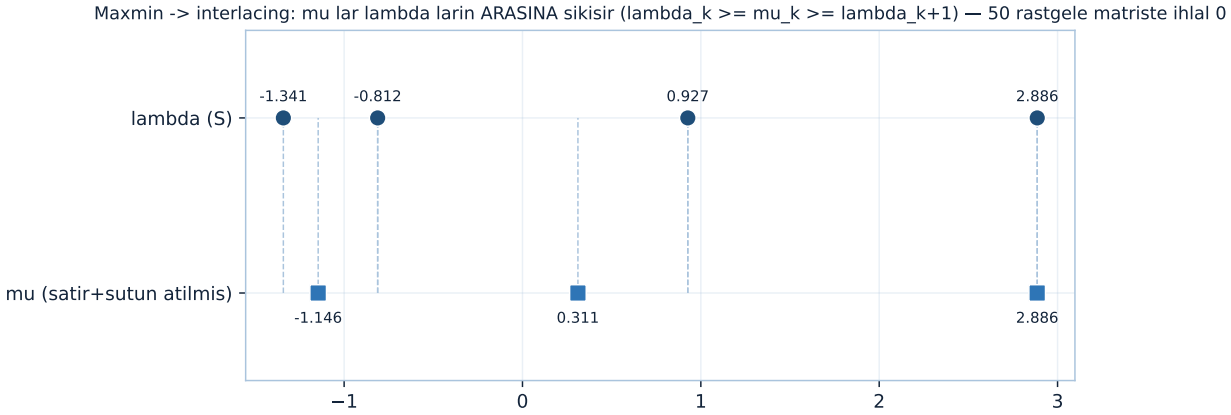
> *"…would lead you… very quickly to the interlacing theorem…"* — Strang, 11:31

Sezgi: $\lambda_k = \max_{\dim V = k} \min_{x \in V} R(x)$. Şimdi S'yi perturbe et (rank-1 ekle) veya bir satır+sütun at (alt-matris, Ders 15). Kısıtlama, alt-uzay seçim havuzunu daraltır veya genişletir; ama maks-min bir adım kayar. Örneğin bir satır+sütun atmak, alt-uzayları bir boyut daha kısıtlar — yeni λ'lar eski λ'ların **arasına** sıkışır:

$$\lambda_k(S) \geq \mu_k(S_{n-1}) \geq \lambda_{k+1}(S)$$

Çünkü min-üzerinden-max yapısında bir kısıt eklemek değeri ne çok yukarı ne çok aşağı oynatabilir — tam interlacing. Ders 16'nın Weyl eşitsizliği de aynı maxmin makinesinden gelir.

Sayı doğrusu üzerinde bu içiçe geçiş açıkça görünür (Şekil 26.6): 4×4 rastgele simetrik bir S için özdeğerler $\lambda = [2.886, 0.927, -0.812, -1.341]$, son satır+sütun atılınca elde edilen 3×3 alt-matrisin özdeğerleri $\mu = [2.886, 0.311, -1.146]$ — her μ_k tam olarak λ_k ile λ_{k+1} arasına sıkışır. 50 rastgele matriste ihlal sayısı 0.



Şekil 26.6: Maxmin -> interlacing: satir+sutun atinca mu lar lambda larin ARASINA siksir (lambda_k >= mu_k >= lambda_k+1) — 50 rastgele matriste ihlal 0.

> **Builder Notu — Bir İlke İki Teorem**
>
> Maxmin'in interlacing'i "ücretsiz" üretmesi, doğru soyutlamanın gücüdür: tek bir ilke (Courant-Fischer) hem özdeğer tanımını hem perturbasyon teoremlerini verir. ML köprüsü: NYU paralel dersinin spektral graf evrişim ağları (GCN), graf Laplacian'ının özdeğerlerini kullanır; bu özdeğerler maxmin ile tanımlanır ve graf değişince (kenar ekle/çıkar) interlacing onların nasıl kayacağını sınırlar — dinamik/gürültülü graflarda GCN kararlılığının teorik temeli. **NYU H13 GCN (konuk Bresson):** graf Laplacian özdeğerleri maxmin ile tanımlanır; λ_2 = Fiedler değeri spektral kümelemenin anahtarı; interlacing dinamik graflarda spektrum kararlılığını sınırlar.

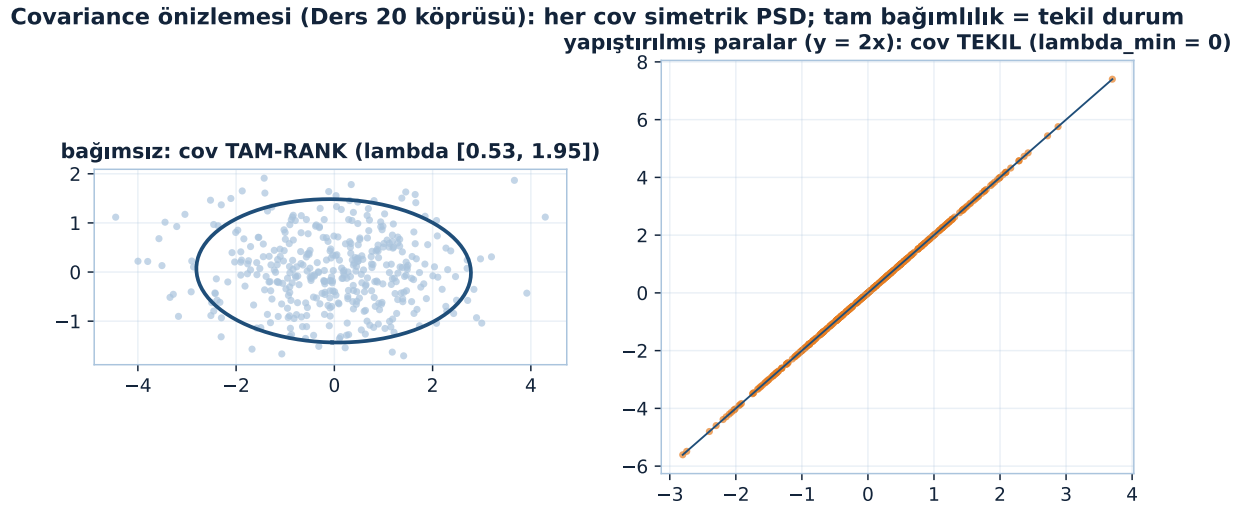
26.9 Covariance Önizlemesi (Ders 20 Köprüsü)

Strang dersin sonunda yeni konuya — istatistik ve **covariance matrisi**ne — köprü kurar. Mean ve variance bilinen kavramlar; sıradaki adım covariance (değişkenler arası ilişki). Anahtar gerçek:

> *"…that covariance matrix… will be symmetric positive definite, or semidefinite."* — Strang, 51:36

Covariance matrisi her zaman simetrik ve pozitif yarı-tanımlıdır (Ders 5). Yarı-tanımlı (tekil) durum, değişkenler **tam bağımlı** olduğunda ortaya çıkar (Strang'ın benzetmesiyle: "birbirine yapıştırılmış paralar"). Bağımsız değişkenlerde köşegen, bağımlılıkta köşegen-dışı terimler dolar.

Bunu iki örnek gösterir (Şekil 26.7): solda bağımsız değişkenlerde covariance tam-rank PSD'dir (özdeğerler $[0.53, 1.95]$), elips iki yöne de açılır; sağda "yapıştırılmış paralar" ($y = 2x$) tam bağımlılığında covariance **TEKİL** olur ($\lambda_{\text{min}} = 0$), elips bir doğruya çöker — $C = \begin{bmatrix} v & 2v \\ 2v & 4v \end{bmatrix}$ yapısı tam bağımlılığı kodlar.



Şekil 26.7: Covariance önizlemesi (Ders 20 köprüsü): bağımsız değişkenlerde kovaryans matrisi tam-rank PSD (özdeğerler [0.53, 1.95]); 'yapıştırılmış paralar' (y = 2x) tam bağımlılığında kovaryans TEKİL olur (lambda_min = 0). Her kovaryans simetrik ve PSD'dir — bu da Ders 20'nin başlangıç noktasıdır.

> Builder Notu — Yapıştırılmış Paralar
>
> Covariance matrisinin pozitif yarı-tanımlı olması, tüm bu lineer cebir bloğunu istatistiğe bağlar: öz-değerleri varyans yönlerini (PCA, Ders 7), pozitif tanımlılığı geçerli bir olasılık yapısını garanti eder. ML köprüsü: Gauss dağılımı, Mahalanobis mesafesi, Kalman filtresi (Ders 14) ve PCA hep covariance matrisinin spektral yapısına dayanır — bir sonraki blok (istatistik → optimizasyon → derin öğrenme) buradan başlar.

26.10 Bu Dersin Özeti

- **Eyer = derin öğrenme:** yüksek boyutta minimumdan çok eyer var; gradient descent'in eyerlerden kaçışı önemli.
- **Rayleigh örneği:** $S = \text{diag}(5, 3, 1)$ → maks 5 (λ_1, e_1), min 1 (λ_3, e_3), eyer 3 (λ_2, e_2).
- **Değerler = özdeğer, yerler = özvektör:** Rayleigh'in tüm kritik noktaları özvektör. Uç özdeğerler kolay, ara (eyer) zor.
- **Eyer tanımı:** gradyan $\nabla R = 0$ + Hessian belirsiz (karışık işaretli özdeğer).
- **Maxmin (Courant-Fischer):** $\lambda_k = \max_{\dim V = k} \min_{x \in V} R(x)$. Örnekte $ _{2} = $ max-min $= 3$.
- **Interlacing:** maxmin'den düşer — S perturbe/alt-matris olunca özdeğerler $\lambda_k(S) \geq \mu_k(S_{n-1}) \geq \lambda_{k+1}(S)$ arasına sıkışır. Weyl de aynı kaynaktan.
- **Covariance önizleme:** simetrik pozitif (yarı-)tanımlı; yarı-tanımlı = tam bağımlı değişkenler (Ders 20).

! Tek Bir Cümle

Bir simetrik matrisin aradaki özdeğerleri Rayleigh bölümünün eyer noktalarıdır ve maxmin ilkesiyle ($ _{k} = $ boyut-k alt-uzaylar üzerinde maks, alt-uzay içinde min) maks-min olarak yazılabilir — bu da interlacing teoremini doğrudan kanıtlar.

26.11 Kontrol Soruları

i Soru 1: S = diag(5, 3, 1) için Rayleigh bölümünün maks, min ve eyer değerleri nedir, hangi vektörlerde

Cevap: Maks = 5, $x = (1, 0, 0)$ (λ_1, e_1); min = 1, $x = (0, 0, 1)$ (λ_3, e_3); eyer = 3, $x = (0, 1, 0)$ (λ_2, e_2). Değerler özdeğerler, yerler özvektörlerdir. Maks tüm ağırlığı en büyük köşegene (5), min en küçüğe (1), eyer ortadakine (3) yükleyince çıkar.

i Soru 2: Bir kritik noktanın eyer olduğunu Hessian'dan nasıl anlarsın

Cevap: Eyerde gradyan $\nabla R = 0$ (tüm birinci türevler sıfır) **ve** Hessian (ikinci türev matrisi, simetrik) **belirsizdir** — özdeğerleri karışık işaretli. Minimumda Hessian pozitif tanımlı, maksimumda negatif tanımlı; eyerde hem pozitif hem negatif özdeğer var (bazı yönde yukarı, bazı yönde aşağı).

i Soru 3: Maxmin (Courant-Fischer) ilkesiyle λ_2 nasıl yazılır

Cevap: $\lambda_2 = \max_{\dim V=2} \min_{0\neq x\in V} R(x)$: tüm 2-boyutlu alt-uzaylar V üzerinde, her V içinde Rayleigh'in minimumunu al, sonra bu minimumların maksimumunu al. Genel olarak $ _{k} = $ boyut-k alt-uzaylar üzerinde maks, alt-uzay içinde min. Eyer değerini iki kolay işlemin (min sonra maks) bileşimine indirger.

i Soru 4: Maxmin ilkesi interlacing teoremini neden kolayca verir

Cevap: $ _{k} = $ max-min yapısında bir kısıt eklemek (S'yi perturbe etmek veya bir satır+sütun atmak) alt-uzay seçim havuzunu daraltır/genişletir; bu, maks-min değerini sadece bir adım oynatabilir — ne çok yukarı ne çok aşağı. Sonuç: yeni özdeğerler eski özdeğerlerin arasına sıkışır, $\lambda_k(S) \geq \mu_k(S_{n-1}) \geq \lambda_{k+1}(S)$. Weyl eşitsizliği de aynı maxmin makinesinden gelir.

26.12 Egzersizler

1. **Rayleigh kritik noktaları.** $S = \text{diag}(6, 4, 2)$. $R(x)$'in maks, min ve eyer değerlerini ve bunlara karşılık gelen vektörleri yaz. Eyer değeri hangi özdeğer? *(İpucu: maks 6, min 2, eyer 4 = λ_2.)*

2. **2B minimum.** $S = \text{diag}(6, 4, 2)$, $V = \text{span}\{e_1, e_3\}$ (yani $(u, 0, w)$ vektörleri). Bu V üzerinde R'nin minimumunu bul. Maxmin yarışında bu V kazanır mı ($\lambda_2 = 4$ için)? *(İpucu:* $\text{span}\{e_1, e_3\}$ *min = 2; kazanmaz.)*

3. **Maxmin doğrulaması.** Egzersiz 2'deki S için, $\lambda_2 = 4$'ü veren kazanan 2B alt-uzayı bul ve o uzaydaki minimumun 4 olduğunu göster. *(İpucu: kazanan* span$\{e_1, e_2\}$*, min = 4 =* λ_2*.)*

4. **Minimax dual.** Maxmin'in ikizi minimax: $\lambda_k = \min_{\dim V = n-k+1} \max_{x \in V} R(x)$. $S = \text{diag}(5, 3, 1)$ için λ_2'yi minimax formuyla (3-boyutta $n - k + 1 = 2$ boyutlu uzaylar) hesapla; yine 3 çıktığını doğrula. Minimax ikizinin sayısal kanıtı (Şekil 26.4): 2000 rastgele 2B uzayın TÜM max'ları $\lambda_2 = 3$'ün üstünde (min-of-maxs 3.000012), kazanan span$\{q_2, q_3\}$ tam 3.0.

5. **(Ders 20 habercisi)** Bu derste covariance matrisinin simetrik pozitif yarı-tanımlı olduğunu gördük. Peki covariance matrisi tam olarak nasıl tanımlanır? Mean'den sapmaların hangi çarpımı? Ne zaman tekil (yarı-tanımlı) olur? Bir tahmin yaz — Ders 20 "tanımlar ve eşitsizlikler" (covariance, mean, variance) ile istatistik bloğunu açıyor.

26.13 Sonraki Ders İçin Hazırlık

Ders 20: Tanımlar ve Eşitsizlikler (Mean, Variance, Covariance). Lineer cebir bloğu kapandı; istatistik başlıyor. Strang covariance matrisini tanımlar (mean'den sapmaların dış-çarpım ortalaması), pozitif yarı-tanımlılığını kanıtlar ve temel olasılık eşitsizliklerini (Markov, Chebyshev) işler — PCA, Gauss dağılımı ve gradient descent'in (Ders 22+) istatistiksel temeli.

> ⚠ Ders 20 Öncesi Yapılacak
>
> Bu dersin covariance önizlemesini (Şekil 26.7) gözden geçir: bağımsız değişkende cov tam-rank, tam bağımlılıkta ("yapıştırılmış paralar", $y = 2x$) tekil ($\lambda_{\text{min}} = 0$). Ders 20 bu PSD yapısını tanımdan kanıtlayacak — lineer cebir bloğunun (özdeğer/SVD/eyer/maxmin) istatistiğe açılan kapısı buradan başlıyor.

26.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Eyer = derin öğrenme	gradient descent eyerden kaçış (Ders 22)	1m00
Rayleigh örneği	$S = \text{diag}(5, 3, 1)$: maks 5, min 1, eyer 3	4m05
Değerler = özdeğer	kritik noktalar özvektör, değerler özdeğer	7m38
Eyer tanımı	$\nabla R = 0$ + Hessian belirsiz	8m10
Maxmin (Courant-Fischer)	$\lambda_k = \max_{\dim V = k} \min_{x \in V} R(x)$	10m38
Örnek $\lambda_2 = 3$	span$\{e_1, e_2\}$ min = 3 = maks-min	12m08
Maxmin → interlacing	$\lambda_k(S) \geq \mu_k(S_{n-1}) \geq \lambda_{k+1}(S)$	11m31
Covariance önizleme	simetrik pozitif (yarı-)tanımlı	51m36

26.15 ML Bağlantıları Özeti

- **Eyer geometrisi:** derin ağ kayıp yüzeyinde minimumdan çok eyer; SGD gürültüsü (Ders 25) ve negatif eğrilik yönleri eyerden kaçışı sağlar.
- **Maxmin / minimax:** GAN (üretici-ayırıcı), adversarial/robust eğitim hep min-maks dengesi; Courant-Fischer bu yapının spektral kökeni.
- **Spektral graf / GCN (NYU paralel ders):** graf Laplacian özdeğerleri maxmin ile tanımlanır; λ_2 (Fiedler değeri) graf bölme ve spektral kümelemenin (Ders 35) anahtarı; interlacing graf değişiminde spektrum kararlılığını sınırlar.
- **PCA köprüsü:** maxmin "en iyi k-boyutlu alt-uzay" karakterizasyonu = Eckart-Young'ın (Ders 7) özdeğer ikizi; spektral gömme aynı iskelet.
- **İkinci-derece optimizasyon:** Hessian özdeğer işaretleri (eyer mi minimum mu) Newton ve eyer-kaçış yöntemlerinin temeli (Ders 18 pivot/atalet).
- **Covariance → istatistik bloğu:** pozitif yarı-tanımlı covariance (Ders 20), Gauss dağılımı, Kalman filtresi (Ders 14), PCA — sıradaki blok buradan başlar.
- **Geriye köprü:** Ders 18 (Rayleigh, eyer, KKT), Ders 15 (interlacing), Ders 16 (Weyl), Ders 7 (Eckart-Young), Ders 5 (pozitif tanımlı).

! Maks-Min Eyeri Çözer, İnterlacing'i Armağan Eder

> *"…write it as the maximum of a minimum."* — Strang, 10:38 — Eyer noktası, doğru alt-uzay seçildiğinde bir maks-min'e dönüşür; bu ilke ara özdeğerleri tanımlar, interlacing'i kanıtlar ve spektral graf yöntemlerinin kapısını açar.

27 Tanımlar ve Eşitsizlikler — Mean, Variance, Covariance

İstatistik bloğu açılışı: Markov, Chebyshev ve covariance matrisi

> **Bölüm bilgisi**
>
> Bu ders lineer cebir bloğunu kapatıp **istatistik bloğunu açar**: Strang'in Ders 20 videosu (≈55 dk) ve OCW Lecture 20 temel alınmıştır. Okuma süresi ≈34 dk; önkoşul Ders 19 (covariance önizlemesi).

27.1 Bu Derste Ne Var?

Lineer cebir bloğu kapandı; **istatistik** başlıyor. Bu ders olasılığın temel tanımlarını (beklenen değer, varyans) ve iki büyük eşitsizliği (Markov, Chebyshev) tazeler, sonra **covariance matrisi**ne ulaşır — derin öğrenmenin istatistiksel temeli.

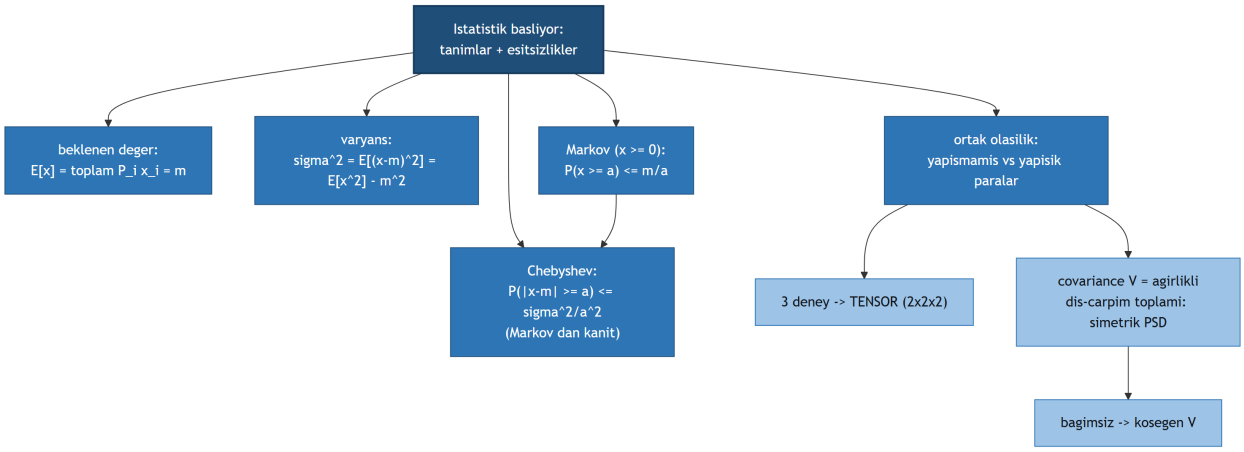
Beş sonuç:

1. **Beklenen değer:** $E[x] = \sum_i P_i x_i = m$; **varyans** $\sigma^2 = E[(x-m)^2] = E[x^2] - m^2$.
2. **Markov eşitsizliği** ($x \geq 0$): $P(x \geq a) \leq \frac{m}{a}$.
3. **Chebyshev eşitsizliği:** $P(|x-m| \geq a) \leq \frac{\sigma^2}{a^2}$ — Markov'dan $y = (x-m)^2$ ile kanıtlanır.
4. **Ortak olasılık:** bağımsız (yapışmamış) vs bağımlı (yapışık) paralar; 3 deney → **tensör** (3-yollu dizi).
5. **Covariance matrisi:** $V = \sum P_{ij}\,(\text{sapma})(\text{sapma})^\top$; köşegen = varyanslar, köşegen-dışı = σ_{xy}; bağımsız → köşegen; simetrik pozitif yarı-tanımlı.

> *"…part of deep learning as we get there."* — Strang, 0:28

Kavram haritası (Şekil 27.1) dersin akışını gösterir: merkezdeki "istatistik başlıyor" düğümünden beş dal (beklenen değer, varyans, Markov, Chebyshev, ortak olasılık) çıkar; Markov, Chebyshev'i besler (kanıt oku), ortak olasılık ise hem tensöre hem covariance matrisine açılır.

> **Builder Notu — İstatistik Bloğunun Kapısı**
>
> Burası kitabın ikinci yarısının başlangıcı: 19 dersin lineer cebiri artık olasılığa hizmet ediyor. **Covariance matrisi = veri yapısı** — PCA (Ders 7), Mahalanobis mesafesi, Gauss dağılımı, Kalman filtresi (Ders 14) hep bu simetrik pozitif yarı-tanımlı matrise dayanır. **Markov/Chebyshev = konsantrasyon eşitsizlikleri** — ML genelleme sınırlarının (generalization bounds) ataları; "örneklem ortalaması gerçek ortalamaya ne kadar yakın?" sorusunu yanıtlar. **Tensör** — derin öğrenmenin temel veri yapısı (PyTorch/NumPy ndarray); 3+ yollu dizi ilk kez burada çıkıyor. Geriye köprü: Stat 110 (mean/variance/Markov/Chebyshev



Şekil 27.1: Ders 20 kavram haritasi: “Istatistik basliyor: tanimlar + esitsizlikler” merkezde; beklenen deger E[x], varyans sigma^2, Markov esitsizligi, Chebyshev (Markov dan kanit) ve ortak olasilik (yapismamis vs yapisik paralar) dallarda; uc deneyin tensore tasmasi, covariance V nin agirlikli dis-carpim toplami olarak simetrik PSD olusu ve bagimsizligin kosegen V vermesi ayri dugumlerde.

— §4.B), Ders 5 (pozitif tanımlı), Ders 19 (covariance önizleme).

27.2 1. Beklenen Değer (Mean)

İstatistik bloğu başlıyor — derin öğrenmenin de bir parçası.

> *“…part of deep learning as we get there.”* — Strang, 0:28

Örneklem ortalaması (sample mean) veriden hesaplanır; **beklenen değer** ise olasılıklarla ağırlıklı ortalamadır. x değeri P_1 olasılıkla x_1, …, P_n olasılıkla x_n alıyorsa:

$$E[x] = P_1 x_1 + P_2 x_2 + \cdots + P_n x_n = m$$

E sembolü her yerde kullanılır — pratik bir kısaltma. Genel olarak herhangi bir $f(x)$ fonksiyonunun beklenen değeri de olasılıklarla ağırlıklıdır:

$$E[f(x)] = \sum_i P_i \, f(x_i)$$

Builder Notu — Riskin Dili E Sembolü

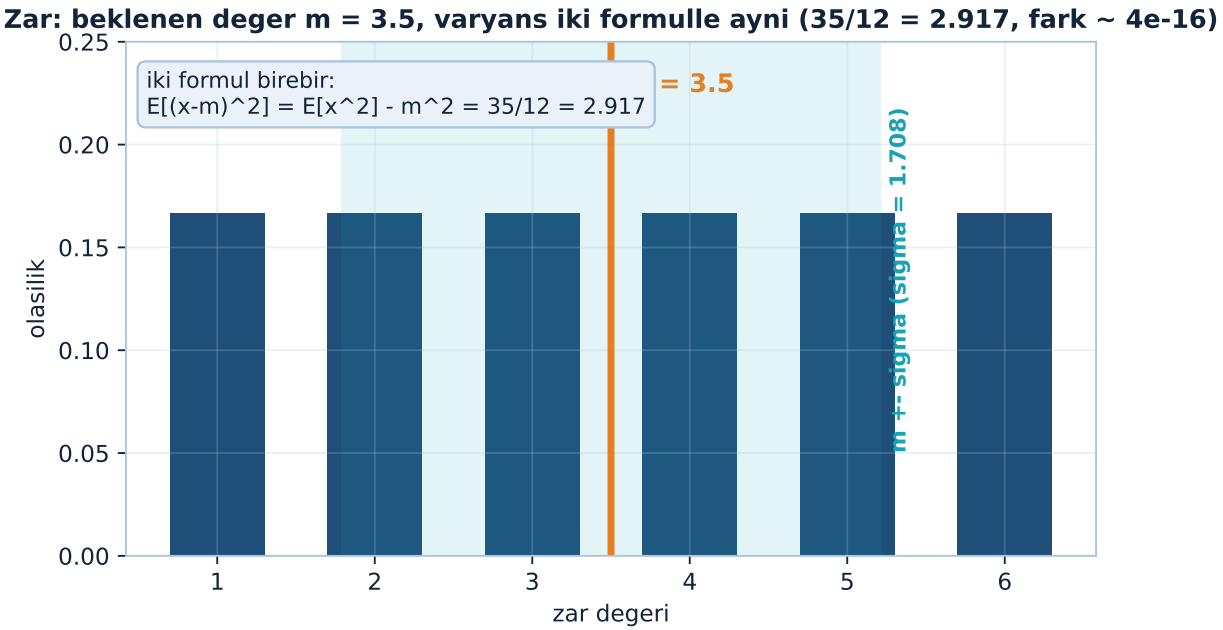
Beklenen değer = olasılık-ağırlıklı ortalama; örneklem ortalaması bunun veriden tahminidir. ML köprüsü: bir kayıp fonksiyonunun beklenen değerini (risk) minimize etmek istatistiksel öğrenmenin tanımıdır; SGD (Ders 25) bu beklenen değeri mini-batch örneklem ortalamasıyla tahmin eder. $E[\cdot]$ gösterimi tüm makine öğrenmesi teorisinin dili.

27.3 2. Varyans

Varyans, beklenen değerin özel bir hâli: ortalamadan uzaklığın karesinin beklenen değeri:

$$\sigma^2 = E[(x-m)^2] = \sum_i P_i(x_i - m)^2$$

Kare şart — işaretten bağımsız bir yayılım ölçüsü verir. m = ortalama (mean). Varyans, dağılımın ortalama etrafında ne kadar saçıldığını ölçer.



Şekil 27.2: Zar dağılımı: beklenen değer m = 3.5 ve varyansın iki formülle birebir aynı çıkışı (35/12 ≈ 2.917).

> **Builder Notu — Saçılımın Karesi**
>
> Varyans = ortalamadan kare-uzaklığın ortalaması. ML köprüsü: bias-variance ayrışımı (model hatasının iki kaynağı), gradyan gürültüsünün varyansı (SGD yakınsama hızını belirler), ve PCA'da varyans-maksimizasyonu — hepsi bu tek tanıma dayanır. Yüksek varyans = aşırı-öğrenme sinyali.

27.4 3. Varyans için İkinci Formül

Kareyi açıp yeniden düzenleyerek varyansın çok kullanışlı (ve hesaplaması hızlı) ikinci biçimi çıkar:

$$\sigma^2 = E[x^2] - m^2$$

"…the expected value of x squared minus m squared. It's just algebra." — Strang, 7:56

İspat lise cebiri: $\sum P_i(x_i - m)^2$ açılınca $\sum P_i x_i^2 - 2m\sum P_i x_i + m^2 \sum P_i$. Son terimde $\sum P_i = 1$, orta terimde $\sum P_i x_i = m$, yani $-2m \cdot m + m^2 = -m^2$. Geriye $E[x^2] - m^2$ kalır.

Zar örneği (Şekil 27.2), bu iki formülün birebir aynı sonucu verdiğini sayısal olarak doğrular: $m = 3.5$, her iki yoldan $\sigma^2 = 35/12 \approx 2.917$, aradaki fark yalnızca yuvarlama düzeyinde ($\sim 4 \times 10^{-16}$). $m \pm \sigma$ bandı ($\sigma \approx 1.708$) dağılımın çoğunu kapsar.

> **Builder Notu — Tek Geçişte Varyans**
>
> "Karelerin ortalaması eksi ortalamanın karesi" formülü tek geçişte (online) varyans hesaplamayı sağlar — $\sum x^2$ ve $\sum x$ biriktirip sonda birleştir. ML köprüsü: batch normalization ve running statistics (çalışan istatistikler) tam bu formülü kullanır; veri akışında ortalama ve varyansı tek geçişte günceller.

27.5 4. Markov Eşitsizliği

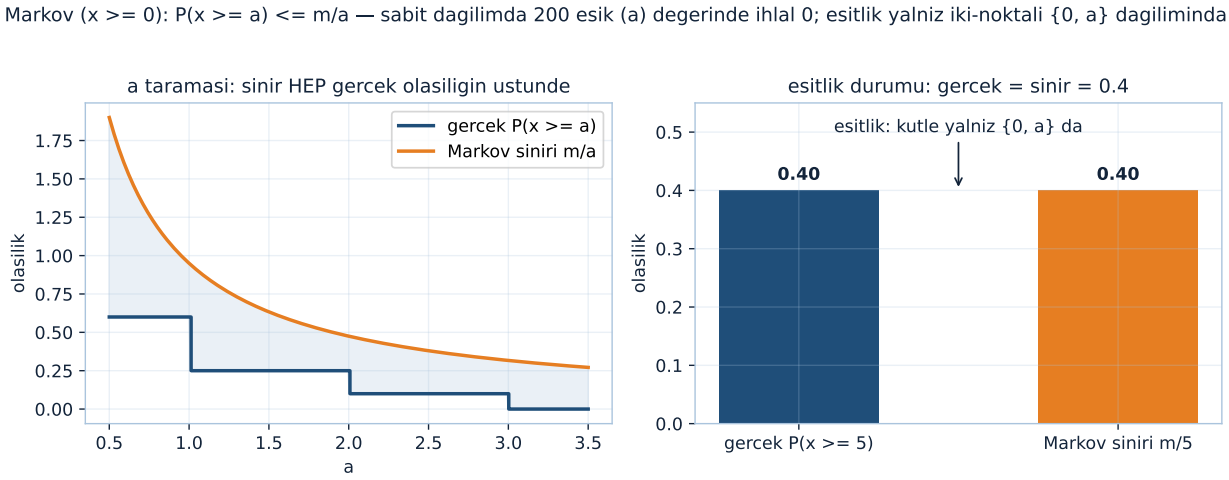
İstatistiğin iki büyük eşitsizliğinden ilki Markov'dan (1900'lerin büyük Rus olasılıkçısı):

> *"And the first one is due to Markov."* — Strang, 8:46

Negatif olmayan çıktılar için (tüm $x_i \geq 0$), x'in a'dan büyük olma olasılığını ortalama ile sınırlar:

$$P(x \geq a) \leq \frac{m}{a} \qquad (x \geq 0)$$

a büyüdükçe olasılık düşer (daha fazlasını istiyoruz). Örnek: m = 1, a = 3 → $P(x \geq 3) \leq 1/3$. İspat sezgisi: $x_3 P_3 + x_4 P_4 + \cdots \leq \sum x_i P_i = m$; tüm terimler negatif olmadığından ve toplam m olduğundan, "a'yı aşan" kısım m/a'yı geçemez.



Şekil 27.3: Markov eşitsizliği iki yüzü. Sol: dört-noktalı dağılımda ($m = 0.95$) gerçek $P(x \geq a)$ basamak eğrisi (navy) ile Markov sınırı m/a (turuncu) — sınır eşiğin her değerinde gerçek olasılığın üstünde, aradaki gri bölge sınırın gevşekliği. Sağ: eşitliğin sağlandığı tek aile — kütle yalnız $\{0, a\}$'da yoğunlaşırsa ($a = 5$, $P = [0.6, 0.4]$) gerçek olasılık ile sınır birebir çakışır (her ikisi de 0.40).

Markov'un iki yüzü (Şekil 27.3): solda sabit bir dört-noktalı dağılımda eşik a, $0.5 \to 3.5$ boyunca (200 nokta) tarandıkça gerçek $P(x \geq a)$ basamak eğrisi sınır m/a eğrisinin daima altında kalır — taranan 200 eşik değerinin hiçbirinde ihlal yok (ihlal sayısı 0). Sağda eşitliğin sağlandığı tek aile gösterilir: kütle yalnız $\{0, a\}$ iki noktasına toplanırsa gerçek olasılık sınıra dokunur ($a = 5$ için her ikisi de 0.40).

> Builder Notu — Az Varsayım Zayıf Sınır
>
> Markov, en az varsayımla (sadece $x \geq 0$) en zayıf ama en genel sınırı verir. ML köprüsü: konsantrasyon eşitsizliklerinin (concentration inequalities) atası; genelleme sınırları, PAC öğrenme ve "kötü olay" olasılıklarını sınırlamak hep Markov-tipi argümanla başlar. Gauss gibi negatif değer alan dağılımlara uygulanamaz — sınırı orada Chebyshev devralır.

27.6 5. Chebyshev Eşitsizliği

İkinci büyük eşitsizlik Chebyshev'den (dönemin diğer büyük Rus olasılıkçısı). Markov'un $x \geq 0$ varsayımını yapmaz:

> *"Chebyshev is the other great Russian probabilist of the time."* — Strang, 21:37

Ortalamadan **uzaklığı** (her iki yöne) sınırlar — varyans devreye girer:

$$P(|x - m| \geq a) \leq \frac{\sigma^2}{a^2}$$

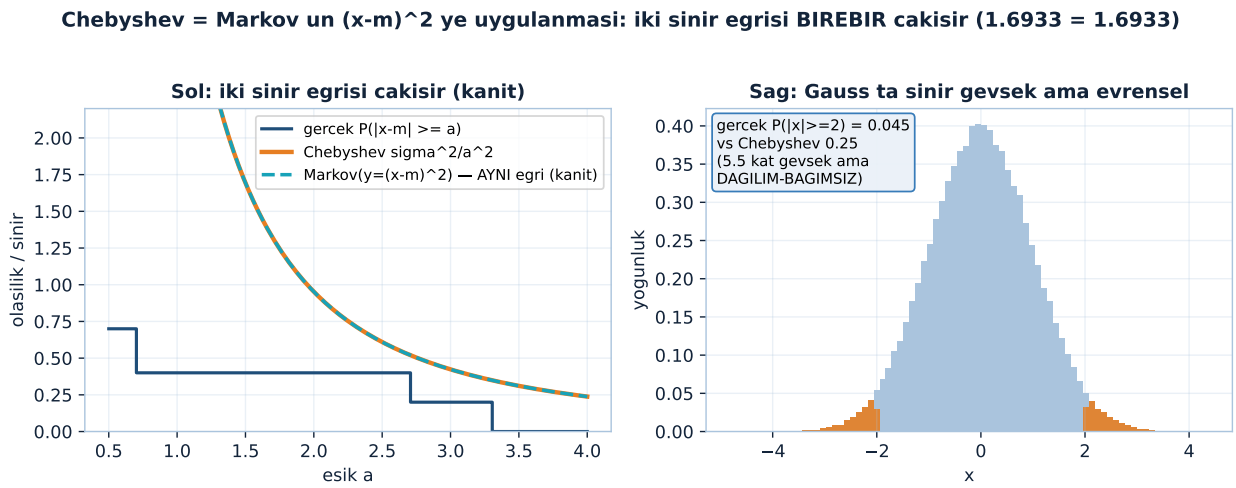
Zekice olan: ispat doğrudan Markov'dan gelir.

> *"…the proof of Chebyshev comes directly from Markov."* — Strang, 25:05

Yeni bir çıktı tanımla: $y = (x - m)^2$ (negatif olmayan! aynı olasılıklarla). Markov'u y'ye uygula: $P(y \geq a^2) \leq E[y]/a^2 = \sigma^2/a^2$. Ama $y \geq a^2$ demek $|x - m| \geq a$ demektir. Sonuç Chebyshev.

Kanıt görseli (Şekil 27.4) tam da bu hamleyi gösterir: solda Chebyshev sınırı σ^2/a^2 (turuncu) ile "Markov'u $y = (x-m)^2$'ye uygula" eğrisi (teal kesik) birebir çakışır — sayısal tanık $1.6933 = 1.6933$. Sağda standart normal için Chebyshev sınırı işler: gerçek $P(|x| \geq 2) \approx 0.045$, sınır 0.25 (yaklaşık 5,5 kat gevşek). Bu gevşeklik bedel: Markov negatif değerli Gauss'a uygulanamaz, ama Chebyshev her dağılıma uygulanır.

> Builder Notu — Markov'u Kareye Uygula
>
> Chebyshev mutlak değer/uzaklık ile çalışır, yani işaretsiz — varyans temelli. "Markov'u $(x - m)^2$'ye uygula" hamlesi, negatif-olmayan bir dönüşüm bulup genel bir eşitsizliği özelden türetmenin klasik örneği. ML köprüsü: büyük sayılar yasasının (LLN) niceliksel hâli; örneklem ortalamasının gerçek ortalamaya yakınsama hızını (1/n) Chebyshev verir — SGD'nin neden işe yaradığının temeli.



Şekil 27.4: Chebyshev = Markov'un $(x-m)^2$'ye uygulanması: gerçek kuyruk (navy basamak) iki sınırın altında kalır; Chebyshev σ^2/a^2 (turuncu) ile Markov($y=(x-m)^2$) eğrisi (teal kesik) BİREBİR çakışır — kanıtın kendisi. Sağda standart normalde gerçek $P(|x| \geq 2) = 0{,}045$, Chebyshev sınırı $0{,}25$ (5,5 kat gevşek ama dağılım-bağımsız).

27.7 6. Ortak Olasılık: Yapışmamış ve Yapışık Paralar

Covariance'a geçmeden önce **ortak olasılık** (joint probability). İki para at; sonuçları bir 2×2 matriste topla. **Yapışmamış** (bağımsız) paralar: her kombinasyon eşit olasılıklı:

$$P_{\text{unglued}} = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix}$$

Yapışık (tam bağımlı) paralar: biri yazı gelirse diğeri de yazı; sadece YY veya TT mümkün:

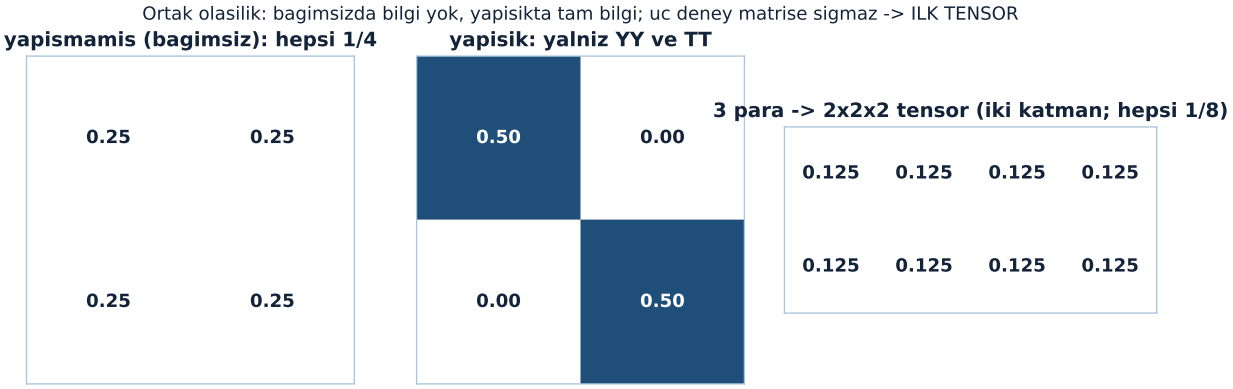
$$P_{\text{glued}} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}$$

Bir paranın sonucunu bilmek diğeri hakkında: bağımsızda hiçbir şey, bağımlıda her şeyi söyler. Üç deney (üç para) yaparsan, ortak olasılıklar artık bir matrise sığmaz — üç indis gerekir:

> *"…we're seeing for the first time a tensor."* — Strang, 37:56

Üç para → 2×2×2 bir **tensör** (8 girdi); bağımsızsa her biri 1/8. Tensör = çok-yollu matris (satır, sütun, katman).

İki para matrisi ile ilk tensör (Şekil 27.5): solda yapışmamış paraların dört hücresi de 1/4, ortada yapışık paralarda kütle yalnız YY/TT köşegeninde (her biri 1/2), sağda üç para için 2×2×2 tensör iki katman olarak gösterilir (her hücre 1/8).



Şekil 27.5: Ortak olasılık matrisleri ve ilk tensör: yapışmamış (bağımsız) paralarda dört hücre de 1/4, yapışık paralarda kütle yalnız YY ve TT köşegeninde (her biri 1/2), üç paranın deneyi ise artık matrise sığmaz ve 2×2×2 tensöre taşar (her hücre 1/8).

> **Builder Notu — Köşegen-Dışı Bağımlılık Kodlar**
>
> Ortak olasılık matrisinin köşegen-dışı yapısı bağımlılığı kodlar: bağımsız = köşegen-baskın değil, eşit dağılım; bağımlı = sıfırlar belirir. ML köprüsü: tensör derin öğrenmenin temel veri yapısıdır (PyTorch/NumPy ndarray) — bir görüntü batch'i 4-yollu tensördür (batch×kanal×yükseklik×genişlik). Strang'ın "ilk kez tensör görüyoruz" anı, lineer cebirden çok-boyutlu dizilere geçişin kapısı.

27.8 7. Covariance Matrisi

Günün varış noktası:

> *"…what is the covariance matrix?"* — Strang, 42:37

İki deney aynı anda koşar (deney 1 → x, deney 2 → y). Tüm olası (x_i, y_j) çiftleri üzerinde, ortak olasılık P_{ij} ile ağırlıklı, sapmaların **dış-çarpımı** (kolon × satır):

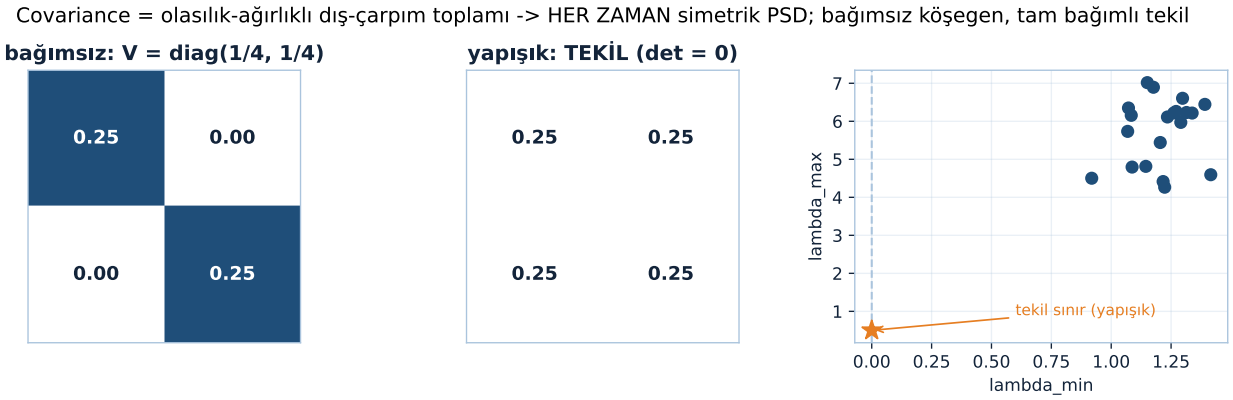
$$V = \sum_{i,j} P_{ij} \begin{bmatrix} x_i - m_x \\ y_j - m_y \end{bmatrix} \begin{bmatrix} x_i - m_x & y_j - m_y \end{bmatrix}$$

İki deney → 2×2 matris. (1,1) girdisi: $(x - m_x)^2$ ağırlıklı toplam = x'in varyansı σ_x^2. (2,2): σ_y^2. Köşegen-dışı (1,2) = (2,1): **kovaryans** σ_{xy} (simetrik):

$$V = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix}$$

Köşegen = ayrı ayrı varyanslar, köşegen-dışı = değişkenler arası ilişki. Dış-çarpım yapısı (Ders 1) covariance'ı doğal olarak simetrik kılar.

Covariance'ın yapısı (Şekil 27.6): solda bağımsız paralarda $V = \text{diag}(1/4, 1/4)$ (köşegen, $\sigma_{xy} = 0$), ortada yapışık paralarda V'nin tüm girdileri 1/4 olduğundan determinant sıfır — tekil/yarı-tanımlı. Sağda 20 rastgele



Şekil 27.6: Covariance = olasılık-ağırlıklı dış-çarpım toplamı: SOL bağımsız paralarda V = diag(1/4, 1/4) (köşegen, σ_xy = 0); ORTA yapışık paralarda V = [[1/4,1/4],[1/4,1/4]] tekil (det = 0, yarı-tanımlı); SAĞ 20 rastgele ortak dağılımın V özdeğer çifti hep λ_min > 0 bölgesinde (PSD), yapışık paranın noktası (0, 0.5) tam tekil sınırda. V her zaman simetrik ve PSD.

ortak dağılımın özdeğer çiftleri $(\lambda_{\text{min}}, \lambda_{\text{max}})$ daima $\lambda_{\text{min}} > 0$ bölgesinde (en küçüğü 0.92), yapışık paranın noktası (0, 0.5) tam tekil sınırda.

> **Builder Notu — Dış-Çarpımların Ağırlıklı Toplamı**
>
> Covariance matrisi = dış-çarpımların olasılık-ağırlıklı toplamı (Ders 1 $uv^\top$ deseni). Köşegen-dışı terimler değişkenlerin birlikte nasıl değiştiğini söyler. ML köprüsü: veri matrisi A için $(1/n)A^\top A$ merkezlenmiş covariance'tır; özvektörleri ana bileşenler (PCA, Ders 7), özdeğerleri varyans miktarları. Gauss dağılımı, Mahalanobis mesafesi ve LDA hep V'ye dayanır.

27.9 8. Pozitif Yarı-Tanımlı ve Bağımsızlık

Dış-çarpım toplamı olduğundan covariance matrisi her zaman **simetrik pozitif yarı-tanımlıdır** (Ders 5): her sapma-dış-çarpımı pozitif yarı-tanımlı, toplamları da öyle. Bağımsız (yapışmamış) deneylerde kovaryans sıfırdır:

> *"…in that case, those are 0."* — Strang, 52:10

$$\text{independent} \;\Rightarrow\; \sigma_{xy} = 0 \;\Rightarrow\; V = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$$

Bağımsızlıkta V köşegen — sadece ayrı varyanslar. Tam bağımlılıkta (yapışık) V tekildir (pozitif **yarı**-tanımlı, det = 0). Ek not: ortak olasılıkları bir indis üzerinden toplamak ($\sum_i P_{ij} = P_j$) **marjinal** olasılıkları verir.

Builder Notu — Sıfır Özdeğer Tam Bağımlılık

"Bağımsız → köşegen covariance" ve "tam bağımlı → tekil (yarı-tanımlı)" ayrımı, covariance'ın öz-değerlerinin bağımlılık yapısını okuduğunu gösterir: sıfır özdeğer = mükemmel doğrusal bağımlılık. ML köprüsü: çoklu-doğrusallık (multicollinearity) tam budur — neredeyse-tekil covariance regresyonu kararsız kılar (Ders 10 kondisyon); PCA bu yönde varyansı sıfıra yakın bileşenleri atar.

27.10 Bu Dersin Özeti

- **Beklenen değer:** $E[x] = \sum P_i x_i = m$; genel olarak $E[f(x)] = \sum P_i f(x_i)$.
- **Varyans:** $\sigma^2 = E[(x-m)^2] = E[x^2] - m^2$ (ikinci formül tek geçişte hesaplanır).
- **Markov ($x \geq 0$):** $P(x \geq a) \leq m/a$. En genel, en zayıf sınır.
- **Chebyshev:** $P(|x-m| \geq a) \leq \sigma^2/a^2$; Markov'u $y = (x-m)^2$'ye uygulayarak kanıtlanır.
- **Ortak olasılık:** bağımsız (köşegen-dışı bilgi yok) vs bağımlı; 3 deney → tensör (2×2×2).
- **Covariance matrisi:** $V = \sum P_{ij}\,(\text{sapma})(\text{sapma})^\top$; köşegen = varyanslar, köşegen-dışı = σ_{xy}; simetrik pozitif yarı-tanımlı.
- **Bağımsızlık → köşegen V** ($\sigma_{xy} = 0$); tam bağımlılık → tekil V.

Tek Bir Cümle

Beklenen değer ve varyans olasılığın temel ölçüleridir; Markov ($P(x \geq a) \leq m/a$) ve Chebyshev ($P(|x-m| \geq a) \leq \sigma^2/a^2$) sapmaları sınırlar; covariance matrisi ise değişkenler arası ilişkiyi simetrik pozitif yarı-tanımlı bir matris olarak kodlar (köşegen = varyans, köşegen-dışı = kovaryans).

27.11 Kontrol Soruları

Soru 1: Varyansın iki formülü nedir ve ikincisi neden hesaplama açısından kullanışlı?

$\sigma^2 = E[(x-m)^2]$ (sapma-kare ortalaması) ve $\sigma^2 = E[x^2] - m^2$ (karelerin ortalaması eksi ortalamanın karesi). İkincisi kullanışlı çünkü $\sum x^2$ ve $\sum x$'i tek veri geçişinde biriktirip sonda birleştirirsin — online/running varyans hesabı. İspatı lise cebiri: kareyi açıp $\sum P_i = 1$ ve $\sum P_i x_i = m$ kullan.

Soru 2: Markov ve Chebyshev eşitsizlikleri arasındaki fark nedir, ve Chebyshev nasıl kanıtlanır?

Markov yalnız $x \geq 0$ için geçerli: $P(x \geq a) \leq m/a$. Chebyshev bu varsayımı yapmaz; ortalamadan uzaklığı (her iki yön) sınırlar: $P(|x-m| \geq a) \leq \sigma^2/a^2$. Chebyshev, Markov'u negatif-olmayan yeni değişken $y = (x-m)^2$'ye uygulayarak kanıtlanır: $P(y \geq a^2) \leq E[y]/a^2 = \sigma^2/a^2$, ve $y \geq a^2 \Leftrightarrow |x-m| \geq a$.

i Soru 3: İki yazı-tura için yapışmamış (bağımsız) ve yapışık (bağımlı) ortak olasılık matrisleri nasıl farklıdır?

Yapışmamış: dört kombinasyon (YY, YT, TY, TT) eşit olasılıklı, her biri 1/4 — köşegen-dışı doludur, bilgi bağımsız. Yapışık: paralar birlikte hareket eder, sadece YY ve TT mümkün (her biri 1/2), YT ve TY sıfır — sadece köşegen dolu. Bir paranın sonucu, bağımsızda diğeri hakkında hiçbir şey, bağımlıda her şeyi söyler.

i Soru 4: Covariance matrisi V neden simetrik pozitif yarı-tanımlıdır ve köşegen-dışı sıfır ne anlama gelir?

$V = \sum P_{ij}\,(\text{sapma})(\text{sapma})^\top$, yani pozitif yarı-tanımlı dış-çarpımların olasılık-ağırlıklı toplamı (Ders 5) — dolayısıyla simetrik pozitif yarı-tanımlı. Köşegende varyanslar (σ_x^2, σ_y^2), köşegen-dışında kovaryans σ_{xy} bulunur. Köşegen-dışı sıfır ($\sigma_{xy} = 0$) değişkenlerin **bağımsız/ilişkisiz** olduğu anlamına gelir; V köşegen matris olur.

27.12 Egzersizler

1. **İki formül.** Bir zar (1–6, eşit olasılık 1/6) için $E[x] = m$'yi ve hem $\sigma^2 = E[(x-m)^2]$ hem $\sigma^2 = E[x^2] - m^2$ ile varyansı hesapla; aynı çıktığını göster.

2. **Markov sınırı.** Ortalaması m = 2 olan negatif-olmayan bir x için $P(x \geq 8)$ en fazla kaç olabilir? Markov sınırı gevşek mi sıkı mı, yorumla.

3. **Chebyshev sınırı.** m = 0, σ = 1 olan bir değişken için $P(|x| \geq 2)$ Chebyshev'e göre en fazla kaçtır? (Gauss'ta gerçek değerle karşılaştır: ~0.046.)

4. **Covariance hesabı.** Yapışık paralar (YY: 1/2, TT: 1/2; Y=1, T=0). x ve y için $m_x = m_y = 1/2$. σ_{xy} kovaryansını hesapla; σ_x^2 ile karşılaştır. V matrisi tekil mi?

5. **(Ders 21 habercisi)** Bu derste beklenen değer/varyans gördük; bunlar bir kayıp fonksiyonunun "ortalama" davranışını tanımlar. Peki bir fonksiyonun minimumunu adım adım nasıl buluruz? Türev sıfır olduğunda mı, yoksa iteratif inişle mi? Bir tahmin yaz — Ders 21 "bir fonksiyonu adım adım minimize etmek" (Newton, gradient descent girişi) ile optimizasyon bloğunu açıyor.

27.13 Sonraki Ders İçin Hazırlık

Ders 21: Bir Fonksiyonu Adım Adım Minimize Etmek. İstatistikten optimizasyona geçiş: bir maliyet fonksiyonunun minimumunu nasıl buluruz? Strang Newton yöntemini (ikinci-derece, hızlı ama Hessian gerektirir) ve gradient descent'in (birinci-derece, ölçeklenebilir) temelini kurar — derin öğrenme eğitiminin (Ders 22+) çekirdek algoritması.

⚠ Lineer cebirden optimizasyona

Bu ders istatistik bloğunun kapısıydı: beklenen değer, varyans, iki eşitsizlik ve covariance matrisi. Sıradaki blok **optimizasyon** — kayıp fonksiyonlarını minimize etme. Ders 21'e gelmeden önce gradyan (gradient) ve türev kavramlarını gözden geçir; gradient descent'in tek satırı "negatif gradyan yönünde adım at" olsa da arkasındaki sezgi tüm derin öğrenme eğitiminin temeli.

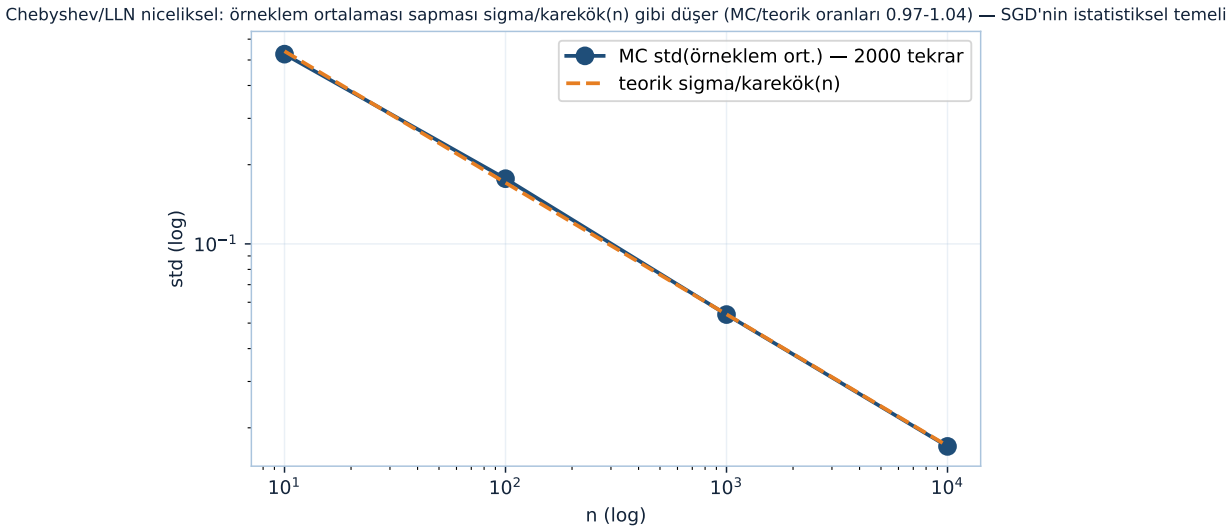
27.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
İstatistik başlıyor	olasılık = derin öğrenmenin parçası	0m28
Beklenen değer	$E[x] = \sum P_i x_i = m$	0m28
Varyans	$\sigma^2 = E[(x-m)^2] = E[x^2] - m^2$	7m56
Markov eşitsizliği	$P(x \geq a) \leq m/a$ $(x \geq 0)$	8m46
Chebyshev eşitsizliği	$P(\lvert x-m\rvert \geq a) \leq \sigma^2/a^2$	21m37
Chebyshev kanıtı	Markov'u $y = (x-m)^2$'ye uygula	25m05
Ortak olasılık / tensör	bağımsız vs bağımlı; 3 deney → 2×2×2 tensör	37m56
Covariance matrisi	$V = \sum P_{ij}\,(\text{sapma})(\text{sapma})^\top$; simetrik PYT	42m37
Bağımsız → köşegen	$\sigma_{xy} = 0 \Rightarrow V$ köşegen	52m10

27.15 ML Bağlantıları Özeti

- **Risk minimizasyonu:** beklenen kayıp $E[\text{loss}]$ minimize etmek = istatistiksel öğrenme; SGD (Ders 25) bunu mini-batch örneklem ortalamasıyla tahmin eder.
- **Konsantrasyon eşitsizlikleri:** Markov/Chebyshev → genelleme sınırları, PAC öğrenme; "örneklem ortalaması gerçek ortalamaya ne kadar yakın?" (LLN'in niceliksel hâli).
- **Covariance / PCA:** veri matrisi A → $(1/n)A^\top A$ covariance; özvektörleri ana bileşenler (Ders 7), özdeğerleri varyanslar; Gauss, Mahalanobis, LDA.
- **Tensör:** derin öğrenmenin temel veri yapısı (batch×kanal×H×W); çok-yollu dizi ilk kez burada.
- **Batch normalization:** $E[x^2] - m^2$ formülü running statistics ile tek geçişte ortalama/varyans.
- **Çoklu-doğrusallık:** tekil (yarı-tanımlı) covariance → kararsız regresyon (Ders 10 kondisyon); PCA sıfıra yakın varyansı atar.
- **Geriye köprü:** Stat 110 §4.B (mean/variance/Markov/Chebyshev/covariance), Ders 5 (pozitif yarı-tanımlı), Ders 7 (PCA), Ders 14 (Kalman covariance), Ders 19 (covariance önizleme).

LLN'in niceliksel yüzü (Şekil 27.7): örneklem ortalamasının standart sapması teorik $\sigma/\sqrt{n}$ eğrisini birebir takip eder (MC/teorik oranları 0.97–1.04 bandında); n 100 kat artınca std 10 kat düşer (std(10)/std(1000) = 9.77 ≈ 10). Bu, Chebyshev'in "örneklem ortalaması gerçek ortalamaya yakınsar" vaadinin sayısal kanıtı ve SGD'nin neden çalıştığının istatistiksel temeli.



Şekil 27.7: Chebyshev/LLN niceliksel: n örnekli örneklem ortalamasının std'si teorik $\sigma/\sqrt{n}$ eğrisini birebir takip eder (MC/teorik oranları 0.97-1.04 bandında). n 10 kat artınca std $\sqrt{10} \approx 3.16$ kat düşer — std(10)/std(1000) = 9.77 ≈ 10. Bu, SGD'nin neden çalıştığının istatistiksel temelidir: daha çok örnek → daha az gürültü.

! İstatistik, lineer cebirin üzerine kurulur

> *"…part of deep learning as we get there."* — Strang, 0:28

Covariance bir matristir, özdeğerleri varyans yönlerini verir; beklenen değer ve varyans olasılığın iki temel ölçüsü, Markov ve Chebyshev sapmaları sınırlayan iki büyük eşitsizliktir. Bu blok lineer cebir ile optimizasyon ve derin öğrenme arasındaki köprü: covariance PCA'ya, konsantrasyon eşitsizlikleri genelleme sınırlarına, tensör ise derin öğrenmenin temel veri yapısına açılır.

28 Bir Fonksiyonu Adım Adım Minimize Etmek

Taylor serisi, Newton yöntemi, yakınsama hızları ve konvekslik

> **Bölüm bilgisi**
>
> Bu ders **optimizasyon bloğunu (Part 6) açar** — derin öğrenmenin kalbindeki "bir fonksiyonu adım adım küçült" fikrini Taylor serisi, Newton yöntemi ve konvekslik üçlüsüyle kurar: Strang'in Ders 21 videosu (≈54 dk) ve OCW Lecture 21 temel alınmıştır. Okuma süresi ≈34 dk; önkoşul Ders 20 (istatistik bloğu).

28.1 Bu Derste Ne Var?

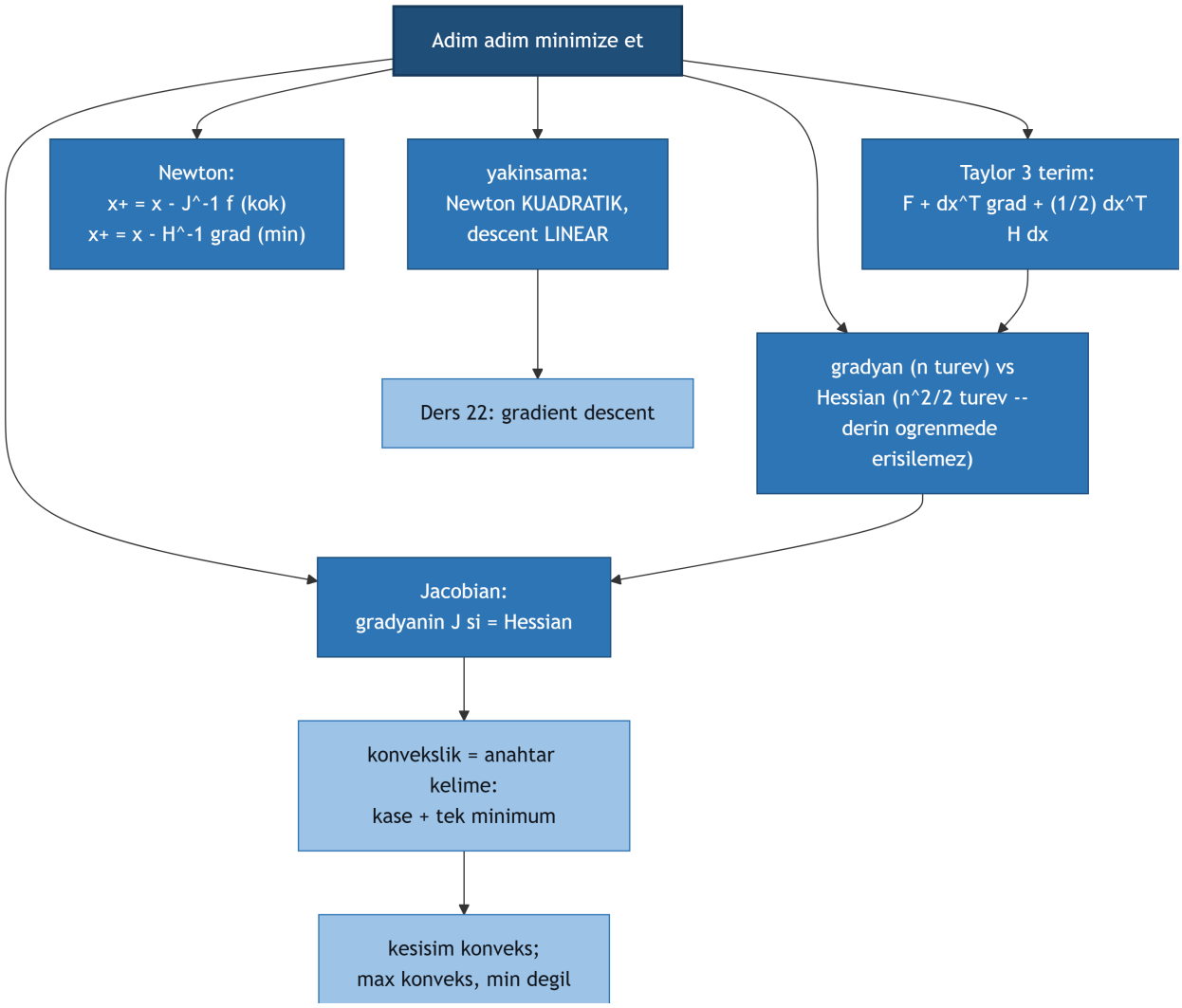
Lineer cebir ve istatistik bloklarından sonra sıra **bir sayıyı küçültmeye** geliyor: bir maliyet fonksiyonu verildiğinde, onu en küçükleyen noktayı adım adım nasıl ararız? Cevap her seferinde aynı: bulunduğun noktada fonksiyonu Taylor serisiyle yaklaşıkla, o yaklaşımın minimumuna atla, tekrar et. Bu derste o döngünün araçlarını (gradyan, Hessian, Jacobian), iki ana yöntemini (Newton ve steepest descent) ve sonucu garanti eden geometrik koşulu (konvekslik) görüyoruz.

Beş sonuç:

1. **Taylor serisi (3 terim):** $F(x+\Delta x) \approx F(x) + (\Delta x)^T \nabla F + \frac{1}{2}(\Delta x)^T H(\Delta x)$; ∇F gradyan, H Hessian (simetrik, n^2 ikinci türev).
2. **Jacobian:** vektör fonksiyon f için birinci türev matrisi J; gradyanın Jacobian'ı = Hessian.
3. **Newton yöntemi:** $f = 0$ için $x_+ = x - J^{-1} f$; minimizasyon için $x_+ = x - H^{-1} \nabla F$.
4. **Yakınsama:** Newton **kuadratik** (hata karelenir, çok hızlı); steepest descent **linear** (hata × sabit < 1).
5. **Konvekslik:** konveks küme (iki nokta arası çizgi kümede kalır), konveks fonksiyon (grafik üstü = konveks küme); kesişim konveks, max konveks ama min değil.

> *"Convexity is the key word for these problems." — Strang, 32:49*

Şekil 28.1 dersin iskeletini gösterir: merkezdeki "adım adım minimize et" fikri Taylor'dan Newton'a, oradan yakınsama hızlarına ve konveksliğe dallanır; Ders 22'nin gradient descent'i ise yakınsama dalından çıkan köprü düğümüdür.



Şekil 28.1: Ders 21 kavram haritasi: “Adim adim minimize et” merkezde; Taylor uc terim (F + dx^T grad + yari dx^T H dx), gradyan (n turev) vs Hessian (n^2/2 turev, derin ogrenmede erisilemez), Jacobian (gradyanin J si = Hessian), Newton iterasyonu (kok icin x - J^-1 f, min icin x - H^-1 grad) ve yakinsama (Newton kuadratik, descent linear) dallarda; konvekslik = kase + tek minimum anahtar kelimesi, kesisim konveks ama max-konveks-min-degil notu ve Ders 22 gradient descent kopru dugumu ayri.

Builder Notu — Optimizasyonun Alet Çantası

Bu dersin her satırı tek bir döngünün parçasıdır: **yaklaşıkla → çöz → tekrarla**. Taylor serisi yaklaşımı verir, gradyan/Hessian onun katsayılarıdır, Newton ve steepest descent çözme stratejileridir, konvekslik ise "bu döngü gerçekten dibe varır mı?" sorusunun cevabıdır. Derin öğrenmenin tüm eğitim mekanizması bu alet çantasının birinci-derece (gradyan) köşesinde yaşar — Hessian milyar parametrede erişilemez olduğu için.

28.2 Taylor Serisi: Gradyan ve Hessian

Optimizasyon, bir fonksiyonun yerel davranışını yaklaşıkla başlar.

> *"…three terms of a Taylor series?" — Strang, 1:44*

Tek değişkende klasik üç-terim Taylor'ı çok değişkene taşı. $x = (x_1, \dots, x_n)$, Δx bir vektör:

$$F(x + \Delta x) \approx F(x) + (\Delta x)^T \nabla F + \tfrac{1}{2}(\Delta x)^T H\,(\Delta x)$$

Lineer terimde **gradyan** ∇F (n kısmi türev vektörü); kuadratik terimde **Hessian** H (ikinci türev matrisi):

$$\nabla F = \left(\frac{\partial F}{\partial x_1}, \dots, \frac{\partial F}{\partial x_n}\right), \qquad H_{jk} = \frac{\partial^2 F}{\partial x_j\, \partial x_k}$$

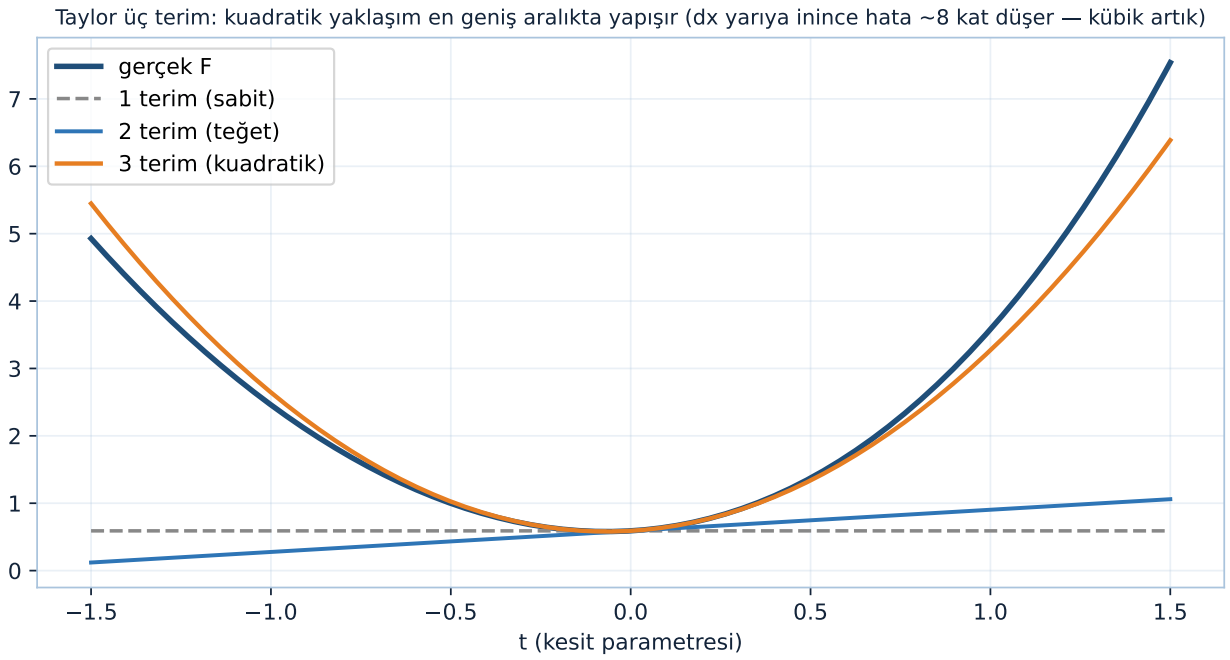
> *"…The Hessian, Hessian matrix." — Strang, 5:19*

Hessian **simetriktir** (karışık türevler eşit, Ders 16). n büyükse gradyan n türev, Hessian $\sim \frac{1}{2}n^2$ türev ister — bu yüzden derin öğrenmede Hessian "erişilemez", sadece gradyan hesaplanır (otomatik türev/backprop ile).

Şekil 28.2 bu üç terimi tek bir kesit üzerinde üst üste koyar: sabit terim noktayı, lineer terim (teğet) eğimi, kuadratik terim (parabol) eğriliği yakalar; kuadratik yaklaşım en geniş aralıkta gerçek fonksiyona yapışır.

Builder Notu — Lineer Terim Ucuz Kuadratik Pahalı

Üç-terim Taylor optimizasyonun tüm algoritmalarının temelidir: lineer terim (gradyan) → gradient descent, kuadratik terim (Hessian) → Newton. Maliyet asimetrisi belirleyicidir: gradyan n türev (ucuz), Hessian $\sim \frac{1}{2}n^2$ türev (pahalı). ML köprüsü: n^2 Hessian milyar-parametreli modellerde imkânsız; bu yüzden pratik derin öğrenme birinci-derece (gradyan) kalır, Hessian'a yaklaşımlar (L-BFGS, K-FAC) kullanır.



Şekil 28.2: Taylor üç terim: sabit < teğet (gradyan) < kuadratik (Hessian) — kuadratik yaklaşım en geniş aralıkta yapışır. $F(x) = x_1^2 + 2x_2^2 + \frac{1}{4}x_1^4 + x_1x_2$ fonksiyonunun $x_0 = (0.7, -0.4)$ noktasından $d = (1,1)/\sqrt{2}$ yönlü 1B kesiti. 1 terim (sabit) noktayı, 2 terim (teğet doğru) eğimi, 3 terim (parabol) eğriliği yakalar; Δx yarıya inince kuadratik yaklaşımın hatası ~8 kat düşer (kübik artık).

28.3 Jacobian: Vektör Fonksiyonun Türevi

Skaler F yerine **vektör** fonksiyon $f = (f_1, \ldots, f_n)$ (n fonksiyon, n değişken). Onun Taylor açılımı:

$$f(x + \Delta x) \approx f(x) + J\,(\Delta x)$$

Buradaki birinci-türev matrisi **Jacobian** J:

> *"…I'm looking for the Jacobian." — Strang, 8:56*

$$J_{jk} = \frac{\partial f_j}{\partial x_k}$$

Önemli paralellik: f, gradyan ∇F'ye karşılık gelir (ikisi de n fonksiyon/n değişken). Dolayısıyla **gradyanın Jacobian'ı = Hessian** (birinci türevin birinci türevi = ikinci türev). Bu, skaler ve vektör dünyaları birbirine bağlar — sayısal tanık olarak gradyanın Jacobian'ı ile doğrudan Hessian arasındaki fark merkez-fark hassasiyetinde maxdiff ~1e-4 düzeyinde kalır.

Şekil 28.7 (Egzersizler bölümünde) bu özdeşliğin somut bir Hessian örneğini gösterir: simetrik ama belirsiz bir H, "gradyanın Jacobian'ı = Hessian" zincirinin sonunda konvekslik testine bağlanır.

> **Builder Notu — Katman Katman Jacobian**
>
> Jacobian = çok-girişli çok-çıkışlı bir fonksiyonun türev matrisi; "gradyanın Jacobian'ı Hessian'dır" özdeşliği skaler/vektör kalkülüsü birleştirir. ML köprüsü: backprop (Ders 27) tam bir Jacobian-çarpımı zinciridir — her katman bir Jacobian, zincir kuralı bunları çarpar. PyTorch'un autograd'ı Jacobian-vektör çarpımları (JVP/VJP) hesaplar.

28.4 Newton Yöntemi: f = 0 Çözmek

n denklem n bilinmeyenli $f(x) = 0$ sistemini çöz.

> *"…I'll start with Newton's method…" — Strang, 10:59*

Taylor'ın lineer kısmını kullan: yeni nokta x_{k+1}'de f sıfır olsun istiyoruz.

$$0 = f(x_k) + J(x_k)\,(x_{k+1} - x_k)$$

Δx'i çöz: $x_{k+1} - x_k = -J^{-1} f(x_k)$. Yani Newton iterasyonu:

$$x_{k+1} = x_k - J(x_k)^{-1}\, f(x_k)$$

Her adımda Jacobian'ı (mevcut noktada) ters çevir, f ile çarp, çıkar. 18.02'de nadiren öğretilir ama gradyan ve Jacobian'ın büyük uygulamasıdır.

Şekil 28.3 sol paneli bu geometriyi gösterir: bir parabolde teğet doğrusunu çözüp onun x-kesişimine atlamak; sağ panel ise hatanın her adımda nasıl karelendiğini taşır (sonraki bölümdeki $\sqrt{9}$ örneğinin tanığı).

> **Builder Notu — Lineerleştir Çöz Tekrarla**
>
> Newton = "lineerleştir, çöz, tekrarla." Her adımda fonksiyonu teğet doğrusuyla değiştirir. ML köprüsü: Newton ikinci-derece optimizasyonun temeli; tam Hessian/Jacobian pahalı olduğundan pratikte yaklaşımlar (quasi-Newton: BFGS, L-BFGS) kullanılır — ters Hessian'ı gradyan farklarından tahmin eder.

28.5 Örnek: √9 Bulmak

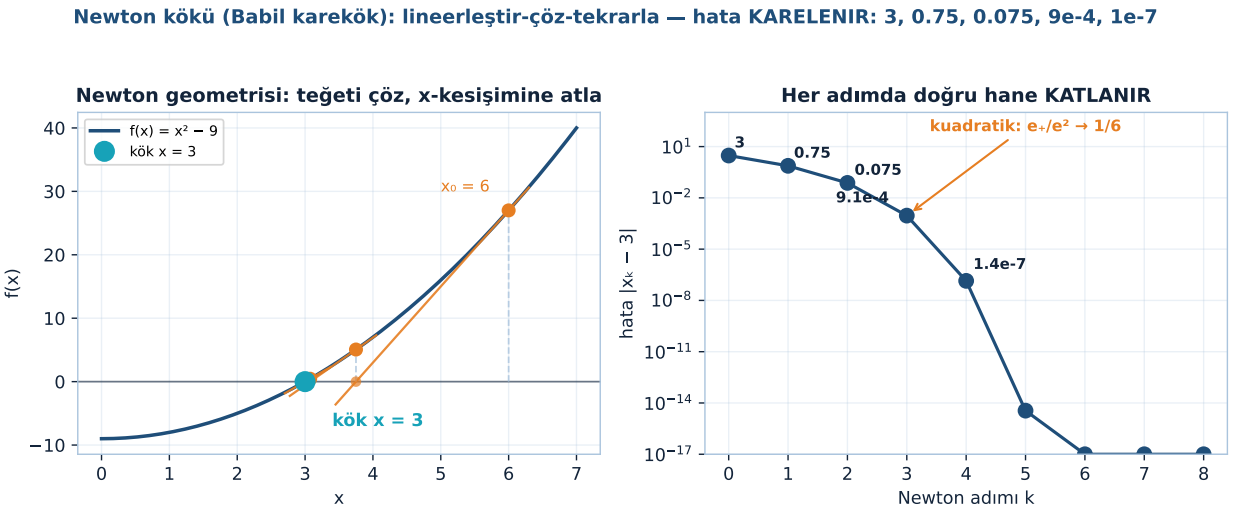
Newton'ı somut bir tek-değişkenli örnekte gör. $f(x) = x^2 - 9$, $f = 0$ çözümü $\sqrt{9} = 3$.

> *"…I want to find the square root of 9." — Strang, 14:21*

Jacobian (türev) $J = 2x$. Newton formülünü uygula ve sadeleştir:

$$x_{k+1} = x_k - \frac{x_k^2 - 9}{2x_k} = \tfrac{1}{2}x_k + \frac{9}{2x_k}$$

Sağlama: $x_k = 3$ ise $x_{k+1} = \frac{1}{2} \cdot 3 + 9/6 = 1.5 + 1.5 = 3$ — sabit nokta, doğru kök. Bu, bilgisayarların karekök hesaplama yöntemidir (Babil yöntemi); birkaç adımda muazzam hassasiyetle yakınsar. $x_0 = 6$'dan başlayınca hata dizisi $3 \to 0.75 \to 0.075 \to 9.1\text{e}{-4} \to 1.4\text{e}{-7}$ ilerler — her adımda doğru hane sayısı katlanır, kuadratik oran e_+/e^2 teorik limit $1/(2\sqrt{c}) = 1/6 = 0.1666$'ya oturur.



Şekil 28.3: Newton kökü (Babil karekök): lineerleştir-çöz-tekrarla — hata KARELENIR: 3, 0.75, 0.075, 9e-4, 1e-7. Sol panel f(x)=x^2−9 parabolünde x_0=6'dan teğeti çözüp x-kesişimine atlama (6 → 3.75 → 3.075 → kök 3); sağ panel hatanın her adımda karelenmesi (kuadratik oran e_+/e^2 → 1/6).

Şekil 28.3 bu örneği bütünüyle taşır: solda $x_0 = 6 \to 3.75 \to 3.075$ teğet adımları kök 3'e iner, sağda semilogy eksende hata her adımda karelenir.

> **Builder Notu — Babil'den Beri Aynı Hile**
>
> $x_{k+1} = \frac{1}{2}(x_k + 9/x_k)$ formülü Newton'ın gücünü gösterir: 3'e **kuadratik** yakınsar (her adımda doğru hane sayısı ikiye katlanır). ML köprüsü: aynı "tahmin et, düzelt, tekrarla" döngüsü tüm iteratif çözücülerin (ve eğitimin) kalbidir; Newton'ın hızı, ikinci-derece bilginin (eğrilik) ne kadar değerli olduğunu kanıtlar.

28.6 Newton ile Minimizasyon

$F(x)$'i minimize etmek = gradyanı sıfırlamak ($\nabla F = 0$). Bu, "$f = 0$" probleminin gradyan versiyonudur: f yerine ∇F, Jacobian yerine onun Jacobian'ı (= Hessian). Newton iterasyonu:

$$x_{k+1} = x_k - H(x_k)^{-1}\, \nabla F(x_k)$$

Yani minimum ararken her adımda Hessian'ı ters çevir, gradyanla çarp, çıkar. Hessian eğriliği (kâsenin ne kadar dik büküldüğünü) bildiğinden adım yönünü ve boyunu birlikte ayarlar — gradient descent'ten çok daha akıllı, ama H'yi hesaplamak/ters almak pahalı. Kuadratik bir kâsede (örn. $A = \text{diag}(1, 10)$) Newton **tek adımda** merkeze $(0, 0)$ varır; kuadratik-olmayan $F21$'de ise $\|\nabla F\| \to 0.0$ değerine 8 adımda iner.

Şekil 28.4 (bir sonraki bölümde) bu tek-adım davranışını steepest descent'in zikzaklı yoluyla yan yana koyar.

> **Builder Notu — Eğriliği Bilen Akıllı Adım**
>
> "$\nabla F = 0$ çözmek için Newton'ı gradyana uygula" — minimizasyon, kök bulmanın özel hâli. ML köprüsü: doğal gradyan (natural gradient) ve K-FAC, Hessian/Fisher bilgisini kullanan yaklaşık-Newton yöntemleridir; tam Newton derin ağlarda imkânsız (H milyar×milyar) ama eğrilik bilgisi optimizasyonu hızlandırır.

28.7 Yakınsama Hızları: Kuadratik vs Linear

İki yöntemin yakınsama hızı çok farklı.

> *"…the convergence rate for Newton's method [will be quadratic]." — Strang, 30:35*

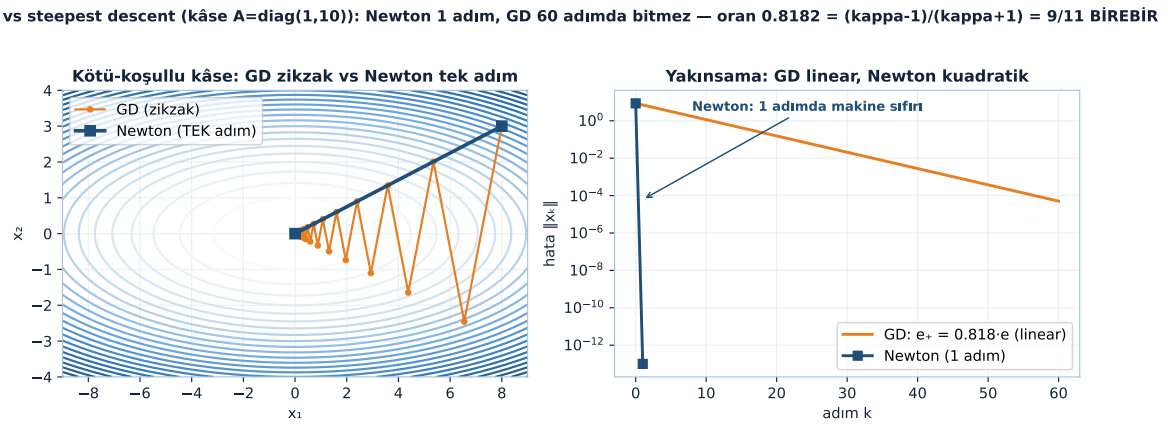
Newton: kuadratik — hata her adımda karelenir ($e \to e^2$), yeterince yakın başlarsan süper hızlı (hane sayısı ikiye katlanır). **Steepest descent (gradient descent): linear** — hata her adımda bir sabitle (< 1) çarpılır; doğru matrisin tersi yerine bir sayı kullandığından süper hız beklenemez.

$$\text{Newton: } e_{k+1} \sim e_k^2, \qquad \text{steepest descent: } e_{k+1} \sim c\, e_k \ (c < 1)$$

Kötü-koşullu kâse $A = \text{diag}(1, 10)$'da, adım boyu $s = 2/11$ ile steepest descent'in oranı $e_+/e = 0.8182 = (\kappa - 1)/(\kappa + 1) = 9/11$ değerine BİREBİR oturur; 60 adımda hata ancak 5e-5'e iner — oysa Newton aynı kâsede tek adımda biter. Arada **Levenberg-Marquardt** gibi "ucuz Newton" yöntemleri var: tam Hessian'ı hesaplamaz, gradyandan görebildiği bir Hessian terimini kapar — tam ikinci-derece değil ama gradient descent'ten iyi.

Şekil 28.4 bu kıyası iki panelde tutar: solda GD konturlar üstünde zikzak çizerken Newton tek adımda merkeze atlar; sağda log-ölçekli hata GD için düz (linear) iner, Newton için tek adımda makine sıfırına çöker.

> **Builder Notu — Az Pahalı Adım mı Çok Ucuz Adım mı**
>
> Kuadratik vs linear ayrımı pratikte belirleyici: Newton az adımda yakınsar ama her adım pahalı (Hessian); gradient descent çok adım ister ama her adım ucuz (sadece gradyan). ML köprüsü: derin öğrenme dev ölçekte olduğundan "çok ucuz adım" kazanır — gradient descent ve türevleri (SGD, Adam, Ders 22-25) standarttır; Levenberg-Marquardt küçük/orta problemlerde (klasik regresyon) hâlâ yaygın.



Şekil 28.4: Newton vs steepest descent (kâse A=diag(1,10)): Newton 1 adım, GD 60 adımda bitmez — oran 0.8182 = (kappa-1)/(kappa+1) = 9/11 BİREBİR. Solda kötü-koşullu kâsenin konturlarında GD zikzak çizerken Newton tek adımda merkeze ulaşır; sağda GD'nin hatası log-ölçekte düz inerek (linear) k=60'ta ~5e-5'e gelir, Newton ise tek adımda makine sıfırına düşer.

28.8 Konvekslik: Küme ve Fonksiyon

Optimizasyon problemlerinin anahtar kelimesi:

> *"Convexity is the key word for these problems." — Strang, 32:49*

İdeal problem: bir **konveks fonksiyonu** bir **konveks küme** üzerinde minimize et. **Konveks küme** tanımı: kümeden herhangi iki nokta al, aralarındaki çizgi tamamen kümede kalıyorsa konvekstir.

> *"…it stays in the set. So that's convexity…" — Strang, 38:19*

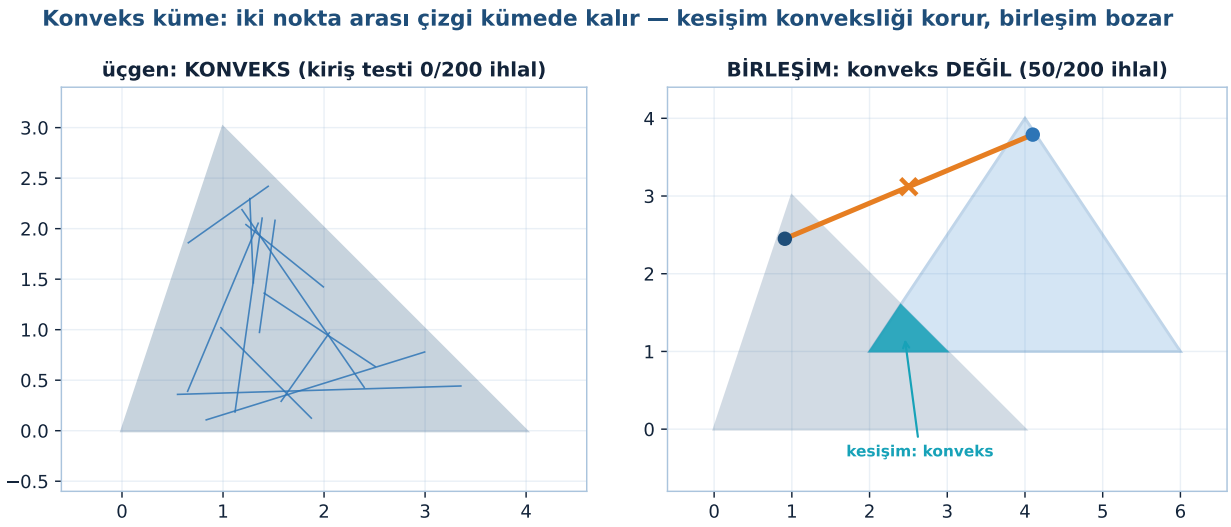
$$x_1, x_2 \in K \;\Rightarrow\; tx_1 + (1-t)x_2 \in K \quad (0 \le t \le 1)$$

Üçgen konvekstir (200 kiriş testinde 0 ihlal); iki üçgenin **birleşimi** genelde değil — kiriş dışarı çıkar (50/200 ihlal); ama **kesişimi** her zaman konvekstir (iki nokta her iki kümede, çizgi her ikisinde kalır → 0/100 ihlal). Kısıt kümesi K örn. $Ax = b$ (afin alt-uzay) olabilir. **Konveks fonksiyon** ise grafiğinin **üstünde ve üzerindeki** noktalar (epigraph) bir konveks küme oluşturuyorsa konvekstir — bir kâse şekli. İki konveks fonksiyonun **maksimumu** konvekstir (orta-nokta ihlali 0), ama **minimumu** değil (60 ihlal, max gap 0.344):

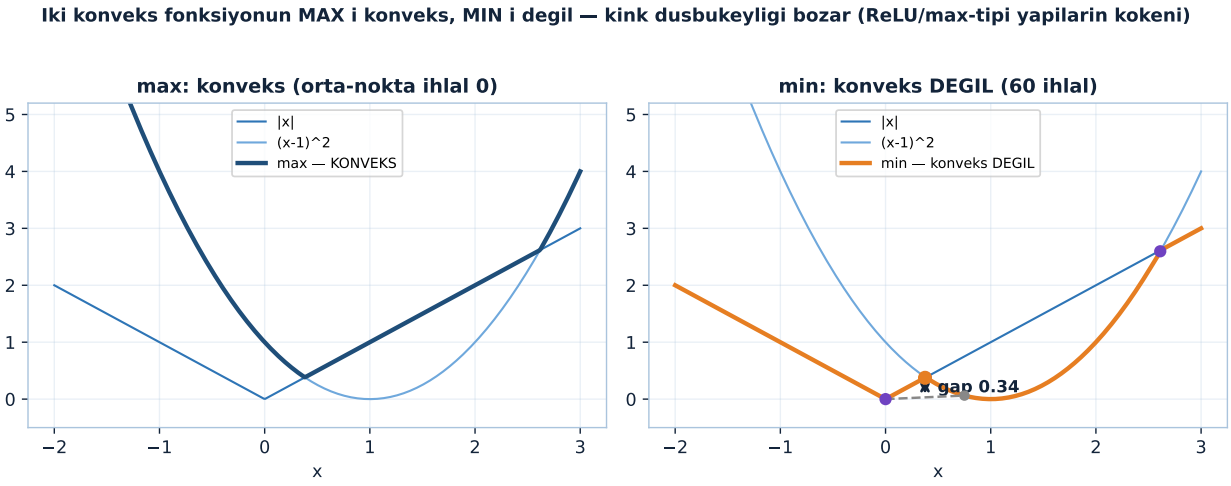
> *"…What about the maximum of the two functions?" — Strang, 49:05*

Minimum bir "kink" (köşe) yapar, dışbükeyliği bozar; maksimum keskin köşeyi yukarı tutar, konveks kalır.

Şekil 28.5 küme tarafını gösterir: tek üçgenin kirişleri içeride kalır, birleşimde turuncu kiriş dışarı çıkar, kesişim (teal) konveks kalır. Şekil 28.6 ise fonksiyon tarafını: $|x|$ ve $(x-1)^2$'nin üst zarfı (max) düzgün konveks, alt zarfı (min) kink'lerle bozulur.



Şekil 28.5: Konveks küme testi: iki nokta arasındaki tüm doğru parçası kümede kalmalı. Sol panel tek üçgen — 12 rastgele kiriş tamamen içeride (200 kiriş testinde 0 ihlal, KONVEKS). Sağ panel iki üçgenin birleşimi — bir ucu birinci üçgende, öbür ucu ikincide olan turuncu kiriş ortada her iki kümenin de dışından geçer (50/200 ihlal, konveks DEĞİL); buna karşılık kesişim bölgesi (teal) her zaman konvekstir (0/100 ihlal).



Şekil 28.6: İki konveks fonksiyonun **max**'ı konvekstir (orta-nokta ihlali 0), **min**'i değildir (60 ihlal, max gap 0.344). Üst zarf düzgün; alt zarftaki kink'ler dışbükeyliği bozar — bu, ReLU/max-tipi yapıların matematiksel kökenidir.

Builder Notu — Kâse Garantisi

Konvekslik = "tek minimum + gradient descent yakınsar" garantisi. Konveks fonksiyonun testi: Hessian her yerde pozitif yarı-tanımlı (Ders 5) — kâse her yönde yukarı büker. ML köprüsü: lojistik regresyon, SVM, LASSO konvekstir (küresel optimum garantili); ama derin ağlar **konveks değildir** (eyer noktaları, çok minimum — Ders 19) ama yine çalışır. Yine de çalışmaları, konveks-olmayan optimizasyonun pratikteki gizemidir. "max konveks, min değil" → ReLU (max-tipi) ağların parçalı-konveks yapısının kökeni.

28.9 Bu Dersin Özeti

- **Taylor (3 terim):** $F(x+\Delta x) \approx F(x) + (\Delta x)^T \nabla F + \frac{1}{2}(\Delta x)^T H(\Delta x)$; ∇F gradyan, H Hessian (simetrik).
- **Jacobian:** vektör f için $J_{jk} = \partial f_j / \partial x_k$; gradyanın Jacobian'ı = Hessian.
- **Newton (f=0):** $x_{k+1} = x_k - J^{-1} f(x_k)$. Örnek $x^2 - 9 \to x_{k+1} = \frac{1}{2}x_k + 9/(2x_k) \to 3$.
- **Newton (min, ∇F=0):** $x_{k+1} = x_k - H^{-1}\nabla F(x_k)$.
- **Yakınsama:** Newton kuadratik ($e \to e^2$), steepest descent linear ($e \to ce$, $c < 1$); Levenberg-Marquardt = ucuz Newton.
- **Konveks küme:** iki nokta arası çizgi kümede kalır; kesişim konveks, birleşim genelde değil.
- **Konveks fonksiyon:** epigraph konveks küme (kâse); test: Hessian pozitif yarı-tanımlı; max konveks, min değil.

Tek Bir Cümle

Bir fonksiyonu minimize etmek Taylor açılımıyla iteratif adım atmaktır — Newton ikinci-dereceyi (Hessian) kullanıp kuadratik yakınsar, gradient descent birinci-derecedir (linear); konvekslik (Hessian pozitif yarı-tanımlı, kâse) tek minimumu ve yakınsamayı garanti eder.

28.10 Kontrol Soruları

Soru 1 — Taylor'da gradyan ve Hessian

Soru: Çok değişkenli Taylor serisinin üç teriminde gradyan ve Hessian nerede çıkar, Hessian'ın özelliği nedir?
Cevap: $F(x+\Delta x) \approx F(x) + (\Delta x)^T \nabla F + \frac{1}{2}(\Delta x)^T H(\Delta x)$. Sabit terim $F(x)$; lineer terimde **gradyan** ∇F (n kısmi türev); kuadratik terimde **Hessian** H (n^2 ikinci türev). Hessian **simetriktir** ($\partial^2 F/\partial x_j \partial x_k = \partial^2 F/\partial x_k \partial x_j$). n büyükse gradyan n türev, Hessian $\sim \frac{1}{2}n^2$ türev ister — derin öğrenmede Hessian erişilemez.

Soru 2 — Newton iterasyonu ve √9

Soru: $f = 0$ çözmek için Newton iterasyonu nedir ve $x^2 - 9$ örneğinde nasıl görünür?
Cevap: $x_{k+1} = x_k - J(x_k)^{-1} f(x_k)$ (Taylor'ın lineer kısmını sıfıra eşitleyip Δx çözülür). $f(x) =$

$x^2 - 9$, $J = 2x$ için: $x_{k+1} = x_k - (x_k^2 - 9)/(2x_k) = \frac{1}{2}x_k + 9/(2x_k)$. $x_k = 3$'te sabit kalır (kök). Babil karekök yöntemi; kuadratik yakınsar (her adımda hane sayısı ikiye katlanır).

i Soru 3 — Newton vs steepest descent hızı

Soru: Newton ve steepest descent (gradient descent) yakınsama hızları nasıl farklıdır, neden?
Cevap: Newton **kuadratik** ($e_{k+1} \sim e_k^2$) — doğru ters Hessian'ı kullandığından her adımda hata karelenir, süper hızlı. Steepest descent **linear** ($e_{k+1} \sim c\, e_k$, $c < 1$) — doğru matrisin tersi yerine sabit bir adım kullandığından her adımda hata sadece bir oranla küçülür. Newton az ama pahalı adım, gradient descent çok ama ucuz adım.

i Soru 4 — Konveks küme ve max/min

Soru: Konveks küme nedir, ve iki konveks fonksiyonun min'i mi yoksa max'ı mı konvekstir?
Cevap: Konveks küme: içinden alınan herhangi iki noktayı birleştiren çizgi tamamen kümede kalır. İki konveks fonksiyonun **maksimumu konvekstir** (keskin köşe yukarı tutulur, kâse korunur); **minimumu konveks değildir** (kesişimde bir kink/köşe oluşur, dışbükeylik bozulur). Konveks kümelerin kesişimi konveks, birleşimi genelde değil.

28.11 Egzersizler

1. **Karekök Newton'ı.** $f(x) = x^2 - 2$ için Newton formülünü yaz ($x_{k+1} = \frac{1}{2}x_k + 1/x_k$). $x_0 = 1$'den başla, iki adım hesapla; $\sqrt{2} \approx 1.41421$'e ne kadar yaklaştın? (Motor tanığı: $1 \to 1.5 \to 17/12 = 1.41667$; iki adım sonra hata 2.45e-3.)

2. **Hessian sayımı.** $F(x_1, x_2, x_3) = x_1^2 + x_2 x_3$ için gradyanı ve Hessian'ı yaz. Hessian simetrik mi? Kaç bağımsız ikinci türev var?

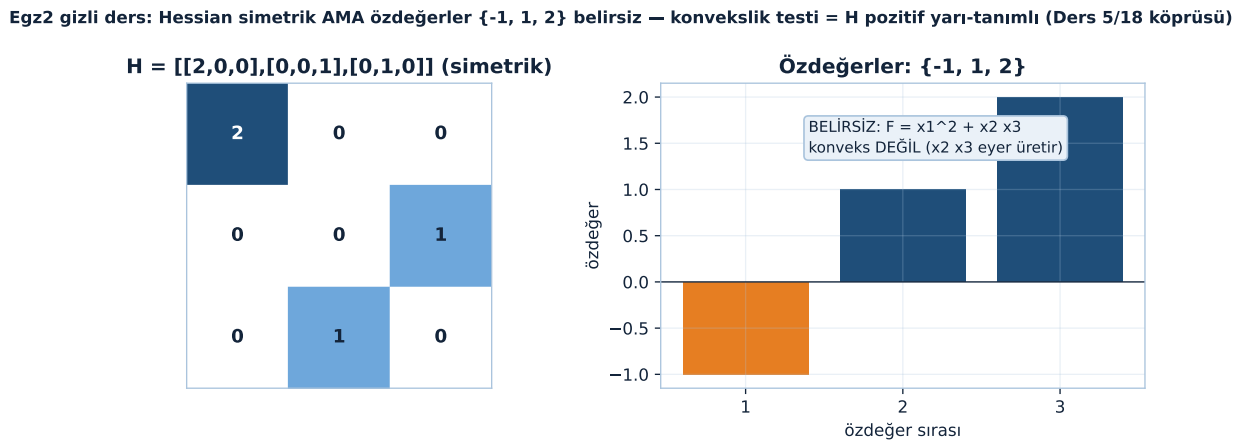
   [motor notu]: H'nin özdeğerleri {−1, 1, 2} — simetrik ama BELİRSİZ; $F = x_1^2 + x_2 x_3$ konveks değildir, $x_2 x_3$ terimi eyer üretir (Ders 18-19 köprüsü).

Şekil 28.7 bu egzersizin gizli dersini görselleştirir: Hessian simetrik olmasına rağmen özdeğer çubukları sıfırın iki yanına dağılır (belirsiz), yani F konveks değildir.

3. **Newton minimizasyon.** $F(x) = \frac{1}{2}(x - 4)^2$ (tek değişken). $H = F'' = 1$, $\nabla F = x - 4$. Newton adımı $x_{k+1} = x_k - H^{-1}\nabla F$'i hesapla; tek adımda minimuma ($x = 4$) ulaştığını göster. (Kuadratik fonksiyonda Newton tek adımda biter — neden?)

4. **Konvekslik testi.** $f(x) = x^4$ konveks mi? (İpucu: $f'' = 12x^2 \geq 0$, motor tanığı: 0 ihlal.) Peki $f(x) = x^3$? (Motor tanığı: 256 ihlal — konveks değil.) İki konveks fonksiyon $|x|$ ve x^2'nin maksimumu konveks mi?

5. **(Ders 22 habercisi)** Bu derste Newton'ın hızlı ama Hessian-pahalı olduğunu gördük. Hessian'ı atıp sadece gradyanla, sabit bir adım boyuyla inersek ne olur? Adım boyu (learning rate) çok büyük/küçük olursa? Bir tahmin yaz — Ders 22 "gradient descent: minimuma doğru aşağı" ile birinci-derece optimizasyonu derinlemesine işliyor.



Şekil 28.7: Egz2 gizli ders: Hessian simetrik AMA özdeğerler {-1, 1, 2} belirsiz — konvekslik testi = H pozitif yarı-tanımlı (Ders 5/18 köprüsü).

28.12 Sonraki Ders İçin Hazırlık

Ders 22: Gradient Descent — Minimuma Doğru Aşağı. Newton'ın pahalı Hessian'ını bırakıp en temel algoritmaya geçiş: $x_{k+1} = x_k - s\nabla F(x_k)$. Strang adım boyunu (learning rate s), yakınsama koşullarını ve kötü-koşullu (uzun-dar kâse) problemlerde zikzak davranışını işler — derin öğrenme eğitiminin çekirdeği.

> **Hazırlık**
>
> Bu dersteki Şekil 28.4 sol panelindeki zikzak yolu zihninde tut: kötü-koşullu kâse $A = \text{diag}(1, 10)$'da steepest descent oranı $9/11 = 0.8182$'ydi. Ders 22 bu sayının nereden geldiğini ($\kappa = \lambda_{\max}/\lambda_{\min}$ kondisyon sayısı, Ders 10) ve adım boyunun yakınsamayı nasıl belirlediğini açacak. Newton'ın "tek adım" lüksünü neden bıraktığımızı (milyar-parametreli H) hatırla.

28.13 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Taylor 3 terim	$F(x+\Delta x) \approx F + (\Delta x)^T\nabla F + \frac{1}{2}(\Delta x)^T H(\Delta x)$	1m44
Hessian	$H_{jk} = \partial^2 F/\partial x_j \partial x_k$; simetrik; $\sim \frac{1}{2}n^2$ türev	5m19
Jacobian	$J_{jk} = \partial f_j/\partial x_k$; gradyanın J'si = Hessian	8m56
Newton (f=0)	$x_+ = x - J^{-1}f(x)$	10m59
√9 örneği	$x_+ = \frac{1}{2}x + 9/(2x) \to 3$ (Babil)	14m21
Newton kuadratik	$e_+ \sim e^2$; steepest descent linear $e_+ \sim ce$	30m35
Konvekslik anahtar	konveks fonksiyon konveks küme üzerinde min	32m49
Konveks küme	iki nokta arası çizgi kümede kalır; kesişim konveks	38m19

Kavram	Formül / Fikir	Strang (dk)
max vs min	iki konveks fonk: max konveks, min değil	49m05

28.14 ML Bağlantıları Özeti

- **Newton vs birinci-derece:** Newton kuadratik (Hessian, az/pahalı adım); derin öğrenme n^2 Hessian'ı hesaplayamaz → gradient descent/SGD (birinci-derece, çok/ucuz adım).
- **Quasi-Newton:** BFGS/L-BFGS ters Hessian'ı gradyan farklarından tahmin eder; doğal gradyan/K-FAC eğrilik bilgisini yaklaşık kullanır.
- **Autodiff = backprop (Ders 27):** gradyan/Jacobian'ı verimli hesaplama; her katman bir Jacobian, zincir kuralı çarpar (JVP/VJP).
- **Konvekslik:** lojistik regresyon/SVM/LASSO konveks (küresel optimum garantili); derin ağlar konveks değil (eyer, çok minimum — Ders 19) ama yine çalışır.
- **Hessian = eğrilik:** kötü-koşullu (özdeğer oranı büyük) Hessian → gradient descent yavaş/zikzak (Ders 22); kondisyon sayısı (Ders 10) yakınsama hızını belirler.
- **max/min konvekslik → ReLU:** max-tipi (ReLU) parçalı-konveks yapı; ağların parçalı-doğrusal geometrisi.
- **Geriye köprü:** Ders 18-19 (Hessian, eyer noktaları), Ders 5 (pozitif yarı-tanımlı = konvekslik testi), Ders 16 (simetrik Hessian), Calculus (Taylor, türev).

! Kapanış

> *"Convexity is the key word for these problems." — Strang, 32:49*

Optimizasyon Taylor açılımıyla başlar; Newton eğriliği (Hessian) kullanıp hızlı yakınsar, gradient descent sadece eğimi kullanır, ve konvekslik tüm bunların tek bir minimuma varmasını garanti eder.

29 Gradient Descent — Minimuma Doğru Aşağı

Adım boyu, line search ve kondisyon sayısının zikzak dersi

> **Bölüm bilgisi**
>
> Bu ders derin öğrenmenin **merkezi algoritmasını** kurar: bir maliyet fonksiyonunu sadece birinci türevle (gradyan) adım adım küçültmek. Strang'in Ders 22 videosu (≈53 dk) ve OCW Lecture 22 temel alınmıştır. Okuma süresi ≈34 dk; önkoşul Ders 21 (Newton, Taylor, konvekslik).

29.1 Bu Derste Ne Var?

Newton'ın pahalı Hessian'ını (Ders 21) bıraktıktan sonra geriye derin öğrenmenin **merkezi algoritması** kalır: bir maliyet fonksiyonunu sadece birinci türevle (gradyan) iniş yönünde adım atarak küçültmek. İkinci türev (Hessian) milyon parametrede erişilemez olduğundan, her adım yalnızca bir gradyan hesabıdır. Strang her şeyi gören tek bir kuadratik örnek seçer ($f = \frac{1}{2}(x^2 + by^2)$), her adımı kapalı-formda çözer ve kondisyon sayısının yakınsama hızını nasıl kontrol ettiğini gözle görülür kılar.

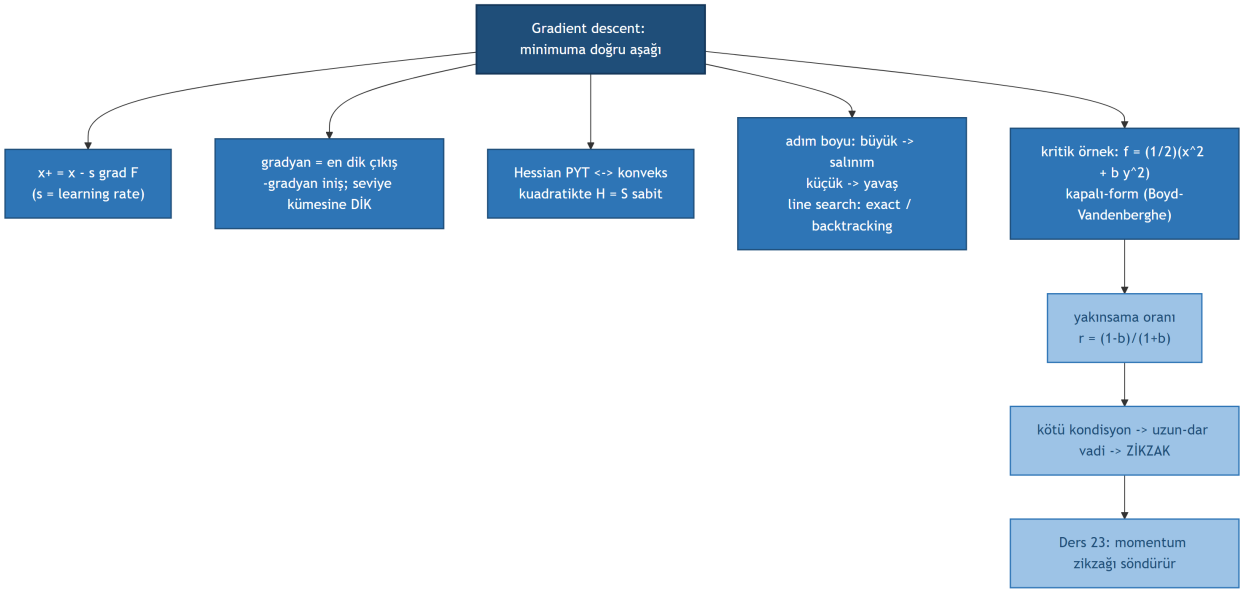
Beş sonuç:

1. **Gradient descent:** $x_+ = x - s\nabla F(x)$; s = adım boyu (learning rate).
2. **Gradyan = en dik çıkış yönü;** $-$gradyan = en dik iniş, seviye kümesine dik.
3. **Hessian $\oplus$ konvekslik:** H pozitif yarı-tanımlı $\Leftrightarrow$ konveks; pozitif tanımlı $\Leftrightarrow$ kesin konveks.
4. **Adım boyu seçimi:** tam çizgi araması (exact line search) veya backtracking; çok büyük $\rightarrow$ salınım, çok küçük $\rightarrow$ yavaş.
5. **Kondisyon sayısı hızı belirler:** örnekte yakınsama oranı $(1-b)/(1+b)$; b küçük (kötü κ) $\rightarrow$ oran $\approx 1 \rightarrow$ yavaş, **zikzak**.

> *"…that central algorithm of neural net deep learning, machine learning…" — Strang, 0:24*

Şekil 29.1 dersin iskeletini gösterir: merkezdeki "minimuma doğru aşağı" fikri tek satırlık güncelleme kuralından gradyanın geometrisine, Hessian/konvekslik testine ve adım boyu seçimine dallanır; kritik kuadratik örnek ise yakınsama oranı r üzerinden kötü kondisyonun yarattığı zikzağa ve Ders 23'ün momentum çözümüne bağlanır.

> **Builder Notu — Tek Satırlık Devrim**
>
> Bu dersin tamamı tek bir satıra indirgenir: $x_+ = x - s\nabla F$. Derin öğrenmenin her ağırlık güncellemesi bu satırdır — gradyan yönü zaten belli, geriye yalnızca "ne kadar gidelim?" (adım boyu s) kalır.



Şekil 29.1: Gradient descent kavram haritası — tek satırlık güncelleme kuralından, gradyanın geometrisine, kondisyon sayısının yarattığı zikzağa ve Ders 23 momentum çözümüne uzanan zincir.

> Newton'ın akıllı ama pahalı Hessian'ını bırakıp sadece eğimle inmek; milyar-parametreli modeller ancak bu ucuz adım sayesinde eğitilebilir. ML köprüsü: PyTorch'ta `w -= lr * w.grad` tam bu satırdır; SGD, Adam, RMSProp hepsi bu iskeletin üzerine adım-boyu/yön ayarlamaları ekler.

29.2 Gradient Descent ve Kritik Örnek

Çok değişken varsa (ikinci türev/Hessian almak için fazla), birinci türevle (gradyan) yetiniriz. Strang her şeyi gösteren tek bir örnek seçer — saf kuadratik, iki bilinmeyen:

$$F(x,y) = \tfrac{1}{2}(x^2 + b\,y^2), \qquad S = \begin{bmatrix} 1 & 0 \\ 0 & b \end{bmatrix}$$

S simetrik, köşegen, özdeğerleri 1 ve b ($b < 1$, küçük olan). Minimum $(0,0)$'da. Kritik büyüklük **kondisyon sayısı** $= \lambda_{\max}/\lambda_{\min} = 1/b$:

> *"…the condition number, which we'll see [is all important in the speed of convergence]…" — Strang, 1:52*

$1/b$ büyükse (b çok küçük), başımız dertte. Bu küçük örnekte adımları tam yazıp ne kadar hızlı yakınsadığımızı göreceğiz.

Builder Notu — Her Şeyi Gören Küçük Örnek

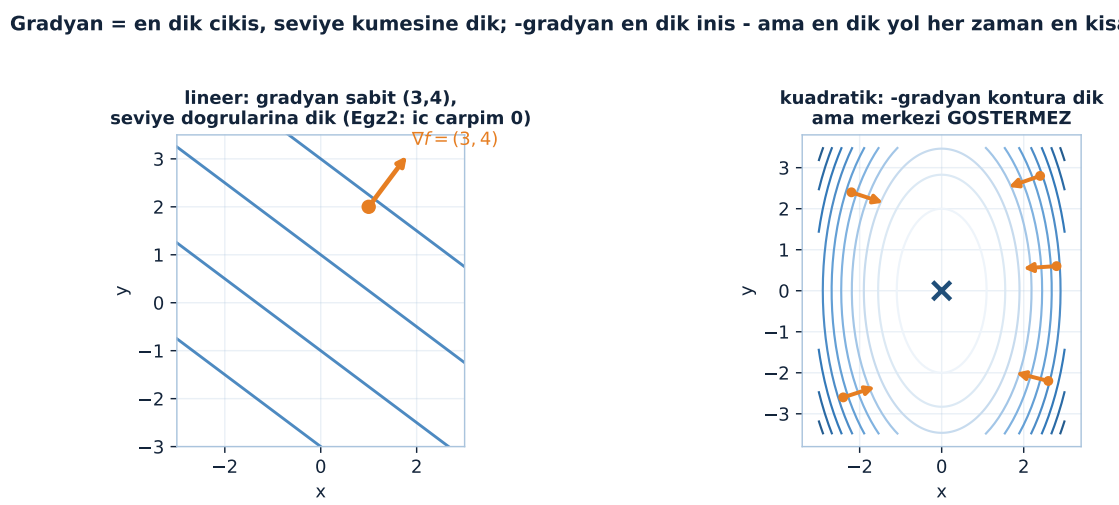
"Her şeyi gören küçük örnek" Strang'ın pedagojik imzası: 2×2 kuadratikte gradient descent'in tüm davranışı (hız, zikzak, kondisyon bağı) kapalı-formda okunur. ML köprüsü: gerçek kayıp yüzeyleri milyon-boyutlu ama yerel olarak kuadratiktir (Taylor, Ders 21); bu örnekteki kondisyon-sayısı dersi doğrudan gerçek eğitime taşınır.

29.3 Gradyanın Anlamı: En Dik İniş

Gradyanı sezgiyle yerleştir. Lineer örnek $f(x, y) = 3x + 4y$: gradyan sabit $(3, 4)$, Hessian sıfır. Gradyan bir yön gösterir:

> *"The gradient direction is the way up." — Strang, 7:14*

Gradyan = yüzeyde **en dik çıkış** yönü; $-$gradyan = **en dik iniş** yönü (steepest descent'teki eksi). Gradyan, seviye kümesine ($f = $ sabit) **diktir**: seviye kümesinde kalırsan f değişmez (gradyan oraya teğet sıfır bileşen), gradyan yönünde gidersen en hızlı değişir.



Şekil 29.2: Lineer fonksiyonun gradyanı sabit (3,4) ve seviye doğrularına diktir (Egzersiz 2: iç çarpım 0). Kuadratikte $-$gradyan her noktada kontura diktir, ama merkezi göstermez: en dik iniş yönü her zaman en kısa yol değildir.

Şekil 29.2 gradyanın iki yüzünü yan yana koyar: solda lineer $f = 3x + 4y$'nin sabit gradyanı $(3, 4)$ seviye doğrularına dik (iç çarpım 0, Egzersiz 2); sağda kuadratik konturlarda $-$gradyan her noktada kontura dik ama merkeze işaret etmez — en dik iniş yönü her zaman en kısa yol değildir.

Builder Notu — Yokuş Yukarı Gradyan Aşağı Eksi

"Gradyan seviye kümesine dik, en dik çıkış yönü" geometrisi optimizasyonun temel sezgisidir: tepeye en hızlı tırmanmak = gradyan, vadiye en hızlı inmek = $-$gradyan. ML köprüsü: backprop (Ders 27) tam bu gradyanı hesaplar; her ağırlık güncellemesi $-$gradyan yönünde küçük bir adımdır. "En dik" yön her

zaman en kısa yol değildir (zikzak) — Newton/momentum bunu düzeltir.

29.4 Hessian ve Konvekslik

Hessian (ikinci türev matrisi) yüzeyin eğriliğini söyler — ve konveksliği belirler:

"Positive definite or semi definite." — Strang, 11:12

Test: H pozitif **yarı-tanımlı** $\Longleftrightarrow$ fonksiyon **konveks**; H pozitif **tanımlı** $\Longleftrightarrow$ **kesin konveks** (her yönde gerçekten yukarı büker). Lineer fonksiyon konvekstir ama kesin değil ($H = 0$). Saf kuadratik için:

$$F(x) = \tfrac{1}{2}x^T Sx - a^T x \;\Rightarrow\; \nabla F = Sx - a, \quad H = S$$

Burada Hessian sabittir ($H = S$, her noktada aynı) — kuadratiğin güzelliği. S pozitif tanımlıysa F kesin konvekstir, tek minimumu vardır: $\nabla F = 0 \Rightarrow Sx = a \Rightarrow x^* = S^{-1}a$.

> Builder Notu — Kuadratiğin Sabit Eğriliği
>
> "Hessian PYT $\Longleftrightarrow$ konveks" testi (Ders 5 + Ders 21) eğitim peyzajını sınıflar. ML köprüsü: kuadratik yaklaşımda $H = S$ sabit; gerçek ağlarda H noktadan noktaya değişir ve çoğu yerde belirsizdir (eyer, Ders 19) — bu yüzden derin öğrenme konveks değildir ama yerel kuadratik analiz (bu örnek) yine de eğitim dinamiğini açıklar.

29.5 Gradient Descent Formülü

Algoritma basit: her adımda negatif gradyan yönünde bir adım at:

$$x_{k+1} = x_k - s_k \nabla F(x_k)$$

s_k = **adım boyu** (step size / learning rate). Tek karar verilmesi gereken şey budur — gradyan yönü zaten belli, geriye "ne kadar gidelim?" kalır. Kuadratik örnekte $\nabla F = (x, by)$ ($S = \text{diag}(1, b)$), yani:

$$(x_{k+1}, y_{k+1}) = (x_k, y_k) - s_k\,(x_k,\, b\,y_k)$$

Her iterasyon mevcut noktadan eğim aşağı küçük bir adım. Milyonlarca parametre için bile her adım sadece bir gradyan hesabı (ucuz) — Newton'ın Hessian-tersi (pahalı) yerine.

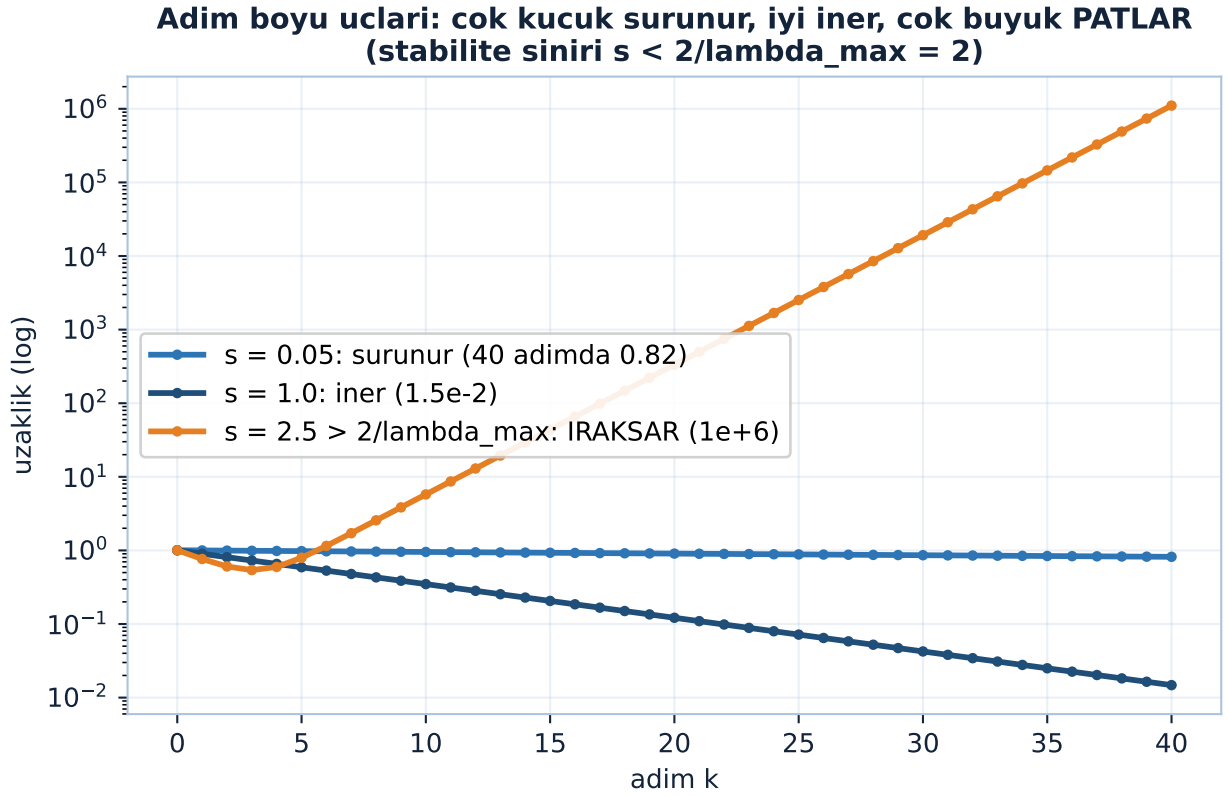
> Builder Notu — PyTorch'un O Satırı
>
> $x_{k+1} = x_k - s\nabla F$ derin öğrenmenin tek satırıdır: her ağırlık güncellemesi budur. ML köprüsü: PyTorch'ta `w -= lr * w.grad` tam bu satır; SGD (Ders 25), Adam, RMSProp hepsi bu iskeletin üzerine adım-boyu/yön ayarlamaları ekler. Ucuz adım (sadece gradyan) sayesinde milyar-parametreli modeller eğitilebilir.

29.6 Adım Boyu (Learning Rate)

Geriye tek karar kalır: adım boyu.

> *"…a step size, the learning rate." — Strang, 34:52*

Sabit alınabilir. **Çok büyük** learning rate → fonksiyon her yere sıçrar, salınır, felaket. **Çok küçük** → adımlar minik, çok uzun sürer. Sorun "tam doğru"yu bulmak. Pratikte büyük problemlerde: uygun bir s tahmin et, bir süre kullan, sonra geriye bakıp (salınım var mı?) ayarla.



Şekil 29.3: Adım boyu (learning rate) üçlüsü, b = 0.1, başlangıç (0.1, 1). Çok küçük adım (s = 0.05) sürünür: 40 adımda hâlâ 0.82 uzakta. İyi adım (s = 1.0) düzgün iner: 1.5e-2. Çok büyük adım (s = 2.5) stabilite sınırını (2/λmax = 2) aşar ve IRAKSAR: $\|p_{40}\| \approx$ 1e+6.

Şekil 29.3 üç adım boyu senaryosunu tek bir semilogy ekranda kıyaslar ($b = 0.1$, başlangıç $(0.1, 1)$): $s = 0.05$ sürünür (40 adımda hâlâ 0.82 uzakta), $s = 1.0$ düzgün iner (1.5e-2), $s = 2.5$ ise stabilite sınırını $(2/\lambda_{\max} = 2)$ aştığı için iraksar ($\|p_{40}\| \approx 10^6$ yukarı fırlar).

> **Builder Notu — En Pahalı Tek Sayı**
>
> Learning rate, derin öğrenmenin **en kritik hiperparametresidir**: çok büyük → kayıp NaN'a patlar; çok küçük → günlerce eğitir. ML köprüsü: learning rate schedule (cosine decay, warmup), adaptif yöntemler (Adam, RMSProp) ve learning-rate-finder araçları bu "tam doğru" arayışını otomatikleştirir. Strang'ın "tahmin et, kullan, geriye bak" reçetesi pratikte hâlâ geçerli.

29.7 Line Search: Exact ve Backtracking

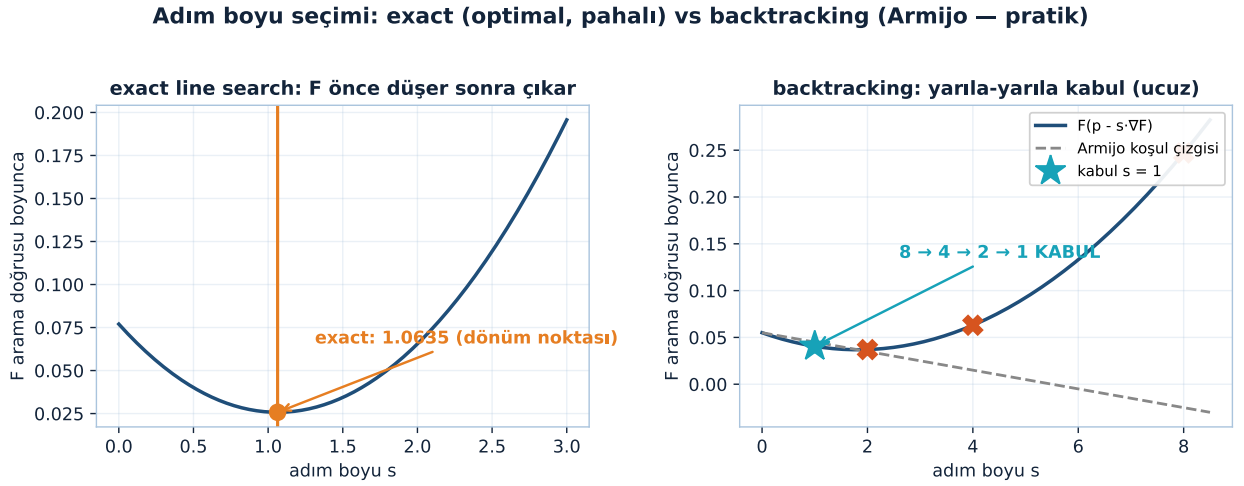
Adım boyunu sistematik seçmenin iki yolu. **Exact line search** (tam çizgi araması): s_k'yi, F'yi arama yönü ($-$gradyan) boyunca **minimum** yapacak şekilde seç.

> *"…an exact line search…" — Strang, 36:09*

Konveks fonksiyonda arama çizgisi boyunca ilerledikçe F önce düşer, bir noktada yükselmeye başlar; exact search tam o dönüm noktasını bulur. Pahalıdır (her adımda bir alt-optimizasyon). **Backtracking line search**: sabit bir s_0 ile başla; çok ileri gittiyse yarıya kes ($s_0 \to \frac{1}{2}s_0 \to \frac{1}{4}s_0 \to \ldots$), tatmin olana dek geri çekil:

$$s_0,\ \tfrac{1}{2}s_0,\ \tfrac{1}{4}s_0,\ \ldots \quad (\text{veya } a\,s_0,\ a^2 s_0, \ldots)$$

Backtracking ucuz ve pratiktir; exact search optimal ama yavaş.



Şekil 29.4: Adım boyu seçimi. Sol: exact line search — arama doğrusu boyunca F bir kuadratik gibi önce düşer sonra çıkar, tam dönüm noktası s = 1.0635'tir (optimal ama her adımda çözülmesi pahalı). Sağ: backtracking (Armijo) — s_0 = 8'den başlayıp 8 → 4 → 2 → 1 yarılayarak s = 1'i kabul eder; gri kesik çizgi Armijo koşulunu (yeterli azalma) gösterir (ucuz ve pratik).

Şekil 29.4 iki stratejiyi yan yana koyar: solda exact line search arama doğrusu boyunca F'nin tam dönüm noktasını ($s = 1.0635$) bulur (optimal ama pahalı); sağda backtracking $s_0 = 8$'den başlayıp $8 \to 4 \to 2 \to 1$ yarılayarak Armijo koşulunu sağlayan $s = 1$'i kabul eder (ucuz ve pratik).

> **Builder Notu — Optimal mi Ucuz mu**
>
> Exact vs backtracking, "optimal ama pahalı" vs "yaklaşık ama ucuz" klasik ödünleşmesidir. ML köprüsü: derin öğrenmede line search neredeyse hiç kullanılmaz (her adım çok pahalı) — bunun yerine sabit/programlı learning rate tercih edilir. Backtracking (Armijo koşulu) klasik optimizasyonda ve L-BFGS'te standarttır; "geri çekil, kontrol et" mantığı trust-region yöntemlerinin de temeli.

29.8 Kritik Örnek: Tam Çözüm

Kuadratik örnekte ($f = \frac{1}{2}(x^2 + by^2)$) exact line search'le başla, başlangıç $(x_0, y_0) = (b, 1)$ — formülleri sadeleştiren akıllı seçim. Strang sonucu Boyd & Vandenberghe'nin *Convex Optimization* kitabından alır:

> *"…the book by Steven Boyd and Vandenberghe called Convex Optimization." — Strang, 42:56*

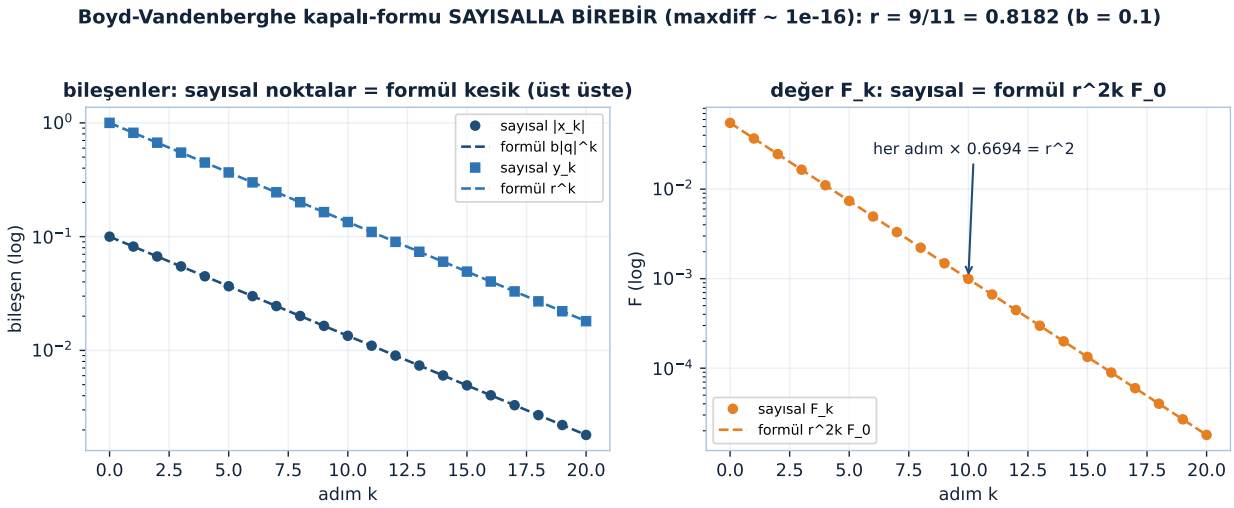
İterasyonlar kapalı-formda çıkar ($q = (b-1)/(b+1)$ ve $r = (1-b)/(1+b)$ kilit oranlar):

$$x_k = b\,q^k \quad (q < 0), \qquad y_k = r^k, \qquad F_k = r^{2k}\,F_0$$

[motor notu]: Notion/Boyd-Vandenberghe özeti y_k'yi q^k yazar; sayısal iz (maxdiff 1e-16) y_k'nin pozitif-monoton r^k olduğunu gösterir — zikzak dar (x) eksendedir; $F_k = r^{2k}F_0$ işaretten bağımsız birebir.

Her adımda x işaret değiştirir ($q < 0$, dar eksende zikzak), y pozitif-monoton sürünür, F sabit r^2 oranıyla çarpılır. İşte gradient descent'in hızı tek bir sayıda: $r = (1-b)/(1+b)$.

> *"…this ratio 1 minus b over 1 plus b is crucial." — Strang, 45:11*



Şekil 29.5: Boyd-Vandenberghe kapalı-formu, sayısal exact-GD ile BİREBİR (b = 0.1, başlangıç (b, 1), r = 9/11 = 0.8182). Sol: sayısal |x_k| navy noktalar formül b|q|^k navy kesik üstüne, sayısal y_k steel kareler formül r^k steel kesik üstüne oturur (maxdiff ~ 1e-16). Sağ: sayısal F_k turuncu noktalar formül r^2k·F_0 turuncu kesik üstüne; her adımda F değeri × 0.6694 = r^2.

Şekil 29.5 Boyd-Vandenberghe kapalı-formunun sayısal exact-GD ile birebir çakıştığını gösterir ($b = 0.1$, $r = 9/11 = 0.8182$, maxdiff $\sim 10^{-16}$): solda sayısal $|x_k|$ ve y_k formül eğrilerinin üstüne oturur, sağda sayısal F_k formül $r^{2k}F_0$ ile çakışır — her adımda F değeri $\times 0.6694 = r^2$.

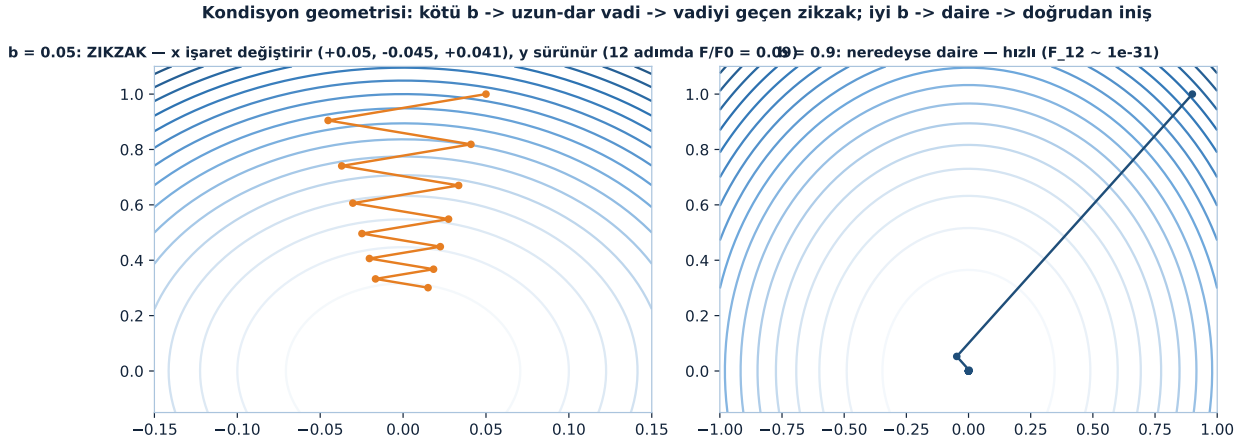
> **Builder Notu — Gözle Görülür Teori**
>
> Kapalı-form çözüm gradient descent teorisini "gözle görülür" kılar: yakınsama geometrik (her adım $\times r$), hız tek orana bağlı. ML köprüsü: gerçek eğitimde de kayıp geometrik düşer (log-lineer eğri); düşüş hızı yerel Hessian'ın kondisyon sayısına bağlıdır — bu örnek o bağı ispatlar.

29.9 Kondisyon Sayısı ve Zikzak

Kilit ders: yakınsama oranı $r = (1-b)/(1+b)$, kondisyon sayısına $(1/b)$ bağlı. $b \approx 1$ (iyi kondisyon): $r ≈ 0/2 = $ küçük → hızlı yakınsama, sorun yok. $b \approx 0$ (kötü kondisyon, $1/b$ büyük): $r \approx 1 \rightarrow$ çok yavaş. Yakınsar ($r < 1$) ama her adımda az ilerler:

$$b \to 1 : r \to 0 \text{ (hizli)}, \qquad b \to 0 : r \to 1 \text{ (cok yavas, zikzak)}$$

Kötü-koşullu durumda seviye kümeleri **uzun-dar bir vadi**dir; gradient descent vadiyi geçerek zikzak çizer — biraz iner, karşı duvara tırmanır, geri döner. İlerleme vadinin uzun ekseni boyunca çok yavaştır.

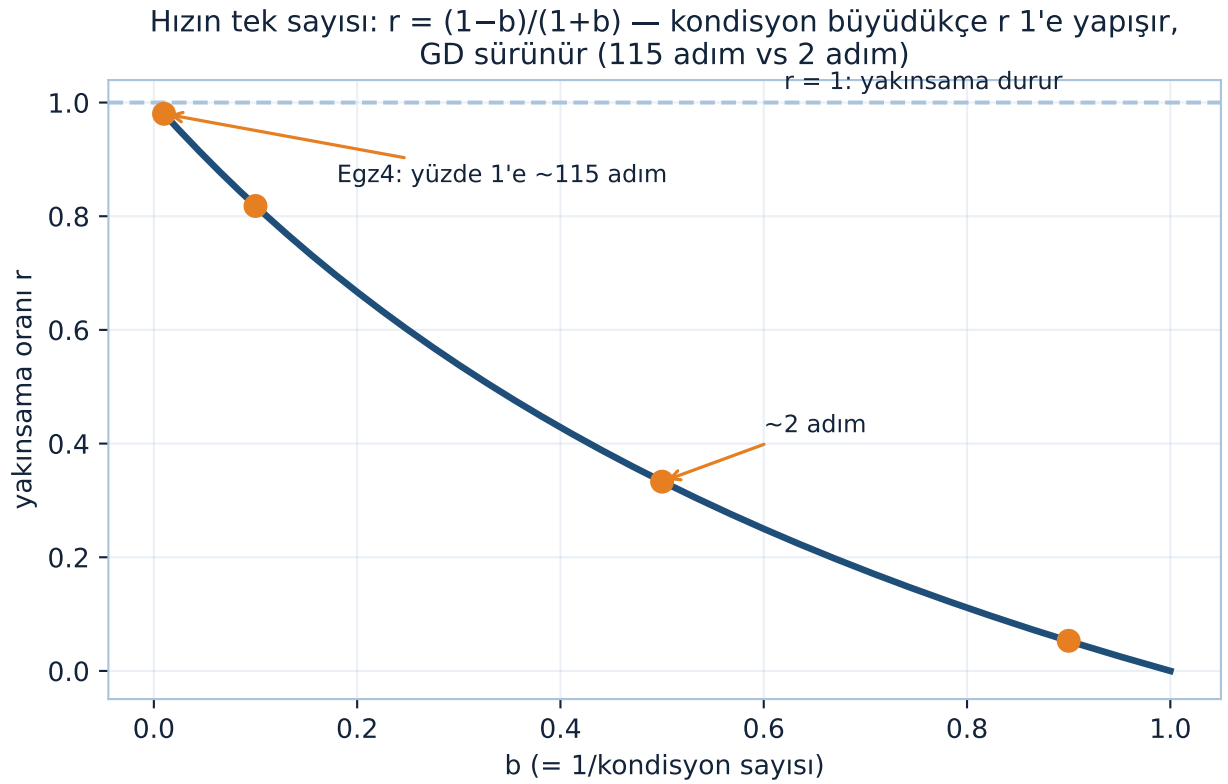


Şekil 29.6: Kondisyon sayısı geometri olarak. Sol (b = 0.05): uzun-dar vadi konturları, exact line search yörüngesi vadiyi geçen ZIKZAK çizer — x her adımda işaret değiştirir (+0.05, -0.045, +0.041, …), y sürünerek iner; 12 adımda F/F0 = 0.09 (yavaş). Sağ (b = 0.9): konturlar neredeyse daire, yörünge 2-3 adımda merkeze iner (F_12 ~ 1e-31). Kötü kondisyon -> uzun-dar vadi -> zikzak; iyi kondisyon -> daire -> doğrudan iniş.

Şekil 29.6 kondisyon sayısını geometriye çevirir: solda $b = 0.05$ uzun-dar vadisinde exact line search yörüngesi vadiyi geçen zikzak çizer (x her adımda işaret değiştirir: $+0.05, -0.045, +0.041, ...$; y sürünür, 12 adımda $F/F_0 = 0.09$); sağda $b = 0.9$ neredeyse daire olduğundan yörünge 2-3 adımda merkeze iner ($F_{12} \approx 10^{-31}$).

Şekil 29.7 yakınsama oranını tek bir eğride özetler: $b = 1/\kappa$ küçüldükçe (kondisyon büyüdükçe) $r = (1-b)/(1+b)$ değeri 1'e yapışır ve GD sürünür — $b = 0.01 \rightarrow r = 0.9802$ (%1'e ~115 adım, Egzersiz 4), $b = 0.1 \rightarrow r = 0.818$, $b = 0.5 \rightarrow r = 0.333$ (~2 adım), $b = 0.9 \rightarrow r = 0.053$.

> **Builder Notu — Vadinin Duvarları**
>
> "Kötü kondisyon → uzun-dar vadi → zikzak" gradient descent'in temel zaafıdır. ML köprüsü: derin ağ kayıp yüzeyleri aşırı kötü-koşulludur (kondisyon sayısı milyonlar); saf gradient descent zikzak yapar. Çözümler: **momentum** (Ders 23) zikzağı söndürür, **batch normalization** peyzajı yeniden ölçekleyip kondisyonu iyileştirir, **Adam** her koordinata ayrı adım boyu verir. Kondisyon sayısı (Ders 10) burada eğitim hızının doğrudan belirleyicisidir.



Şekil 29.7: Yakınsama oranının tek sayısı r = (1−b)/(1+b). b = 1/kondisyon sayısı küçüldükçe (kondisyon büyüdükçe) r → 1 olur ve gradyan inişi sürünür. İşaretli noktalar: b = 0.01 → r = 0.9802 (Egz4: %1'e inmek için ~115 adım), b = 0.1 → r = 0.818, b = 0.5 → r = 0.333 (~2 adım), b = 0.9 → r = 0.053. r = 1 çizgisinde yakınsama tümden durur.

29.10 Bu Dersin Özeti

- **Gradient descent:** $x_{k+1} = x_k - s_k \nabla F(x_k)$; s_k = adım boyu (learning rate).
- **Gradyan:** en dik çıkış yönü, seviye kümesine dik; $-$gradyan = en dik iniş.
- **Hessian $\oplus$ konvekslik:** H PYT $\iff$ konveks; H pozitif tanımlı $\iff$ kesin konveks. Kuadratik: $H = S$ sabit, $x^* = S^{-1}a$.
- **Adım boyu:** çok büyük $\rightarrow$ salınım, çok küçük $\rightarrow$ yavaş; exact line search (optimal, pahalı) veya backtracking ($\frac{1}{2}s_0, \frac{1}{4}s_0 \ldots$).
- **Kritik örnek:** $f = \frac{1}{2}(x^2 + by^2)$, başlangıç $(b, 1)$; $F_k = \left(\frac{1-b}{1+b}\right)^{2k} F_0$.
- **Kondisyon sayısı $(1/b)$ hızı belirler:** $r = (1-b)/(1+b)$; $b \approx 1$ hızlı, $b \approx 0$ (kötü κ) yavaş + zikzak (uzun-dar vadi).

! Tek Bir Cümle

Gradient descent her adımda negatif gradyan yönünde bir adım boyu (learning rate) kadar iner; saf kuadratik örnekte yakınsama oranı tam $r = (1-b)/(1+b)$'dir ve kondisyon sayısı kötüleştikçe ($b \rightarrow 0$) bu oran 1'e yaklaşıp algoritmayı uzun-dar vadide zikzak çizerek yavaşlatır.

29.11 Kontrol Soruları

i Soru 1 — Gradyan, −gradyan ve seviye kümesi

Soru: Gradyan ve $-$gradyan bir yüzeyde hangi yönleri gösterir, seviye kümesiyle ilişkisi nedir?
Cevap: Gradyan ∇F **en dik çıkış** yönünü, $-\nabla F$ **en dik iniş** yönünü gösterir (steepest descent'teki eksi işaret). Gradyan, seviye kümesine ($F = $ sabit) **diktir**: seviye kümesinde hareket edersen F değişmez, gradyan yönünde gidersen F en hızlı değişir. Gradient descent bu yüzden $-$gradyan yönünde adım atar.

i Soru 2 — Hessian ile konvekslik

Soru: Bir fonksiyonun konveks olduğunu Hessian'dan nasıl anlarsın, ve kuadratik $\frac{1}{2}x^T Sx$ için Hessian nedir?
Cevap: H pozitif yarı-tanımlı $\iff$ konveks; H pozitif tanımlı $\iff$ kesin konveks (her yönde yukarı büker). Saf kuadratik $F = \frac{1}{2}x^T Sx - a^T x$ için Hessian **sabittir**: $H = S$ (her noktada aynı). S pozitif tanımlıysa F kesin konveks, tek minimum $\nabla F = Sx - a = 0 \Rightarrow x^* = S^{-1}a$.

i Soru 3 — Adım boyu uçları ve line search

Soru: Adım boyu (learning rate) çok büyük veya çok küçük seçilirse ne olur, line search ne yapar?
Cevap: Çok büyük $\rightarrow$ fonksiyon salınır/patlar (minimumu aşar). Çok küçük $\rightarrow$ adımlar minik, yakınsama çok yavaş. **Exact line search** s_k'yi arama yönünde F'yi minimize edecek şekilde seçer (optimal ama pahalı); **backtracking** sabit s_0'dan başlayıp yarıya kese kese ($\frac{1}{2}s_0, \frac{1}{4}s_0 \ldots$) tatmin olana dek geri çekilir (ucuz, pratik).

Soru 4 — Kondisyon sayısı ve hız

Soru: $f = \frac{1}{2}(x^2 + by^2)$ örneğinde kondisyon sayısı yakınsama hızını nasıl etkiler?
Cevap: Yakınsama oranı $r = (1-b)/(1+b)$; kondisyon sayısı $= 1/b$. $b \approx 1$ (iyi kondisyon) $\rightarrow$ $r \approx 0$ $\rightarrow$ çok hızlı. $b \approx 0$ (kötü kondisyon, $1/b$ büyük) $\rightarrow$ $r \approx 1$ $\rightarrow$ çok yavaş. Kötü durumda seviye kümeleri uzun-dar bir vadidir; gradient descent vadiyi geçerek **zikzak** çizer, uzun eksende çok yavaş ilerler. Momentum/batch norm bunu düzeltir.

29.12 Egzersizler

1. **Gradyan ve minimum.** $F(x, y) = \frac{1}{2}(x^2 + 4y^2)$. Gradyanı ∇F ve Hessian H'yi yaz. Kondisyon sayısı ($b = 4$ ise) kaç? Minimum nerede? (Motor tanığı: $\nabla F = (x, 4y)$, $H = \text{diag}(1, 4)$, $\kappa = 4$, minimum $(0, 0)$.)

2. **Yön ve dik.** $f(x, y) = 3x + 4y$ için gradyanı yaz. Seviye kümesi $3x + 4y = 12$ doğrusuna gradyanın dik olduğunu (iç çarpım) doğrula. (Motor tanığı: $\nabla f = (3, 4) \perp (4, -3)$, iç çarpım 0.)

3. **Tek adım kuadratik.** $F = \frac{1}{2}x^2$ (tek değişken, b yok). $\nabla F = x$. $x_0 = 5$'ten $s = 1$ adım boyuyla bir gradient descent adımı at; minimuma (0) tek adımda ulaştın mı? ($s = 1$ neden burada optimal? — $s = 1/\lambda = 1$ olduğundan tek adım yeter.)

4. **Yakınsama oranı.** $f = \frac{1}{2}(x^2 + by^2)$ için $b = 0.01$ (kötü kondisyon). $r = (1-b)/(1+b)$'yi hesapla; fonksiyonun her adımda kaçta kaçı kaldığını (r^2) bul. ~%1'e inmek için kaç adım gerekir (kabaca)? (Motor tanığı: $r = 0.9802$, ~ 115 adım — bkz. Şekil 29.7; karşılaştırma $b = 0.5 \rightarrow r = 1/3$, %1'e ~ 2.1 adım.)

5. **(Ders 23 habercisi)** Bu derste kötü kondisyonun zikzak yaptırdığını gördük. Peki zikzağı söndürmek için adıma "momentum" (önceki adımın hafızası) eklersek ne olur? Yakınsama oranı $(1-b)/(1+b)$'den nasıl iyileşir? Bir tahmin yaz — Ders 23 "gradient descent'i hızlandırmak" (momentum, Nesterov) ile zikzak sorununu çözüyor.

29.13 Sonraki Ders İçin Hazırlık

Ders 23: Gradient Descent'i Hızlandırmak (Momentum, Nesterov). Bu dersin zikzak sorununu çözüyoruz: adıma momentum (önceki yönün hafızası) eklemek vadiyi geçen salınımları söndürür ve yakınsama oranını $(1-b)/(1+b)$'den $(1-\sqrt{b})/(1+\sqrt{b})$'ye iyileştirir — kondisyon sayısının karekökü kadar hızlanma. Nesterov ivmelendirmesi optimal birinci-derece yöntem.

Hazırlık

Bu dersteki Şekil 29.6 sol panelindeki uzun-dar vadi yörüngesini zihninde tut: $b = 0.05$'te yakınsama oranı $r = (1-b)/(1+b) = 0.905$'ti ve GD vadiyi geçerek zikzak çizdi. Ders 23 momentumun bu oranı nasıl $(1-\sqrt{b})/(1+\sqrt{b})$'ye düşürdüğünü ($\sqrt{\kappa}$ kadar hızlanma) açacak. Kondisyon sayısının ($1/b$, Ders 10) neden eğitim hızının düşmanı olduğunu hatırla.

29.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Gradient descent	derin öğrenmenin merkezi algoritması	0m24
Kondisyon sayısı	$\kappa = \lambda_{\max}/\lambda_{\min} = 1/b$; hızı belirler	1m52
Gradyan = en dik çıkış	$-$gradyan iniş; seviye kümesine dik	7m14
Hessian $\oplus$ konvekslik	H PYT $\Longleftrightarrow$ konveks; pozitif tanımlı $\Longleftrightarrow$ kesin	11m12
Adım boyu / learning rate	$x_+ = x - s\nabla F$; büyük $\rightarrow$ salınım, küçük $\rightarrow$ yavaş	34m52
Line search	exact (optimal, pahalı) / backtracking ($\frac{1}{2}s_0$)	36m09
Kritik örnek	$f = \frac{1}{2}(x^2 + by^2)$, start $(b, 1)$ (Boyd & Vandenberghe)	42m56
Yakınsama oranı	$r = (1-b)/(1+b)$; $b \to 0 \rightarrow$ zikzak (uzun-dar vadi)	45m11

29.15 ML Bağlantıları Özeti

- **Learning rate = en kritik hiperparametre:** çok büyük $\rightarrow$ NaN patlama, çok küçük $\rightarrow$ yavaş; schedule/warmup/Adam otomatikleştirir.
- **Kondisyon = eğitim hızının düşmanı:** kötü-koşullu kayıp yüzeyi zikzak; momentum (Ders 23), batch norm (peyzajı yeniden ölçekler) ve Adam (koordinat-başı adım) düzeltir.
- $x_+ = x - s\nabla F$**:** PyTorch'ta `w -= lr * w.grad`; tüm SGD/Adam/RMSProp bu iskeletin türevi.
- **Birinci-derece zaferi:** ucuz adım (sadece gradyan) $\rightarrow$ milyar parametre eğitilebilir (Newton'ın Hessian'ı imkânsız, Ders 21).
- **Line search kullanılmaz:** derin öğrenmede her adım çok pahalı $\rightarrow$ sabit/programlı learning rate; backtracking klasik optimizasyonda/L-BFGS'te standart.
- **Yerel kuadratik:** gerçek kayıp yüzeyi yerel olarak $\frac{1}{2}x^T Hx$ (Taylor); bu örneğin kondisyon dersi gerçek eğitime doğrudan taşınır.
- **Geriye köprü:** Ders 21 (Newton, steepest descent, konvekslik), Ders 10 (kondisyon sayısı κ), Ders 5 (pozitif tanımlı), Ders 18-19 (Hessian, Rayleigh/eyer).

! Kapanış

> *"…this ratio 1 minus b over 1 plus b is crucial." — Strang, 45:11*

Gradient descent'in tüm hikayesi tek oranda: kondisyon sayısı iyiyse hızlı, kötüyse zikzak; bir sonraki ders momentum ile bu oranı kökten iyileştirir.

30 Gradient Descent'i Hızlandırmak — Momentum

Heavy ball, 2x2 özdeğer analizi, karekök hızlanması ve Nesterov

> **Bölüm bilgisi**
>
> Bu ders Ders 22'nin **zikzak** sorununu çözer: adıma momentum (önceki yönün hafızası) ekleyerek yakınsama oranını kondisyon sayısının karekökü ne indirir. Strang'in Ders 23 videosu (≈49 dk) ve OCW Lecture 23 temel alınmıştır. Okuma süresi ≈34 dk; önkoşul Ders 22 (gradient descent, kondisyon, zikzak).

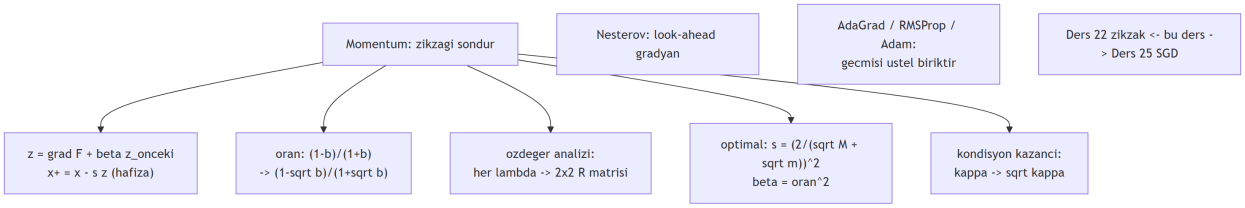
30.1 Bu Derste Ne Var?

Ders 22'nin **zikzak** sorununu çözüyoruz. Çözüm: adıma **momentum** (önceki adımın hafızası) eklemek. Sonuç mucizevi: yakınsama oranı $(1-b)/(1+b)$'den $(1-\sqrt{b})/(1+\sqrt{b})$'ye düşer — kondisyon sayısının **karekökü** kadar hızlanma. Strang her şeyi gören aynı kuadratik örnekte ($F = \frac{1}{2}(x^2 + by^2)$) iki satırlık momentum özyinelemesini özvektör başına 2×2 bir matrise indirger ve optimal s, β'yı kapalı-formda okur.

Beş sonuç:

1. **Momentum:** $z_k = \nabla F(x_k) + \beta z_{k-1}$; $x_{k+1} = x_k - s z_k$. Önceki yönün hafızası (β).
2. **Özdeğer analizi:** her özvektörü izle $\rightarrow 2 \times 2$ matris R; s ve β'yı R'nin en büyük özdeğerini tüm $\lambda \in [m, M]$ için minimize edecek seç.
3. **Optimal:** $s = (2/(\sqrt{M} + \sqrt{m}))^2$, $\beta = ((\sqrt{M} - \sqrt{m})/(\sqrt{M} + \sqrt{m}))^2$; yakınsama oranı $(\sqrt{M} - \sqrt{m})/(\sqrt{M} + \sqrt{m})$.
4. **Hızlanma:** kondisyon κ yerine $\sqrt{\kappa}$ — $b = 1/100$'de oran 0.98'den 0.82'ye düşer.
5. **Nesterov:** gradyanı kaydırılmış (look-ahead) noktada hesapla; alternatif ivmelendirme.

> *"…something called momentum to avoid the zigzag…" — Strang, 2:09*



Şekil 30.1: Ders 23 kavram haritasi — momentum hafizayla zikzagi sondurur, karekok kazanci getirir

Şekil 30.1 dersin iskeletini gösterir: merkezdeki "zikzağı söndür" fikri iki satırlık güncelleme kuralından (hafıza), oran dönüşümünden, özvektör başına 2×2 analizden, optimal (s, β) seçiminden ve $\kappa \to \sqrt{\kappa}$ kondisyon kazancından dallanır; ayrı düğümlerde Nesterov'un look-ahead gradyanı, modern AdaGrad/RMSProp/Adam ailesi ve Ders 22 (zikzak) ile Ders 25 (SGD) köprüleri durur.

> **Builder Notu — Hafızanın Karekök Mucizesi**
>
> Bu dersin tamamı tek bir mucizeye indirgenir: adıma önceki yönün hafızasını eklemek yakınsama oranındaki b'yi $\sqrt{b}$ yapar. $b = 1/100$'de $(1-b)/(1+b) = 0.98$ (sürünür) iken $(1-\sqrt{b})/(1+\sqrt{b}) = 0.82$ (hızlı) olur — gereken adım sayısı κ değil $\sqrt{\kappa}$ ile orantılıdır. ML köprüsü: derin öğrenmenin neredeyse her optimizer'ı bu hafıza terimini kullanır; Adam = momentum + adaptif adım. Strang'ın bu dersi tüm modern optimizer ailesinin matematiksel köküdür.

30.2 Zikzak Sorunu (Hatırlatma)

Ders 22'nin temel formülü: $x_{k+1} = x_k - s\nabla F(x_k)$. Kritik örnekte ($f = \frac{1}{2}(x^2 + by^2)$, küçük b) seviye kümeleri **uzun-ince elipsler**. Steepest descent her adımda elipse dik iner, en içteki elipse teğet olunca durur, sonra ters yöne döner — **zikzak**. Yakınsama bu sayıya bağlıydı:

$$r = \frac{b-1}{b+1}, \qquad F_k = \left(\frac{1-b}{1+b}\right)^{2k} F_0 \quad \text{(momentumsuz)}$$

b küçükse (kötü kondisyon) bu oran 1'e yakın → çok yavaş. Bugünün işi: momentum terimiyle bu oranı iyileştirmek.

> **Builder Notu — İsrafın Adı Zikzak**
>
> Zikzak, "en dik yön her zaman en kısa yol değildir" gerçeğinin görünür hâli (Ders 22). Steepest descent vadiyi dik açıyla geçer, ileri-geri sıçrar; vadi ekseni (asıl ilerleme yönü) boyunca neredeyse hiç gitmez. Momentum tam bu israfı düzeltir. ML köprüsü: NYU paralel kursunda KONUK Defazio'nun optimizasyon dersi de aynı resmi çizer — kötü kondisyon (κ) zikzak yaptırır, momentum bu salınımları söndürür.

30.3 Momentum: Önceki Adımın Hafızası

Çözüm: adıma "geçen adımı hatırlatan" bir terim ekle.

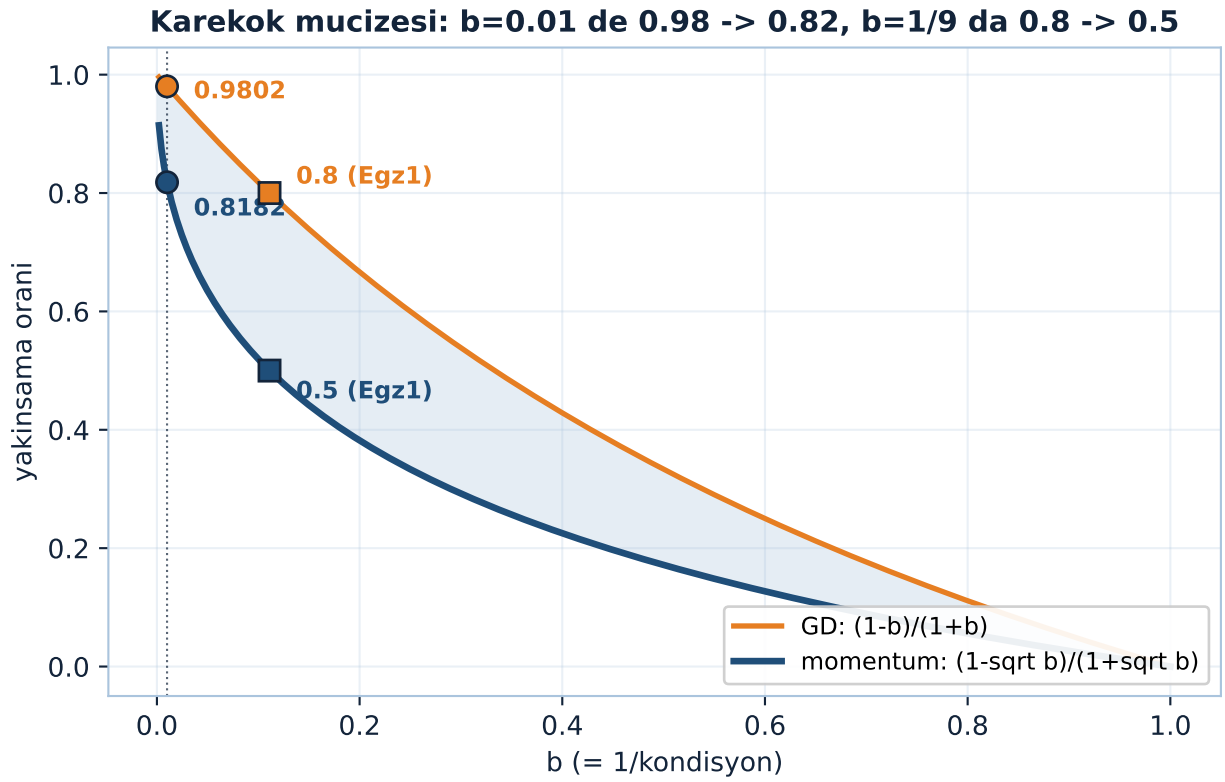
> *"…something called momentum to avoid the zigzag…" — Strang, 2:09*

Ana sonucu önce söyleyelim (cebir sonra). Momentum, sihirli yakınsama oranını kökten değiştirir:

$$\frac{1-b}{1+b} \quad \longrightarrow \quad \frac{1-\sqrt{b}}{1+\sqrt{b}}$$

> *"…changes to 1 minus square root of b over 1 plus square root of b." — Strang, 10:56*

Fark devasa. $b = 1/100$ örneğinde: momentumsuz oran $0.99/1.01 \approx 0.98$ (1'e çok yakın, yavaş); momentumlu $0.9/1.1 \approx 0.82$ (çok daha küçük, hızlı). $\sqrt{b}$ ortaya çıkması, kondisyon sayısının **karekökü** kadar hızlanma demek.

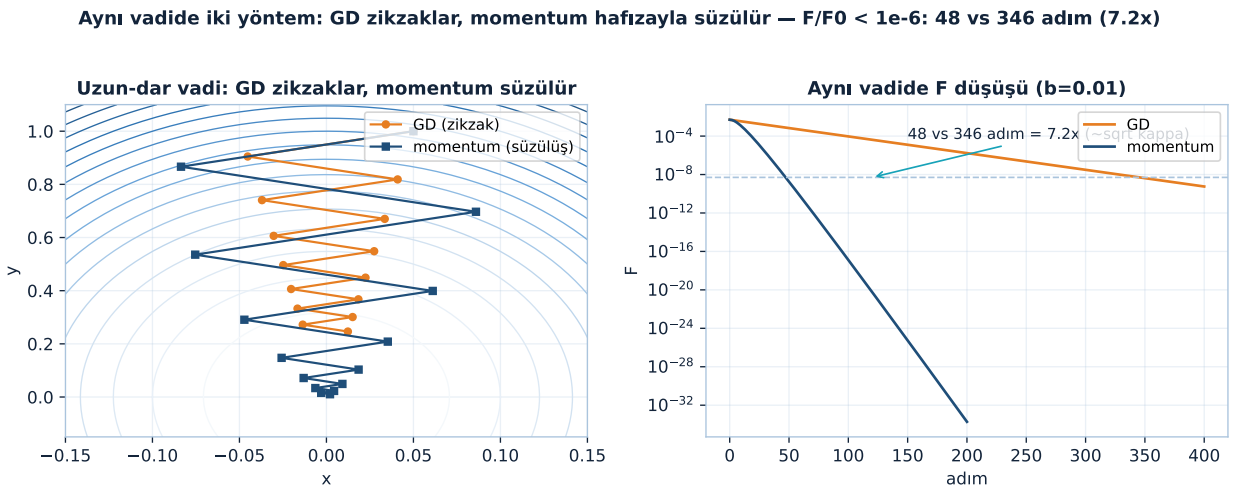


Şekil 30.2: GD ve momentum yakınsama oranları aynı eksende: kondisyonun tersi b ($b = 1/\kappa$) küçüldükçe GD oranı $(1-b)/(1+b)$ (turuncu) yavaşça 1'e tırmanır, momentum oranı $(1-\sqrt{b})/(1+\sqrt{b})$ (navy) ise karekök sayesinde çok daha düşük kalır. $b = 0.01$ noktasında GD oranı 0.9802 iken momentum 0.8182'ye iner; $b = 1/9$'da (Egz1) GD 0.8 iken momentum 0.5'tir. İki eğri arasındaki gri dolgu momentumun her kondisyonda kazandırdığı farkı gösterir.

Şekil 30.2 GD ve momentum oranlarını aynı eksende kıyaslar: $b = 1/\kappa$ küçüldükçe GD oranı $(1-b)/(1+b)$ (turuncu) yavaşça 1'e tırmanır, momentum oranı $(1-\sqrt{b})/(1+\sqrt{b})$ (navy) karekök sayesinde çok daha düşük kalır — $b = 0.01$'de $0.9802 \to 0.8182$, $b = 1/9$'da (Egzersiz 1) $0.8 \to 0.5$.

Şekil 30.3 Ders 22'nin uzun-dar vadisini geri getirir: solda $b = 0.05$'te GD vadiyi geçen zikzak çizer (turuncu) ama momentum hafızasını kullanıp süzülerek iner (navy); sağda $b = 0.01$ kâsesinde $F/F_0 < 10^{-6}$ eşiğine momentum 48 adımda, GD 346 adımda ulaşır — 7.2 kat fark, $\sqrt{\kappa} = 10$ mertebesinde.

> **Builder Notu — Topun Biriken Hızı**
>
> Momentum'un fizik sezgisi: yokuş aşağı yuvarlanan bir top. Top her adımda hız (velocity) biriktirir — zikzak salınımları birbirini götürür (sağa-sola), ama vadi ekseni boyunca ivme birikir (hep aynı yön). ML köprüsü: SGD + momentum derin öğrenmenin standart optimizer'ıdır; "ağır top" (heavy ball) yöntemi tam budur, β momentum katsayısı genelde 0.9.



Şekil 30.3: Aynı vadide iki yöntem. Sol: $b = 0.05$ uzun-dar vadide GD turuncu adımlarla zikzak çizer (Ders 22 deseni), momentum lacivert adımlarla hafızasını kullanıp süzülerek iner. Sağ: $b = 0.01$ kâsesinde F düşüşü (semilog) — GD yavaş, momentum dik. $F/F_0 < 10^{-6}$ eşiğine momentum 48 adımda, GD 346 adımda iner: 7.2 kat fark (mertebe $\sqrt{\kappa} = 10$).

30.4 Momentum Formülü

İki denklemle yaz. Bir arama yönü z_k tut; lider terimi gradyan, üstüne önceki yönün β katı (hafıza):

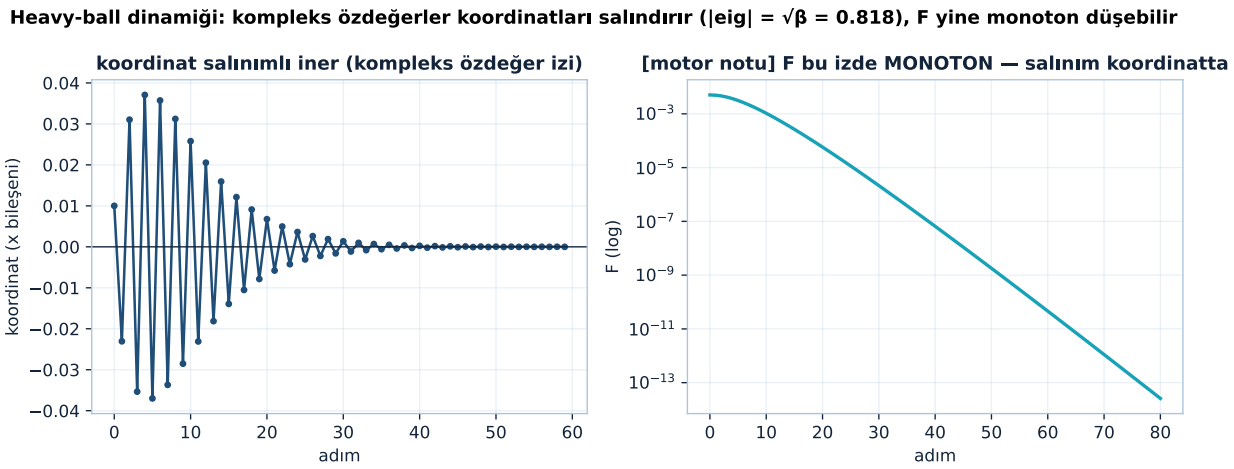
$$z_k = \nabla F(x_k) + \beta\, z_{k-1}, \qquad x_{k+1} = x_k - s\, z_k$$

$\beta = 0$ olsaydı $z_k = \nabla F$ olur, sıradan steepest descent'e dönerdik. $\beta > 0$ ile her adım önceki hareketi "hatırlar" — z_k geçmiş gradyanların üstel-ağırlıklı toplamıdır. İki ayar parametresi: adım boyu s ve momentum katsayısı β.

*[motor notu]: Şekil 30.4 sol panelindeki koordinat 60 adımda 59 işaret değiştirir (sönümlü salınım) — heavy-ball'un kompleks eşlenik özdeğerlerinin izi ($|\text{eig}| = \sqrt{\beta} = 0.8182$). Ama bu izde enerji F **monoton** düşer: salınım koordinattadır, F'de değil — "F yükselir" demek yanlış olur.*

Şekil 30.4 heavy-ball dinamiğinin iki yüzünü ayırır: solda bir koordinat sönümlü salınımla iner (kompleks özdeğer izi), sağda aynı yörüngede F monoton düşer — salınım koordinatta görünür ama enerji düzenli azalır.

> **Builder Notu — Geçmişin Geometrik Toplamı**
>
> $z_k = \nabla F + \beta z_{k-1}$ açılırsa $z_k = \nabla F_k + \beta \nabla F_{k-1} + \beta^2 \nabla F_{k-2} + \ldots$ — geçmiş gradyanların geometrik-ağırlıklı (β azalan) toplamı. ML köprüsü: PyTorch'ta `momentum=0.9` tam bu β; SGD(momentum) bu iki satırı uygular. AdaGrad/RMSProp/Adam farklı ağırlıklandırmalarla aynı "geçmişi biriktir" fikrini kullanır.



Şekil 30.4: Heavy-ball dinamiği: kompleks özdeğerler koordinatları salındırır (|eig| = √β = 0.818), F yine monoton düşebilir. Sol: bir koordinat (λ=1 yönü, reel negatif çift özdeğer −0.818) salınımlı iner — 60 adımda 59 işaret değişimi. Sağ: aynı izde enerji F monoton düşer; salınım koordinatta, F'de değil.

30.5 Özdeğer Analizi: 2×2 Matris R

Cebir nasıl çözülür? S simetrik olduğundan özvektörlerine ayır; **her özvektörü ayrı ayrı izle** — problem skalere iner (her λ için ayrı). Momentum özyinelemesi, (c_k, d_k) katsayıları için 2×2 bir matrisle çarpmaya dönüşür:

"…it's a miracle that the algebra… you really see the value of eigenvectors." — Strang, 9:36

$$\begin{bmatrix} c_{k+1} \\ d_{k+1} \end{bmatrix} = R \begin{bmatrix} c_k \\ d_k \end{bmatrix}, \qquad R = \begin{bmatrix} 1 - s\lambda & \beta \\ -s\lambda & \beta \end{bmatrix}$$

Her adım R ile çarpım. Yakınsama hızı R'nin **özdeğerlerine** bağlı (R^k küçülmeli). R, λ (S'nin özdeğeri) ile s ve β'ya bağlı. İş: s ve β'yı, R'nin en büyük özdeğerini **tüm** $\lambda \in [m, M]$ aralığında en küçük yapacak şekilde seç.

> Builder Notu — Her Özvektör Kendi 2x2'si
>
> "Her özvektörü izle, problem skalere insin" Ders 4'ün köşegenleştirme gücünün doruğu: bir matris özyinelemesi, özvektör başına bağımsız 2×2 sisteme ayrışır. ML köprüsü: bu spektral analiz, optimizer'ların neden farklı özdeğer (eğrilik) yönlerinde farklı davrandığını açıklar — Adam'ın koordinat-başı adım boyu tam bu "her yön ayrı" fikrinin pratik versiyonu.

30.6 Optimal s ve β (Mucize)

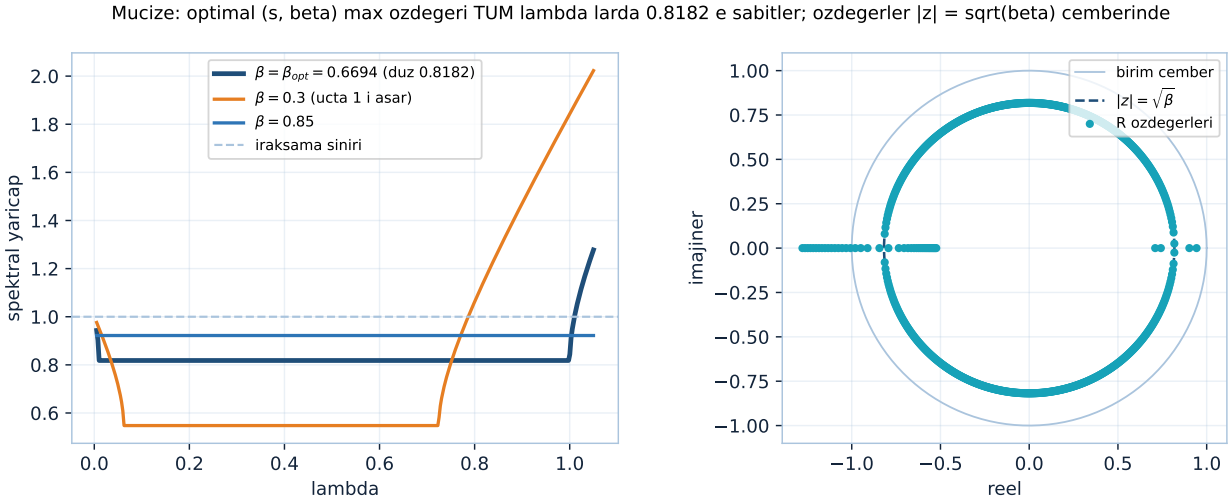
İki parametreyi (s, β) tüm $\lambda \in [m, M]$ için R'nin en büyük özdeğerini minimize edecek şekilde seç — şaşırtıcı derecede temiz bir çözüm çıkar (Strang'ın "mucize" dediği). $m = \lambda_{\min}$, $M = \lambda_{\max}$ cinsinden:

$$s_{\text{opt}} = \left(\frac{2}{\sqrt{M}+\sqrt{m}}\right)^2, \qquad \beta_{\text{opt}} = \left(\frac{\sqrt{M}-\sqrt{m}}{\sqrt{M}+\sqrt{m}}\right)^2$$

Bu seçimle R'nin özdeğerleri (yani adım-başı küçülme oranı) şu sınıra iner:

$$\text{yakinsama orani} = \frac{\sqrt{M}-\sqrt{m}}{\sqrt{M}+\sqrt{m}}$$

Örnek ($M = 1$, $m = b$): $\beta_{\text{opt}} = ((1-\sqrt{b})/(1+\sqrt{b}))^2$, oran $(1-\sqrt{b})/(1+\sqrt{b})$. Momentumsuz $(1-b)/(1+b)$'nin $\sqrt{b}$ versiyonu.



Şekil 30.5: Optimal (s, β) ile R'nin maksimum özdeğeri TÜM $\lambda \in [m, M]$ aralığında 0.8182'ye sabitlenir (eşit-dalgalanma, çift kök). Yanlış β ya yakınsamayı yavaşlatır (steel, $\beta = 0.85$) ya da ıraksatır (turuncu, $\beta = 0.3$ uçta 1'i aşar). Sağda: özdeğerler ara λ'larda kompleks eşlenik olup tam $|z| = \sqrt{\beta}$ çemberi üzerinde durur.

Şekil 30.5 mucizeyi spektral olarak gösterir: solda optimal (s, β) ile R'nin maksimum özdeğeri tüm λ aralığında düz 0.8182'ye sabitlenir (eşit-dalgalanma, çift kök), yanlış β ya yavaşlatır ($\beta = 0.85$, steel) ya da uçta 1'i aşıp ıraksatır ($\beta = 0.3$, turuncu); sağda özdeğerler ara λ'larda kompleks eşlenik olup tam $|z| = \sqrt{\beta}$ çemberi üzerinde durur.

Builder Notu — Polyak'ın Ağır Topu

Bu, Polyak'ın "ağır top" (heavy ball) yönteminin optimal parametreleridir. ML köprüsü: pratikte M, m (Hessian özdeğerleri) bilinmez, bu yüzden $\beta \approx 0.9$ sabit alınır ve s ayarlanır; ama teori, momentum'un neden bu kadar yardımcı olduğunu kanıtlar — kondisyon sayısı kötüleştikçe kazanç büyür.

30.7 Kondisyon Sayısı: √κ Hızlanma

Anahtar kavram kondisyon sayısı $\kappa = \lambda_{\max}/\lambda_{\min} = M/m$ (örnekte $1/b$). κ büyükse problem zor; $\kappa = 1$ ise ($M = m$, \$S = \$ identity katı) problem trivial. Momentum'un kazancı: efektif kondisyon sayısını **karekökü**ne indirir:

$$\kappa = \frac{M}{m} = \frac{1}{b} \quad \Longrightarrow \quad \text{momentumsuz} \sim \kappa, \quad \text{momentumlu} \sim \sqrt{\kappa}$$

$b = 1/100$ ($\kappa = 100$): momentumsuz oran $\approx (\kappa - 1)/(\kappa + 1) \approx 0.98$; momentumlu $\approx (\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1) = 9/11 \approx 0.82$. Yakınsama için gereken adım sayısı $\sqrt{\kappa}$ ile orantılı (κ değil) — büyük κ'da muazzam fark.

> Builder Notu — Karekök Kazancı
>
> "$\kappa \to \sqrt{\kappa}$" momentum'un teorik vaadidir ve birinci-derece yöntemler için neredeyse optimaldir (Nesterov bunu kanıtladı). ML köprüsü: derin ağlarda κ astronomik (milyonlar); momentum olmadan eğitim pratik değil. NYU paralel kursunda KONUK Defazio momentum sönümlemesini tam bu çerçevede anlatır — batch normalization κ'yı küçülterek, momentum $\sqrt{\kappa}$'ya indirerek birlikte eğitimi mümkün kılar; ikisi de "kötü kondisyon" düşmanına karşı.

30.8 Nesterov İvmelendirmesi

Momentum'a alternatif bir hızlandırma, Rus matematikçi Nesterov'dan (makaleleri zor okunur ama içerik ciddi):

> *"…Nesterov's idea." — Strang, 42:08*

Fikir benzer (önceki değeri kullan) ama bir incelikle: gradyanı **mevcut** noktada değil, ileriye **kaydırılmış** (look-ahead) bir y_k noktasında hesapla:

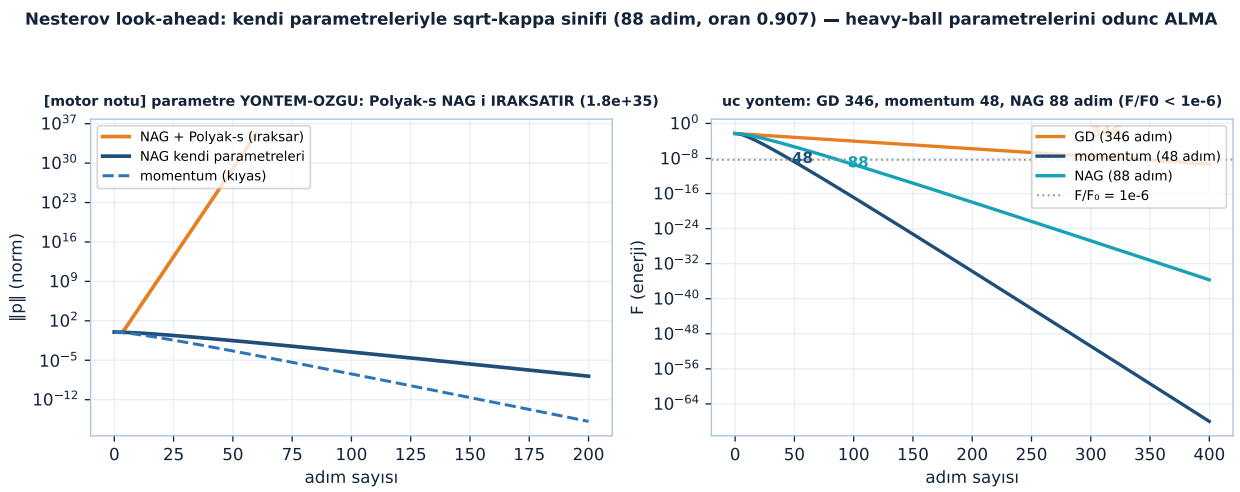
$$x_{k+1} = y_k - s\,\nabla F(y_k)$$

> *"…the gradient is being evaluated at some different point." — Strang, 44:09*

y_k, x'lerden bir ekstrapolasyonla (momentum yönünde bir adım ileri) elde edilir. Yani "önce momentum yönünde bak, oradaki eğimi ölç, sonra adım at." Bu look-ahead, momentum'un aşırı-atlama (overshoot) eğilimini frenler ve aynı $\sqrt{\kappa}$ hızlanmasını verir — birinci-derece yöntemler için **optimal**.

*[motor notu]: heavy-ball'un optimal $s = 3.3058$ değeri NAG'a taşınırsa yöntem **ıraksar** ($\|p_{60}\| = 1.8 \times 10^{35}$) — parametre yöntem-özgüdür, ödünç alınamaz. NAG kendi standart seçimiyle ($s = 1/M = 1$, $\beta = (\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1) = 9/11$) 88 adımda yakınsar (empirik oran 0.9074): GD'den çok hızlı, momentum mertebesinde.*

Şekil 30.6 bu yöntem-özgülüğü gösterir: solda Polyak'ın optimal s'i NAG'a taşınınca norm patlar (turuncu, ıraksar) ama NAG kendi parametreleriyle düzgün iner (navy); sağda üç yöntem yan yana — GD 346, momentum 48, NAG 88 adımda $F/F_0 < 10^{-6}$ eşiğine ulaşır.



Şekil 30.6: Nesterov look-ahead: kendi parametreleriyle √κ sınıfı (88 adım, oran 0.907) — heavy-ball parametrelerini ödünç ALMA. Sol: Polyak'ın optimal s'i NAG'a taşınınca ıraksar (1.8e+35); NAG kendi seçimiyle (s=1, β=9/11) düşer. Sağ: GD 346, momentum 48, NAG 88 adımda F/F_0 < 1e-6.

> **Builder Notu — Önce Bak Sonra Adım At**
>
> Nesterov'un "önce bak, sonra adım at" hilesi momentum'dan biraz daha kararlıdır (overshoot'u düzeltir). ML köprüsü: Nesterov Accelerated Gradient (NAG) PyTorch'ta `nesterov=True` ile mevcut; teoride birinci-derece yöntemlerin yakınsama hızının alt sınırına ulaşır — daha hızlısı (sadece gradyanla) mümkün değil.

30.9 AdaGrad, Adam ve Modern Optimizer'lar

Momentum tek bir önceki adımı (z_{k-1}) hatırlar. Modern yöntemler **tüm** geçmiş değerleri basit bir formülle biriktirir:

> *"…it goes under the names adagrad, or others." — Strang, 41:36*

AdaGrad, RMSProp, Adam: her geçmiş gradyana ayrı katsayı vermeden (o devasa iş olurdu), üstel-ağırlıklı ortalamalarla geçmişi özetler. Adam = momentum (birinci moment, geçmiş gradyan ortalaması) + adaptif adım boyu (ikinci moment, geçmiş gradyan-kare ortalaması). Her koordinata kendi efektif learning rate'ini verir — kötü kondisyona karşı momentum'dan bir adım öteye.

> **Builder Notu — Momentum'dan Adam'a**
>
> Momentum → AdaGrad → RMSProp → Adam evrim çizgisi: hepsi "geçmiş gradyanları biriktir" fikrinin varyantları. ML köprüsü: Karpathy serisinde micrograd saf SGD ile başlar, nanoGPT ise AdamW kullanır — bu ders tam o geçişin teorisidir. Momentum (1. moment) zikzağı söndürür, RMSProp bileşeni (2. moment) her koordinatı ölçekleyip kondisyonu düzeltir; Adam bugün derin öğrenmenin (özellikle Transformer'ların) varsayılan optimizer'ıdır ve Strang'ın bu dersi tüm bu ailenin matematiksel kökenidir.

30.10 Bu Dersin Özeti

- **Zikzak sorunu:** kötü kondisyonlu (uzun-ince elips) problemde steepest descent yavaş; oran $(1-b)/(1+b) \approx 1$.
- **Momentum:** $z_k = \nabla F(x_k) + \beta z_{k-1}$, $x_{k+1} = x_k - s z_k$; önceki yönün hafızası (β).
- **Özdeğer analizi:** her özvektör → 2×2 matris R; s, β'yı R'nin max özdeğerini tüm $\lambda \in [m, M]$ için minimize edecek seç.
- **Optimal:** $s = (2/(\sqrt{M}+\sqrt{m}))^2$, $\beta = ((\sqrt{M}-\sqrt{m})/(\sqrt{M}+\sqrt{m}))^2$; yakınsama oranı $(\sqrt{M}-\sqrt{m})/(\sqrt{M}+\sqrt{m})$.
- **$\sqrt{\kappa}$ hızlanma:** kondisyon $\kappa = M/m$ yerine $\sqrt{\kappa}$; oran $(1-b)/(1+b) \to (1-\sqrt{b})/(1+\sqrt{b})$.
- **Nesterov:** gradyanı look-ahead noktada hesapla ($x_{k+1} = y_k - s\nabla F(y_k)$); optimal birinci-derece.
- **Modern:** AdaGrad/RMSProp/Adam tüm geçmişi üstel-ağırlıkla biriktirir (Adam = momentum + adaptif adım).

> **!** Tek Bir Cümle
>
> Momentum, adıma önceki yönün hafızasını (βz_{k-1}) ekleyerek zikzağı söndürür ve yakınsama oranını $(1-b)/(1+b)$'den $(1-\sqrt{b})/(1+\sqrt{b})$'ye — kondisyon sayısının karekökü kadar — iyileştirir; Nesterov bunu look-ahead gradyanla yapar, Adam ise momentumu adaptif adım boyuyla birleştirir.

30.11 Kontrol Soruları

> **i** Soru 1 — Momentum terimi ve sezgisi
>
> **Soru:** Momentum terimi gradient descent formülünü nasıl değiştirir ve sezgisi nedir?
> **Cevap:** $z_k = \nabla F(x_k) + \beta z_{k-1}$, $x_{k+1} = x_k - s z_k$. Arama yönü z_k, gradyana ek olarak önceki yönün β katını içerir (hafıza). Sezgi: yokuş aşağı yuvarlanan top — zikzak salınımlar (sağ-sol) birbirini götürür, vadi ekseni boyunca ivme (velocity) birikir. $\beta = 0$ ise sıradan steepest descent'e döner.

> **i** Soru 2 — Oran iyileşmesi ve kondisyon
>
> **Soru:** Momentum yakınsama oranını nasıl iyileştirir, kondisyon sayısıyla ilişkisi nedir?
> **Cevap:** Oran $(1-b)/(1+b)$'den $(1-\sqrt{b})/(1+\sqrt{b})$'ye düşer. Kondisyon sayısı $\kappa = M/m = 1/b$ cinsinden: momentumsuz hız $\sim \kappa$, momentumlu $\sim \sqrt{\kappa}$. Yani momentum efektif kondisyon sayısını **karekök**üne indirir. Büyük κ'da ($b \approx 0$) kazanç devasa: yakınsama için gereken adım sayısı $\sqrt{\kappa}$ ile orantılı olur (κ yerine).

> **i** Soru 3 — 2×2 matris R ve özvektörler
>
> **Soru:** Momentum analizinde 2×2 matris R nereden gelir, neden özvektörler kullanılır?
> **Cevap:** S simetrik olduğundan özvektörlerine ayrılır; her özvektör ayrı izlenince problem skalere iner (her λ için bağımsız). Momentum özyinelemesi (c_k, d_k) katsayıları için 2×2 matris R ile çarpıma dönüşür. Yakınsama R^k'ya, yani R'nin özdeğerlerine bağlıdır. s, β bu özdeğerleri tüm $\lambda \in [m, M]$ için minimize edecek seçilir — köşegenleştirme (Ders 4) problemi parçalara ayırır.

> **i** Soru 4 — Nesterov ile momentum farkı
>
> **Soru:** Nesterov ivmelendirmesi momentum'dan nasıl farklıdır?
> **Cevap:** Momentum gradyanı **mevcut** x_k noktasında hesaplar; Nesterov **look-ahead** y_k noktasında (momentum yönünde bir adım ileri) hesaplar: $x_{k+1} = y_k - s\nabla F(y_k)$. "Önce momentum yönünde bak, oradaki eğimi ölç, sonra adım at." Bu, momentum'un aşırı-atlama (overshoot) eğilimini frenler ve birinci-derece yöntemler için optimal yakınsamayı verir.

30.12 Egzersizler

1. **Oran karşılaştırması.** $b = 1/9$ için momentumsuz oran $(1-b)/(1+b)$ ve momentumlu $(1-\sqrt{b})/(1+\sqrt{b})$'yi hesapla. Hangisi kaç kat daha küçük? Yakınsama için kabaca kaç kat daha az adım? (Motor tanığı: GD 0.8, momentum 0.5; adım çarpanı $\log 0.5/\log 0.8 = 3.106\times$ — bkz. Şekil 30.7.)

2. **Momentum açılımı.** $z_k = \nabla F_k + \beta z_{k-1}$ özyinelemesini $z_0 = \nabla F_0$'dan başlayarak z_2'ye kadar aç. z_2'nin $\nabla F_2, \nabla F_1, \nabla F_0$ cinsinden (β ağırlıklı) ifadesini yaz. (Motor tanığı: $z_2 = \nabla F_2 + \beta\nabla F_1 + \beta^2\nabla F_0$, özyineleme = açık form, maxdiff $\sim 10^{-15}$.)

3. **Optimal β.** $M = 1$, $m = 0.04$ ($\kappa = 25$) için $\beta_{\text{opt}} = ((\sqrt{M}-\sqrt{m})/(\sqrt{M}+\sqrt{m}))^2$'yi hesapla. ($\sqrt{m} = 0.2$.) β yaklaşık kaç? Tipik derin öğrenme değeri 0.9 ile karşılaştır. (Motor tanığı: $\beta_{\text{opt}} = (0.8/1.2)^2 = 4/9 = 0.4444$; pratik sabit 0.9'dan farklı, optimal κ'ya bağlı.)

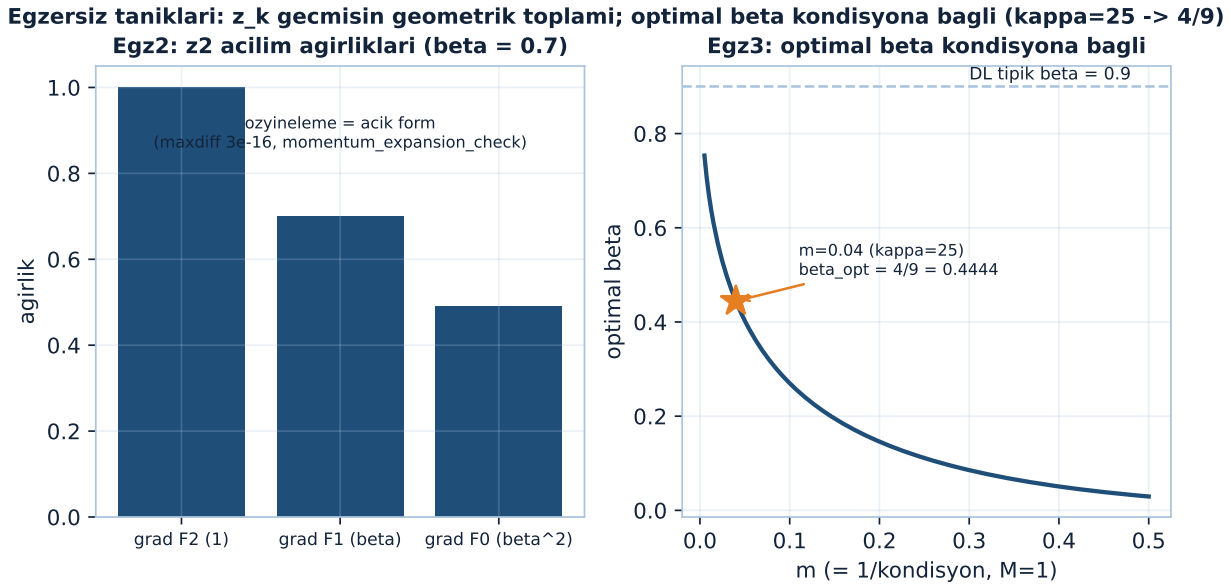
4. **β = 0 sınırı.** Momentum formülünde $\beta = 0$ alırsan ne olur? Yakınsama oranı $(\sqrt{M}-\sqrt{m})/(\sqrt{M}+\sqrt{m})$ hangi ifadeye döner? (İpucu: $\beta = 0$ momentum'u kapatır, iz birebir GD'ye döner, maxdiff 0; oran $(1-b)/(1+b)$'ye iner.)

5. **(Ders 24 habercisi)** Şimdiye kadar kısıtsız (veya basit kısıtlı) minimizasyon gördük. Peki amaç ve kısıtların **hepsi doğrusalsa**? Bir doğrusal hedefi doğrusal eşitsizlik kısıtları altında optimize etmek — köşelerde çözüm? Bir tahmin yaz — Ders 24 "lineer programlama ve iki-kişilik oyunlar" ile bu özel ama çok önemli sınıfı işliyor.

Şekil 30.7 iki egzersizi sayısal olarak doğrular: solda Egz2'nin açılım ağırlıkları $[1, \beta, \beta^2] = [1, 0.7, 0.49]$ özyinelemenin açık formla birebir eşleştiğini gösterir (maxdiff $\sim 10^{-16}$); sağda optimal $\beta(m)$ eğrisi kondisyona bağlıdır — $m = 0.04$ ($\kappa = 25$) noktasında $\beta_{\text{opt}} = 4/9 = 0.4444$ (turuncu yıldız), DL pratiğinin sabit $\beta = 0.9$'undan farklı.

30.13 Sonraki Ders İçin Hazırlık

Ders 24: Lineer Programlama ve İki-Kişilik Oyunlar. Optimizasyonun özel ama merkezî sınıfı: doğrusal hedef + doğrusal kısıtlar. Çözüm bir köşededir (simplex yöntemi köşeden köşeye yürür). Dualite (primal-dual), ve iki-kişilik sıfır-toplamlı oyunların minimax dengesi (Ders 19 maxmin bağı) — von Neumann'ın minimax teoremi.



Şekil 30.7: Egzersiz tanıkları — sol: Egz2'nin açılım ağırlıkları $[1, \beta, \beta^2] = [1, 0.7, 0.49]$ özyinelemenin açık formla birebir eşleştiğini gösterir (maxdiff 3e-16, `momentum_expansion_check`); sağ: optimal $\beta(m)$ eğrisi kondisyona bağlıdır — $m = 0.04$ ($\kappa = 25$) noktasında $\beta_{\text{opt}} = 4/9 = 0.4444$ (turuncu yıldız), DL pratiğinin sabit $\beta = 0.9$'undan farklı.

> **Hazırlık**
>
> Bu dersteki Şekil 30.3 sol panelindeki süzülen momentum yörüngesini zihninde tut: kısıtsız bir kuadratiği gradyanla (artı hafıza) minimize ettik ve oran $\sqrt{\kappa}$'ya indi. Ders 24 ise tamamen farklı bir geometriye geçer: hedef ve kısıtların hepsi doğrusalsa minimum içte değil bir **köşede** olur. Ders 19'un maxmin (Courant-Fischer) fikrini hatırla — iki-kişilik oyunların minimax dengesi oraya bağlanacak.

30.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Zikzak (momentumsuz)	oran $(1-b)/(1+b) \approx 1$, yavaş	2m09
Momentum	$z = \nabla F + \beta z_{-1}$; $x_+ = x - sz$	2m09
√b mucizesi	$(1-b)/(1+b) \to (1-\sqrt{b})/(1+\sqrt{b})$	10m56
Özvektör analizi	her $\lambda \to 2 \times 2$ R; max özdeğeri minimize et	9m36
Optimal s, β	$s = (2/(\sqrt{M}+\sqrt{m}))^2$, $\beta = ((\sqrt{M}-\sqrt{m})/(\sqrt{M}+\sqrt{m}))^2$	37m00
Kondisyon √κ	$\kappa \to \sqrt{\kappa}$; adım sayısı $\sim \sqrt{\kappa}$	30m50
Nesterov	look-ahead: $x_+ = y - s\nabla F(y)$	42m08
AdaGrad / Adam	tüm geçmişi üstel-ağırlıkla biriktir	41m36

30.15 ML Bağlantıları Özeti

- **SGD + momentum = standart:** derin öğrenmede neredeyse her optimizer momentum kullanır ($\beta \approx 0.9$); zikzağı söndürür, yakınsamayı $\sqrt{\kappa}$'ya hızlandırır.
- **Adam = momentum + adaptif:** 1. moment (momentum, geçmiş gradyan) + 2. moment (RMSProp, geçmiş gradyan-kare); koordinat-başı learning rate; Transformer'ların varsayılanı.
- **Nesterov (NAG):** look-ahead gradyan; birinci-derece yöntemlerin optimal yakınsama hızı (PyTorch `nesterov=True`).
- **Fizik sezgisi:** momentum = yokuş aşağı yuvarlanan top, hız biriktirir; salınımlar söner, ivme birikir.
- **$\sqrt{\kappa}$ teorisi:** momentum efektif kondisyonu kareköke indirir; batch norm κ'yı küçültür — ikisi birlikte derin ağ eğitimini mümkün kılar.
- **Köşegenleştirme gücü:** özvektör başına 2×2 analiz (Ders 4); Adam'ın koordinat-başı adımının teorik kökeni.
- **Geriye köprü:** Ders 22 (zikzak, kondisyon, oran), Ders 4 (özvektör/köşegenleştirme), Ders 10 (kondisyon sayısı κ), Ders 5 (pozitif tanımlı S).

! Kapanış

> *"…changes to 1 minus square root of b over 1 plus square root of b." — Strang, 10:56*

Momentum tek bir hafıza terimiyle kondisyon sayısını karekökünü indirir; bu küçük ekleme, gradient descent'i pratik bir algoritmadan derin öğrenmeyi mümkün kılan bir araca dönüştürür.

31 Lineer Programlama ve İki-Kişilik Oyunlar

Köşe çözümleri, max-flow min-cut dualitesi ve von Neumann minimax

> **Bölüm bilgisi**
>
> Bu ders optimizasyonun özel ama merkezî bir sınıfını işler: doğrusal hedef + doğrusal kısıtlar (lineer programlama). Anahtar kavram **dualite** — max-flow = min-cut ve iki-kişilik sıfır-toplamlı oyunların minimax dengesi hep aynı ilkedir. Strang'in Ders 24 videosu (≈53 dk) ve OCW Lecture 24 temel alınmıştır. Okuma süresi ≈34 dk; önkoşul Ders 23 (gradient descent, momentum, kondisyon).

31.1 Bu Derste Ne Var?

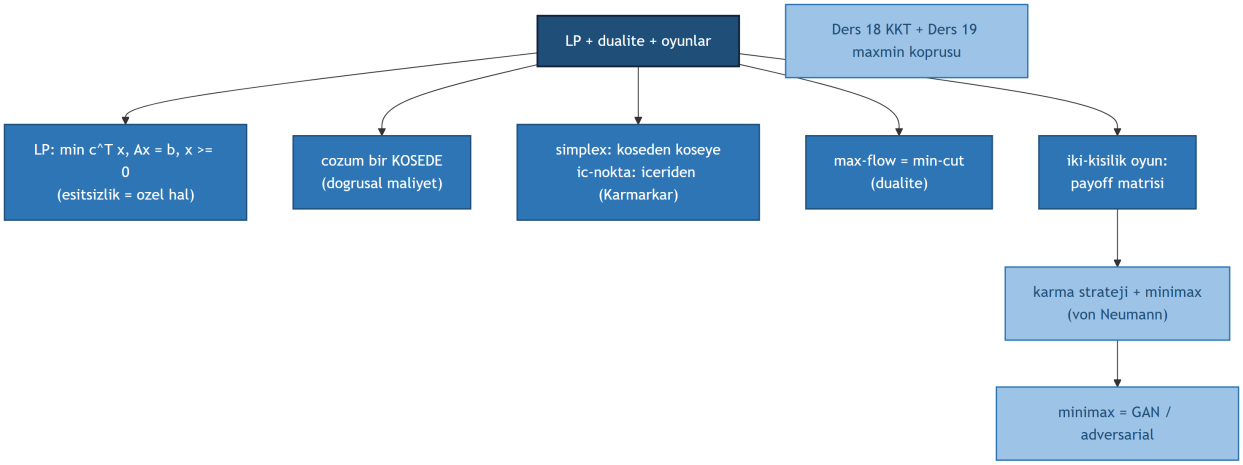
Bu derste beş sonuç var:

1. **LP:** min $c^T x$, kısıt $Ax = b$ ve $x \geq 0$ (eşitsizlik kısıtı LP'yi "özel" yapar). Çözüm bir **köşe**dedir.
2. **Algoritmalar:** simplex (Dantzig, köşeden köşeye) ve iç-nokta (Karmarkar, feasible set içinden).
3. **Max-flow min-cut:** ağın maksimum akışı = minimum kesim kapasitesi — klasik dualite örneği.
4. **Dualite:** her akış ≤ her kesim; maksimum = minimum olduğunda optimal.
5. **İki-kişilik oyun:** payoff matrisi; x satır seçer, y sütun; saddle point yoksa **karma strateji** (mixed strategy) + minimax.

> *"Linear programming is a very famous part of optimization." — Strang, 1:57*

Şekil 31.1 dersin iskeletini gösterir: merkezdeki "LP + dualite + oyunlar" fikri, eşitsizlik kısıtının LP'yi özel kıldığı doğrusal hedeften, doğrusal maliyetin çözümü bir köşeye ittiği geometriden, simplex (köşeden köşeye) ile iç-nokta (Karmarkar) algoritma ikiliğinden, max-flow = min-cut dualitesinden ve payoff matrisli iki-kişilik oyundan dallanır; ayrı düğümlerde karma strateji + minimax (von Neumann), minimax = GAN/adversarial köprüsü ve Ders 18 (KKT) + Ders 19 (maxmin) bağı durur.

> **Builder Notu — Her Min'in Bir Maks İkizi**
>
> - **Dualite = optimizasyonun kalbi** — her minimizasyon probleminin bir maksimizasyon ikizi vardır; ikisi eşitse optimal kanıtlanmıştır (max-flow min-cut, primal-dual).
> - **LP ∈ P** — polinom zamanda çözülür (ellipsoid, Khachiyan); simplex ortalama-durumda hızlı (Spielman smoothed analysis).
> - **Minimax = GAN** — iki-kişilik sıfır-toplamlı oyun; üretici-ayırıcı (Ders 23) ve adversarial eğitim aynı minimax yapısı.



Şekil 31.1: Ders 24 kavram haritasi — lineer programlama, max-flow min-cut dualitesi ve iki-kisilik oyunlar tek bir minimax cercevesinde bulusur.

- **Geriye köprü:** Ders 18 (KKT, Lagrange kısıtlı optimizasyon), Ders 19 (maxmin/minimax — özdeğer versiyonu), Ders 21 (konvekslik, feasible set). Paralel: 6.006 Ders 16 (ağırlıklı en kısa yol — LP dualitesi).

31.2 Lineer Programlama Nedir?

Optimizasyonun en ünlü özel sınıfı:

> *"Linear programming is a very famous part of optimization." — Strang, 1:57*

Doğrusal bir maliyet fonksiyonunu, doğrusal kısıtlar altında minimize et:

$$\min\ c^T x \quad \text{s.t.} \quad Ax = b, \quad x \geq 0$$

x bilinmeyen ($n \times 1$), c maliyet vektörü, A ise $m \times n$ ($m < n$, kare-tersinir DEĞİL). Maliyet doğrusal, eşitlik kısıtları doğrusal — ama $x \geq 0$ (vektör eşitsizliği: her $x_i \geq 0$) işi doğrusal-olmayan, "özel" yapar. Bu iki kısıt birlikte **uygunluk kümesi** (feasible set) K'yi tanımlar.

Builder Notu — Eşitsizlik Köşe Yaratır

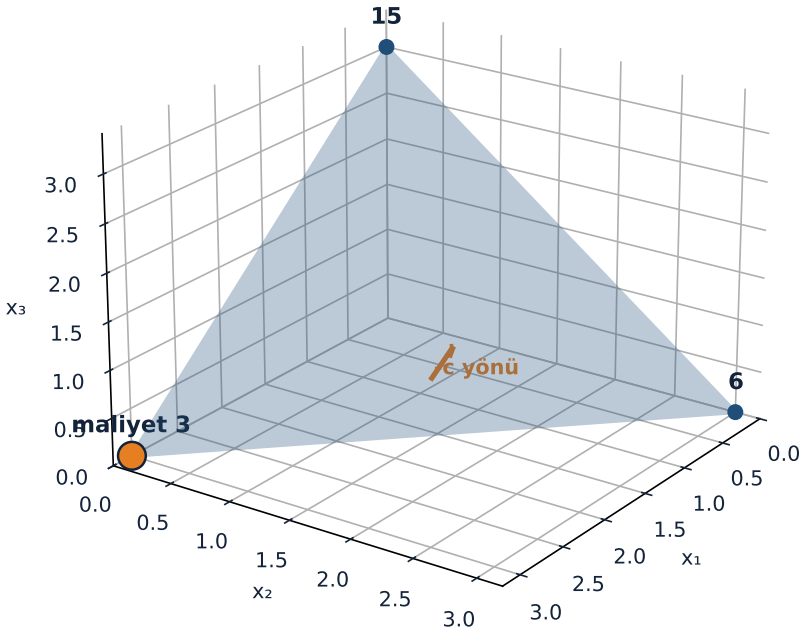
"$x \geq 0$ eşitsizliği LP'yi özel kılar" — eşitsizlik kısıtları çözümü kısıt bölgesinin sınırına/köşesine iter. ML köprüsü: negatif-olmama kısıtı her yerde (olasılıklar, ağırlıklar, kaynaklar ≥ 0); LP, kaynak tahsisi, üretim planlama ve (dual formda) SVM gibi problemlerin temelidir.

31.3 Geometri: Çözüm Bir Köşededir

$n = 3$'te görselleştir. $x \geq 0$ → uzayın 1/8'i (oktant). Tek kısıt ($Ax = b$) bir düzlem; oktantla kesişimi bir **üçgen**. Örnek: min $x_1 + 2x_2 + 5x_3$, kısıt $x_1 + x_2 + x_3 = 3$, $x \geq 0$. Düzlem eksenleri $(3, 0, 0)$, $(0, 3, 0)$, $(0, 0, 3)$'te keser. Maliyet doğrusal olduğundan minimum bir **köşede**dir:

> *"...one of these three corners is the winner." — Strang, 8:15*

Üç köşenin maliyeti: $(3, 0, 0) \rightarrow 3$, $(0, 3, 0) \rightarrow 6$, $(0, 0, 3) \rightarrow 15$. Kazanan $x^* = (3, 0, 0)$, maliyet 3. Doğrusal fonksiyon uçlarda (köşelerde) ekstremumdadır — iç noktada değil.



Şekil 31.2: Feasible set = oktant ∩ duzlem ($x_1 + x_2 + x_3 = 3$) = üçgen. Doğrusal maliyet köşeye kaçar: köşe maliyetleri (3,0,0)→3, (0,3,0)→6, (0,0,3)→15; kazanan köşe (3,0,0), maliyet 3 (linprog çözümü birebir köşede).

Şekil 31.2 LP'nin temel teoremini görselleştirir: feasible set, oktant ($x \geq 0$) ile düzlemin ($x_1 + x_2 + x_3 = 3$) kesişimi olan bir üçgendir; üç köşenin maliyeti $3/6/15$ ve doğrusal maliyet köşeye kaçtığı için kazanan köşe $(3, 0, 0)$ (turuncu marker), minimum maliyet 3 — linprog çözümü bu köşeyle birebir aynı çıkar.

> **Builder Notu — Doğrusal Maliyet Uçlara Kaçar**
>
> "Doğrusal maliyet → köşede optimal" LP'nin temel teoremi: optimum daima bir köşede (vertex) bulunur. ML köprüsü: bu, LASSO'nun (ℓ^1 regülarizasyon, Ders 8/16) neden seyrek çözüm verdiğini açıklar — ℓ^1 topu köşelidir, optimum köşeye (çok sıfırlı) düşer. Köşe = seyreklik geometrisi.

31.4 Algoritmalar: Simplex ve İç-Nokta

Köşeyi bulmak yeter — ama büyük m, n'de **üstel** sayıda köşe var, hepsini sayamayız. İki algoritma ailesi:

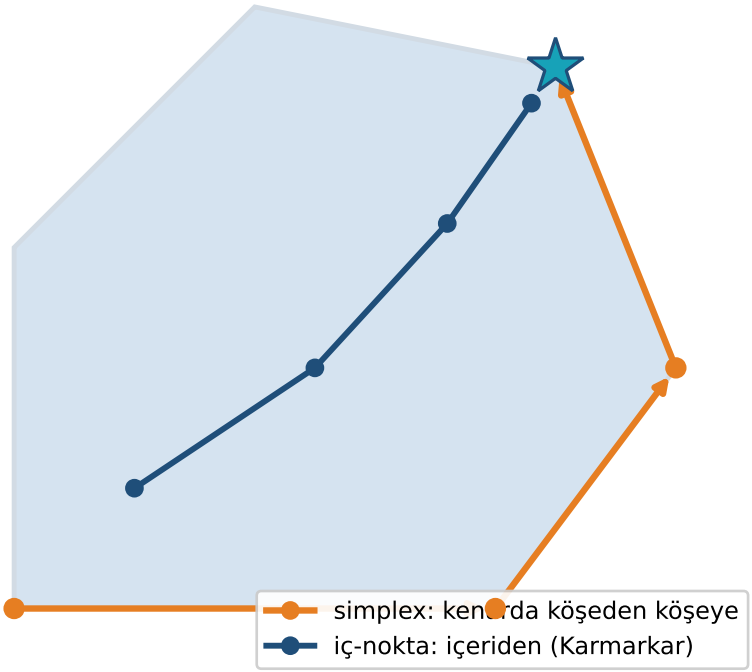
Simplex (Dantzig): bir köşeden başla, maliyeti düşüren bir kenar boyunca yürü, sonraki köşeye var, tekrarla — kenarlar üzerinde steepest descent.

> *"…the simplex method, which finds a corner." — Strang, 10:45*

İç-nokta (interior point, Karmarkar): kenarlardan değil, uygunluk kümesinin **içinden** geç; bir arama yönü seç, sınıra çarpmadan dur, Newton/kalkülüsle minimize et.

> *"…interior point method." — Strang, 13:46*

Karmarkar'ın 1984'teki yöntemi (New York Times'a çıkacak kadar ses getirdi) iç-nokta fikrini popülerleştirdi. Karmaşıklık: simplex en-kötü durumda üstel ama **ortalama-durumda polinom** (Spielman smoothed analysis); ayrıca LP'nin **P** sınıfında olduğu kanıtlandı (ellipsoid yöntemi, Khachiyan — ünlü Rus sonucu).



Şekil 31.3: İki strateji aynı köşeye varır. Şematik politopun kenarı boyunca simplex (turuncu oklar) köşeden köşeye sıçrar — Dantzig'in yöntemi. İç-nokta yolu (navy) ise politopun içinden kıvrılarak aynı optimum köşeye (teal yıldız) yaklaşır — Karmarkar'ın 1984 yaklaşımı. Aynı çözüm, iki ayrı geometri.

Şekil 31.3 iki algoritma stratejisini aynı politopta karşılaştırır: simplex (turuncu oklar) feasible kümenin kenarı boyunca köşeden köşeye sıçrar (Dantzig), iç-nokta yolu (navy) ise politopun içinden kıvrılarak aynı optimum köşeye (teal yıldız) yaklaşır (Karmarkar, 1984) — aynı çözüm, iki ayrı geometri.

> Builder Notu — Kenardan mı İçinden mi
>
> Simplex (kenar) vs iç-nokta (iç) ikiliği, optimizasyonun iki büyük stratejisini temsil eder. ML köprüsü: simplex'in "kenarda steepest descent"i, gradient descent'in (Ders 22) kısıtlı versiyonu; iç-nokta yöntemleri Newton'ı (Ders 21) kısıt bölgesi içinde kullanır. Modern büyük-ölçek LP/QP çözücüleri (SVM eğitimi dahil) iç-nokta tabanlıdır.

31.5 Max-Flow Problemi

LP'nin en güzel örneği: bir ağda kaynaktan (source) hedefe (sink) **maksimum akış** gönder. Her kenarın bir **kapasitesi** var; o kenardaki akış kapasiteyi aşamaz:

$$\max \left(\text{sink akisi}\right) \quad \text{s.t.} \quad 0 \leq \left(\text{kenar akisi}\right) \leq \left(\text{kapasite}\right)$$

Bu bir tamsayı (integer) LP'dir ama dikkat çekici bir özelliği var: kesirli akışa izin versen bile **optimal akış tamsayı** çıkar — yani tamsayı kısıtını "güvenle gevşetebilirsin" (LP rahatlatması), simplex/iç-nokta uygulayabilirsin. Bu, max-flow'u verimli çözülebilir kılar.

Şekil 31.4 motor ağında max-flow = min-cut = 14 sonucunu çizer; aşağıda dualite bölümünde optimal akışların tamsayı çıktığını ve kesim kümelerinin akışı nasıl sınırladığını ayrıntılandıracağız.

> Builder Notu — Tamsayı Bedava Çıkar
>
> Max-flow'un "LP optimumu tamsayı çıkar" özelliği (tümleşik-modülerlik, total unimodularity) nadir ve değerlidir: genelde tamsayı programlama NP-zordur, ama max-flow polinomdur. ML köprüsü: graf-kesim (graph cut) görüntü segmentasyonu, ikili etiketleme (binary labeling) ve enerji minimizasyonu max-flow'a indirgenir — bilgisayarlı görüde standart araç.

31.6 Dualite: Max-Flow = Min-Cut

Bir akışın en fazla ne kadar olabileceğini nasıl sınırlarız? **Kesim** (cut) ile: ağı kaynak-tarafı ve hedef-tarafı düğümlere ayır; tüm akış bu kesimi geçmek zorunda. Kesimin kapasitesi = onu geçen kenarların toplamı. Her akış ≤ her kesim kapasitesi.

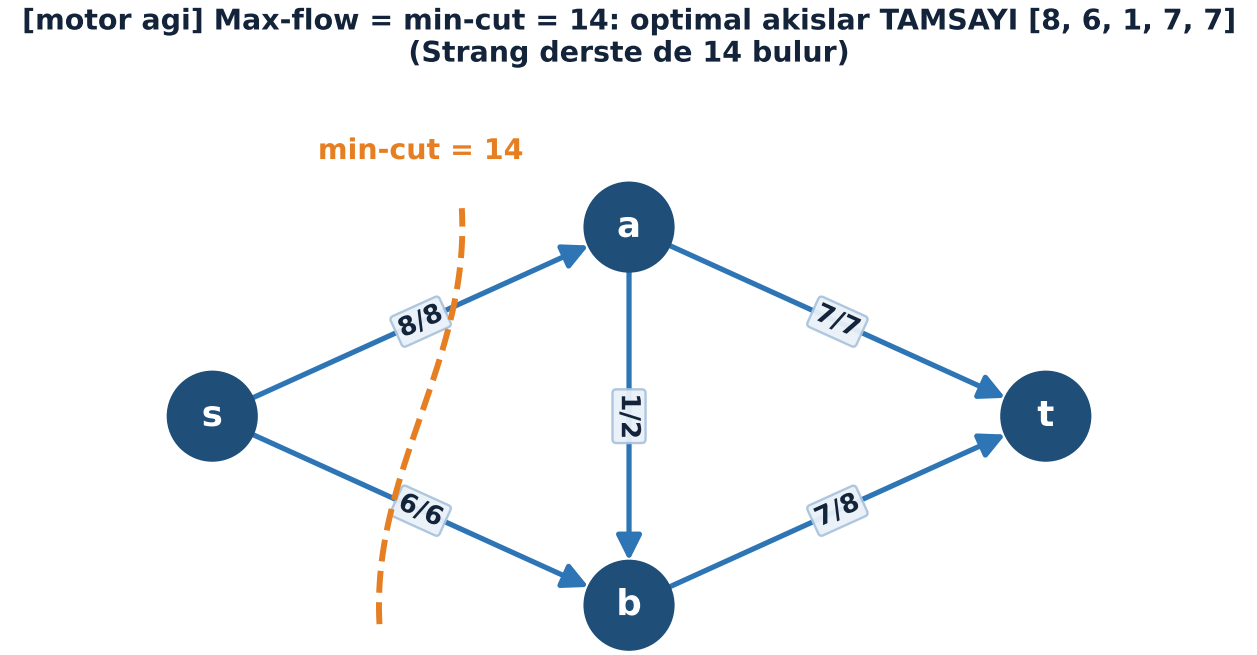
> *"…that's what duality is, that any flow has [to be ≤ the capacity of any cut]." — Strang, 37:32*

Maksimum akış ile minimum kesim **çakıştığında** optimal kanıtlanır:

> *"The maximum flow turned out to be 14[, and the minimum cut turned out to be 14]." — Strang, 34:45*

$$\max \text{ flow} = \min \text{ cut}$$

İşte dualite: bir minimizasyon (en küçük kesim) ile bir maksimizasyon (en büyük akış) aynı sayıya varır. Akış 14'ü geçebiliyorsam ve hiçbir kesim 14'ün altında değilse, 14 optimaldir. Büyük ağda min-cut görünmez ama hızlı bulunabilir.

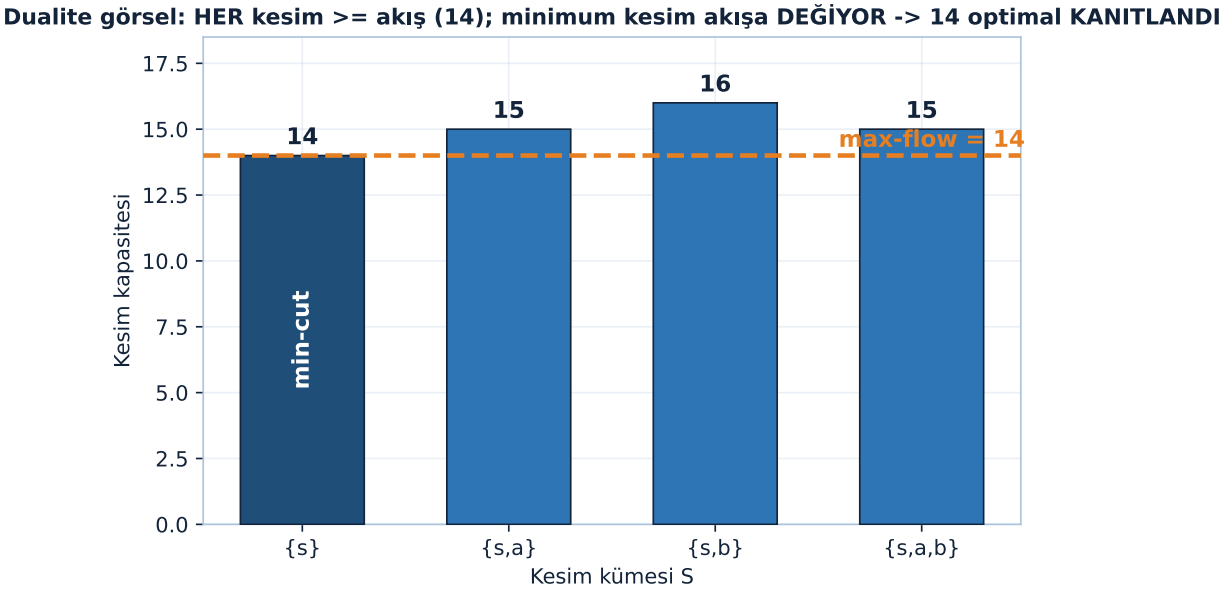


Şekil 31.4: [motor ağı] Max-flow = min-cut = 14: optimal akışlar TAMSAYI [8, 6, 1, 7, 7], kesim {s} en dar boğaz (Strang derste de 14 bulur). Her kenarda "akış/kapasite" etiketi; turuncu kesik eğri {s} kesimini ayırır.

Şekil 31.4 dualiteyi somut bir ağda gösterir: kaynak s'ten hedef t'ye giden LP-optimal akışlar tamsayı $[8, 6, 1, 7, 7]$ çıkar, her kenarda "akış/kapasite" etiketi durur, turuncu kesik eğri $\{s\}$ kesimini ayırır ve max-flow = min-cut = 14. Bu motor ağıdır — Strang'ın derste çizdiği ağ farklı olabilir, ama o da aynı 14 değerine varır; dualite mesajı birebir aynıdır.

Şekil 31.5 dualitenin kanıt mantığını çubuklarla gösterir: dört s-t kesiminin kapasiteleri $\{s\} \to 14$, $\{s, a\} \to 15$, $\{s, b\} \to 16$, $\{s, a, b\} \to 15$; hiçbir kesim akışın (yatay turuncu çizgi, 14) altına inmez ve minimum kesim ($\{s\}$, navy çubuk) akışa tam değer — bu yüzden 14 hem maksimum akış hem minimum kesimdir, optimal kanıtlanmıştır.

> **Builder Notu — En Dar Boğaz Kanıttır**
>
> Max-flow min-cut, dualitenin en görsel örneği: "ne kadar gönderebilirim" (primal, maks) = "en dar boğaz" (dual, min). ML köprüsü: dualite optimizasyonun her yerinde — SVM'nin primal (margin) ve dual (destek vektörleri) formları, Lagrange dualitesi (Ders 18 KKT). 6.006 Ders 16'nın ağırlıklı en kısa yol problemi de bir LP dualitesi örneğidir (Phase 2 paralel ders köprüsü).



Şekil 31.5: Dualite görsel: HER kesim >= akış (14); minimum kesim akışa DEĞİYOR -> 14 optimal KANITLANDI

31.7 İki-Kişilik Sıfır-Toplamlı Oyunlar

Aynı dualite oyun teorisinde de var:

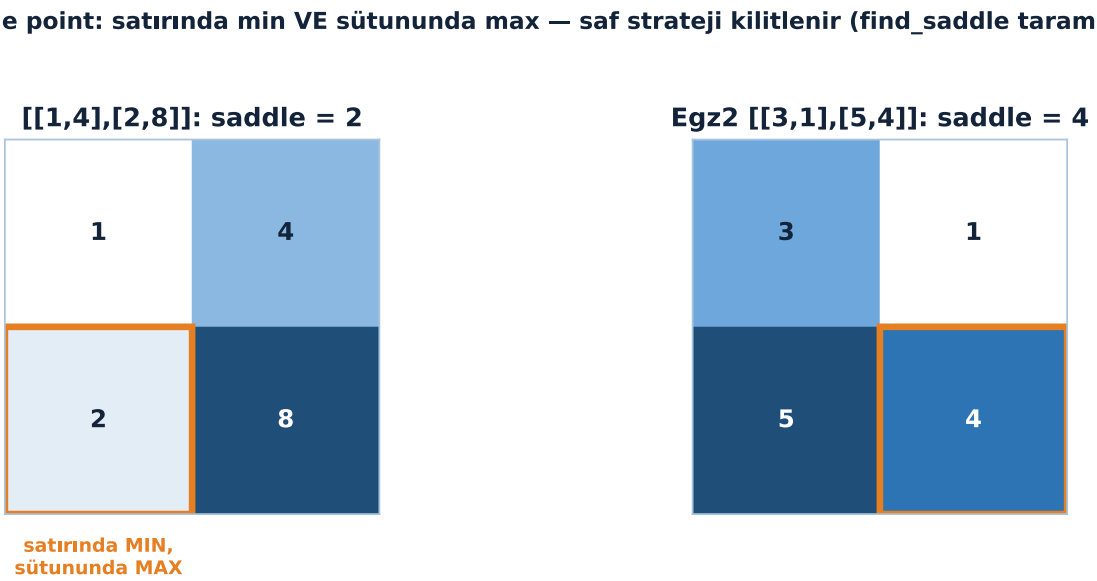
> *"…two-person games…" — Strang, 2:41*

İki oyuncu x ve y, bir **payoff (ödeme) matrisi** üzerinde. x bir **satır** seçer, y bir **sütun**. Ödeme x'ten y'ye akar — sıfır-toplamlı (x'in ödediği y'ye gider, üçüncü taraf yok). x **minimize** eder (ödeyen), y **maksimize** eder (alan) — tıpkı LP'nin primal/dual'i gibi. Basit örnek (payoff $[[1,4],[2,8]]$): y büyük sayı ister $\rightarrow$ 2. sütun; x küçük ister $\rightarrow$ 1. satır; sonuç 2'de kilitlenir. Bu bir **saddle point**: y'nin sütununda min, x'in satırında max.

Şekil 31.6 saddle point'i iki payoff matrisinde gösterir: solda $[[1,4],[2,8]]$ için saddle hücresi 2 (turuncu çerçeve — satırında min, sütununda max), sağda Egzersiz 2'nin $[[3,1],[5,4]]$ matrisinde saddle 4; `find_saddle` taraması bu hücreleri otomatik bulur.

[motor notu]: Rol adlandırması transkriptte gevşek; Notion'un "x küçük ister $\rightarrow$ 1. satır, sonuç 2" akışı kendi sayılarıyla birebir tutarlı değil ((1. satır, 2. sütun) = 4 olurdu). Sayılar (saddle 2, $q = 2/3$, değer $10/3$) klasik SATIR-ALIR (maximin) / SÜTUN-ÖDER (minimax) konvansiyonunda motor-tanıklıdır: saddle = (satır 2, sütun 1) = 2 satırında min ve sütununda max'tır (`find_saddle`), ve `game_value_lp` iki yönden $10/3 = 10/3$ verir.

> **Builder Notu — Oyunun Primal'i Dual'i**
>
> "x minimize (satır), y maksimize (sütun)" oyun yapısı LP primal-dual'in birebir karşılığı. ML köprüsü: GAN'lar (Ders 23) tam iki-kişilik oyun — üretici (generator) kayıbı minimize, ayırıcı (discriminator) maksimize; eğitim bu oyunun dengesini (Nash/saddle) arar. Adversarial robustness de min-maks bir oyun.



Şekil 31.6: Saddle point: satırında min VE sütununda max — saf strateji kilitlenir (find_saddle taraması). Sol [[1,4],[2,8]]: saddle = 2; sağ Egz2 [[3,1],[5,4]]: saddle = 4.

31.8 Karma Strateji ve Minimax

Her matriste saddle point yoktur. Payoff $[[1,4],[8,2]]$: y 8'i sever (2. sütun), ama x 2. satırı seçince y 4'ü görüp 1. sütuna kayar — saf strateji kilitlenmez. Çözüm **karma strateji** (mixed strategy):

> *"Mixed strategy..." — Strang, 43:00*

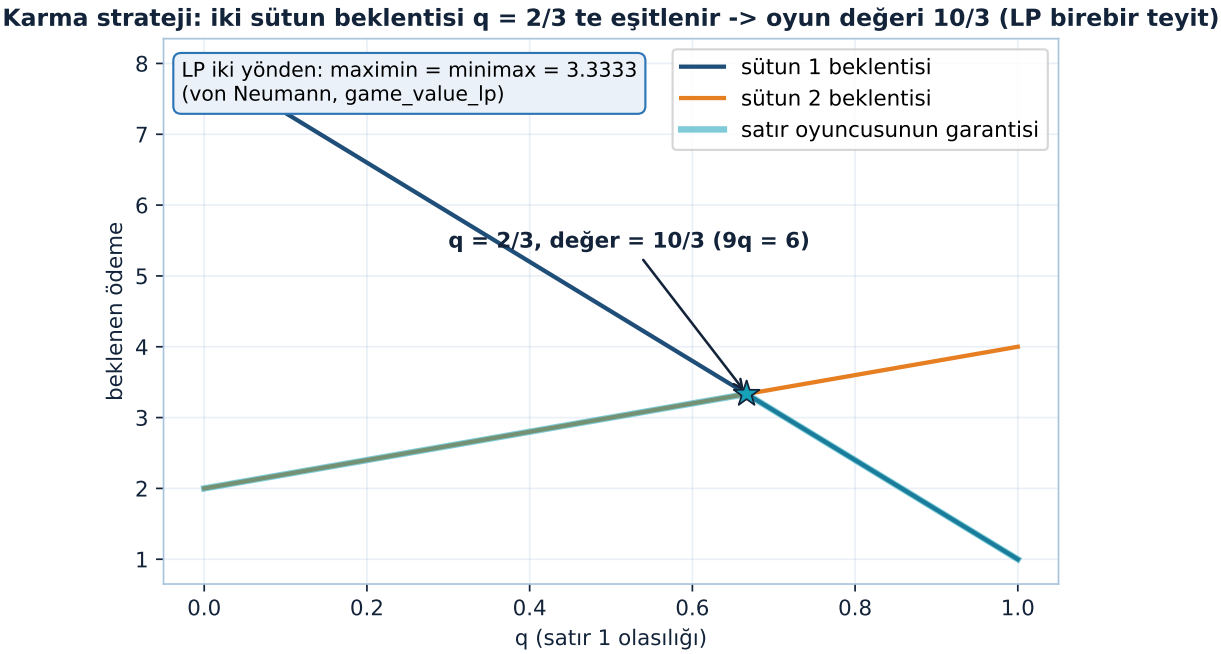
y sütunları p ve $1-p$ olasılıkla, x satırları q ve $1-q$ ile karıştırır. Optimal nokta, karışık seçeneklerin **eşit** olduğu yerdir (rakip avantaj bulamasın). Örnekte $9p = 6 \to p = 2/3$; oyunun değeri $10/3$. Bu, **minimax teoremi** (von Neumann): en iyi karma stratejiyle

$$\max_y \min_x (\text{payoff}) = \min_x \max_y (\text{payoff})$$

Maks-min = min-maks. Bu eşitlik tam olarak LP dualitesidir — iki-kişilik sıfır-toplamlı oyun bir LP'ye eşdeğerdir.

Şekil 31.7 karma stratejiyi klasik oyun grafiğiyle gösterir: $[[1,4],[8,2]]$ için iki sütun beklentisi $q\cdot 1+(1-q)\cdot 8$ (navy) ve $q\cdot 4+(1-q)\cdot 2$ (turuncu) $q = 2/3$ noktasında eşitlenir; satır oyuncusunun garantisi (alt zarf, teal) tam orada maksimum olur ve oyun değeri $10/3$ (teal yıldız) — $9q = 6$ denkleminin çözümü. LP iki yönden maximin = minimax = 3.3333 verir (`game_value_lp`), von Neumann minimaxı birebir teyitlenir.

> **Builder Notu — von Neumann'ın Dengesi**
>
> von Neumann'ın minimax teoremi (maks-min = min-maks) oyun teorisinin kurucu sonucu ve LP dualitesinin ikizidir. ML köprüsü: bu, Ders 19'un eigenvalue maxmin'inin (Courant-Fischer) oyun-teorik kuzenidir; GAN eğitimi, robust optimizasyon ve adversarial örnekler hep bu min-maks dengesini



Şekil 31.7: Karma strateji: iki sütun beklentisi q = 2/3 te eşitlenir -> oyun değeri 10/3 (LP birebir teyit)

çözmeye çalışır. Karma strateji = olasılıksal politika (stochastic policy), pekiştirmeli öğrenmenin temeli.

31.9 Bu Dersin Özeti

- **LP:** min $c^T x$, kısıt $Ax = b$, $x \geq 0$. Eşitsizlik kısıtı LP'yi özel yapar; feasible set K.
- **Köşe çözümü:** doğrusal maliyet → optimum bir köşede (vertex). Örnek min $x_1 + 2x_2 + 5x_3$, $x_1 + x_2 + x_3 = 3$ → $(3, 0, 0)$, maliyet 3.
- **Algoritmalar:** simplex (Dantzig, köşeden köşeye) + iç-nokta (Karmarkar, içeriden). LP $\in$ P (ellipsoid).
- **Max-flow min-cut:** kaynak→hedef maksimum akış = minimum kesim kapasitesi; tamsayı optimum (gevşetilebilir).
- **Dualite:** her akış ≤ her kesim; maks = min optimal. Min (kesim) = maks (akış).
- **İki-kişilik oyun:** payoff matrisi; x satır (min), y sütun (max). Saddle yoksa karma strateji.
- **Minimax (von Neumann):** $\max_x \min_y = \min_y \max_x$; iki-kişilik sıfır-toplamlı oyun = LP dualitesi.

! Tek Bir Cümle

Lineer programlama doğrusal bir hedefi ($c^T x$) doğrusal kısıtlar ($Ax = b$, $x \geq 0$) altında bir köşede minimize eder; dualite (max-flow = min-cut, primal-dual) optimali kanıtlar ve iki-kişilik sıfır-toplamlı oyunların minimax dengesi (maxmin = minmax) bu LP dualitesinin ta kendisidir.

31.10 Kontrol Soruları

Soru 1 — Köşe çözümü ve özel kısıt

Soru: Lineer programlamada çözüm neden bir köşede bulunur, ve hangi kısıt LP'yi "doğrusal-olmayan" yapar?
Cevap: Maliyet $c^T x$ doğrusaldır; doğrusal bir fonksiyon bir konveks bölge (politop) üzerinde ekstremumunu daima bir **köşede** (vertex) alır — iç noktada değil. $x \geq 0$ eşitsizlik kısıtı LP'yi "özel/doğrusal-olmayan" yapar: çözümü kısıt bölgesinin köşesine iter (eşitlik kısıtları tek başına bir düzlem verir, eşitsizlik onu köşeli politopa böler).

Soru 2 — Simplex ve iç-nokta farkı

Soru: Simplex ve iç-nokta yöntemleri nasıl farklıdır?
Cevap: Simplex (Dantzig) uygunluk politopunun **kenarları** üzerinde yürür: bir köşeden başlar, maliyeti düşüren kenar boyunca sonraki köşeye gider (kenarda steepest descent). **İç-nokta** (Karmarkar) politopun **içinden** geçer: arama yönü seçer, sınıra çarpmadan durur, Newton/kalkülüsle ilerler. Simplex en-kötü üstel ama ortalama polinom; iç-nokta polinom garantili.

Soru 3 — Max-flow min-cut dualitesi

Soru: Max-flow min-cut dualitesi ne der ve neden optimali kanıtlar?
Cevap: Maksimum akış (kaynak→hedef) = minimum kesim kapasitesi. Her akış, ağı ikiye bölen herhangi bir kesimi geçmek zorundadır, dolayısıyla **akış ≤ her kesim kapasitesi**. Bir akış değeri 14'e ulaşıyorsa ve hiçbir kesim 14'ün altında değilse, 14 optimaldir — maks (akış) ile min (kesim) çakıştığında ikisi de kanıtlanmış olur. Bu dualitedir.

Soru 4 — Saddle yoksa minimax

Soru: İki-kişilik sıfır-toplamlı oyunda saddle point yoksa ne yapılır, minimax teoremi ne der?
Cevap: Saf strateji kilitlenmiyorsa **karma strateji** (mixed strategy): oyuncular satır/sütunları olasılıklarla (p, q) karıştırır; optimal nokta karışık seçeneklerin eşit olduğu yerdir. von Neumann'ın **minimax teoremi**: en iyi karma stratejiyle $\max_y \min_x(\text{payoff}) = \min_x \max_y(\text{payoff})$. Bu maks-min = min-maks eşitliği, iki-kişilik oyunun bir LP'ye eşdeğer olduğunu (LP dualitesi) gösterir.

31.11 Egzersizler

1. **Köşe maliyeti.** $\min 4x_1 + x_2 + 3x_3$, kısıt $x_1 + x_2 + x_3 = 6$, $x \geq 0$. Üç köşeyi $(6,0,0)$, $(0,6,0)$, $(0,0,6)$ ve maliyetlerini yaz. Kazanan köşe ve minimum maliyet nedir? (Motor tanığı: maliyetler $24/6/18$; kazanan $(0,6,0)$, maliyet 6; linprog köşeyle birebir.)

2. **Saddle point.** Payoff matrisi $[[3,1],[5,4]]$. y sütun (max), x satır (min) seçer. Bir saddle point var mı? Varsa hangi (satır, sütun) ve oyunun değeri kaç? (Motor tanığı: saddle (satır 2, sütun 2) = 4 — satırında min, sütununda max; bkz. Şekil 31.6 sağ panel.)

3. **Karma strateji.** Payoff $[[1,4],[8,2]]$ için y'nin karma stratejisi p (1. sütun), $1-p$ (2. sütun). x'in iki satır beklentisini eşitle: $p+8(1-p) = 4p+2(1-p)$. p'yi ve oyun değerini bul (Strang'ın $2/3$, $10/3$ sonucuyla karşılaştır). (Motor tanığı: $q = 2/3$, değer $10/3$; `game_value_lp` maximin = minimax = 3.333333 birebir.)

4. **Dualite mantığı.** Bir ağda bir akış değeri 12 buldun ve kapasitesi 12 olan bir kesim gördün. Bu, 12'nin maksimum akış olduğunu kanıtlar mı? Neden? (İpucu: akış ≤ her kesim — bir akış bir kesim kapasitesine değiyorsa, ikisi de optimaldir.)

5. **(Ders 25 habercisi)** Şimdiye kadar gradyanı tüm veriyle (full batch) hesapladık. Peki milyonlarca eğitim örneği varsa her adımda hepsini kullanmak çok pahalıysa? Rastgele küçük bir alt-küme (mini-batch) yeter mi? Bir tahmin yaz — Ders 25 "stochastic gradient descent" (konuk Prof. Sra) ile derin öğrenmenin asıl çalışan algoritmasını işliyor.

31.12 Sonraki Ders İçin Hazırlık

Ders 25: Stochastic Gradient Descent (konuk: Suvrit Sra). Derin öğrenmenin kritik algoritması: gradyanı tüm veriyle değil, rastgele bir **mini-batch** ile tahmin et. Hem hesaplama olarak ucuz hem de beklenen-değer (olasılıksal) problemleri çözebilir. Konuk Prof. Sra anlatır; quote'lar "— Sra, X:XX".

> ⚠ Hazırlık
>
> Ders 25 dersi **konuk Prof. Suvrit Sra** verir — quote'lar Strang'a değil Sra'ya atfedilecek ("— Sra, X:XX"). Bu dersteki Egzersiz 5'in sorduğu soruyu zihninde tut: milyonlarca örnekte tüm veriyle gradyan hesaplamak pahalıysa, rastgele bir mini-batch yeterli mi? Sra tam bu soruyu işleyecek; gradient descent'i (Ders 22-23) "stochastic" hale getirmek hem hesaplamayı ucuzlatır hem de beklenen-değer problemlerini çözer.

31.13 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
LP nedir	$\min c^T x$, $Ax = b$, $x \geq 0$	1m57
Köşe çözümü	doğrusal maliyet → optimum köşede (vertex)	8m15
Simplex (Dantzig)	köşeden köşeye, kenarda steepest descent	10m45
İç-nokta (Karmarkar)	feasible set içinden, Newton	13m46
Max-flow	kaynak→hedef max akış; akış ≤ kapasite	34m45
Dualite	max-flow = min-cut; akış ≤ her kesim	37m32
İki-kişilik oyun	payoff; x satır (min), y sütun (max)	2m41
Karma strateji / minimax	$\max_x \min_y = \min_y \max_x$ (von Neumann)	43m00

31.14 ML Bağlantıları Özeti

- **Dualite her yerde:** SVM primal (margin) / dual (destek vektörleri); Lagrange dualitesi (Ders 18 KKT); max-flow min-cut.
- **Graph cut / segmentasyon:** max-flow min-cut görüntü segmentasyonu ve enerji minimizasyonunda (binary labeling) standart araç.
- **Minimax = GAN:** iki-kişilik sıfır-toplamlı oyun; üretici-ayırıcı (Ders 23), adversarial robustness, pekiştirmeli öğrenme (karma strateji = stokastik politika).
- **Köşe = seyreklik:** doğrusal-maliyet köşe çözümü → LASSO'nun ℓ^1 seyrekliği (Ders 8/16); ℓ^1 topu köşeli.
- **LP $\in$ P:** ellipsoid (Khachiyan) polinom; iç-nokta yöntemleri büyük-ölçek LP/QP (SVM eğitimi) çözücülerinin temeli.
- **Paralel kurs:** 6.006 Ders 16 ağırlıklı en kısa yol = bir LP dualitesi örneği (Phase 2 köprü).
- **Geriye köprü:** Ders 18 (KKT, Lagrange), Ders 19 (maxmin/minimax — özdeğer versiyonu), Ders 21 (konvekslik, feasible set), Ders 22 (steepest descent — simplex kenarda).

! Kapanış

> *"…that's what duality is, that any flow has [to be ≤ the capacity of any cut]." — Strang, 37:32*

Dualite optimizasyonun kalbi: her minimizasyonun bir maksimizasyon ikizi vardır; ikisi eşitse optimal kanıtlanır — max-flow = min-cut, LP primal-dual ve oyunların minimax dengesi hep aynı ilke.

32 Stochastic Gradient Descent

Konuk ders, Suvrit Sra: finite-sum, confusion region ve mini-batch

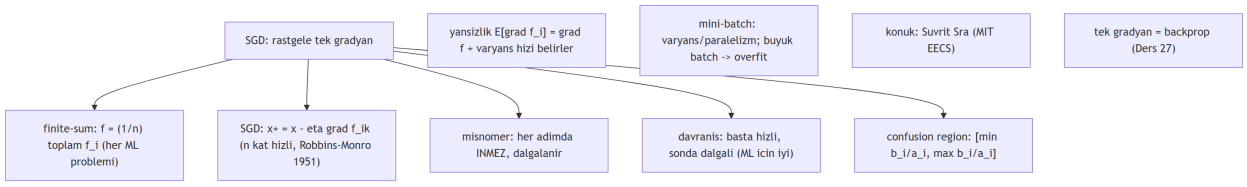
> **Bölüm bilgisi**
>
> Bu bir **konuk dersidir**: ana içeriği EECS'ten Prof. **Suvrit Sra** (MIT EECS, 6.036 hocası) anlatır; Strang yalnız konuğu takdim eder. Ders, derin öğrenmenin asıl çalışan algoritmasını işler: stochastic gradient descent (SGD) — 1951'den (Robbins-Monro) beri var, gradient descent'ten **tek satır** farkı var ama tüm büyük-ölçek ML'i sürüyor. Strang'in Ders 25 videosu (≈53 dk) ve OCW Lecture 25 temel alınmıştır. Okuma süresi ≈35 dk; önkoşul Ders 24 (lineer programlama, dualite). **Ana içerik alıntıları Sra'ya atfedilir ("— Sra, X:XX").**

32.1 Bu Derste Ne Var?

Bu konuk dersinde beş sonuç var:

1. **Finite-sum problem:** ML hedefi $f(x) = \frac{1}{n}\sum_i f_i(x)$; n (veri) ve d (boyut) çok büyük.
2. **SGD fikri:** tüm gradyan yerine **rastgele tek** veri noktasının gradyanı; bir iterasyon n kat hızlı.
3. **Misnomer:** SGD her adımda **inmez** (dalgalanır); adım boyuna çok duyarlı.
4. **Davranış:** başta hızlı ilerler, optimuma yakın dalgalanır (confusion region) — ML için iyi (aşırı-öğrenmeyi önler).
5. **Unbiased + variance:** stokastik gradyan gerçeğin yansız tahmini ama hızı **varyans** belirler; mini-batch varyansı azaltır + paralelizm.

> *"Stochastic gradient descent is actually a misnomer. At every step it doesn't do any descent." — Sra, 18:48*



Şekil 32.1: Ders 25 kavram haritası: SGD rastgele tek gradyanla calisir; finite-sum yapisi, n kat hizlanma, misnomer (her adimda inmez), dalgali davranis ve confusion region temel fikirler. Yansizlik + varyans hizi belirler; mini-batch varyans/paralelizmi yonetir. Konuk: Suvrit Sra. Tek gradyan = backprop (Ders 27).

Şekil 32.1 dersin iskeletini gösterir: merkezdeki "SGD: rastgele tek gradyan" fikri, her ML problemini ortak bir biçime sokan finite-sum yapısından, bir iterasyonu n kat hızlandıran SGD adımından (Robbins-Monro 1951), her adımda inmeyip dalgalanan misnomer davranışından, başta hızlı sonda dalgalı seyirden ve confusion region'dan dallanır; ayrı düğümlerde yansızlık + varyans, mini-batch (varyans/paralelizm; büyük batch → overfit), konuğun adı Suvrit Sra ve tek gradyanın backprop'la (Ders 27) hesaplandığı köprüsü durur.

Builder Notu — Tek Satırlık İkinci Devrim

- **SGD = derin öğrenmenin motoru** — her PyTorch/TensorFlow eğitimi SGD veya türevi (Adam, Ders 23); tek stokastik gradyan **backprop** ile hesaplanır (Ders 27).
- **Mini-batch = paralelizm** — GPU'da her çekirdek bir gradyan; büyük batch hızlı ama aşırı-öğrenme riski (generalization gap).
- **Yansızlık + varyans** — Ders 20'nin beklenen değer/varyansı burada doğrudan: stokastik gradyan $E[\nabla f_i] = \nabla f$ ama varyans yakınsamayı belirler.
- **Geriye köprü:** Ders 22 (GD), Ders 23 (momentum/Adam), Ders 20 (beklenen değer/varyans), Ders 9 (least squares). Paralel: NYU H4 (Defazio SGD/momentum pratik bakış).

32.2 Strang'ın Takdimi ve SGD Nedir

Strang konuğu tanıtır: Prof. Suvrit Sra (EECS, 6.036 hocası), SGD anlatacak. Buradan sonra içerik Sra'ya aittir. Sra'nın açılışı: SGD en **eski** optimizasyon yöntemlerinden biri (Cauchy'ye dayanır), sınıfta gördüğünüz gelişmiş yöntemlerden çok daha basit — ama hâlâ büyük-ölçek ML eğitiminin "**the**" yöntemi. Gradient descent'ten farkı tek bir satır; o tek satır tüm derin öğrenme araç kutularını sürüyor.

Builder Notu — Yetmiş Yıllık Genç Fikir

"En basit yöntem hâlâ en yaygın" SGD'nin paradoksu: 70 yıllık fikir, modern derin öğrenmenin motoru. ML köprüsü: PyTorch'ta `optim.SGD` bir satır; gelişmiş Adam/RMSProp (Ders 23) bile SGD iskeletinin üzerine kurulu. Basitlik + ölçeklenebilirlik, karmaşıklığı yener.

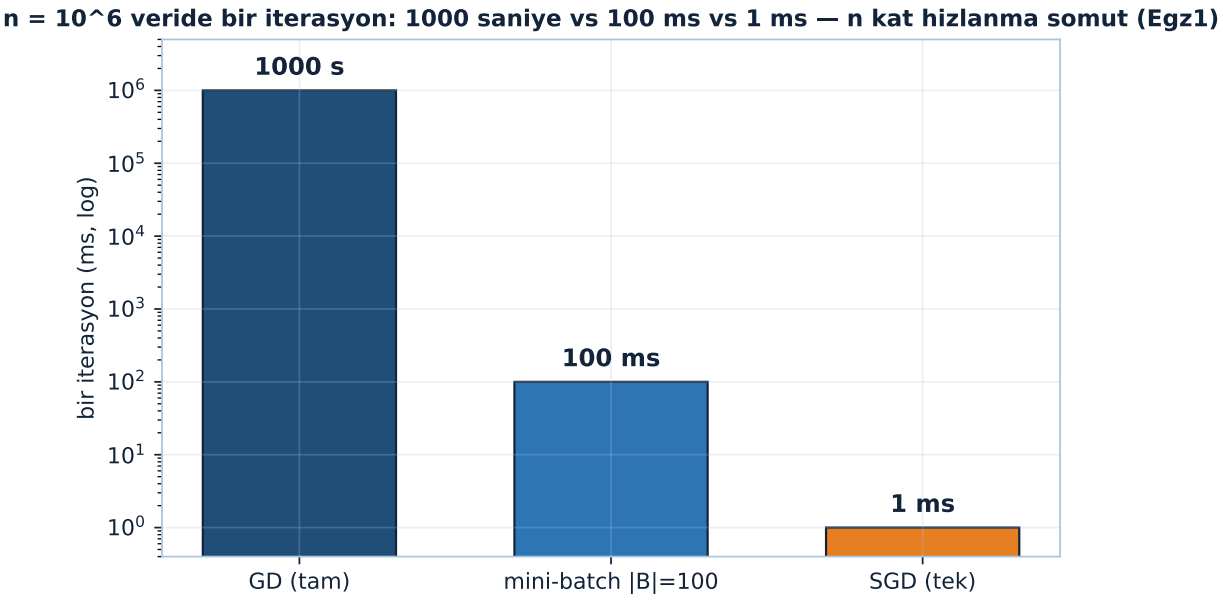
32.3 Finite-Sum Problem (ML Hedefi)

Tüm ML optimizasyon problemleri aynı biçimde:

$$\min_x \; f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

"…also called finite sum problems…" — Sra, 4:01

Eğitim verisi: $a_1, \dots, a_n$ (özellikler), $y_1, \dots, y_n$ (etiketler: ± 1 sınıflandırma veya reel regresyon). Her f_i tek bir veri noktasının kaybı. **Büyük-ölçek ML** = hem n (veri sayısı: milyon-milyar) hem d (boyut: piksel/özellik, milyona-milyara) büyük. Örnekler hep bu finite-sum: least squares, LASSO, SVM, derin sinir ağları (f_i = tüm ağ mimarisi + kayıp), maksimum olabilirlik.



Şekil 32.2: n = 10^6 veride bir iterasyon: GD 1000 saniye, mini-batch(100) 100 ms, SGD 1 ms — n kat hizlanma somut.

Şekil 32.2 finite-sum'ın hesaplama bedelini somutlaştırır: $n = 10^6$ veri noktasında bir gradyan 1 ms sürüyorsa, tam gradient descent bir iterasyonu 10^6 ms = 1000 saniye, mini-batch ($|B| = 100$) 100 ms, saf SGD (tek nokta) yalnız 1 ms alır — log-ölçekli çubuklar tam n kat hızlanmayı gösterir. Bu sayı, neden tam gradyanın (n terim) milyar-örnekli bir veride imkânsız olduğunu açıklar.

> **Builder Notu — Her Şey Bir Finite-Sum**
>
> "Her ML problemi bir finite-sum" birleştirici görüş: least squares'ten derin ağlara kadar hepsi $\frac{1}{n}\sum f_i$. ML köprüsü: bu yapı SGD'yi evrensel kılar — f_i ister kuadratik (regresyon) ister milyar-parametreli transformer kaybı olsun, aynı algoritma çalışır. n ve d'nin ikisinin de büyük olması, tam gradyanı (n terim) hesaplamayı imkânsız kılar — SGD'nin doğuş sebebi.

32.4 SGD Fikri: Rastgele Tek Nokta

Tam gradient descent her adımda **tüm** n terimin gradyanını toplar — büyük n'de bir adım saatler/günler sürebilir. Fikir (sınıftan çıktı):

> *"One example at a time." — Sra, 14:38*

Her iterasyonda rastgele bir i_k seç, sadece o tek veri noktasının gradyanını kullan:

$$\text{GD:}\quad x_{k+1} = x_k - \eta \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x_k) \qquad \text{SGD:}\quad x_{k+1} = x_k - \eta_k \nabla f_{i_k}(x_k)$$

Bir iterasyon artık n **kat** hızlı (n bir milyarsa, muazzam). Bu fikir Robbins ve Monro'dan (1951) — ve hâlâ kullandığımız en gelişmiş yöntem.

Şekil 32.3 bu tek-satır farkın izini aynı 2D least squares probleminde çizer: GD düz iner, SGD ise x^* çevresinde zikzaklayan ucuz adımlar atar (figürün ayrıntısı bir sonraki "Misnomer" bölümünde).

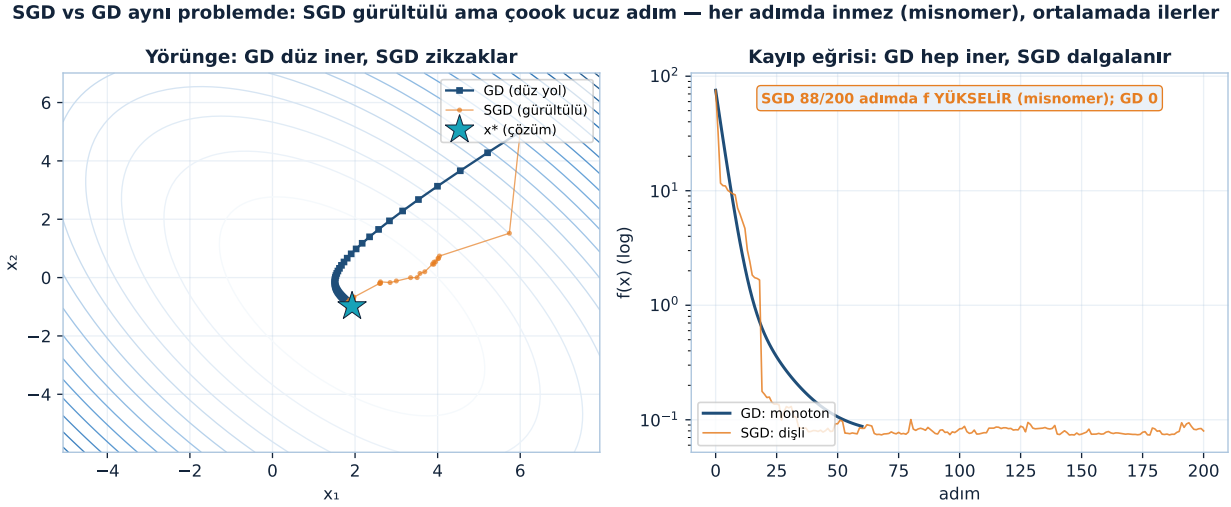
> **Builder Notu — Milyarda Bir Yeter**
>
> "Tüm gradyan yerine rastgele bir tane" SGD'nin tek-satır devrimi. ML köprüsü: bir milyar örnekli veri kümesinde tam gradyan bir adım = bir milyar gradyan; SGD bir adım = bir gradyan. Eğitim, veriyi tek tek (veya mini-batch) tarayarak ilerler — "epoch" = veri kümesinin bir tam geçişi.

32.5 "Misnomer": Her Adımda İnmez

SGD aslında yanlış-adlandırılmış: gradient descent her adımda **iner** (maliyeti düşürür); SGD bunu yapmaz.

> *"Stochastic gradient descent is actually a misnomer. At every step it doesn't do any descent." — Sra, 18:48*

Tek bir rastgele noktanın gradyanı tüm fonksiyonu temsil etmediğinden, bazı adımlar maliyeti yükseltir, bazıları düşürür — **dalgalanır** (fluctuate), ama stokastik olarak yine de optimuma doğru ilerler. Gradient descent'ten çok daha **adım-boyu duyarlı**: çok küçük $\eta \rightarrow$ neredeyse hareketsiz, kararlı görünür; çok büyük $\eta \rightarrow$ optimum çevresinde çılgın salınım.



Şekil 32.3: SGD vs GD aynı problemde: SGD gürültülü ama çoook ucuz adım — her adımda inmez (misnomer), ortalamada ilerler. Sol: GD düz iner, SGD x* (teal yıldız) çevresinde zikzaklayan bir bulut bırakır. Sağ: GD monoton düşer, SGD %88/200 adımda f'yi YÜKSELTİR; GD 0.

Şekil 32.3 misnomer'i sayıyla kanıtlar: solda aynı LS kontur haritasında GD düz iner (navy kare), SGD ise x^* (teal yıldız) çevresinde zikzaklayan gürültülü bir bulut bırakır (turuncu); sağda kayıp eğrileri log-ölçekte — GD monoton düşerken SGD dişlidir ve 200 adımın **88'inde** f'yi gerçekten YÜKSELTİR (GD'de 0 yükselme). Her adımda inmez ama ortalamada optimuma ilerler — "descent" adı tam da bu yüzden yanlıştır.

Builder Notu — İnmeyen İniş

"Her adımda inmez ama ortalamada ilerler" SGD'nin doğası: gürültülü ama doğru yöne. ML köprüsü: eğitim kayıp eğrisi (loss curve) bu yüzden pürüzsüz değil, dişlidir (gürültülü iner); learning rate çok büyükse kayıp patlar/salınır, çok küçükse öğrenme durur. Learning rate seçimi SGD'nin en kritik ve hâlâ açık problemi.

32.6 SGD Davranışı: Hızlı Başlangıç, Sonda Dalgalanma

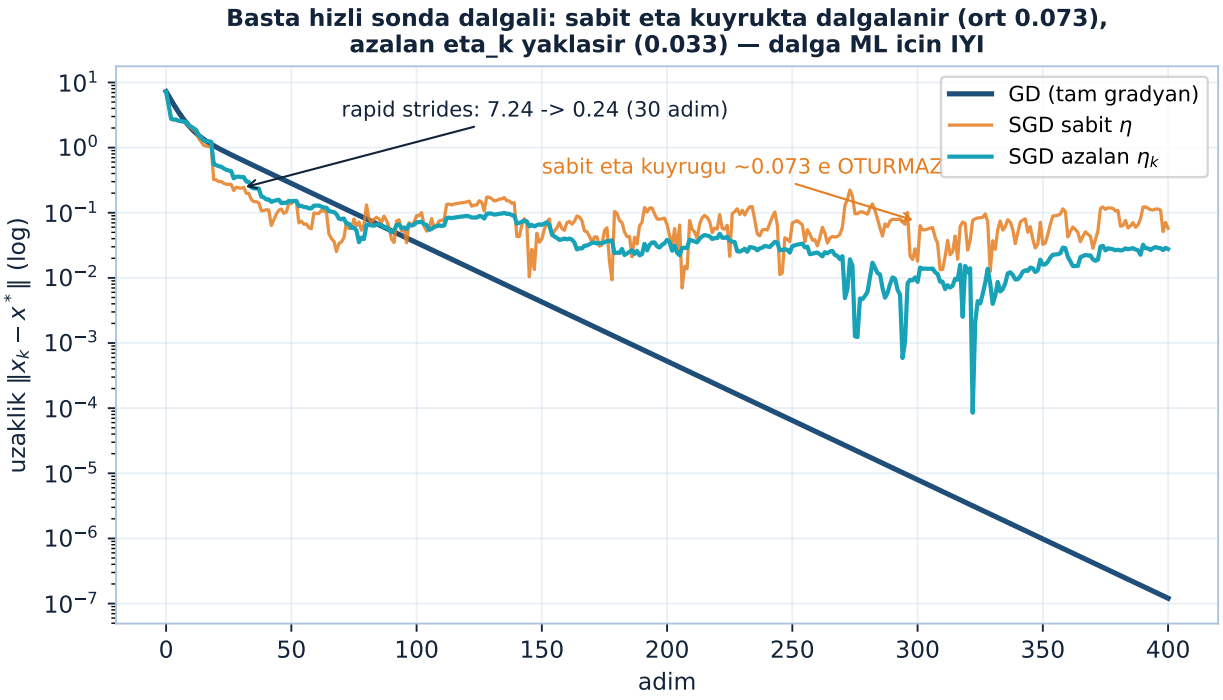
SGD'nin karakteristik deseni:

> *"…in the beginning, it makes rapid strides." — Sra, 23:30*

Başlangıçta hızlı ilerler (eğitim kaybı süper hızlı düşer), sonra optimuma yaklaşınca dalgalanır — takılır, kaosa girer ya da çılgınca davranır. İlginç olan: bu **ML için iyi**.

> *"…you actually care about finding solutions that work well on unseen data." — Sra, 24:04*

ML'de amaç optimizasyon problemini mükemmel çözmek değil, **görülmemiş veride** iyi çalışan çözüm bulmaktır. Optimizasyonu aşırı iyi çözmek = aşırı-öğrenme (overfitting). Hızlı kaba ilerleme + sonda durulma tam istenen.



Şekil 32.4: Başta hızlı sonda dalgalı: sabit η kuyruğa oturur (ort 0.073), azalan η_k yaklaşır (0.033) — ML için dalga İYİ (genelleme).

Şekil 32.4 bu deseni uzaklık eğrileriyle gösterir: $\|x_k - x^*\|$ üç iz halinde log-ölçekte — GD (navy) pürüzsüz iner, SGD sabit-η (turuncu) başta **rapid strides** yapar ($d_0 = 7.24 \to d_{30} = 0.24$, %97 düşüş 30 adımda)

ama kuyrukta ortalama 0.073'e oturup 0'a inMEZ, azalan-η_k (teal) ise platoyu 0.033'e indirir. Sabit adım optimuma yapışmaz; azalan adım yaklaştırır — ama ML için bu dalga bir kusur değil, erdemdir.

> Builder Notu — Mükemmel Çözme Genelle
>
> "Optimumu mükemmel çözme, genelle" SGD'nin gizli erdemi: dalgalanma bir kusur değil, düzenleyici (regularizer). ML köprüsü: SGD gürültüsü düz minimumlara (flat minima) yönelir — bunlar genelde daha iyi genelleşir; tam gradient descent keskin minimumlara saplanıp aşırı-öğrenebilir. "Erken durdurma" (early stopping) da aynı mantık: optimuma varmadan dur.

32.7 Confusion Region (Karışıklık Bölgesi)

Neden başta hızlı, sonda dalgalı? 1B least squares sezgisi: $f_i = (a_i x - b_i)^2$, her bir bileşen kendi minimumuna $x = b_i/a_i$'da ulaşır. Gerçek çözüm x^* bu tekil minimumların aralığındadır:

$$x^* \in \left[\min_i \frac{b_i}{a_i},\ \max_i \frac{b_i}{a_i}\right]$$

"…within this range of the individual mins and max is where [the solution lies]." — Sra, 29:22

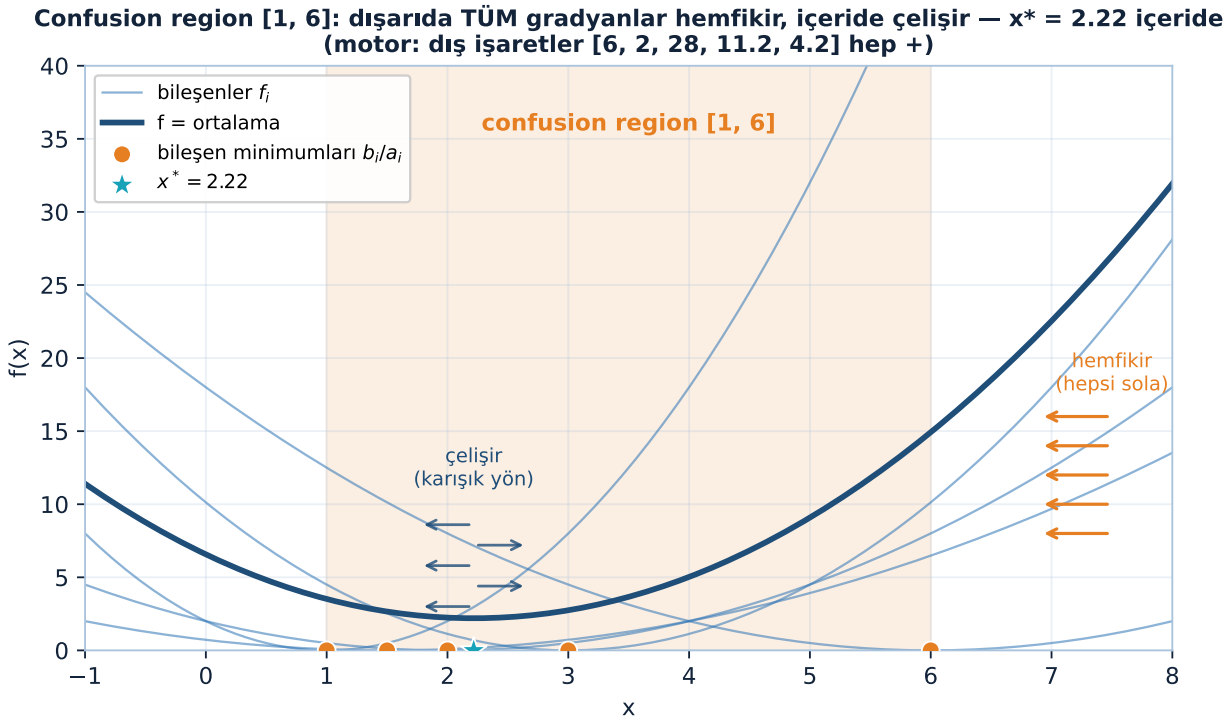
Dışarıda (bu aralığın dışında): tüm f_i'lerin gradyanları aynı yönü gösterir — hangi noktayı seçersen seç, doğru yöne inersin (hızlı, güvenli ilerleme). **İçeride** (confusion region): farklı f_i'ler farklı yönler söyler — rastgele seçtiğin nokta seni bazen ileri bazen geri iter (dalgalanma). SGD bu yüzden uzaktan hızlı yaklaşır, yakında kararsızlaşır.

Şekil 32.5 sezgiyi 1B paraboller üzerinde gösterir: beş bileşen $f_i = \frac{1}{2}(a_i x - b_i)^2$ (steel ince eğriler) ve ortalama f (navy kalın); bileşen minimumları b_i/a_i (turuncu markerlar, değerler 2/6/1/3/1.5) ve confusion aralığı $[1, 6]$ (turuncu gölge) içinde $x^* = 2.22$ (teal yıldız) durur. **Dışarıda** ($x = 8$): motor beş gradyanı $[6,\ 2,\ 28,\ 11.2,\ 4.2]$ verir — hepsi pozitif, yani hepsi sola (hemfikir, hızlı iniş). **İçeride** (x^*): işaretler $[0.22,\ -3.78,\ 4.87,\ -1.76,\ 0.46]$ — karışık yönlü (çelişir, dalgalanma). SGD bu yüzden uzaktan hızlı, yakında kararsızdır.

> Builder Notu — Hemfikir Dışarısı Kavgalı İçerisi
>
> "Dışarıda gradyanlar hemfikir, içeride çelişir" confusion region SGD davranışının zarif açıklaması. ML köprüsü: bu, mini-batch boyutunun neden önemli olduğunu açıklar — daha büyük batch confusion region'ı küçültür (gradyanlar ortalanır, daha az çelişki) ama birazdan göreceğimiz gibi bu her zaman iyi değil.

32.8 Yansızlık ve Varyans

SGD neden çalışır? İki istatistiksel sebep (Ders 20). Birincisi **yansızlık** (unbiased): rastgele seçilen tek gradyanın beklenen değeri tam gerçek gradyandır:



Şekil 32.5: Confusion region [1, 6]: dışarıda TÜM gradyanlar hemfikir (hızlı iniş), içeride çelişir (dalgalanma) — x* = 2.22 içeride.

$$E[\nabla f_{i_k}(x)] = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x) = \nabla f(x)$$

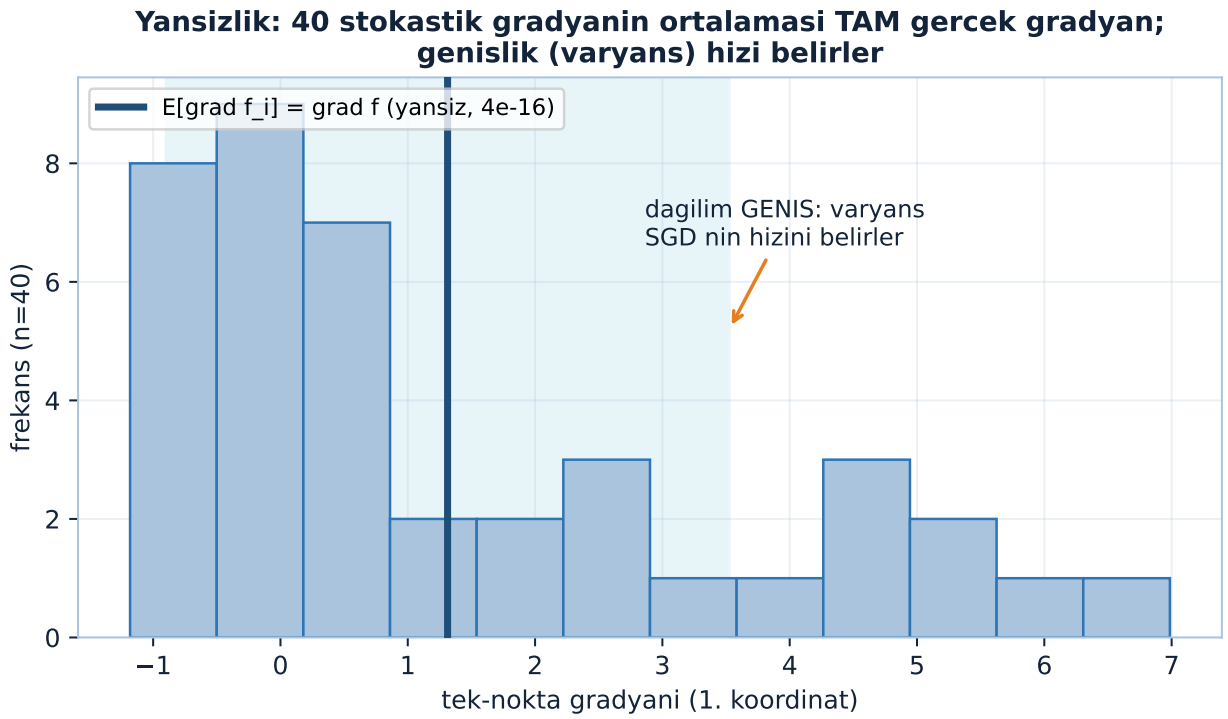
Yani ortalamada doğru yöne gidiyoruz. Ama bu yetmez:

> *"…beyond this unbiasedness [the amount of noise is controlled]…" — Sra, 38:03*

İkincisi **varyans**: yansız ama varyans devasaysa (bir adım $+\infty$, diğeri $-\infty$, ortalama 0) işe yaramaz. SGD'nin hızını, stokastik gradyanların **varyansı** belirler. Son yılların araştırma cephesi: yansızlığı koruyup varyansı düşüren yöntemler (variance reduction: SVRG, SAGA).

Şekil 32.6 yansızlığı ve varyansı tek bir histogramla gösterir: $x = (3, -2)$ noktasında 40 tek-nokta gradyanının 1. koordinatı (steel çubuklar) geniş bir dağılım çizer; navy dikey çizgi onların ortalamasıdır ve tam gerçek gradyana eşittir — motor maxdiff 4×10^{-16} ile yansızlığı birebir doğrular. Ama dağılımın genişliği (teal $\pm$std bandı) büyüktür: ortalama doğru olsa da tek bir örnek çok gürültülüdür, ve işte bu **varyans** SGD'nin hızını belirler.

> **Builder Notu — Yansız Ama Gürültülü**
>
> "Yansız ama varyans önemli" Ders 20'nin beklenen değer/varyansının doğrudan uygulaması: $E[\text{tahmin}]$ = doğru (yansız), ama $\text{Var}[\text{tahmin}]$ yakınsama hızını belirler. ML köprüsü: variance reduction yöntemleri (SVRG) ve momentum (Ders 23) ikisi de gradyan gürültüsünü azaltır; Adam'ın ikinci momenti (gradyan-kare ortalaması) de bir varyans-uyarlaması.



Şekil 32.6: Yansizlik: 40 stokastik gradyanin ortalamasi TAM gercek gradyan (4e-16); genisligi (varyans) SGD nin hizini belirler.

32.9 Mini-Batch ve Paralelizm

Tek nokta yerine küçük bir grup (mini-batch B) kullan; gradyanları ortala:

$$x_{k+1} = x_k - \eta_k \frac{1}{|B|} \sum_{i \in B} \nabla f_i(x_k)$$

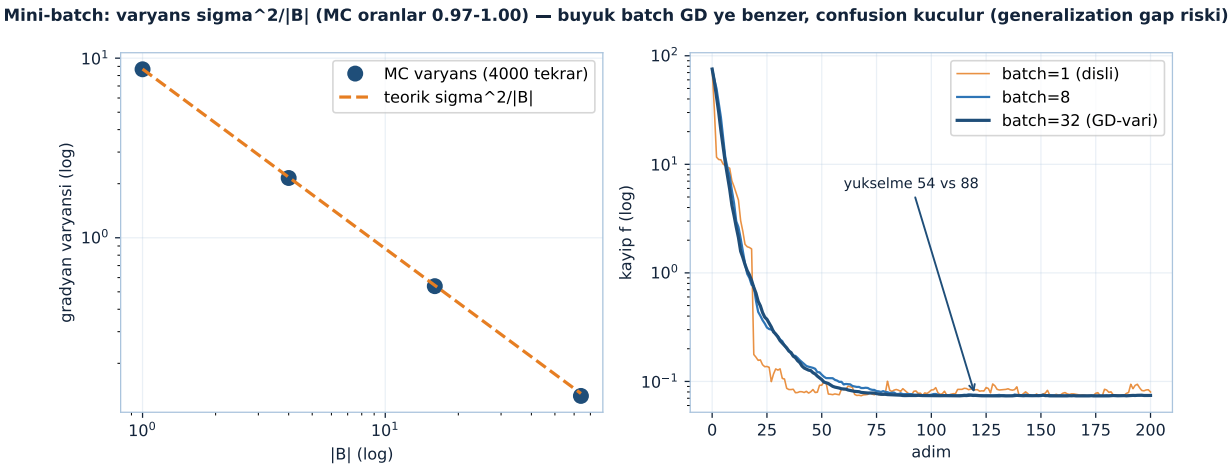
"Averaging things reduces the variance." — Sra, 44:51

Ortalama almak **varyansı azaltır**. $|B| = 1$ saf SGD, $|B| = n$ tam gradient descent, arası pratikte kullanılan. **İki rastgelelik:** with replacement (yerine koyarak, IID — teori bunu analiz eder) vs without replacement (pre-shuffle + stream — tüm araç kutuları bunu kullanır, ama IID-olmadığından analizi açık problem). Mini-batch'in asıl değeri **paralelizm**: GPU'da her çekirdek bir gradyan hesaplar. Ama tehlike:

"…your method starts resembling gradient descent." — Sra, 47:56

Çok büyük mini-batch → tam gradient descent'e benzer → confusion region aşırı küçülür → **aşırı-öğrenme**, test hatası kötüleşir (large-batch generalization gap). Optimal mini-batch boyutu hâlâ açık soru.

Şekil 32.7 ortalamanın bedelini ve faydasını iki panelde gösterir: solda Monte Carlo varyansı (4000 tekrar, navy markerlar) ile teorik $\sigma^2/|B|$ doğrusu (turuncu kesik) $|B| = 1, 4, 16, 64$ için neredeyse tam örtüşür — oranlar $[0.997,\ 0.990,\ 0.989,\ 0.969]$, yani ortalama varyansı tam $|B|$ kat düşürür. Sağda üç SGD izi: $|B| = 1$ (turuncu, dişli, 88 yükselme), $|B| = 8$ (steel), $|B| = 32$ (navy, GD-vari, yalnız 54 yükselme —



Şekil 32.7: Mini-batch: varyans sigma^2/|B| (MC oranlar 0.97-1.00) — buyuk batch GD ye benzer, confusion kuculur (generalization gap riski)

confusion küçülür). Büyük batch daha düzgün iner ama GD'ye benzedikçe genelleme açığı (generalization gap) doğar.

> **Builder Notu — Ortalamanın Bedeli**
>
> "Büyük batch hızlı ama genelleme kötü" derin öğrenmenin meşhur ikilemi: paralelizm için büyük batch isteriz, ama confusion region'ı yok edince düz-minimum arayışı (genelleme) bozulur. ML köprüsü: pratikte batch boyutu 32–8192 arası ayarlanır; büyük-batch eğitimi (warmup, LARS/LAMB optimizer'ları) bu açığı kapatmaya çalışır. Tek stokastik gradyanın nasıl hesaplandığı? **Backprop** (Ders 27).

32.10 Bu Dersin Özeti

- **Finite-sum:** ML hedefi $f(x) = \frac{1}{n}\sum f_i(x)$; n (veri) ve d (boyut) çok büyük. Least squares/LASSO/SVM/DNN hepsi bu.
- **SGD:** $x_{k+1} = x_k - \eta_k \nabla f_{i_k}(x_k)$, i_k rastgele; bir adım n kat hızlı (Robbins-Monro 1951).
- **Misnomer:** her adımda inmez, dalgalanır; adım boyuna çok duyarlı (motor: 88/200 adım yükselir).
- **Davranış:** başta hızlı, optimuma yakın dalgalı; ML için iyi (genelleme, aşırı-öğrenme önler).
- **Confusion region:** $x^* \in [\min b_i/a_i, \max b_i/a_i] = [1, 6]$; dışarıda gradyanlar hemfikir (hızlı), içeride çelişir (dalgalı).
- **Yansızlık + varyans:** $E[\nabla f_{i_k}] = \nabla f$ (maxdiff 4e-16); hızı varyans belirler (variance reduction: SVRG).
- **Mini-batch:** ortalama varyansı $\sigma^2/|B|$ azaltır; $|B| = 1$ SGD, $|B| = n$ GD; paralelizm (GPU); büyük batch → overfit (generalization gap). Tek gradyan = backprop (Ders 27).

> **Tek Bir Cümle**
>
> Stochastic gradient descent, finite-sum hedefin tüm gradyanı yerine rastgele bir veri noktasının (veya mini-batch'in) yansız ama gürültülü gradyanını kullanır — n kat ucuz, başta hızlı, optimuma yakın

dalgalı; yakınsamayı ve genellemeyi varyans ile mini-batch boyutu belirler.

32.11 Kontrol Soruları

Soru 1 — Tek satır farkı ve n kat hız

Soru: SGD, gradient descent'ten hangi tek değişiklikle elde edilir ve neden n kat hızlıdır?
Cevap: GD tüm n terimin gradyanını toplar: $x_{k+1} = x_k - \eta \frac{1}{n} \sum \nabla f_i$. SGD bunun yerine **rastgele tek** bir i_k seçip yalnız onun gradyanını kullanır: $x_{k+1} = x_k - \eta_k \nabla f_{i_k}$. Bir iterasyon n gradyan yerine 1 gradyan hesapladığından n **kat** hızlıdır (n milyarsa devasa kazanç). Robbins-Monro (1951).

Soru 2 — Misnomer ve GD'den fark

Soru: SGD neden "misnomer" (yanlış-adlandırma), ve davranışı GD'den nasıl farklıdır?
Cevap: Gradient descent her adımda maliyeti **düşürür** (descends); SGD bunu yapmaz — tek rastgele gradyan tüm fonksiyonu temsil etmediğinden bazı adımlar maliyeti yükseltir, **dalgalanır** (motor: 200 adımın 88'i yükselir). Ortalamada doğru yöne gider ama her adımda değil. Ayrıca adım boyuna çok daha duyarlıdır: başta hızlı ilerler, optimuma yakın dalgalanır.

Soru 3 — Confusion region

Soru: Confusion region nedir, SGD'nin başta hızlı/sonda dalgalı davranışını nasıl açıklar?
Cevap: 1B least squares'te her bileşen $f_i = (a_i x - b_i)^2$ minimumuna b_i/a_i'da ulaşır; gerçek x^* bu değerlerin $[\min, \max]$ aralığındadır (confusion region). Dışarıda tüm gradyanlar aynı yönü gösterir → hangi noktayı seçersen seç doğru yöne hızlı inersin. İçeride gradyanlar çelişir → rastgele seçim seni bazen ileri bazen geri iter → dalgalanma. Uzaktan hızlı, yakında kararsız.

Soru 4 — Yansızlık, varyans ve mini-batch

Soru: Stokastik gradyanın yansızlığı ne demek, varyans ve mini-batch nasıl devreye girer?
Cevap: Yansızlık: $E[\nabla f_{i_k}] = \frac{1}{n} \sum \nabla f_i = \nabla f$, yani rastgele gradyan ortalamada gerçek gradyana eşit (Ders 20; motor maxdiff 4e-16). Ama yansızlık yetmez — **varyans** büyükse yakınsama yavaş/kaotik. SGD'nin hızı varyansa bağlıdır. **Mini-batch** birkaç gradyanı ortalayarak varyansı $\sigma^2/|B|$'ye azaltır (+GPU paralelizmi); ama çok büyük batch confusion region'ı yok edip aşırı-öğrenmeye yol açar (generalization gap).

32.12 Egzersizler

1. **Hız karşılaştırması.** $n = 10^6$ veri noktası, her gradyan 1 ms sürüyor. Tam gradient descent bir iterasyonu kaç saniye? SGD (tek nokta) bir iterasyonu kaç ms? Mini-batch ($|B| = 100$) kaç ms? (Motor tanığı: GD 10^6 ms = 1000 s; SGD 1 ms; $|B| = 100$ → 100 ms; bkz. Şekil 32.2.)

2. **Yansızlık.** $f = \frac{1}{3}(f_1 + f_2 + f_3)$, gradyanları $\nabla f_1 = 2$, $\nabla f_2 = -4$, $\nabla f_3 = 8$ (1B, bir noktada). Tam gradyan ∇f nedir? Rastgele tek-nokta gradyanının beklenen değeri buna eşit mi (yansızlık)? (Motor tanığı: $\nabla f = (2 - 4 + 8)/3 = 2$; tek-nokta beklentisi $\frac{1}{3}(2) + \frac{1}{3}(-4) + \frac{1}{3}(8) = 2$ — yansız.)
3. **Confusion region.** 1B least squares $f_i = (a_i x - b_i)^2$, (a, b) çiftleri: $(1, 2), (1, 6), (2, 2)$. Her bileşenin minimumu b_i/a_i'yı bul. x^* hangi $[\min, \max]$ aralığında olmalı? (Motor tanığı: minimumlar $2, 6, 1 \rightarrow$ aralık $[1, 6]$; $x^* = \Sigma ab/\Sigma a^2 = 12/6 = 2.0$, aralık içinde.)
4. **Mini-batch varyansı.** Bağımsız stokastik gradyanların varyansı σ^2. $|B|$ büyüklüğünde mini-batch ortalamasının varyansı kaça düşer? (İpucu: bağımsız ortalamanın varyansı $\sigma^2/|B|$.) (Motor tanığı: $|B| = 100 \rightarrow \sigma^2/100$; MC oranları 0.97–1.00 ile birebir; bkz. Şekil 32.7 sol panel.)
5. **(Ders 26 habercisi)** SGD'nin minimize ettiği f_i, derin sinir ağının kaybıydı ($f_i = \text{DNN}(a_i, x)$ kaybı). Peki bir sinir ağı **tam olarak** nasıl bir fonksiyon? Katmanlar, ağırlıklar, doğrusal-olmayan aktivasyon nasıl birleşir? Bir tahmin yaz — Ders 26 "derin öğrenme için sinir ağlarının yapısı" ile öğrenme fonksiyonunu inşa ediyor.

32.13 Sonraki Ders İçin Hazırlık

Ders 26: Derin Öğrenme için Sinir Ağlarının Yapısı. SGD'nin minimize ettiği fonksiyonu açıyoruz: bir sinir ağı = doğrusal katmanlar (ağırlık matrisleri) + doğrusal-olmayan aktivasyon (ReLU) zinciri. Strang öğrenme fonksiyonu $F(x, v)$'yi, katmanların geometrisini ve neden derinliğin işe yaradığını inşa eder.

> ⚠ Hazırlık
>
> Ders 26 ile konuk dersi biter, anlatım Strang'a döner. Bu dersteki Egzersiz 5'in sorduğu soruyu zihninde tut: SGD'nin minimize ettiği f_i bir sinir ağının kaybıysa, o ağ tam olarak nasıl bir fonksiyondur? Strang, doğrusal katmanlar (ağırlık matrisleri) ile doğrusal-olmayan aktivasyonun (ReLU) zincirini, öğrenme fonksiyonu $F(x, v)$'yi ve derinliğin neden işe yaradığını inşa edecek — SGD'nin "ne"yi optimize ettiğinin cevabı.

32.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Konuşmacı (dk)
Finite-sum problem	$f(x) = \frac{1}{n}\sum f_i(x)$; n, d büyük	Sra 4m01
SGD fikri	$x_+ = x - \eta \nabla f_{i_k}$; rastgele tek nokta, $n\times$ hızlı	Sra 14m38
Misnomer	her adımda inmez, dalgalanır	Sra 18m48
Hızlı başlangıç	başta rapid strides, sonda dalgalı	Sra 23m30
Confusion region	$x^* \in [\min b_i/a_i, \max b_i/a_i]$	Sra 29m22
Yansızlık + varyans	$E[\nabla f_{i_k}] = \nabla f$; hızı varyans belirler	Sra 38m03
Mini-batch	ortalama varyansı azaltır + paralelizm	Sra 44m51
Büyük batch → overfit	GD'ye benzer; generalization gap	Sra 47m56

Kavram	Formül / Fikir	Konuşmacı (dk)
Backprop	tek stokastik gradyanı hesaplar (Ders 27)	Sra 50m25

32.15 ML Bağlantıları Özeti

- **SGD = derin öğrenmenin motoru:** her PyTorch/TensorFlow eğitimi SGD/türevi; epoch = veri kümesinin bir tam geçişi.
- **Variance reduction:** SVRG/SAGA yansızlığı koruyup varyansı düşürür; momentum (Ders 23) ve Adam'ın 2. momenti de gürültü azaltır.
- **Mini-batch paralelizmi:** GPU'da her çekirdek bir gradyan; batch boyutu 32–8192; büyük-batch eğitimi (warmup, LARS/LAMB) generalization gap'i kapatmaya çalışır.
- **SGD gürültüsü = düzenleyici:** düz minimumlara (flat minima) yönelir → daha iyi genelleme; erken durdurma da aynı mantık.
- **Yansız tahmin / varyans:** Ders 20'nin beklenen değer/varyansının doğrudan uygulaması (mini-batch örneklem ortalaması).
- **Backprop köprüsü:** tek stokastik gradyan backprop ile hesaplanır (Ders 27) — bazıları "backprop" deyince SGD'yi kasteder ama backprop sadece gradyan hesaplama algoritmasıdır.
- **Geriye köprü:** Ders 22 (GD), Ders 23 (momentum/Adam), Ders 20 (beklenen değer/varyans/Markov), Ders 9 (least squares). Paralel: NYU H4 (Defazio'nun SGD/momentum pratik bakışı).

! Kapanış

> *"Stochastic gradient descent is actually a misnomer. At every step it doesn't do any descent."*
> *— Sra, 18:48*

SGD her adımda inmez ama ortalamada doğru yöne gider; bu gürültü hem ölçeklenebilirliğin hem de iyi genellemenin sırrıdır — derin öğrenmeyi mümkün kılan tek-satırlık devrim.

33 Derin Öğrenme için Sinir Ağlarının Yapısı

Öğrenme fonksiyonu, ReLU, origami ve universality

> **Bölüm bilgisi**
>
> Bu ders, SGD'nin minimize ettiği şeyi açıklar: bir sinir ağı = **affine katman** (ağırlık matrisi $Ax + b$) + **doğrusal-olmayan aktivasyon** (ReLU) zinciri; sonuç **öğrenme fonksiyonu** $F(x)$ sürekli parçalı-doğrusaldır. Strang'in Ders 26 videosu (≈53 dk) ve OCW Lecture 26 temel alınmıştır. Okuma süresi ≈34 dk; önkoşul Ders 25 (SGD, finite-sum). Anlatım konuk dersinden Strang'a döner.

33.1 Bu Derste Ne Var?

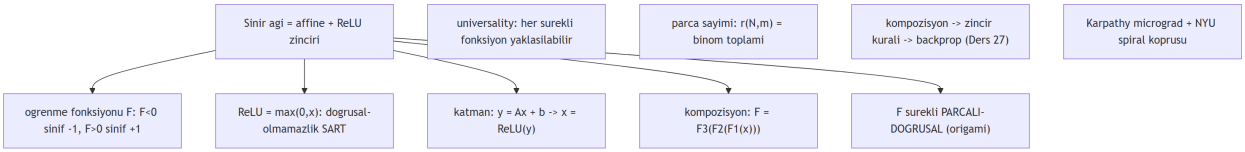
SGD'nin minimize ettiği fonksiyonu açıyoruz: **öğrenme fonksiyonu** $F(x)$. Bir sinir ağı = **affine katman** (ağırlık matrisi $Ax + b$) + **doğrusal-olmayan aktivasyon** (ReLU) zinciri. F sürekli **parçalı-doğrusal** bir fonksiyondur.

Beş sonuç:

1. **Öğrenme fonksiyonu:** $F(x)$ eğitim verisini öğrenir; sınıflandırmada $F(x) < 0 \Rightarrow$ sınıf -1, $F(x) > 0 \Rightarrow +1$. Her noktayı doğru yapma (aşırı-öğrenme).
2. **ReLU:** $\text{ReLU}(x) = \max(0, x)$; doğrusal-olmamazlık ŞART (yoksa sadece düzlemle ayırır, spiral çözülemez).
3. **Katman:** $y = Ax + b$ (affine) $\to x = \text{ReLU}(y)$ (bileşen-bileşen); $F = F_3(F_2(F_1(x)))$ kompozisyon.
4. **F = sürekli parçalı-doğrusal:** origami gibi katlanmış; universality — yeterli katlamayla her sürekli fonksiyon yaklaşılabilir.
5. **Parça sayımı:** N katlama, m boyut $\to r(N, m) = \sum \binom{N}{k}$ (binom); derinlik ifade gücünü artırır.

> *"…constructing this function F which learns the training data…" — Strang, 1:41*

Şekil 33.1 dersin iskeletini gösterir: merkezdeki "sinir ağı = affine + ReLU zinciri" fikrinden, sınıf kararını veren öğrenme fonksiyonu F'den, doğrusal-olmamazlığı sağlayan ReLU'dan, kompoze olan katmanlardan ve sürekli parçalı-doğrusal (origami) çıkan F'den dallanır; ayrı düğümlerde her sürekli fonksiyonu yaklaşan universality, binom toplamı parça sayımı $r(N, m)$, kompozisyondan zincir kuralına oradan backprop'a (Ders 27) uzanan köprü ve Karpathy micrograd + NYU spiral paraleli durur.



Şekil 33.1: Ders 26 kavram haritasi: sinir agi = affine + ReLU zinciri. Ogrenme fonksiyonu F sinif kararini verir; ReLU dogrusal-olmamazligi saglar; katmanlar kompoze olur ve F surekli parcali-dogrusal (origami) cikar. Universality her surekli fonksiyonu yaklasir; parca sayimi r(N,m) binom toplamidir; kompozisyon zincir kuralini -> backprop (Ders 27) getirir. Karpathy micrograd + NYU spiral koprusu.

Builder Notu — SGD'nin Minimize Ettiği Şey

- F **= ağırlık matrisleri + ReLU zinciri** — her PyTorch `nn.Sequential` tam bu; A'lar öğrenilen ağırlıklar, SGD (Ders 25) onları ayarlar.
- **ReLU doğrusal-olmamazlığı = ifade gücü** — doğrusal katmanlar üst üste hâlâ doğrusaldır; ReLU olmadan derinlik anlamsız.
- **Universality + parça sayımı** — derin ağların neden her fonksiyonu öğrenebildiği; derinlik parça sayısını üstel artırır (expressivity).
- **Kompozisyon → zincir kuralı → backprop** (Ders 27): $F = F_3 \circ F_2 \circ F_1$ türevi zincir kuralıyla. Paralel: NYU H1 spiral sınıflandırma, Karpathy micrograd kompozisyon.

33.2 Öğrenme Fonksiyonu F(x)

Tüm sistem tek bir hedefe yöneliktir:

> *"…constructing this function F which learns the training data…" — Strang, 1:41*

Eğitim verisini öğrenen, sonra test verisine uygulanan bir **öğrenme fonksiyonu** F kur. İkili sınıflandırmada (kedi/köpek, ± 1): $F(x)$ sınıf -1 için **negatif**, sınıf $+1$ için **pozitif** olsun. Her örneği doğru yapmaya çalışmak şart değil — tuhaf bir örneği zorla doğru yapmak **aşırı-öğrenme** (overfitting) olur. Amaç neredeyse tüm durumları kapsayan kuralı bulmak.

$$F(x) < 0 \Rightarrow -1, \qquad F(x) > 0 \Rightarrow +1$$

Builder Notu — Kuralı Öğren Noktayı Değil

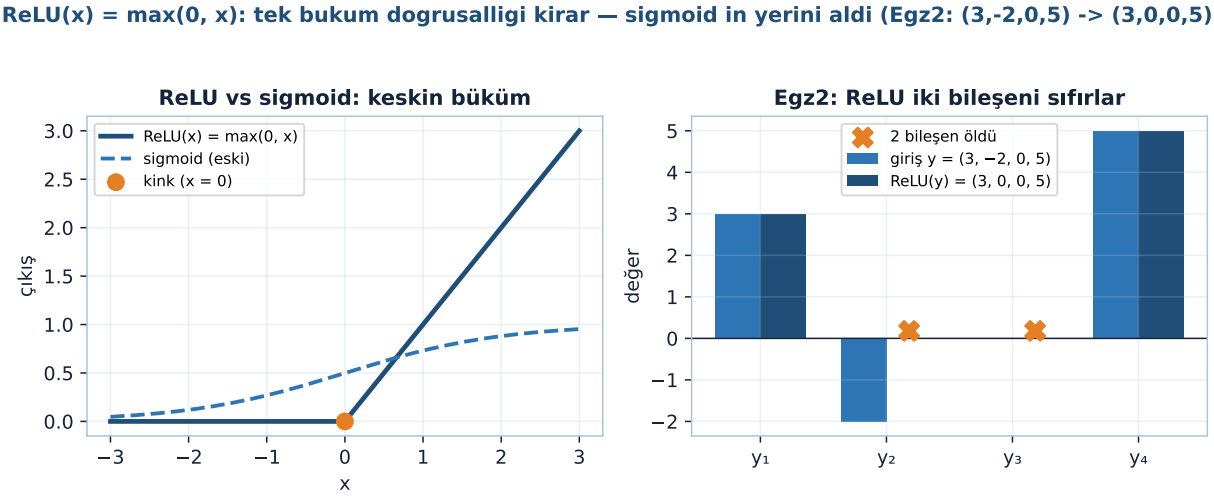
"Her noktayı doğru yapma, kuralı öğren" derin öğrenmenin temel gerilimi: eğitim hatasını sıfırlamak ≠ iyi model. ML köprüsü: F'nin sıfır-kümesi $\{F(x) = 0\}$ sınıfları ayıran karar sınırıdır (decision boundary); SGD (Ders 25) A ağırlıklarını bu sınır veriyi ayıracak şekilde ayarlar. Mucize: pratikte test verisinde de iyi çalışır.

33.3 ReLU ve Doğrusal-Olmamazlık

TensorFlow Playground örneği: iç içe sınıflar (mavi içeride, turuncu dışarıda; ya da iki spiral). Doğrusal sınıflandırıcı (düzlem) bunları ayıramaz — **doğrusal-olmamazlık** şart. Modern aktivasyon fonksiyonu ReLU:

> *"…ReLU is a function of x is the maximum, the larger of 0 and x." — Strang, 10:13*

$$\text{ReLU}(x) = \max(0, x)$$



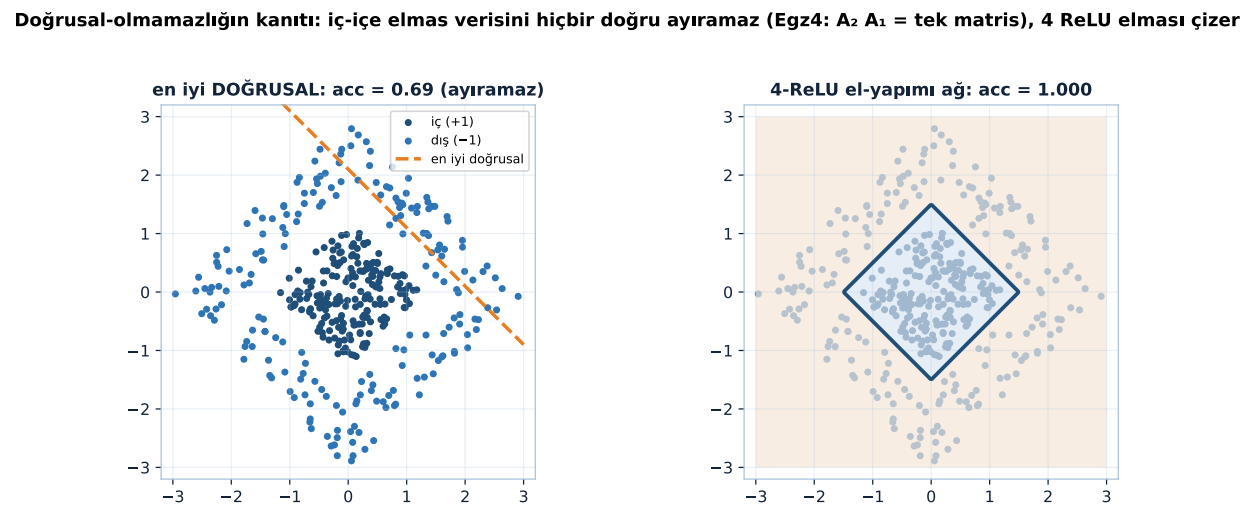
Şekil 33.2: ReLU(x) = max(0, x): tek büküm doğrusallığı kırar — sigmoid'in yerini aldı (Egz2: (3, −2, 0, 5) → (3, 0, 0, 5), 2 bileşen öldü).

Neden kritik: tüm katmanlar doğrusal olsaydı, bileşkeleri yine doğrusal olurdu (bir düzlem) — spirali ayıramazdık. ReLU her katmana doğrusal-olmamazlık enjekte eder; ancak o zaman ağ doğrusal-olmayan karar sınırları (örn. $r - 5$ gibi yarıçap fonksiyonları) öğrenebilir. Şekil 33.3 bunu kanıtlar: iç-içe elmas verisini en iyi doğrusal sınıflandırıcı bile ayıramaz (acc < 0.75), oysa elle kurulan 4-ReLU ağ $F = c - |x| - |y|$ tam ayırır (acc $= 1.000$).

> **Builder Notu — Doğrusalı Kıran Tek Büküm**
>
> "Doğrusal × doğrusal = doğrusal, ReLU kırar" derin öğrenmenin neden derinlik gerektirdiğinin özü: ReLU olmadan 100 katman bile tek bir matrise çöker. ML köprüsü: NYU H1'in spiral sınıflandırması (Phase 2 paralel ders) tam bu Playground örneği — doğrusal-olmayan aktivasyon olmadan spiral öğrenilemez; ReLU bugün varsayılan (sigmoid'in yerini aldı).

33.4 Katman Yapısı: Affine + ReLU

Bir katman iki adımdan oluşur. Önce **affine** dönüşüm (ağırlık matrisi + bias): özellik vektörü x_0 (5 bileşen) bir A_1 matrisiyle (6×5, yani 30 ağırlık) çarpılıp bir b_1 bias'ı eklenir:



Şekil 33.3: Doğrusal-olmamazlığın kanıtı: iç-içe elmas verisini hiçbir doğru ayıramaz (Egz4: A_2 A_1 = tek matris), 4 ReLU elması çizer. Sol — en iyi doğrusal sınıflandırıcı acc = 0.69 (ayıramaz); sağ — 4-ReLU el-yapımı ağ F = c − |x| − |y| acc = 1.000.

$$y_1 = A_1 x_0 + b_1 \qquad (A_1 : 6 \times 5,\ b_1 : 6 \times 1)$$

$30 + 6 = 36$ ağırlık (öğrenilecek parametre). Sonra **ReLU** bileşen-bileşen uygulanır (6 kopya):

$$x_1 = \text{ReLU}(y_1) = \max(0, y_1)$$

Derin ağ = bu katmanı tekrar tekrar yığmak. Katman 1 verinin temel özelliklerini, katman 2 daha ince detayları, katman 3 daha da incesini öğrenir — derinliğin gücü bu. Şekil 33.4 hem tek katmanın 36 parametresini hem de katmanların kompozisyon zincirini gösterir.

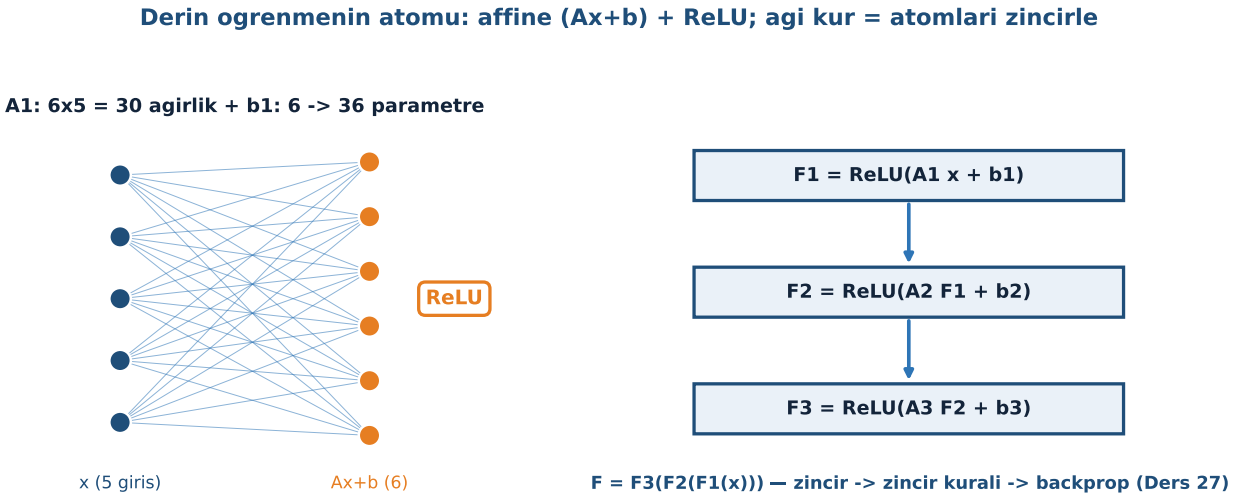
> Builder Notu — Derin Öğrenmenin Atomu
>
> "Affine ($Ax + b$) sonra ReLU" tek bir katman; A ve b öğrenilen ağırlıklar. ML köprüsü: PyTorch'ta `nn.Linear(5,6)` tam $A_1 x_0 + b_1$, ardından `nn.ReLU()`; gerçek ağlarda 36 yerine on binlerce-milyonlarca parametre, onlarca-yüzlerce katman. Affine + nonlinearity ikilisi tüm derin öğrenmenin atomu.

33.5 Kompozisyon: F = $F_3(F_2(F_1(x)))$

Katmanları zincirle. Her katman bir fonksiyon $F_k(v) = \text{ReLU}(A_k v + b_k)$; tüm ağ bunların **kompozisyonu**:

"F of x is going to be F3… of F2 of F1 of x…" — Strang, 25:44

$$F(x) = F_3(F_2(F_1(x))), \qquad F_k(v) = \text{ReLU}(A_k v + b_k)$$



Şekil 33.4: Derin öğrenmenin atomu: affine (Ax+b) + ReLU; ağı kur = atomları zincirle (kompozisyon). Sol: 6×5 ağırlık + 6 bias = 36 parametre. Sağ: F = F3(F2(F1(x))) zinciri → zincir kuralı → backprop (Ders 27).

"Zincir" kelimesi kasıtlı: bir fonksiyonun fonksiyonunun fonksiyonu = kompozisyon, ve türevini **zincir kuralı** verir (Ders 27 backprop). Tarihsel not: eskiden aktivasyon sigmoid (S-eğrisi) idi:

> *"…the original functions were sigmoids, like S curves…" — Strang, 27:33*

ama deneyler ReLU'nun daha iyi çalıştığını gösterdi.

> **Builder Notu — Zincirin Kendisi**
>
> $F = F_3 \circ F_2 \circ F_1$ kompozisyonu derin öğrenmenin iskeleti; "deep" = çok katmanlı kompozisyon. ML köprüsü: kompozisyon → zincir kuralı → backprop (Ders 27); bu tam Karpathy'nin micrograd'ı (fonksiyon kompozisyonu + geri-türev) ve fast.ai'nin manuel backprop'u (Phase 2 paralel dersler). Sigmoid→ReLU geçişi derin öğrenme devrimini (2010'lar) tetikleyen pratik buluşlardan.

33.6 F Sürekli Parçalı-Doğrusaldır

F matematiksel olarak ne tür bir fonksiyon? Affine (sürekli) ve ReLU (sürekli, parçalı-doğrusal) kompozisyonu olduğundan:

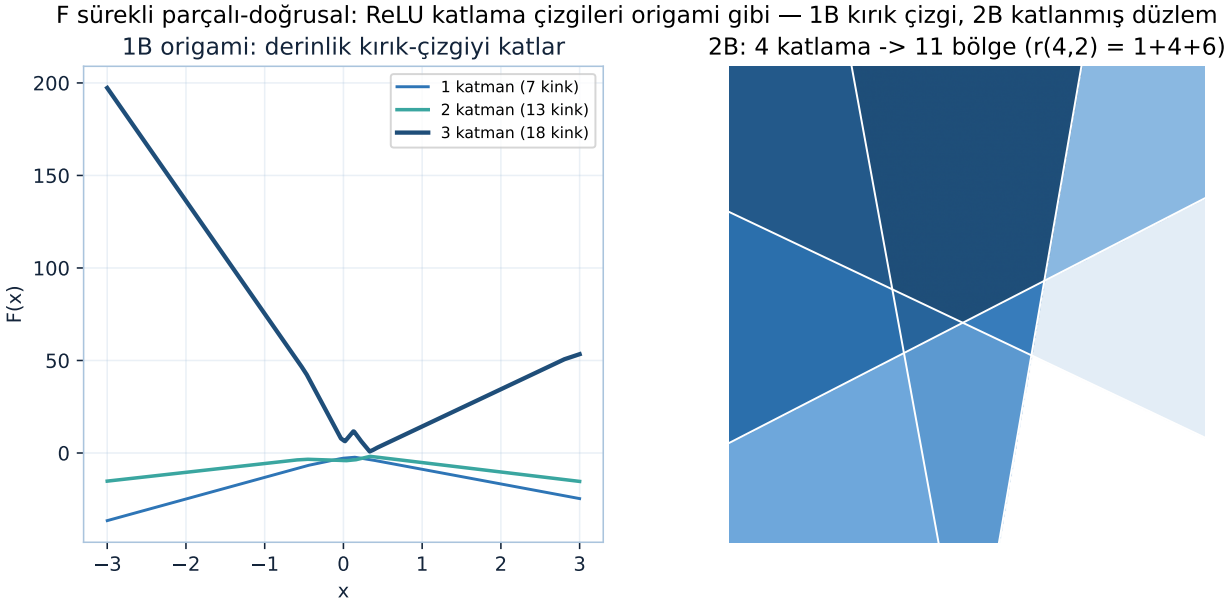
> *"…a continuous piecewise linear function." — Strang, 29:04*

F bir sürekli parçalı-doğrusal fonksiyondur.

Her parça düz (doğrusal), parçalar kenarlarda birleşir (sürekli). Görsel benzetme: **origami**.

> *"This is the theory of origami almost." — Strang, 32:20*

Düz bir kâğıdı düz çizgiler boyunca katlamak gibi. ReLU her katmanda "katlama" (fold) çizgileri yaratır; F bu katlamalarla parçalanmış düz yüzeylerden oluşur. 1B'de "kırık çizgi" (broken line), 2B'de katlanmış düzlem. Şekil 33.5 hem 1B'de derinlikle artan kink sayısını hem de 2B'de 4 katlamanın 11 bölge yaratışını gösterir.



Şekil 33.5: F sürekli parçalı-doğrusal: ReLU katlama çizgileri origami gibi — 1B kırık çizgi, 2B katlanmış düzlem. Sol: derinlik arttıkça (1/2/3 katman) kırık-çizgi katlanır, kink sayısı artar. Sağ: 2B'de 4 katlama düzlemi 11 bölgeye böler (r(4,2) = 1+4+6).

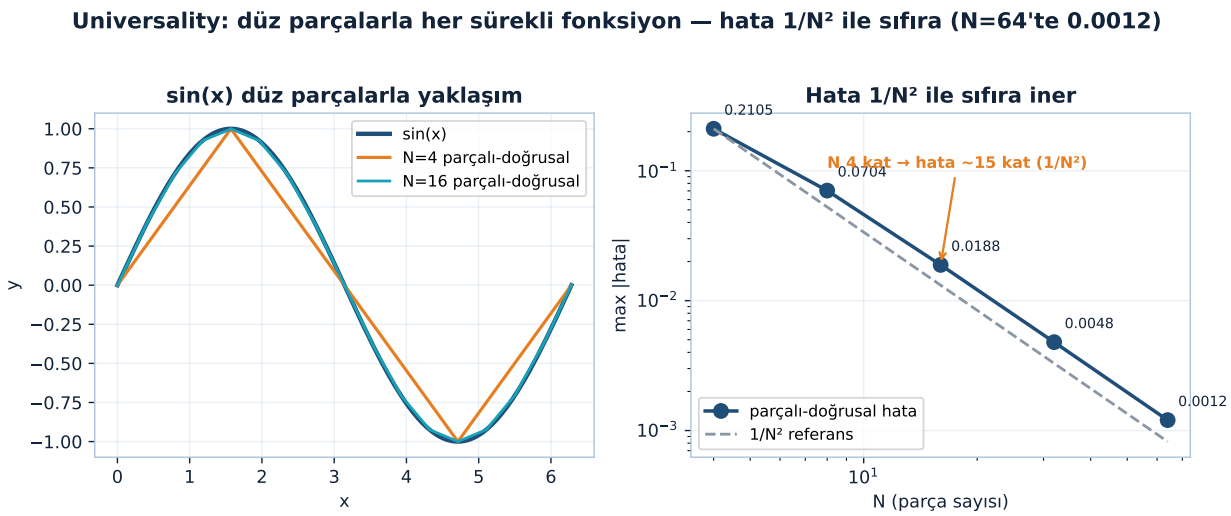
> **Builder Notu — Origami Matematiği**
>
> "F = sürekli parçalı-doğrusal" ReLU ağlarının kesin matematiksel karakterizasyonu: eğri değil, düz parçaların birleşimi. ML köprüsü: bu yüzden ReLU ağları girdi uzayını çokyüzlü (polytope) bölgelere böler, her bölgede doğrusaldır; ağın "karar sınırı" bu parçaların kenarlarıdır. Modern yorumlanabilirlik (interpretability) araştırması bu bölge yapısını inceler.

33.7 Universality Teoremi

Bu fonksiyon sınıfı ne kadar güçlü? Her fonksiyonu üretebilir mi? Tam olarak değil — sadece sürekli parçalı-doğrusal olanları. Ama:

> *"…the universality theorem would be to say that any function [can be approximated]…" — Strang, 39:41*

Universality (evrensellik): herhangi bir sürekli fonksiyon ($\sin x$, ne olursa olsun), yeterli katlamayla (yeterli nöron/katman) **istenildiği kadar yakından** yaklaşıklanabilir. Düz parçalarla bir eğriyi istediğin hassasiyette örtebilirsin — yeterince çok küçük düz parça kullan. Şekil 33.6 bunu sayısal olarak doğrular: parça sayısı N büyüdükçe hata $1/N^2$ ile sıfıra iner (N=64'te 0.0012).



Şekil 33.6: Universality: düz parçalarla her sürekli fonksiyon — hata 1/N^2 ile sıfıra (N=64'te 0.0012).

> **Builder Notu — Her Eğriye Yetecek Katlama**
>
> Universal approximation teoremi derin öğrenmenin teorik güvencesi: yeterli kapasiteyle ağ her sürekli fonksiyonu öğrenebilir. ML köprüsü: bu "öğrenebilir mi?" sorusunu çözer (evet); ama "verimli öğrenir mi, genelleşir mi?" ayrı sorular (SGD, Ders 25; genelleme). Universality gerek-koşul, yeterli-koşul değil — pratikte derinlik + SGD + veri birlikte çalışır.

33.8 Parça Sayımı: Binom Formülü

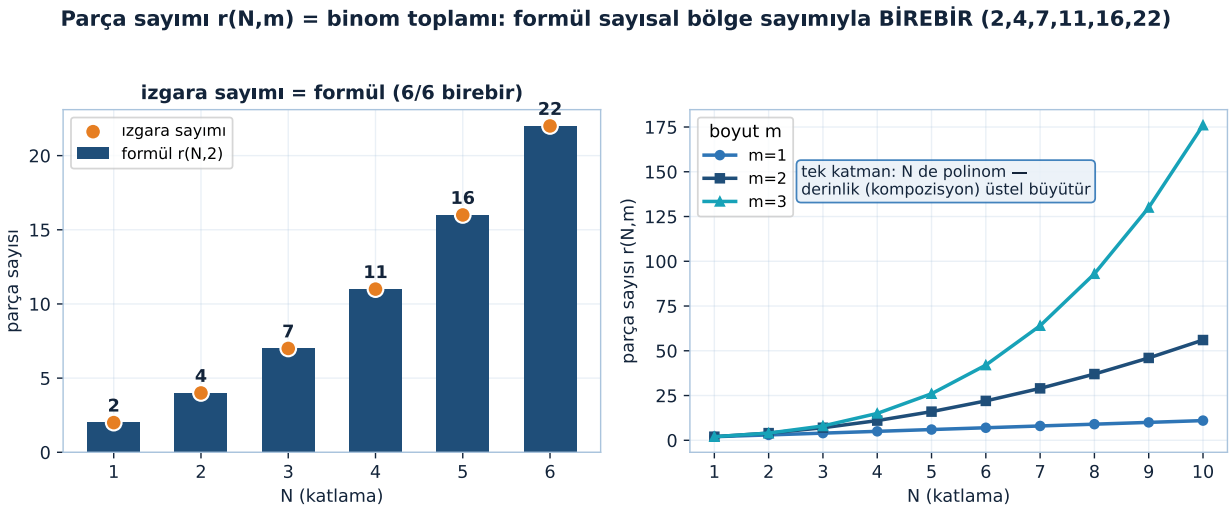
F ne kadar "kıvrımlı" olabilir? Düz parça (flat piece) sayısını say. m-boyutlu uzayı N kez katla (N fold). 2B'de bir düzlemi katla: 1 katlama → 2 parça, 2 katlama → 4, 3 katlama → 7, 4 katlama → 11. Bir özyineleme (her yeni katlama mevcut çizgileri keserek yeni parçalar ekler) ve sonuç binom sayıları:

> *"…it turns out it's binomial numbers — N 0, N 1, up to N m." — Strang, 47:00*

$$r(N,m) = \binom{N}{0} + \binom{N}{1} + \cdots + \binom{N}{m}$$

$m = 2$ (düzlem) için: $r = \binom{N}{0} + \binom{N}{1} + \binom{N}{2} = 1 + N + N(N-1)/2$. ($N = 4 \to 1 + 4 + 6 = 11$ ✓.) Şekil 33.7 formülün sayısal bölge sayımıyla birebir $(2, 4, 7, 11, 16, 22)$ örtüştüğünü ve derinliğin parça sayısını üstel büyüttüğünü gösterir. Parça sayısı boyut m'de N ile **polinom**, ama katman sayısıyla (derinlikle) bileşke alındığında **üstel** büyür — derin ağların ifade gücü (expressivity) buradan.

> **Builder Notu — Binomun İfade Gücü**
>
> "Parça sayısı = binom toplamı" ağın geometrik kapasitesini ölçer. ML köprüsü: derin (çok katmanlı) ağların sığ (tek katman, çok nöron) ağlardan daha verimli olmasının sebebi — derinlik parça sayısını



Şekil 33.7: Parça sayımı r(N,m) = binom toplamı: formül sayısal bölge sayımıyla BİREBİR (2,4,7,11,16,22). Sol — N=1..6 katlama için formül r(N,2)=[2,4,7,11,16,22] (navy çubuk); ızgara sign-pattern sayımı (turuncu) tam üstüne oturur (6/6 birebir). Sağ — boyut m=1/2/3 için r(N,m): tek katman N'de polinom kalır; derinlik (kompozisyon) parça sayısını üstel büyütür.

üstel artırır (her katman önceki katlamaları çarpar), sığlık sadece polinom. "Neden derin?" sorusunun matematiksel cevabı.

33.9 Derinlik ve Epoch

Neden çok katman? Katmanlar bilgiyi ayrıştırır:

> *"…you can separate what layer one learns about the data and from what layer two learns…" — Strang, 22:29*

Katman 1 temel özellikleri (kenarlar), katman 2 daha karmaşık desenleri (şekiller), katman 3 daha soyut kavramları öğrenir — hiyerarşik temsil. Eğitim **epoch**'larla ölçülür:

> *"…one epoch is the number of steps that matches [the training data size]…" — Strang, 12:58*

Bir epoch = SGD adımlarının (mini-batch'lerin) tüm eğitim verisini bir kez taraması. Spiral gibi zor problemler binlerce epoch ister.

Builder Notu — Kenardan Kavrama

"Katman 1 kenar, katman 2 şekil, katman 3 kavram" hiyerarşik özellik öğrenme derin öğrenmenin sezgisel gücü. ML köprüsü: CNN'lerde (Ders 32) bu somut — ilk katmanlar kenar/doku, son katmanlar nesne dedektörleri; transfer öğrenme bu hiyerarşiyi yeniden kullanır. Epoch, learning rate schedule ve early stopping eğitim döngüsünün pratik parametreleri.

33.10 Bu Dersin Özeti

- **Öğrenme fonksiyonu $F(x)$:** eğitim verisini öğrenir; sınıflandırmada $F < 0 \to -1$, $F > 0 \to +1$. Her noktayı doğru yapma (aşırı-öğrenme).
- **ReLU:** $\text{ReLU}(x) = \max(0, x)$; doğrusal-olmamazlık şart (yoksa derinlik tek matrise çöker).
- **Katman:** $y = Ax + b$ (affine, ağırlık matrisi + bias) $\to x = \text{ReLU}(y)$ (bileşen-bileşen).
- **Kompozisyon:** $F = F_3(F_2(F_1(x)))$, $F_k(v) = \text{ReLU}(A_k v + b_k)$; zincir → zincir kuralı → backprop (Ders 27).
- **F sürekli parçalı-doğrusal:** origami; ReLU katlama çizgileri yaratır.
- **Universality:** yeterli katlamayla her sürekli fonksiyon yaklaşıklanabilir.
- **Parça sayımı:** $r(N, m) = \sum \binom{N}{k}$; $m = 2 \to 1 + N + N(N-1)/2$; derinlik ifade gücünü üstel artırır (motor: $r(N, 2) = 2, 4, 7, 11, 16, 22$ ızgara sign-pattern sayımıyla birebir).

! Tek Bir Cümle

Bir sinir ağı, affine katmanların ($Ax+b$) ve doğrusal-olmayan ReLU'nun zincirlenmiş kompozisyonudur ($F = F_3 \circ F_2 \circ F_1$); sonuç F sürekli parçalı-doğrusal bir öğrenme fonksiyonudur, yeterli katmanla her sürekli fonksiyonu yaklaşıklayabilir (universality) ve derinlik parça sayısını üstel artırarak ifade gücünü büyütür.

33.11 Kontrol Soruları

i Soru 1 — Katmanın iki adımı ve ReLU neden zorunlu

Soru: Bir sinir ağı katmanı hangi iki adımdan oluşur, ve ReLU neden zorunludur?
Cevap: (1) **Affine** dönüşüm: $y = Ax + b$ (ağırlık matrisi $A\times$ girdi $+$ bias b). (2) **ReLU** aktivasyonu: $x = \text{ReLU}(y) = \max(0, y)$, bileşen-bileşen. ReLU zorunlu çünkü **doğrusal-olmamazlık** sağlar — onsuz tüm katmanlar doğrusal olurdu ve bileşkeleri tek bir matrise (tek düzleme) çökerdi; spiral gibi doğrusal-ayrılamaz verileri ayıramazdık.

i Soru 2 — F ne tür bir fonksiyon ve neden

Soru: $F(x) = F_3(F_2(F_1(x)))$ ne tür bir matematiksel fonksiyondur ve neden?
Cevap: F **sürekli parçalı-doğrusaldır**. Her katman affine (sürekli, doğrusal) + ReLU (sürekli, parçalı-doğrusal) bileşimidir; süreklilerin kompozisyonu sürekli, doğrusal/parçalı-doğrusalların kompozisyonu parçalı-doğrusaldır. Görsel: origami — düz parçalar kenarlarda (ReLU katlama çizgileri) birleşir.

i Soru 3 — Universality teoremi ve sınırı

Soru: Universality teoremi ne der, sınırı nedir?
Cevap: Yeterli nöron/katman (yeterli katlama) ile herhangi bir **sürekli** fonksiyon istenildiği kadar yakından yaklaşıklanabilir — düz parçalarla eğriyi istediğin hassasiyette örtersin. Sınır: bu "öğrenilebilir mi?" sorusunu (evet) çözer ama "verimli öğrenir mi, genelleşir mi?" ayrı sorulardır (gerek-koşul, yeterli-koşul değil).

i Soru 4 — Parça sayısı ve derinlikle büyüme

Soru: ReLU ağında düz parça sayısı nasıl hesaplanır, derinlikle nasıl büyür?
Cevap: m boyutta N katlama için $r(N,m) = \binom{N}{0} + \binom{N}{1} + \cdots + \binom{N}{m}$ (binom toplamı); $m = 2$'de $r = 1 + N + N(N-1)/2$ ($N = 4 \to 11$ parça). Tek katmanda parça sayısı N ile polinom; ama katmanlar bileşke alındığında (derinlik) **üstel** büyür — her katman önceki katlamaları çarpar. Bu, derin ağların sığ ağlardan neden daha ifade-güçlü olduğunu açıklar.

33.12 Egzersizler

1. **Parametre sayımı.** Girdi 5 boyut, gizli katman 8 nöron, çıktı 1. A_1 (8×5) $+b_1$ (8) ve A_2 (1×8) $+b_2$ (1) kaç ağırlık eder? Toplam parametre? (Motor tanığı: A_1 $40 + 8 = 48$, A_2 $8 + 1 = 9$, toplam $48 + 9 = 57$; bkz. `param_count([5,8,1])`.)

2. **ReLU hesabı.** $y = (3, -2, 0, 5)^\top$. ReLU(y) nedir? Kaç bileşen "öldü" (sıfırlandı)? (Motor tanığı: ReLU$(y) = (3, 0, 0, 5)$; 2 bileşen (-2 ve 0) sıfırlandı; bkz. Şekil 33.2 sağ panel.)

3. **Parça sayımı.** $m = 2$ düzlem için $N = 5$ katlama kaç parça verir? $r(5,2) = \binom{5}{0} + \binom{5}{1} + \binom{5}{2}$'yi hesapla. (Motor tanığı: $1+5+10 = 16$; ders dizisi $r(N,2) = 2, 4, 7, 11, 16, \ldots$ ile uyumlu; $r(N,1) = N+1$.)

4. **Doğrusal-olmamazlık.** İki katmanı ReLU olmadan birleştir: $A_2(A_1 x) = (A_2 A_1)x$. Bunun neden tek bir doğrusal katmana eşdeğer olduğunu ve spirali ayıramayacağını açıkla. (Motor tanığı: $A_2(A_1 x) = (A_2 A_1)x$ maxdiff $\approx 10^{-12}$; iç-içe elmas verisinde en iyi doğrusal acc < 0.75, 4-ReLU ağ acc $= 1.000$; bkz. Şekil 33.3.)

5. **(Ders 27 habercisi)** $F = F_3(F_2(F_1(x)))$ bir kompozisyon; SGD bunu eğitmek için ∇F (ağırlıklara göre gradyan) gerektirir. Kompozisyonun türevi hangi kuralla hesaplanır? Bu hesabı verimli organize eden algoritma ne? Bir tahmin yaz — Ders 27 "backpropagation: kısmi türevleri bulmak" ile zincir kuralının matris organizasyonunu işliyor.

33.13 Sonraki Ders İçin Hazırlık

Ders 27: Backpropagation — Kısmi Türevleri Bulmak. SGD'nin ihtiyaç duyduğu gradyanı (∇F, ağırlıklara göre) verimli hesaplama: zincir kuralının organize edilmiş hâli. $F = F_3 \circ F_2 \circ F_1$ kompozisyonunun türevi, katman Jacobian'larının çarpımıdır. **Phase 2'nin dört tanığı buluşur:** Strang'ın matris-çarpım dili = Karpathy micrograd `_backward` = fast.ai manuel `.g` = NYU Jacobian zinciri — dört yol, aynı backprop.

⚠ Hazırlık

Bu dersteki Egzersiz 5'in sorduğu soruyu zihninde tut: $F = F_3 \circ F_2 \circ F_1$ kompozisyonunun ağırlıklara göre türevi hangi kuralla, hangi algoritmayla verimli hesaplanır? Ders 27 cevabı veriyor — zincir kuralının matris-çarpım hâli backpropagation. Bu, Phase 2 boyunca dört ayrı kursta (18.065, Karpathy, fast.ai, NYU) gördüğümüz aynı fikrin tek noktada buluştuğu yerdir; backprop'un "neden çarpım" olduğunu Ders 26'nın kompozisyon zinciri ($F = F_3 \circ F_2 \circ F_1$) önceden açıklar.

33.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Öğrenme fonksiyonu F	$F < 0 \to$ sınıf -1, $F > 0 \to +1$	1m41
ReLU	$\max(0, x)$; doğrusal-olmamazlık şart	10m13
Katman	$y = Ax + b$ (affine) $\to x = \text{ReLU}(y)$	20m23
Kompozisyon	$F = F_3(F_2(F_1(x)))$; $F_k = \text{ReLU}(A_k v + b_k)$	25m44
Sürekli parçalı-doğrusal	origami; ReLU katlama çizgileri	29m04
Universality	her sürekli fonksiyon yaklaşıklanabilir	39m41
Parça sayımı	$r(N, m) = \binom{N}{0} + \cdots + \binom{N}{m}$	47m00
Derinlik	katman 1 kenar, 2 şekil, 3 kavram	22m29
Epoch	veri kümesinin bir tam geçişi	12m58

33.15 ML Bağlantıları Özeti

- F **= ağırlık matrisleri + ReLU zinciri:** PyTorch `nn.Sequential(nn.Linear, nn.ReLU, …)`; SGD (Ders 25) A'ları ayarlar.
- **ReLU = ifade gücü:** doğrusal-olmamazlık olmadan derinlik anlamsız; sigmoid→ReLU geçişi derin öğrenme devrimini tetikledi.
- **Universality + parça sayımı:** derin ağlar her fonksiyonu öğrenebilir; derinlik parça sayısını üstel artırır (sığ ağlardan verimli).
- **Hiyerarşik özellik:** katman 1 kenar, katman 2 şekil, katman 3 kavram (CNN'lerde somut, Ders 32); transfer öğrenme bu hiyerarşiyi yeniden kullanır.
- **Kompozisyon → backprop:** $F = F_3 \circ F_2 \circ F_1$ türevi zincir kuralıyla (Ders 27); Karpathy micrograd + fast.ai manuel backprop + NYU Jacobian zinciri ile aynı.
- **Parçalı-doğrusal geometri:** ReLU ağı girdi uzayını polytope bölgelere böler; yorumlanabilirlik araştırmasının konusu.
- **Geriye köprü:** Ders 25 (SGD, f_i = NN kaybı), Ders 21 (zincir kuralı → backprop), Ders 2 (kompozisyon/çarpanlama). Paralel: NYU H1 (spiral sınıflandırma), Karpathy micrograd (fonksiyon kompozisyonu).

! Kapanış

> *"…constructing this function F which learns the training data…" — Strang, 1:41*

Bir sinir ağı, affine katmanlar ve ReLU'nun kompozisyonundan ibarettir; bu basit yapı ($Ax + b$, sonra $\max(0, \cdot)$) yeterince derinleştiğinde her fonksiyonu öğrenebilen evrensel bir makine olur.

34 Backpropagation — Kısmi Türevleri Bulmak

Ters-mod otomatik türev, matris-zinciri sırası ve dört tanığın buluşması

> **Bölüm bilgisi**
>
> Bu ders, SGD'nin (Ders 25) her adımda ihtiyaç duyduğu **gradyanı** verimli hesaplayan algoritmayı kurar: **backpropagation** = ters-mod otomatik türev (reverse-mode AD); zincir kuralının matris-çarpım hâli. Strang'in Ders 27 videosu (≈52 dk) ve OCW Lecture 27 temel alınmıştır. Okuma süresi ≈34 dk; önkoşul Ders 26 (sinir ağı = kompozisyon $F = F_3 \circ F_2 \circ F_1$). Bu ders optimizasyon ve derin öğrenme bloğunun kapanışıdır.

34.1 Bu Derste Ne Var?

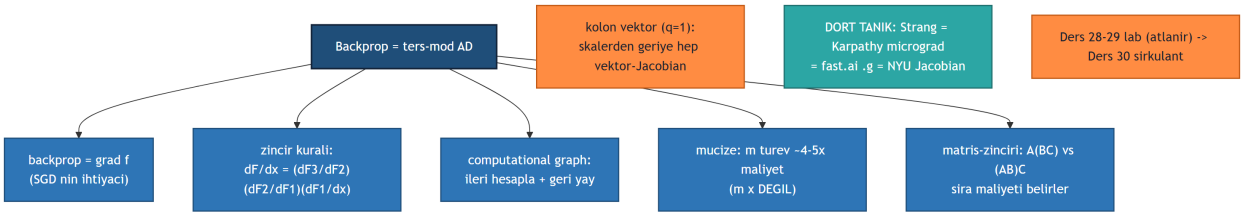
SGD'nin (Ders 25) her adımda ihtiyacı olan **gradyanı** verimli hesaplama: **backpropagation** = ters-mod otomatik türev (reverse-mode AD). Bu, sinir ağlarını haritaya koyan hesap.

Beş sonuç:

1. **Backprop = grad f:** tüm kısmi türevler $(\partial F/\partial x_1, \ldots, \partial F/\partial x_m)$; her steepest descent adımı gradyan ister.
2. **Zincir kuralı:** $F = F_3(F_2(F_1(x)))$ türevi $dF/dx = (dF_3/dF_2)(dF_2/dF_1)(dF_1/dx)$.
3. **Computational graph:** ileri (F'yi hesapla) + geri ($dF/dF = 1$'den başlayıp türevleri geri yay).
4. **Mucize:** ters-mod, m değişkenin TÜM türevlerini $m\times$ değil **~4-5×** maliyetle hesaplar (zincir parçaları yeniden kullanılır).
5. **Neden hızlı:** matris-zinciri $(AB)C$ vs $A(BC)$ — sıra önemli; ters-mod **doğru sıra** (skaler çıktıdan geriye).

> *"…you could compute n first derivatives with about four or five times the cost, not n times." — Strang, 30:03*

Şekil 34.1 dersin iskeletini gösterir: merkezdeki "backprop = ters-mod AD" fikrinden, SGD'nin ihtiyacı olan grad f'yi zincir kuralıyla hesaplamaya, ileri hesaplayıp geri yayan computational graph'a, m türevi $m\times$ değil ~4-5× maliyetle veren mucizeye ve sıranın maliyeti belirlediği matris-zincirine $(A(BC)$ vs $(AB)C)$ dallanır; ayrı düğümlerde skalerden geriye hep vektör-Jacobian olan kolon-vektör durumu, DÖRT TANIK (Strang = Karpathy micrograd = fast.ai `.g` = NYU Jacobian) buluşması ve Ders 28-29 lab (atlanır) → Ders 30 sirkülant köprüsü durur.



Şekil 34.1: Ders 27 kavram haritasi: backprop = ters-mod otomatik turev. SGD nin ihtiyaci olan grad f yi zincir kuraliyla hesaplar; computational graph ileri hesaplar sonra geri yayar; mucize m turev ~4-5x maliyetle (m x DEGIL) gelir; matris-zinciri A(BC) vs (AB)C sira maliyeti belirler. Kolon vektor (q=1): skalerden geriye hep vektor-Jacobian. DORT TANIK: Strang = Karpathy micrograd = fast.ai .g = NYU Jacobian. Ders 28-29 lab (atlanir) -> Ders 30 sirkulant.

Builder Notu — Dört Yol Tek Gradyan

Backprop = **autograd** — PyTorch/TensorFlow/JAX'in `.backward()`'ı; reverse-mode AD, vektör-Jacobian çarpımları (VJP). Derin öğrenmeyi pratik kılan algoritma (Hinton + öncesi AD). **DÖRT TANIK BULUŞUR (Phase 2 sentezi):** Strang'ın matris-zinciri = Karpathy micrograd `_backward` = fast.ai manuel `.g` = NYU Jacobian zinciri — dört yol, aynı backprop. **Matris-zinciri sırası** ($(AB)C$ vs $A(BC)$ maliyet farkı) ters-mod'un neden verimli olduğunun özü; skaler kayıptan geriye çarpmak = vektör×matris (ucuz), matris×matris değil (pahalı). **Geriye köprü:** Ders 25 (SGD gradyan ister), Ders 26 (F kompozisyon), Ders 21 (zincir kuralı/Jacobian), Ders 2 (kompozisyon).
Tek cümle: Backpropagation, $F = F_3 \circ F_2 \circ F_1$ kompozisyonunun gradyanını zincir kuralını ters sırada (skaler çıktıdan geriye) uygulayarak hesaplar; bu matris-zinciri "doğru sıra" seçimi, tüm türevleri tek değişkeninkinin sadece 4-5 katı maliyetle verir — derin öğrenmeyi mümkün kılan algoritma.

34.2 Backprop = Gradyan Hesaplama

Backpropagation tek bir iş yapar: gradyanı hesaplamak.

> *"This is all to compute grad f..." — Strang, 8:39*

Tüm kısmi türevler $\partial F/\partial x_1, \ldots, \partial F/\partial x_m$. Steepest descent'in (Ders 22) her adımı gradyan ister; hızlı hesaplayamazsan battın. Çözüm:

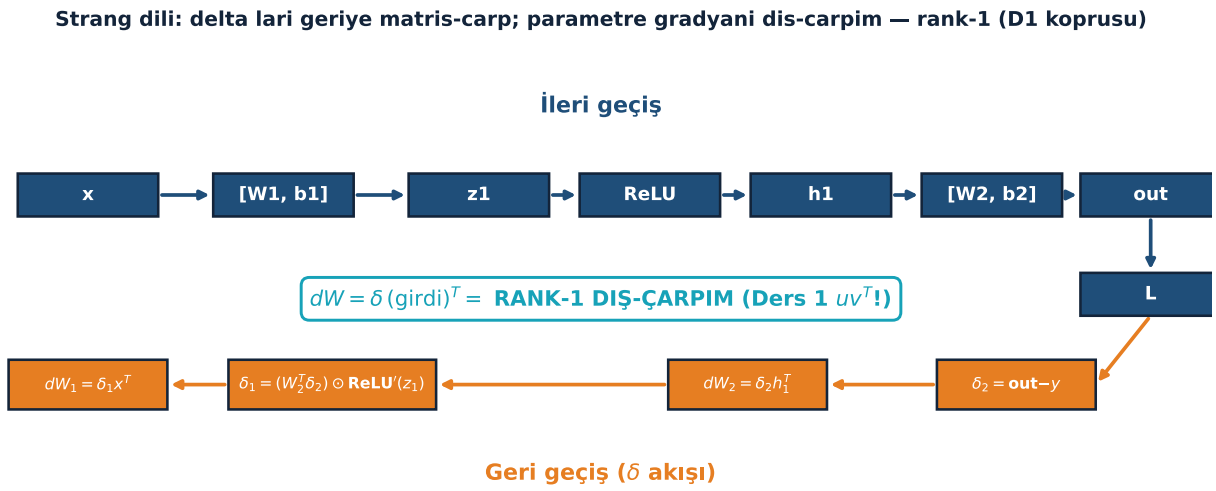
> *"...automatic differentiation in reverse mode..." — Strang, 9:45*

$$\nabla F = \left(\frac{\partial F}{\partial x_1}, \ldots, \frac{\partial F}{\partial x_m} \right)$$

Backprop = **ters-mod otomatik türev** (reverse-mode AD). Sinir ağlarını haritaya koyan keşif buydu (Hinton'a büyük pay; ama daha önce "Automatic Differentiation" adıyla çalışılmıştı). Hızlı gradyan = uygulanabilir derin öğrenme.

Şekil 34.2 Strang'ın matris diliyle aynı hesabı verir: ileri geçişte $x \to [W_1, b_1] \to z_1 \to \text{ReLU} \to h_1 \to [W_2, b_2] \to \text{out} \to L$ akar; geri geçişte δ'lar geriye matris-çarpılır ($\delta_2 = \text{out} - y$, $\delta_1 = (W_2^\top \delta_2) \odot$

ReLU$'(z_1)$) ve parametre gradyanı $dW = \delta\,(\text{girdi})^\top$ bir **rank-1 dış-çarpım** olur — bu tam Ders 1'in $uv^\top$ köprüsü.



Şekil 34.2: Strang δ-geri-yayılımı şeması — δ'ları geriye matris-çarpımıyla yay; parametre gradyanı dW = δ·(girdi)^T = rank-1 dış-çarpım (Ders 1 uv^T köprüsü).

> **Builder Notu — Eğitim Değil Türev Hesabı**
>
> "Backprop sadece gradyan hesaplar" — eğitim algoritması SGD'dir (Ders 25), backprop ise SGD'nin ihtiyacı olan gradyanı sağlar. ML köprüsü: bazıları "backprop" deyince tüm eğitimi kasteder, ama backprop yalnızca türev-hesaplama algoritmasıdır; PyTorch'ta `loss.backward()` tam bu, `optimizer.step()` ise SGD adımı.

34.3 Zincir Kuralı: F = $F_3(F_2(F_1(x)))$

Ders 26'dan: F bir kompozisyon. Türevini zincir kuralı verir. Strang'ın basit örneği (Christopher Olah'ın blogundan ilham):

> *"…x cubed times x plus 2y." — Strang, 12:04*

$F = x^3(x + 2y)$, iki fonksiyonun çarpımı (çarpım kuralı + kuvvet kuralı + doğrusal kombinasyon). Kompozisyon $F = F_3(F_2(F_1(x)))$ için zincir kuralı:

> *"…of course, the chain rule." — Strang, 14:58*

$$\frac{dF}{dx} = \frac{dF_3}{dF_2}\Big|_{F_2(F_1(x))} \cdot \frac{dF_2}{dF_1}\Big|_{F_1(x)} \cdot \frac{dF_1}{dx}\Big|_x$$

Her çarpan doğru noktada değerlendirilir. Bu, kalkülüsün özü: birkaç temel türevi (x^n, sin, e^x) bil, gerisini zincir kuralıyla birleştir. Bu hesabın somut grafik hâlini Şekil 34.3 bir sonraki bölümde gösterir.

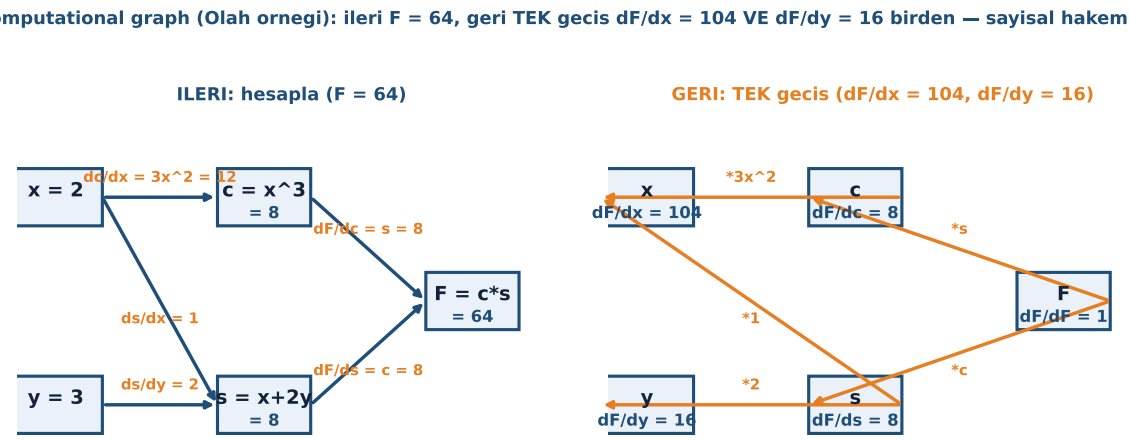
> **Builder Notu — Kalkülüsün Tek Numarası**
>
> "F kompozisyon → türev zincir kuralı" backprop'un matematiksel çekirdeği. ML köprüsü: her sinir ağı katmanı bir fonksiyon; gradyan, katman türevlerinin (Jacobian'ların) zincir-çarpımıdır. Karpathy'nin micrograd'ı tam bunu yapar — her işlem bir düğüm, türevi yerel olarak bilinir, zincir kuralı bağlar.

34.4 Computational Graph: İleri ve Geri

Örneği bir **hesaplama grafına** (computational graph) çevir. $F = x^3(x + 2y)$ için ara değişkenler: $c = x^3$ ve $s = x + 2y$, sonra $F = c \cdot s$. **İleri geçiş** (forward): x, y'den c, s, F'yi sırayla hesapla. **Geri geçiş** (backward): çıktıdan başla, türevleri geriye yay:

$$\frac{dF}{dF} = 1, \quad \frac{dF}{dc} = s, \quad \frac{dF}{ds} = c$$

($F = c \cdot s$ çarpım olduğundan $\partial F/\partial c = s$, $\partial F/\partial s = c$.) Sonra yerel türevler $\partial c/\partial x$, $\partial c/\partial y$, $\partial s/\partial x$, $\partial s/\partial y$ ile çarpıp zinciri tamamla. Kilit: **tek bir geri geçiş**, hem dF/dx hem dF/dy'yi birden verir — her değişken için ayrı zincir değil.

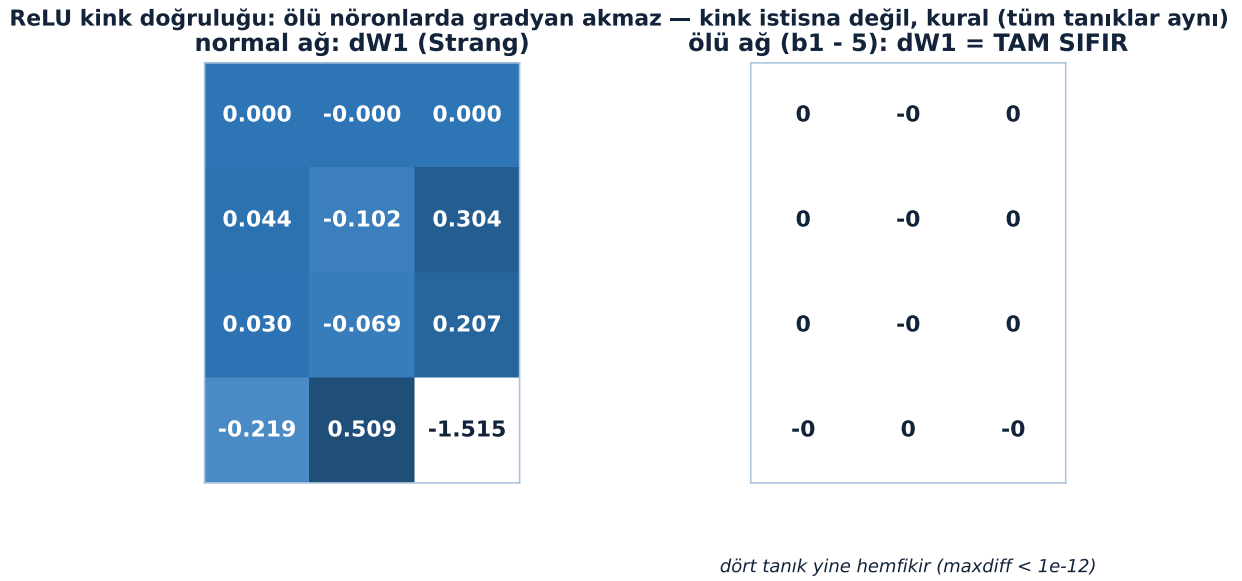


Şekil 34.3: Computational graph (Olah örneği): ileri F = 64, geri TEK geçiş dF/dx = 104 VE dF/dy = 16 birden — sayısal hakem birebir.

Şekil 34.3 sol panelde ileri geçişi ($x = 2, y = 3 \to c = 8, s = 8, F = 64$) ve sağ panelde geri geçişi gösterir: $dF/dF = 1$'den başlayıp tek geçişte $dF/dx = 104$ VE $dF/dy = 16$ birden çıkar; motor ile merkez-fark sayısal hakem birebir örtüşür ($dF/dx = s \cdot 3x^2 + c = 8 \cdot 12 + 8 = 104$, $dF/dy = 2c = 16$).

Bu doğruluk yalnızca düzgün noktalarda değil, ReLU'nun **kink** (kırılım) noktalarında da geçerli: ölü nöronlarda gradyan akmaz, ve dört tanık bu istisnai durumda bile hemfikirdir.

Şekil 34.4 solda normal ağın dW_1 gradyanını (Strang δ dış-çarpımı), sağda $b_1 - 5$ ile tüm ReLU'lar öldürülünce dW_1 = TAM SIFIR olduğunu gösterir — dört tanık yine hemfikir (maxdiff $< 10^{-12}$). Kink istisna değil, kuraldır.



Şekil 34.4: ReLU kink doğruluğu: ölü nöronlarda gradyan akmaz — kink istisna değil, kural. Solda normal ağın dW1 gradyanı (Strang δ dış-çarpımı); sağda b1−5 ile tüm ReLU ölü olunca dW1 = TAM SIFIR. Dört tanık yine hemfikir (maxdiff < 1e-12).

> **Builder Notu — Bir Geri Geçiş Hepsi**
>
> "İleri hesapla, geri türev yay" computational graph backprop'un iskeleti. ML köprüsü: PyTorch dinamik graf kurar (her işlem bir düğüm), `.backward()` grafı ters topolojik sırada gezip her düğümde yerel türevi zincirler. Karpathy micrograd'da `_backward()` tam bu — her düğüm kendi katkısını geriye ekler.

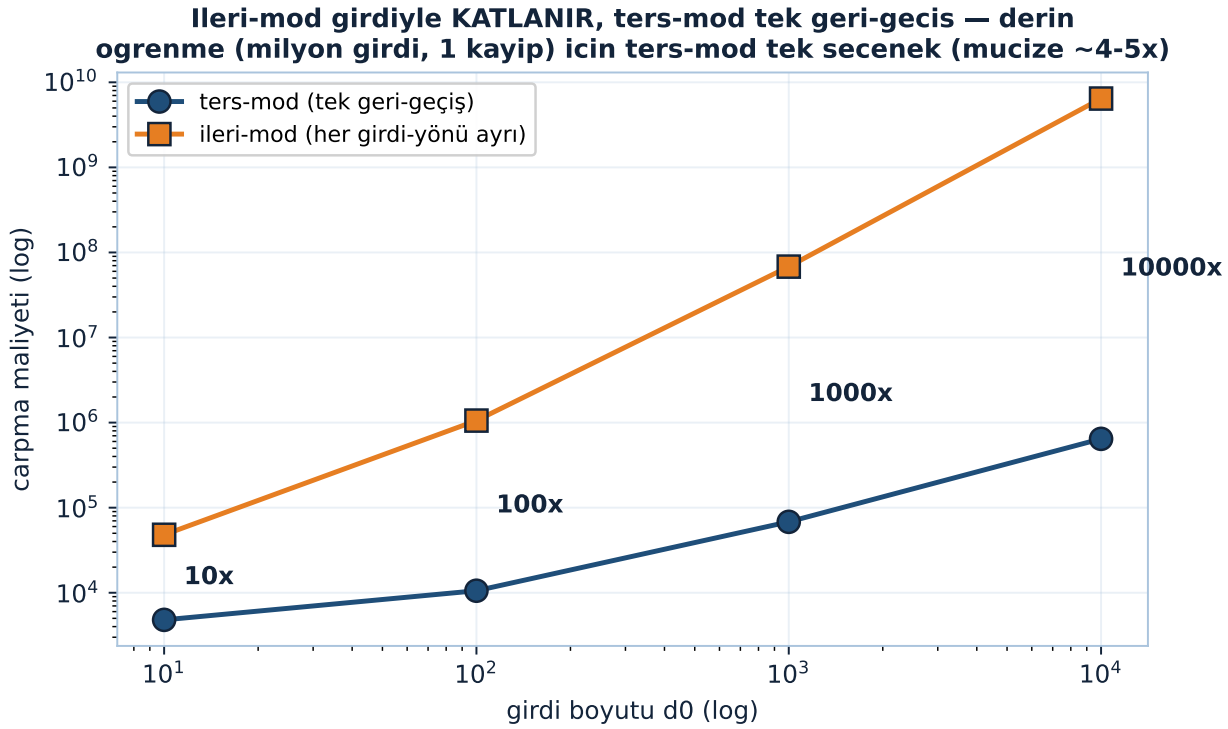
34.5 Ters-Mod Mucizesi

Asıl şaşırtıcı sonuç maliyet hakkında:

> *"…you could compute n first derivatives with about four or five times the cost, not n times." — Strang, 30:03*

m değişkenli bir fonksiyonun TÜM kısmi türevlerini, tek değişkeninkinin sadece **4-5 katı** maliyetle hesaplarsın — m katı değil. 100 değişken için 100 zincir kuralı beklerken, ters-mod bunu ~5 kat maliyetle verir. Neden? Çünkü geriye giderken zincirin **parçalarını yeniden kullanıyoruz**: zincir "daha geniş" ama "daha uzun değil", tekrarlanmıyor. İleri-mod (her girdiye ayrı zincir) m kat maliyetli olurdu; ters-mod (skaler çıktıdan geriye) hepsini tek seferde toplar.

Şekil 34.5 bunu ölçek taramasıyla kanıtlar: zincir $[d_0, 64, 64, 1]$ için girdi boyutu d_0 büyüdükçe ters-mod (navy) tek geri-geçiş kalır, ileri-mod (turuncu) ise her girdi-yönü için zinciri yeniden koşar ve d_0 katına katlanır; oran her noktada tam girdi sayısına eşittir ($d_0 = 1000 \rightarrow 1000\times$). Derin öğrenmede girdi milyonlarca parametre, çıktı tek kayıptır — ters-mod tek seçenektir.



Şekil 34.5: Ters-mod maliyetinin girdi-bağımsızlığı: zincir $[d_0, 64, 64, 1]$ için girdi boyutu d_0 taranıyor. Ters-mod (navy) tek geri-geçiştir; maliyeti d_0 ile doğrusal artar ama hep TEK geçiş kalır (4800 → 644160 çarpma). İleri-mod (turuncu) ise her girdi-yönü için zinciri yeniden koşar, bu yüzden ters-mod maliyetinin d_0 katına KATLANIR. Oran her noktada tam girdi sayısına eşittir: $d_0 = 10 \to 10\times$, $d_0 = 100 \to 100\times$, $d_0 = 1000 \to 1000\times$ (68.160 vs 68.160.000), $d_0 = 10000 \to 10000\times$. Derin öğrenmede girdi milyonlarca parametre, çıktı tek kayıptır — ters-mod tek seçenektir, mucizesi tüm gradyanı bir geri-geçişte (~4-5× ileri-geçiş maliyeti) vermesidir.

Builder Notu — Dört-Beş Kat Mucizesi

"Tüm gradyan, tek türevin 4-5 katı maliyeti" backprop'un neden devrim olduğunu açıklar: milyon-parametreli ağda gradyanı bir ileri+bir geri geçişle alırsın (milyon ayrı türev değil). ML köprüsü: bu yüzden büyük modeller eğitilebilir; ileri-mod AD (girdi sayısıyla ölçeklenir) küçük-girdi/çok-çıktı için, ters-mod (çıktı sayısıyla ölçeklenir) çok-girdi/skaler-çıktı (kayıp) için — derin öğrenme tam ikinci durum.

34.6 Neden Hızlı: Matris-Zinciri Sırası

Ters-mod'un neden hızlı olduğunu daha basit bir örnek gösterir: üç matrisi çarpmak.

> *"…one way could be way faster than another way is when I'm multiplying three matrices." — Strang, 34:53*

ABC çarpımı (A: $m \times n$, B: $n \times p$, C: $p \times q$). İki sıra, aynı sonuç, çok farklı maliyet:

$$A(BC) : \ npq + mnq \qquad (AB)C : \ mnp + mpq$$

Matris çarpımı **birleşmelidir** (associative) — sonuç aynı — ama işlem sayısı sıraya göre 1000 kat değişebilir. Hangi sıra ucuz? Boyutlara bağlı. Zincir kuralı da bir matris (Jacobian) çarpımı zinciridir; doğru sırada çarpmak backprop'un sırrı. Sonraki bölümde bu farkı kolon-vektör durumunda Şekil 34.6 sayılarla gösterir.

Builder Notu — Birleşmeli Ama Eşit-Maliyetli Değil

"Matris çarpımı birleşmeli ama maliyet sıraya bağlı" backprop'un derin sebebi. ML köprüsü: backprop = Jacobian'ların zinciri $J_3 J_2 J_1$; ters-mod bunları **sağdan-sola değil**, çıktıdan (skaler kayıp) başlayıp **sola doğru** vektör-Jacobian çarpımlarıyla hesaplar — her adım vektör×matris (ucuz). Matris-zinciri optimizasyonu (matrix chain order) klasik dinamik programlama problemidir (6.006 paralel).

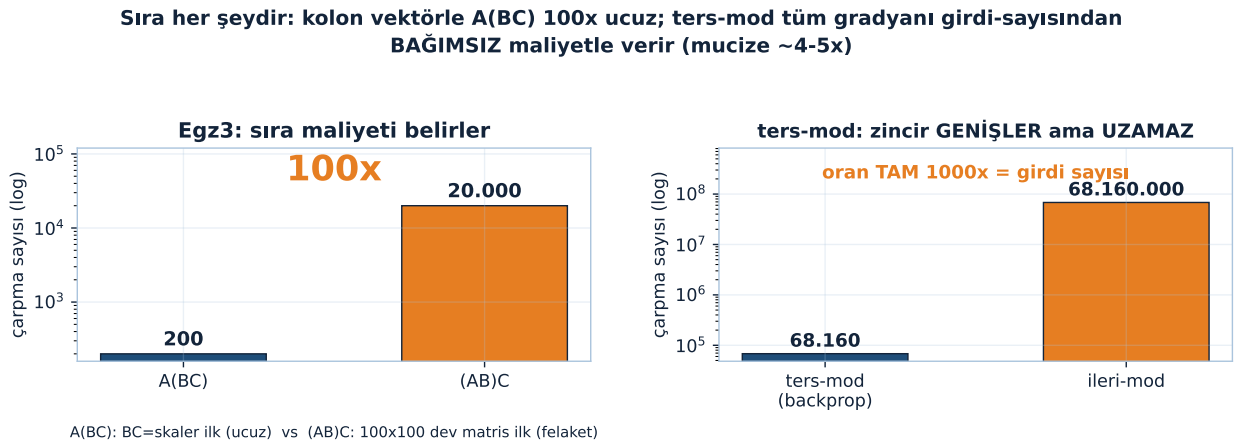
34.7 Kolon Vektör ve Ters-Mod

En önemli özel durum: C bir **kolon vektör** ($q = 1$). O zaman iki sıra dramatik ayrışır:

$$A(BC) : \ n(m+p) \quad \text{(ucuz)} \qquad\qquad (AB)C : \ mp(n+1) \quad \text{(felaket)}$$

$A(BC)$: önce BC (matris × vektör → vektör), sonra $A\times$ vektör → vektör. Hep matris-vektör, ucuz. $(AB)C$: önce AB (matris × matris → **büyük** $m \times p$ matris!), sonra × vektör. Büyük matris çarpımı boşuna — felaket.

> *"…back propagation is the right order…" — Strang, 52:16*



Şekil 34.6: Sıra her şeydir: kolon vektörle A(BC) 100x ucuz; ters-mod tüm gradyanı girdi-sayısından BAĞIMSIZ maliyetle verir (mucize ~4-5x). Sol — Egz3: A(BC)=200 çarpma vs (AB)C=20.000 (BC=skaler ilk ucuz; AB=100x100 dev matris ilk felaket). Sağ — zincir [1000, 64, 64, 1]: ters-mod 68.160 vs ileri-mod 68.160.000, oran TAM 1000x = girdi sayısı; ters-mod zincir genişler ama uzamaz.

Backprop tam $A(BC)$ sırasını seçer: skaler çıktıdan (kayıp) başla, geriye doğru her katmanın Jacobian'ıyla vektör çarp — asla büyük Jacobian matrislerini açıkça oluşturma. İşte ters-mod'un "doğru sıra"sı.

Şekil 34.6 solda Egzersiz 3'ün maliyetini gösterir: A (100×1), B (1×100), C (100×1) için $A(BC) = 200$ çarpma vs $(AB)C = 20.000$ — BC önce hesaplanırsa skaler (ucuz), AB önce hesaplanırsa 100×100 dev matris (felaket); tam **100×** fark. Sağda zincir $[1000, 64, 64, 1]$ için ters-mod 68.160 vs ileri-mod $68.160.000$, oran tam **1000×** = girdi sayısı; ters-mod zincir genişler ama uzamaz.

> **Builder Notu — Skalerden Geriye Hep Vektör**
>
> "Kolon vektörle başla, geriye matris-vektör çarp" ters-mod AD'nin tüm verimliliği buradan. ML köprüsü: derin öğrenmede çıktı bir skaler kayıptır ($q = 1$); ters-mod (backprop) bu yüzden ideal — gradyanı bir geri geçişte, hiç büyük Jacobian oluşturmadan toplar. İleri-mod büyük ara matrisleri oluştururdu (felaket). Bu, JAX/PyTorch'un `grad/.backward()` tasarımının matematiksel gerekçesi.

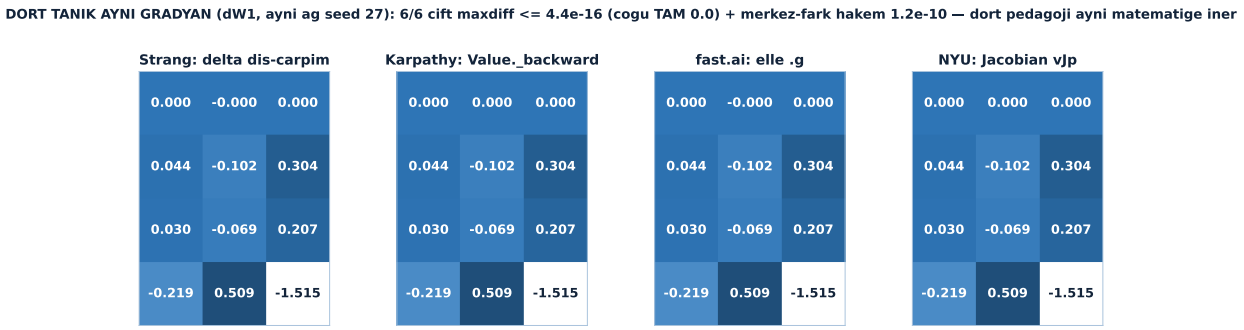
34.8 Dört Tanık Buluşur (Phase 2 Sentezi)

Backpropagation, Phase 2'nin "**farklı yollar aynı yere varır**" tezinin doruğudur. Dört ayrı ders, dört ayrı dil, **tek bir backprop**:

Tanık	Dil	Backprop nasıl görünür
Strang (Ders 27)	Matris/lineer cebir	Zincir kuralı = Jacobian-matris çarpımı; ters-mod = doğru matris-zinciri sırası
Karpathy micrograd	Kod (Python)	Her `Value` düğümünde `_backward()`; ters topolojik geçiş
fast.ai (L13-14)	Elle "from scratch"	Her tensörde `.g` gradyanı; geriye elle hesap

Tanık	Dil	Backprop nasıl görünür
NYU (H2, LeCun)	Teori/Jacobian	Her katman bir Jacobian; backprop = Jacobian zincir-çarpımı

Dördü de aynı şeyi söyler: $F = F_3 \circ F_2 \circ F_1$ kompozisyonunun gradyanı, katman türevlerinin (Jacobian'ların) **ters sırada** çarpımıdır. Strang'ın matris-çarpım dili, Karpathy'nin `_backward`'ı, Howard'ın elle `.g`'si ve LeCun'un Jacobian zinciri — aynı matematik, dört pencere.



Şekil 34.7: DÖRT TANIK AYNI GRADYAN: aynı ağ (x(3)→W1(4×3)→ReLU→W2(1×4)→L, seed 27) için dW_1 gradyanı dört farklı pedagojiyle hesaplanıyor — Strang'ın delta dış-çarpımı, Karpathy'nin micrograd `Value._backward`'ı, fast.ai'nin elle `.g` gradyanı, NYU'nun Jacobian vJp'si. Dört panel ortak ölçekte (vmin/vmax) BİREBİR aynı: 6/6 çift maxdiff $\leq 4.4 \times 10^{-16}$ (çoğu TAM 0.0) + merkez-fark hakem 1.2×10^{-10}. Phase 2 tezinin sayısal ispatı — dört yol aynı matematiğe iner.

Şekil 34.7 bu tezin sayısal ispatıdır: aynı ağ (seed 27) için dW_1 gradyanı dört farklı pedagojiyle hesaplanır — Strang'ın delta dış-çarpımı, Karpathy'nin `Value._backward`'ı, fast.ai'nin elle `.g`'si, NYU'nun Jacobian vJp'si. Dört panel ortak ölçekte birebir aynıdır: 6/6 çift maxdiff $\leq 4.4 \times 10^{-16}$ (çoğu TAM 0.0) + merkez-fark hakem 1.2×10^{-10}. Dört pedagoji aynı matematiğe iner.

> **Builder Notu — Aynı Matematiğin Dört Penceresi**
>
> Bu dörtlü buluşma Phase 2'nin felsefesidir: aynı kavramı (backprop) matris cebiri, kod, elle hesap ve teori dillerinden görmek anlayışı sağlamlaştırır. ML köprüsü: bir kavramı dört yoldan tanımak, onu gerçekten "bilmek"tir — micrograd'ı yazabilmek (Karpathy), elle türetebilmek (fast.ai), matris diliyle ifade etmek (Strang) ve Jacobian olarak görmek (NYU) aynı backprop'un dört yüzü. Bir sonraki gördüğün autograd kütüphanesi bu dördünün de uygulaması.

34.9 Bu Dersin Özeti

- **Backprop = grad f:** tüm kısmi türevler ($\partial F/\partial x_i$); ters-mod otomatik türev (reverse-mode AD); SGD'nin gradyanını sağlar.
- **Zincir kuralı:** $F = F_3(F_2(F_1(x))) \rightarrow dF/dx = (dF_3/dF_2)(dF_2/dF_1)(dF_1/dx)$, her çarpan doğru noktada.
- **Computational graph:** ileri (F'yi hesapla) + geri ($dF/dF = 1$'den türevleri yay); tek geri geçiş tüm türevleri verir.

- **Mucize:** m değişkenin tüm türevleri $m\times$ değil ~4-5× maliyetle (zincir parçaları yeniden kullanılır).
- **Matris-zinciri:** ABC sırası maliyeti değiştirir; $A(BC)$ vs $(AB)C$. Kolon vektörle ($q = 1$) $A(BC)$ ucuz, $(AB)C$ felaket.
- **Ters-mod = doğru sıra:** skaler kayıptan geriye, vektör-Jacobian çarpımları (büyük Jacobian oluşturma).
- **Dört tanık:** Strang matris-zinciri = Karpathy micrograd `_backward` = fast.ai manuel `.g` = NYU Jacobian zinciri.

! Tek Bir Cümle

Backpropagation, $F = F_3 \circ F_2 \circ F_1$ kompozisyonunun gradyanını zincir kuralını ters sırada (skaler çıktıdan geriye, vektör-Jacobian çarpımlarıyla) uygulayarak hesaplar; bu "doğru matris-zinciri sırası" tüm türevleri tek değişkeninkinin sadece 4-5 katı maliyetle verir — derin öğrenmeyi mümkün kılan algoritma.

34.10 Kontrol Soruları

i Soru 1 — Backprop ne hesaplar ve eğitimdeki rolü

Soru: Backpropagation tam olarak ne hesaplar ve eğitimdeki rolü nedir?
Cevap: Backprop **gradyanı** hesaplar: tüm kısmi türevler $\nabla F = (\partial F/\partial x_1, \ldots, \partial F/\partial x_m)$, ters-mod otomatik türevle. Eğitim algoritması SGD'dir (Ders 25); backprop ise SGD'nin her adımda ihtiyaç duyduğu gradyanı sağlar. PyTorch'ta `loss.backward()` = backprop, `optimizer.step()` = SGD adımı. İkisi ayrı işlerdir.

i Soru 2 — İleri ve geri geçiş ne yapar

Soru: Computational graph'ta ileri ve geri geçiş ne yapar?
Cevap: **İleri geçiş** girdiden ara değişkenleri ($c = x^3$, $s = x + 2y$) ve çıktıyı ($F = c \cdot s$) hesaplar. **Geri geçiş** çıktıdan başlar ($dF/dF = 1$), türevleri ters topolojik sırada yayar ($dF/dc = s$, $dF/ds = c$, sonra yerel türevlerle çarp). Kilit: tek bir geri geçiş hem dF/dx hem dF/dy'yi (tüm gradyanı) birden verir — her değişken için ayrı zincir değil.

i Soru 3 — Neden m× değil ~4-5× maliyet

Soru: Ters-mod neden tüm türevleri $m\times$ değil ~4-5× maliyetle hesaplar?
Cevap: Geriye giderken zincirin **parçaları yeniden kullanılır** — zincir "daha geniş ama daha uzun değil", tekrarlanmaz. İleri-mod her girdiye ayrı zincir kurardı (m kat). Ters-mod skaler çıktıdan geriye tek geçişte tüm girdilere göre türevleri toplar; ortak ara hesaplar bir kez yapılır. Bu yüzden m değişkenin gradyanı tek türevin 4-5 katı maliyetinde.

i Soru 4 — A(BC) vs (AB)C kolon vektörde neden farklı

Soru: Matris-zinciri $A(BC)$ vs $(AB)C$, C kolon vektörken neden bu kadar farklıdır?
Cevap: $A(BC)$: önce BC (matris × vektör → vektör), sonra $A\times$ vektör → vektör — hep matris-vektör,

ucuz (maliyet $n(m+p)$). $(AB)C$: önce AB (matris × matris → büyük $m \times p$ matris!), sonra × vektör — büyük matris çarpımı boşuna, felaket ($mp(n+1)$). Backprop tam $A(BC)$ sırasını seçer: skaler kayıptan geriye, hiç büyük Jacobian oluşturmadan vektör-Jacobian çarpımları. Ters-mod'un "doğru sıra"sı budur.

34.11 Egzersizler

1. **Zincir kuralı.** $F(x) = \sin(x^2)$. Bunu $F_2(F_1(x))$ yaz ($F_1 = x^2$, $F_2 = \sin$). dF/dx'i zincir kuralıyla hesapla ($dF_2/dF_1 \cdot dF_1/dx$). (Motor tanığı: $dF/dx = 2x\cos(x^2)$; merkez-fark sayısal türevle birebir, fark $< 10^{-6}$.)

2. **Computational graph.** $F = x^3(x+2y)$. $c = x^3$, $s = x+2y$. $\partial F/\partial c$, $\partial F/\partial s$'yi yaz. Sonra $\partial c/\partial x$, $\partial s/\partial x$, $\partial s/\partial y$'yi yaz. Geri geçişle $\partial F/\partial x$ ve $\partial F/\partial y$'yi birleştir. (Motor tanığı: $x = 2, y = 3$ için $c = 8$, $s = 8$, $F = 64$; $\partial F/\partial x = s \cdot 3x^2 + c = 104$, $\partial F/\partial y = 2c = 16$; bkz. Şekil 34.3.)

3. **Matris-zinciri.** A (100×1), B (1×100), C (100×1). $A(BC)$ ve $(AB)C$ maliyetlerini (çarpma sayısı) hesapla. Hangisi kaç kat ucuz? (BC bir skaler mi?) (Motor tanığı: $A(BC) = n(m+p) = 200$ vs $(AB)C = mp(n+1) = 20.000$; BC önce skaler olduğundan $A(BC)$ tam **100×** ucuz; bkz. Şekil 34.6 sol panel.)

4. **İleri vs ters mod.** Bir fonksiyon 1000 girdi, 1 çıktı (skaler kayıp). Gradyanı (1000 türev) ileri-mod ve ters-mod ile hesaplamanın maliyetini karşılaştır. Hangisi derin öğrenme için doğru? (Motor tanığı: zincir $[1000, 64, 64, 1]$ için ters-mod 68.160 çarpma vs ileri-mod 68.160.000; oran tam **1000×** = girdi sayısı; ters-mod maliyeti girdi sayısından bağımsız, derin öğrenme için doğru seçim; bkz. Şekil 34.6 sağ panel.)

5. **(Sonraki — Ders 30)** Optimizasyon + DL bloğu kapandı. Ders 28-29 kayıtsız lab oturumlarıdır (atlanır). Ders 30 yeni bir konuya geçer: rank-1 matris tamamlama ve **sirkülant** matrisler. Bir sirkülant matrisin (her satır bir öncekinin kaydırılmışı) özvektörleri ne olabilir? Bir tahmin yaz.

34.12 Sonraki Ders İçin Hazırlık

Ders 30: Rank-Bir Matris Tamamlama, Sirkülantlar! (Ders 28-29 kayıt edilmemiş lab oturumlarıdır.) Optimizasyon/DL bloğu kapandı; Strang sinyal-işleme blokuna geçer: sirkülant matrisler (kaydırma yapısı), özvektörleri Fourier matrisidir (Ders 31), ve evrişimin (Ders 32 CNN) matris temeli.

⚠ Hazırlık

Bu dersin Egzersiz 5'inin sorduğu soruyu zihninde tut: her satırı bir öncekinin kaydırılmışı olan bir **sirkülant** matrisin özvektörleri ne olabilir? Ders 30, optimizasyon/DL bloğundan sinyal-işleme bloğuna geçişi başlatır; sirkülantların özvektörleri Fourier matrisi (Ders 31) ve bu yapı evrişimin (Ders 32 CNN) lineer-cebir temelidir. Backprop'un zincir kuralını bitirdik; sıradaki blok kaydırma simetrisini işliyor.

34.13 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Backprop = grad f	ters-mod otomatik türev (reverse-mode AD)	9m45
Zincir kuralı	$dF/dx = (dF_3/dF_2)(dF_2/dF_1)(dF_1/dx)$	14m58
Örnek	$F = x^3(x+2y)$; $c = x^3$, $s = x+2y$, $F = cs$	12m04
Mucize	m türev $m\times$ değil ~4-5× maliyet	30m03
Matris-zinciri	$A(BC)$ vs $(AB)C$; sıra maliyeti değiştirir	34m53
Kolon vektör (q=1)	$A(BC)$ ucuz, $(AB)C$ felaket; ters-mod doğru sıra	52m16
Kaynak	Christopher Olah blog (computational graph)	42m17
Dört tanık	Strang = Karpathy = fast.ai = NYU (aynı backprop)	—

34.14 ML Bağlantıları Özeti

- **Backprop = autograd:** PyTorch/TensorFlow/JAX `.backward()`/`grad`; reverse-mode AD, vektör-Jacobian çarpımları (VJP).
- **Ters-mod = çok-girdi/skaler-çıktı:** derin öğrenme tam bu (milyon parametre, 1 kayıp); ileri-mod küçük-girdi/çok-çıktı için. Maliyet çıktı sayısıyla ölçeklenir → kayıp skaler olduğundan ucuz.
- **Matris-zinciri sırası:** backprop = Jacobian'ları doğru sırada (skalerden geriye) çarpmak; büyük ara Jacobian'ları asla oluşturma.
- **Computational graph:** PyTorch dinamik graf; her işlem düğüm, `_backward` yerel türev (Karpathy micrograd birebir bu).
- **DÖRT TANIK (Phase 2 tezi):** Strang matris-zinciri = Karpathy `_backward` = fast.ai manuel `.g` = NYU Jacobian — aynı backprop, dört dil.
- **Hinton + AD:** backprop'un yeniden keşfi derin öğrenmeyi başlattı; "Automatic Differentiation" olarak önceden de vardı.
- **Geriye köprü:** Ders 25 (SGD gradyan ister), Ders 26 (F kompozisyon), Ders 21 (zincir kuralı/Jacobian), Ders 2 (kompozisyon). Paralel: Karpathy micrograd, fast.ai L13-14, NYU H2.

! Kapanış

> *"…back propagation is the right order…" — Strang, 52:16*

Backprop, zincir kuralını doğru matris sırasında (skaler çıktıdan geriye) uygulamaktan ibarettir; bu basit "sıra" seçimi tüm gradyanı tek geçişte verir ve derin öğrenmeyi mümkün kılan algoritmadır — optimizasyon ve DL bloğunun kapanışı.

35 Rank-Bir Matris Tamamlama, Sirkülantlar

Bipartite graph ile tamamlanabilirlik, cyclic shift P ve evrişimin matris hâli

> **i** Bölüm bilgisi
>
> Bu ders iki yeni konu açar: **rank-1 matris tamamlama** (bir matrisin bazı girdileri verilince kalanı rank-1 olacak şekilde doldurulabilir mi?) ve **sirkülant matrisler** (her satır bir öncekinin döngüsel kaydırılmışı — sinyal-işleme/CNN bloğunun temeli). Strang'in Ders 30 videosu (≈50 dk) ve OCW Lecture 30 temel alınmıştır. Okuma süresi ≈34 dk. **Ders 28-29 kayıt edilmemiş lab oturumlarıdır; sinyal-işleme bloğu bu derste başlıyor.** Önkoşul Ders 1 ($uv^\top$ dış-çarpım), Ders 16 (matris tamamlama) ve Ders 4 (özvektör).

35.1 Bu Derste Ne Var?

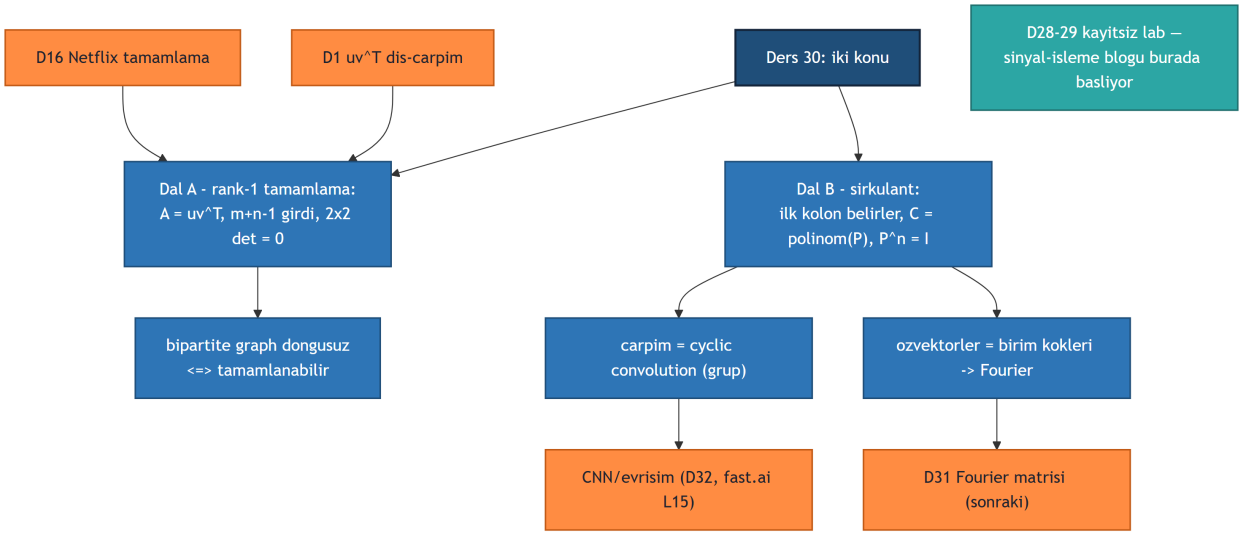
İki konu. **(A) Rank-1 matris tamamlama:** bir matrisin bazı girdileri verilince, kalanı rank-1 olacak şekilde doldurulabilir mi? **(B) Sirkülant matrisler:** her satır bir öncekinin döngüsel kaydırılmışı; sinyal-işleme/CNN bloğunun temeli.

Beş sonuç:

1. **Rank-1 tamamlama:** $A = uv^\top$, $a_{ij} = u_i v_j$; $m + n - 1$ serbest parametre. Rank-1 $\Leftrightarrow$ tüm 2×2 alt-determinantlar = 0.
2. **Bipartite graph:** satırlar + sütunlar düğüm, verilen girdiler kenar; tamamlanabilir $\Leftrightarrow$ **döngü (cycle) yok**.
3. **Sirkülant C:** ilk kolon tümünü belirler (cyclic shift P); $C = c_0 I + c_1 P + c_2 P^2 + c_3 P^3$, $P^n = I$.
4. **Sirkülant çarpımı = cyclic convolution:** sirkülantlar grup oluşturur (CD de sirkülant).
5. **Özvektörler = P'nin özvektörleri = birim kökleri** ($1, -1, i, -i$ for $n = 4$) → Fourier matrisi (Ders 31).

> *"…every circulant matrix is a polynomial in P…" — Strang, 32:45*

Şekil 35.1 dersin iskeletini gösterir: merkezdeki "iki konu" fikrinden iki dala ayrılır — Dal A (rank-1 tamamlama: $A = uv^\top$, $m + n - 1$ girdi yeter, her 2×2 alt-determinant sıfır; bipartite graph döngüsüz ise tamamlanabilir) ve Dal B (sirkülant: ilk kolon tüm matrisi belirler, $C = $ polinom(P), $P^n = I$; çarpım = cyclic convolution grup işlemi; özvektörler birim kökleri → Fourier); köprü düğümleri D1 $uv^\top$ dış-çarpım, D16 Netflix tamamlama, D31 Fourier matrisi (sonraki) ve CNN/evrişim (D32, fast.ai L15) iki dalı önceki ve sonraki derslere bağlar; ayrı bir teal düğüm D28-29 kayıtsız lab oturumlarını işaretler — sinyal-işleme bloğu burada başlar.



Şekil 35.1: Ders 30 kavram haritasi: iki konu tek koprude bulusuyor. Dal A rank-1 tamamlama (A = uv^T, m+n-1 girdi yeter, her 2x2 alt-determinant sifir; bipartite graph dongusuz ise tamamlanabilir). Dal B sirkulant matrisler (ilk kolon tum matrisi belirler, C = polinom(P) ve P^n = I; carpim = cyclic convolution = grup islemi; ozvektorler birim kokleri -> Fourier). Koprular: D1 uv^T dis-carpim, D16 Netflix tamamlama, D31 Fourier matrisi (sonraki), CNN/evrisim (D32, fast.ai L15). D28-29 kayitsiz lab atlanir; sinyal-isleme blogu bu derste basliyor.

Builder Notu — İki Konu Tek Köprü

Bu ders iki bağımsız gibi görünen konuyu tek bir derste topluyor; ama ikisi de **yapı → cebir** köprüsünün örneği. ML köprüsü: rank-1 tamamlama, Ders 16'nın Netflix matris tamamlamasının kombinatoryal/yapısal versiyonudur (hangi gözlem desenleri tamamlamaya izin verir?); sirkülant matrisler ise evrişimin (CNN, Ders 32) ve FFT'nin (Ders 31) lineer-cebir temelidir. **Geriye köprü:** Ders 1 ($uv^\top$ dış-çarpım, kolon×satır), Ders 16 (matris tamamlama/nuclear norm), Ders 4 (özvektör). İleriye: Ders 31 (Fourier — sonraki).
Tek cümle: Rank-1 matris tamamlama bir bipartite graph'ın döngüsüz olmasıyla mümkündür; sirkülant matrisler ise tek bir döngüsel kaydırma P'nin polinomudur, çarpımları döngüsel evrişimdir ve özvektörleri birim kökleridir (Fourier).

35.2 Rank-1 Matris Tamamlama Problemi

İlk konu, Prof. Rao'nun lab'ından doğan bir soru:

> *"…can you complete it to a rank 1 matrix?" — Strang, 1:22*

Bir matrisin bazı girdileri verilmiş (geri kalanı boş). Boşları, sonuç **rank-1** olacak şekilde doldurabilir misin? Rank-1 matris dış-çarpımdır (Ders 1):

$$A = uv^\top, \qquad a_{ij} = u_i\, v_j$$

Kaç girdi serbestçe verilebilir? m tane u, n tane v var; ama u'yu yeniden ölçekleyip $u_1 = 1$ yapabiliriz (bir serbestlik tekrarlı). Yani $m + n - 1$ serbest parametre. $m = n = 3$ için 5 girdi. Bu girdiler sıfırdan farklı olmalı (sıfır verilen bir konum, o satır/sütunu sıfıra zorlardı). Bu beş girdinin nasıl zincirleme dokuz girdiye açıldığını Şekil 35.2 bir sonraki bölümde sayılarla gösterir.

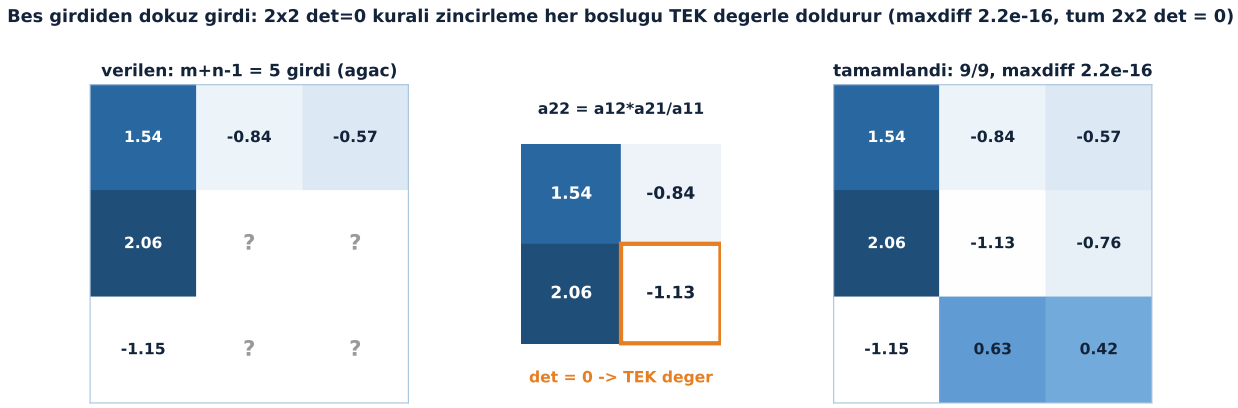
> **Builder Notu — Dış-Çarpımın Geri Dönüşü**
>
> "Rank-1 = $uv^\top$, $m + n - 1$ serbestlik" Ders 1'in dış-çarpım görüşünün geri dönüşü. ML köprüsü: bu, Ders 16'nın Netflix matris tamamlamasının (eksik kullanıcı-film matrisini düşük-rank doldur) kombinatoryal/yapısal versiyonu — hangi gözlem desenleri tamamlamaya izin verir? Öneri sistemlerinde "hangi girdiler yeterli?" sorusu.

35.3 2×2 Determinant Kuralı

Rank-1 olmanın testi: **her 2×2 alt-matris tekildir** (determinantı sıfır). Çünkü rank-1'de tüm kolonlar bir kolonun katı, tüm satırlar bir satırın katı:

$$\det \begin{bmatrix} a_{ij} & a_{ik} \\ a_{lj} & a_{lk} \end{bmatrix} = 0 \quad (\text{her } 2 \times 2)$$

Bu, boş girdileri doldurmanın anahtarı: bir 2×2'nin üç girdisi biliniyorsa, dördüncüsü determinantı sıfır yapacak **tek** değerdir ($a_{ik} = a_{ij} \cdot a_{lk}/a_{lj}$). Bilinen girdilerden başlayıp 2×2 kuralıyla komşuları zincirleme doldurursun.



Şekil 35.2: Beş girdiden dokuz girdi: 2×2 det=0 kuralı zincirleme her boşluğu TEK değerle doldurur — solda verilen m+n−1 = 5 girdilik ağaç deseni, ortada $a_{22} = a_{12} \cdot a_{21}/a_{11}$ kuralı (det=0 → tek değer), sağda 9/9 geri kazanılmış A_hat (maxdiff 2.2e-16, tüm 2×2 det ≈ 0).

Şekil 35.2 solda verilen 5 girdilik ağaç desenini (satır-0 + kolon-0; boş hücreler "?"), ortada $a_{22} = a_{12} \cdot a_{21}/a_{11}$ kuralını (det=0 → tek değer, turuncu vurgu), sağda tamamlanmış A_{hat}'ı gösterir: beş girdiden başlayıp 2×2 det=0 kuralını zincirleyerek dokuz girdinin tamamı geri kazanılır, maxdiff 2.2×10^{-16} ve A_{hat}'ın tüm 2×2 determinantları ≈ 0. 4×5'te de 8 girdiden 20 girdi aynı kesinlikle gelir.

> **Builder Notu — Rank'ı Küçük Pencerelerden Oku**
>
> "Rank-1 $\iff$ tüm 2×2 det = 0" rank'ın yerel karakterizasyonu: rankı global SVD'den değil, küçük determinantlardan oku. ML köprüsü: bu kural, matris tamamlamanın neden bazı desenlerde benzersiz (her boşluk tek değer) bazılarında imkânsız (çelişen kısıtlar) olduğunu açıklar — düşük-rank tamamlamanın belirlenebilirlik (identifiability) koşulu.

35.4 Bipartite Graph ve Tamamlanabilirlik

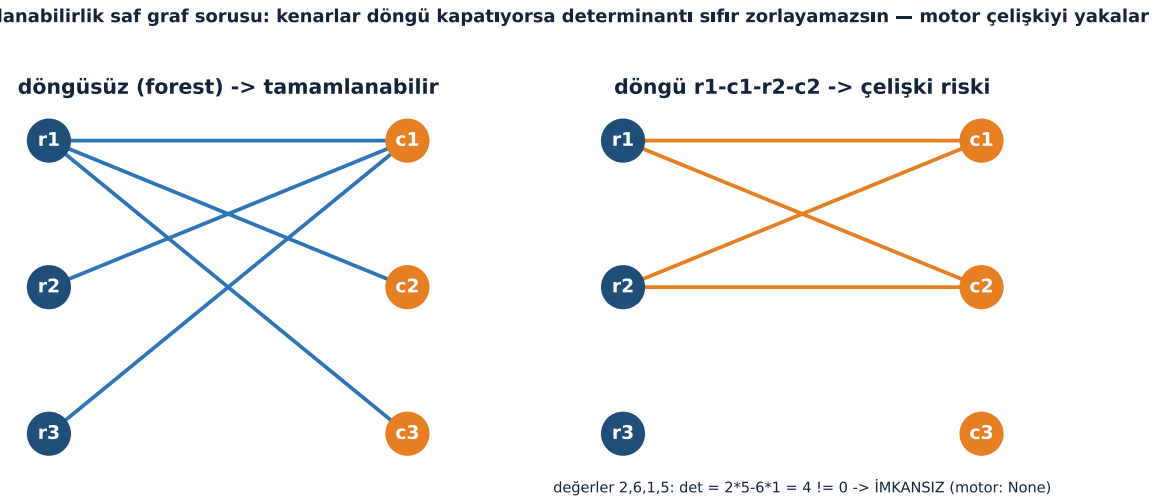
Hangi konumlar tamamlamaya izin verir? Bir kombinatorikçinin görüşü: problemi bir **bipartite graph**a (iki-parçalı çizge) çevir.

> *"…this is called a bipartite graph." — Strang, 12:02*

Satırları bir parça (m düğüm), sütunları diğer parça (n düğüm) yap. Verilen her (i, j) girdisi, satır-i ile sütun-j arasına bir **kenar** koyar. $m + n - 1$ girdi $\rightarrow$ $m + n - 1$ kenar. Kural: matris tamamlanabilir $\iff$ bu graf **döngüsüzdür** (cycle yok, yani bir orman/ağaç). Bir döngü, tüm girdileri verilmiş bir 2×2 (veya daha büyük) demektir — orada determinantı sıfır zorlayamazsın, çelişki çıkar.

$$\text{tamamlanabilir} \iff \text{bipartite graph dongusuz (forest)}$$

Döngüsüz graf, 2×2 kuralını çelişkisiz zincirlemene izin verir; her boş girdi tek değerle belirlenir.



Şekil 35.3: Tamamlanabilirlik saf graf sorusu: kenarlar döngü kapatıyorsa determinantı sıfır zorlayamazsın — motor çelişkiyi yakalar

Şekil 35.3 solda döngüsüz (forest) ağaç desenini gösterir — beş kenar bir orman oluşturur, tamamlanabilir; sağda $(1,1), (1,2), (2,1), (2,2)$ kenarları bir döngü kapatır (r1-c1-r2-c2). Bu döngüde değerler $2, 6, 1, 5$ verilince $\det = 2 \cdot 5 - 6 \cdot 1 = 4 \neq 0$ olduğundan rank-1 tamamlama **imkânsızdır** (motor `None` döner); determinantı sıfır zorlayamazsın.

> **Builder Notu — Cebir Graf Olunca**
>
> "Tamamlanabilir $\iff$ bipartite graph döngüsüz" zarif bir cebir-graf köprüsü: matris yapısı (rank-1 doldurulabilirlik) saf grafik teorisine (döngü var mı?) indirgenir. ML köprüsü: graf-tabanlı düşünme öneri sistemlerinde merkezî — kullanıcı-ürün bipartite grafı, eksik kenarları (tahmin) tamamlamak; döngü/bağlılık yapısı hangi tahminlerin güvenilir olduğunu belirler. Cebir $\leftrightarrow$ graf çevirisi Phase 2'nin tekrarlayan teması.

35.5 Sirkülant Matris ve Cyclic Shift P

İkinci konu: **sirkülant** (dolaşımlı) matris. Her kolon, bir öncekinin **döngüsel kaydırılmışı**dır — yani ilk kolonu verirsen tüm matris belli. Anahtar yapı taşı döngüsel kaydırma matrisi P:

> *"…the key matrix in this is really a cyclic shift matrix." — Strang, 28:21*

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

P bir vektörü bir aşağı kaydırır, en alttaki döngüsel olarak tepeye döner. P bir permütasyon matrisidir. P^2 iki kaydırma, P^3 üç kaydırma. P'nin kuvvetlerinin bir sonraki bölümde $P^4 = I$ ile başa döndüğünü Şekil 35.4 gösterir.

> **Builder Notu — Bir Kolon Bütün Matris**
>
> "İlk kolon → tüm matris, cyclic shift P" sirkülantın sıkıştırılmış temsili: $n \times n$ matris yerine n sayı (ilk kolon). ML köprüsü: bu, evrişimin (convolution) matris hâlidir — bir filtreyi tüm konumlara kaydırarak uygula; CNN'in ağırlık paylaşımı (weight sharing) tam bu döngüsel/kaydırmalı yapı. Toeplitz matrisi (Ders 32) sirkülantın döngüsüz akrabasıdır.

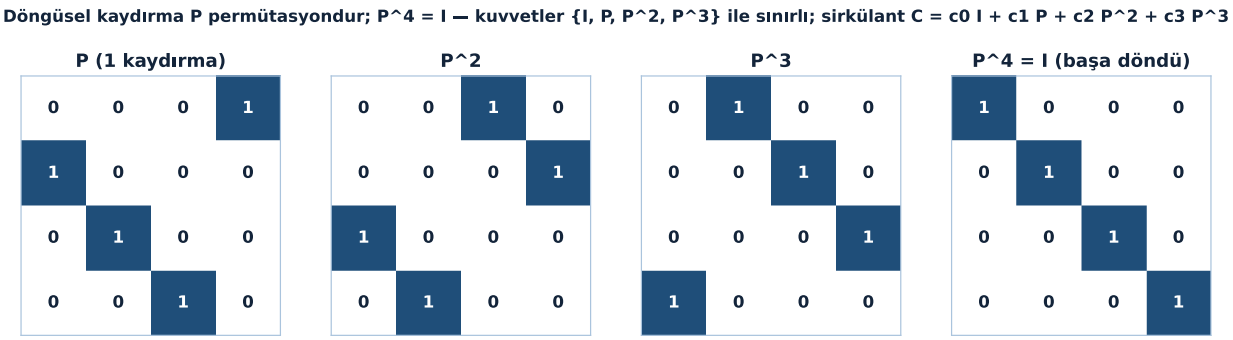
35.6 C = Polinom(P), $P^n = I$

Her sirkülant matris, P'nin bir **polinomu**dur (köşegenlere c_0, c_1, c_2, c_3 koymak = P kuvvetlerinin kombinasyonu):

> *"…every circulant matrix is a polynomial in P…" — Strang, 32:45*

$$C = c_0 I + c_1 P + c_2 P^2 + c_3 P^3$$

Kritik özellik: $n \times n$'de $P^n = I$ (n kaydırma başa döner). Yani P'nin kuvvetleri $\{I, P, \ldots, P^{n-1}\}$ ile sınırlı — derecesi n'i aşan terimler döngüsel olarak geri sarar. Bu sayede sirkülantlar bir **grup** oluşturur: iki sirkülantın çarpımı (iki polinomun çarpımı) yine bir sirkülanttır ($P^n = I$ ile dereceyi n'in altında tutarak).



Şekil 35.4: Döngüsel kaydırma P permütasyondur; P^4 = I — kuvvetler {I, P, P^2, P^3} ile sınırlı; sirkülant C = c0 I + c1 P + c2 P^2 + c3 P^3 olarak bu dört kuvvetin doğrusal birleşimidir.

Şekil 35.4 P, P^2, P^3, P^4 kuvvetlerini sırayla gösterir: her kaydırma 1'leri bir köşegen ilerletir ve $P^4 = I$ başa döner. Demek ki P'nin kuvvetleri $\{I, P, P^2, P^3\}$ ile sınırlıdır; sirkülant $C = c_0 I + c_1 P + c_2 P^2 + c_3 P^3$ tam bu dört kuvvetin doğrusal birleşimidir. Motor tanığı: P döngüsel kaydırma, $P^4 = I$, $P^3 \neq I$ ve $P \cdot (1, 2, 3, 4) = (4, 1, 2, 3)$ (bir aşağı, sarmalı).

> **Builder Notu — Matris Dünyasından Polinom Dünyasına**
>
> "Sirkülant = polinom(P), $P^n = I$" cebirsel kalbi: matris dünyasını polinom dünyasına çevirir. ML köprüsü: bu, evrişimin neden polinom çarpımı (ve Fourier'de basit çarpma) olduğunu açıklar; $P^n = I$ döngüselliği, ayrık Fourier dönüşümünün (DFT, Ders 31) periyodikliğinin kaynağı. Polinom $\leftrightarrow$ matris $\leftrightarrow$ sinyal üç dil aynı şey.

35.7 Cyclic Convolution

Sirkülant çarpımı, polinom çarpımıdır — yani **evrişim** (convolution). Örnek: $(3, 1, 2) \circledast (4, 6, 1)$, gizli polinomlar $(3 + x + 2x^2)(4 + 6x + x^2)$:

$$(3, 1, 2) * (4, 6, 1) = (12, 22, 17, 13, 2)$$

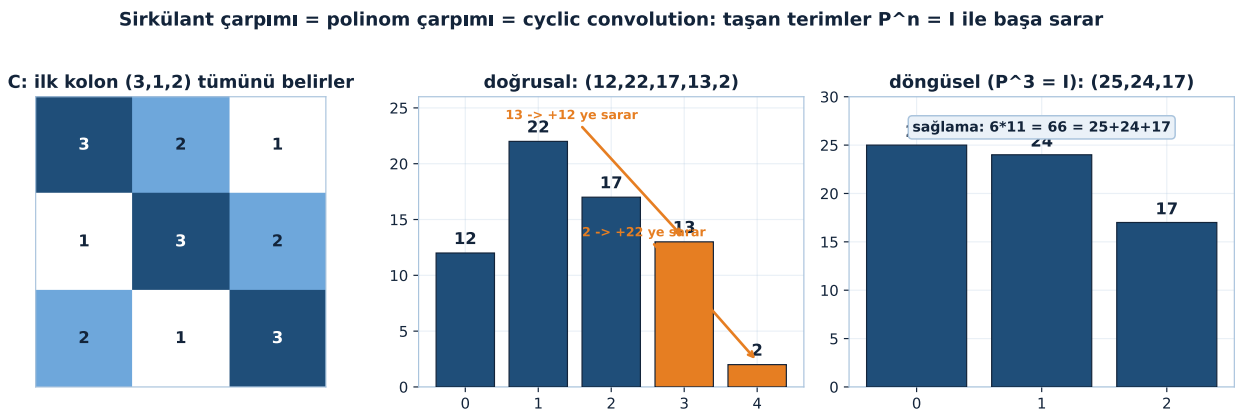
(İlkokul çarpması: terim terim çarp-topla.) Sirkülantlarda $P^n = I$ olduğundan **döngüsel** (cyclic) evrişim: taşan terimler başa sarar.

> *"…I'm just doing cyclic convolution actually." — Strang, 36:48*

$$(3, 1, 2) \circledast (4, 6, 1) = (12 + 13,\ 22 + 2,\ 17) = (25, 24, 17)$$

$n = 3$'te $P^3 = I$, böylece 5-uzunluklu sonuç 3'e sarılır. Hoş bir sağlama: rakam-toplamları çarpımı korunur — $(3 + 1 + 2)(4 + 6 + 1) = 6 \cdot 11 = 66 = 25 + 24 + 17$. Matris çarpımı = döngüsel evrişim = polinom çarpımı (mod $P^n - 1$).

Şekil 35.5 solda $C = \text{circulant}(3, 1, 2)$ heatmap'ini (ilk kolon tümünü belirler), ortada doğrusal evrişimi $(12, 22, 17, 13, 2)$ — taşan son iki terim 13 ve 2 turuncu, sarmalama okları $13 \to +12$ ve $2 \to +22$ — ve



Şekil 35.5: Sirkülant çarpımı = polinom çarpımı = cyclic convolution: taşan terimler P^n = I ile başa sarar — (3,1,2) çark (4,6,1) = (25,24,17). Doğrusal evrişim (12,22,17,13,2); taşan son iki terim 13 ve 2 başa sarınca döngüsel sonuç (25,24,17). Sağlama: 6*11 = 66 = 25+24+17.

sağda döngüsel sonucu $(25, 24, 17)$ gösterir. Sağlama: $6 \cdot 11 = 66 = 25 + 24 + 17$. Taşan terimler $P^n = I$ ile başa sarar; matris çarpımı = polinom çarpımı = cyclic convolution.

> Builder Notu — Üç Dilin Aynı Çarpımı
>
> "Sirkülant çarpımı = cyclic convolution = polinom çarpımı" üç dilin birliği. ML köprüsü: CNN'lerin (Ders 32) konvolüsyon katmanı tam budur; fast.ai L15'in im2col/conv2d'si konvolüsyonu matris çarpımına çevirir. FFT (Ders 31) konvolüsyonu $O(n \log n)$'de yapar — sinyal işleme ve büyük-çekirdek CNN'lerin hızı buradan.

35.8 Özvektörler = Birim Kökleri (Fourier)

Sirkülant C'nin özvektörleri ne? C = P'nin polinomu olduğundan, **C'nin özvektörleri P'nin özvektörleriyle aynı** (P'nin özvektörü, P^2'nin ve P^3'ün de özvektörü):

> *"...the eigenvectors of C are the same as the eigenvectors of P." — Strang, 45:01*

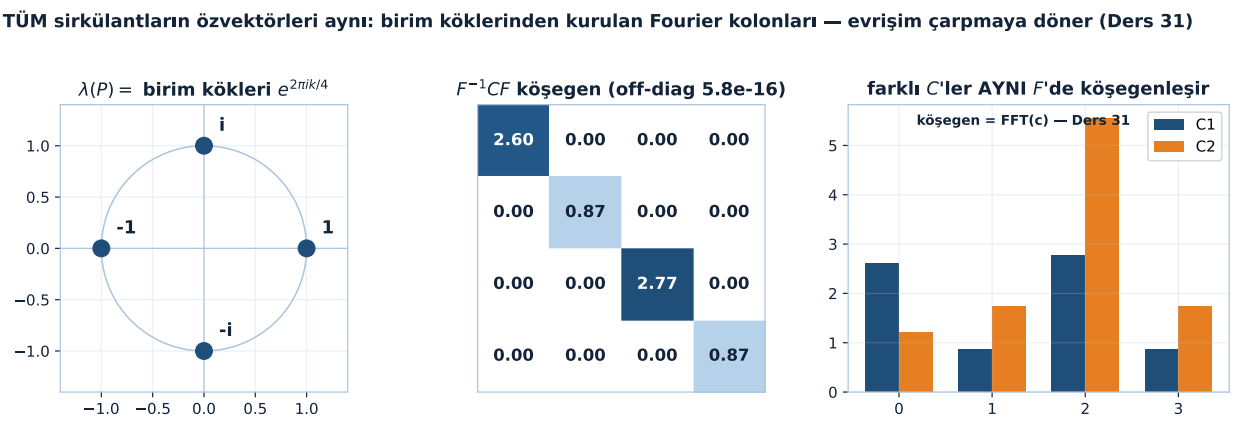
Demek ki tek soru: P'nin özvektörleri/özdeğerleri ne? P döngüsel kaydırma; özdeğerleri **birim kökleri** (roots of unity). $n = 4$ için $(1, 1, 1, 1) \to \lambda = 1$; $(1, -1, 1, -1) \to \lambda = -1$; ve karmaşık $(1, i, -1, -i) \to \lambda = i, \lambda = -i$:

> *"...the eigenvectors are the four roots of 1." — Strang, 48:42*

$$\lambda(P) = 1, -1, i, -i = e^{2\pi ik/n}, \quad k = 0, 1, 2, 3$$

Özvektörler birim köklerinden kurulur — bunlar **Fourier matrisinin** kolonlarıdır (Ders 31). Yani TÜM sirkülantlar aynı özvektörleri (Fourier) paylaşır.

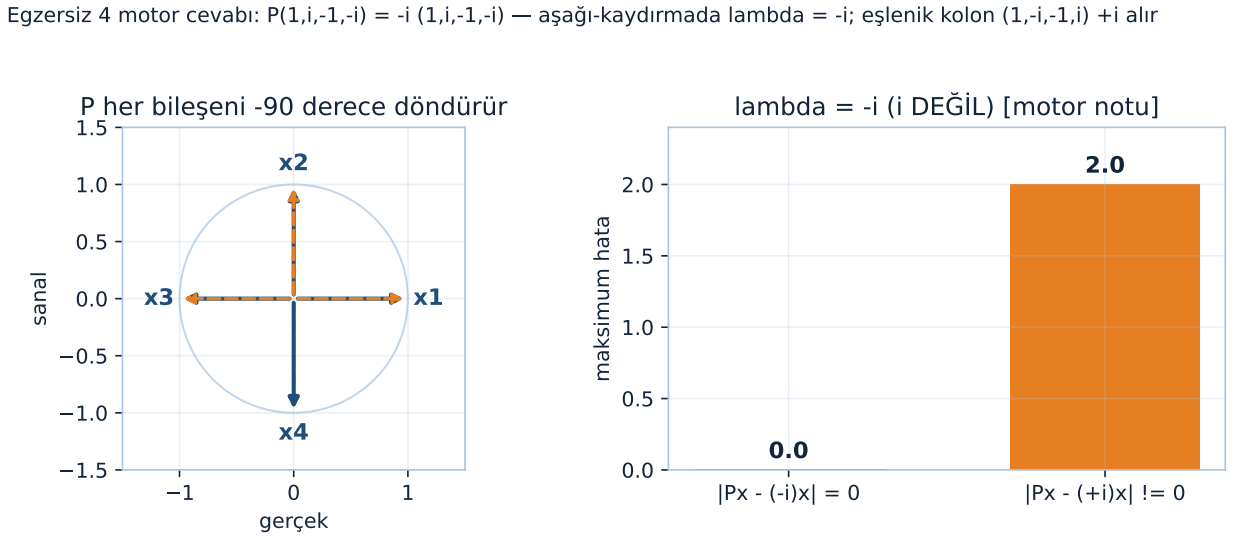
Şekil 35.6 solda $\lambda(P)$'yi birim çemberdeki dört birim kökü $(1, i, -1, -i)$ olarak, ortada $F^{-1}CF$'in köşegen olduğunu (köşegen-dışı 5.8×10^{-16}), sağda iki FARKLI sirkülantın AYNI F'de köşegenleştiğini (köşegen



Şekil 35.6: TÜM sirkülantların özvektörleri aynı: birim köklerinden kurulan Fourier kolonları — evrişim çarpmaya döner (Ders 31). Sol: $\lambda(P)$ birim çemberdeki dördüncü birim kökleri. Orta: $F^{-1}CF$ köşegen (off-diag 5.8e-16). Sağ: iki farklı sirkülant AYNI F'de köşegenleşir, köşegen = FFT(c).

$=$ FFT(c), Ders 31) gösterir. Motor tanığı: $F^{-1}CF$ köşegen-dışı 5.8×10^{-16}, köşegen $=$ np.fft.fft(c) birebir, ve farklı sirkülant da aynı F'de köşegenleşir (ortak özvektör tabanı).

Bu dersin son sorusu — sirkülant özvektörünü P ile çarpınca hangi özdeğer çıkar — Egzersiz 4'te somutlaşır; Şekil 35.7 motor cevabını verir.



Şekil 35.7: Egzersiz 4 motor cevabı: P(1,i,-1,-i) = -i (1,i,-1,-i) — aşağı-kaydırmada lambda = -i; eşlenik kolon (1,-i,-1,i) +i alır. Sol panelde x bileşenleri (navy) ve Px bileşenleri (turuncu kesik) her biri -90° dönmüş; sağ panelde |Px - (-i)x| = 0 (tam tutar) ile |Px - (+i)x| = 2 (tutmaz) karşılaştırması.

Şekil 35.7 solda $x = (1, i, -1, -i)$ bileşenlerini (navy) ve Px bileşenlerini (turuncu kesik) gösterir — P her bileşeni $-90°$ döndürür; sağda iki hata barı $|Px - (-i)x| = 0$ (tam tutar) ile $|Px - (+i)x| \neq 0$ $(= 2$, tutmaz) karşılaştırır.

motor notu — Egz4 özdeğeri ve §7'deki iki kök

Notion §7'deki "$(1, i, -1, -i) \to \lambda = i, \lambda = -i$" cümlesi iki karmaşık özdeğeri birden anar (P'nin spektrumunda hem $+i$ hem $-i$ vardır) ve Egzersiz 4 "(i ile mi?)" sorusunu açık bırakır. Bu aşağı-kaydırma P (yani $(Px)_i = x_{(i-1) \bmod n}$) için motor cevabı kesindir: $x = (1, i, -1, -i)$ özvektörünün özdeğeri $\lambda = -i$'dir (i DEĞİL) — $P \cdot x = -i \cdot x$. Eşlenik kolon $(1, -i, -1, i)$ ise $\lambda = +i$ alır. İki kök birden P'nin spektrumunda yaşar; ama bu spesifik kolon-özvektör eşleşmesi $(1, i, -1, -i) \mapsto -i$ şeklindedir.

Builder Notu — Bütün Sirkülantların Ortak Tabanı

"Tüm sirkülantların özvektörleri = Fourier" sinyal işlemenin temel teoremi: Fourier tabanında her sirkülant köşegenleşir, evrişim çarpmaya döner. ML köprüsü: bu, FFT'nin (Ders 31) ve frekans-uzayı filtrelemenin temeli; konvolüsyon teoremi (zaman uzayında evrişim = frekans uzayında çarpma) doğrudan buradan. Graf sinyal işleme (NYU spektral GCN, Ders 19/35) bu fikri graf Laplacian özvektörlerine genelleştirir. Köşegen değerleri tam FFT(c) olduğundan, Ders 31'in $F^{-1}CF = \text{diag}(\text{FFT}(c))$ köprüsü buradan açılır.

35.9 Bu Dersin Özeti

- **Rank-1 tamamlama:** $A = uv^\top$, $a_{ij} = u_i v_j$; $m + n - 1$ serbest parametre. Rank-1 $\iff$ tüm 2×2 det = 0.
- **Bipartite graph:** satır+sütun düğüm, girdi=kenar; tamamlanabilir $\iff$ döngüsüz (forest).
- **Sirkülant C:** ilk kolon tümünü belirler; cyclic shift P; $C = c_0 I + c_1 P + c_2 P^2 + c_3 P^3$; $P^n = I$.
- **Grup + cyclic convolution:** iki sirkülantın çarpımı sirkülant; çarpım = döngüsel evrişim = polinom çarpımı (mod $P^n - 1$).
- **Özvektörler:** C'nin özvektörleri = P'nin özvektörleri = birim kökleri ($1, -1, i, -i$ for $n = 4$) = Fourier matrisi.
- **Tüm sirkülantlar aynı özvektörleri (Fourier) paylaşır** → evrişim çarpmaya döner (Ders 31).

Tek Bir Cümle

Rank-1 matris tamamlama bir bipartite graph'ın döngüsüz olmasıyla mümkündür (her boş girdi 2×2 det=0 ile tek değer alır); sirkülant matrisler ise tek bir döngüsel kaydırma P'nin polinomudur, çarpımları döngüsel evrişimdir ve özvektörleri birim kökleridir — yani hepsi Fourier matrisinde köşegenleşir.

35.10 Kontrol Soruları

Soru 1 — Kaç girdi serbest ve rank-1'in testi

Soru: Rank-1 matris tamamlamada kaç girdi serbestçe verilebilir ve rank-1'in testi nedir?
Cevap: $m + n - 1$ girdi ($A = uv^\top$'de m tane u + n tane v, ama $u_1 = 1$ normalize edilince bir serbestlik düşer). Verilen girdiler sıfırdan farklı olmalı. Rank-1 testi: **tüm 2×2 alt-determinantlar sıfır** olmalı (tüm kolonlar bir kolonun katı). Bir 2×2'nin üç girdisi biliniyorsa, dördüncüsü det=0 yapacak tek değerdir.

Soru 2 — Bipartite graph tamamlanabilirliği nasıl belirler

Soru: Bipartite graph rank-1 tamamlanabilirliği nasıl belirler?
Cevap: Satırlar bir parça, sütunlar diğer parça düğümleri; verilen her (i, j) girdisi satır-$i \leftrightarrow$ sütun-j kenarı. Matris tamamlanabilir $\iff$ bu graf **döngüsüzdür** (forest/ağaç). Döngü, tüm girdileri verilmiş bir alt-yapı demektir; orada determinantı sıfır zorlayamazsın (çelişki). Döngüsüz graf, 2×2 kuralını çelişkisiz zincirlemene izin verir.

Soru 3 — Sirkülant neden P'nin polinomu ve neden grup

Soru: Sirkülant matris neden P'nin polinomudur ve neden grup oluşturur?
Cevap: Sirkülant C, köşegenlerinde c_0, c_1, c_2, c_3 taşır; bunlar tam olarak P kuvvetlerinin katsayılarıdır: $C = c_0 I + c_1 P + c_2 P^2 + c_3 P^3$ (P = döngüsel kaydırma). $P^n = I$ (n kaydırma başa döner) olduğundan, iki sirkülantın çarpımı (iki polinomun çarpımı) yine n'den küçük dereceli bir polinom = sirkülanttır. Bu yüzden grup: çarpım kapalı, I birim eleman.

Soru 4 — Sirkülantın özvektörleri ve neden önemli

Soru: Sirkülant matrisin özvektörleri nedir ve bu neden önemli?
Cevap: C = P'nin polinomu olduğundan C'nin özvektörleri = P'nin özvektörleri. P döngüsel kaydırmanın özdeğerleri **birim kökleridir** ($e^{2\pi ik/n}$; $n = 4$ için $1, -1, i, -i$), özvektörleri **Fourier matrisinin** kolonları. Önemli çünkü TÜM sirkülantlar aynı (Fourier) özvektörleri paylaşır — Fourier tabanında her sirkülant köşegenleşir, dolayısıyla **evrişim çarpmaya dönüşür** (konvolüsyon teoremi, FFT temeli, Ders 31).

35.11 Egzersizler

1. **2×2 tamamlama.** Bir 2×2 matrisin üç girdisi: $a_{11} = 2$, $a_{12} = 6$, $a_{21} = 1$. Rank-1 olması için a_{22} ne olmalı? (det=0 kuralı: $a_{22} = a_{12} \cdot a_{21}/a_{11}$.) (Motor tanığı: $a_{22} = 6 \cdot 1/2 = 3.0$.)

2. **Bipartite döngü.** 3×3 matriste $(1,1), (1,2), (2,1), (2,2)$ girdileri verilmiş. Bunu bipartite graf olarak çiz; bir döngü var mı? Rank-1 tamamlama mümkün mü, neden? (Motor tanığı: bu dört girdi r1-c1-r2-c2 döngüsü kapatır; değerler $2, 6, 1, 5$ ile $\det = 2 \cdot 5 - 6 \cdot 1 = 4 \neq 0 \rightarrow$ tamamlama **imkânsız** (motor `None`). Aynı döngü $2, 6, 1, 3$ (det=0) ile çelişkisiz geçer, ama bu şans, kural değil; bkz. Şekil 35.3.)

3. **Cyclic convolution.** $(1,2) \circledast (3,4)$ döngüsel evrişimini ($n=2, P^2=I$) hesapla. (Önce normal evrişim $(1,2) * (3,4)$, sonra sarmala.) (Motor tanığı: doğrusal $(3,10,8) \rightarrow$ döngüsel $(3+8,10) = (11,10)$; sağlama $3 \cdot 7 = 21 = 11+10$.)

4. **Sirkülant özvektör.** 4×4 cyclic shift P için $(1,i,-1,-i)$ vektörünü P ile çarp (bir aşağı kaydır, sarmala). Hangi λ ile özvektör? (i ile mi?) (**[motor notu]:** Bu aşağı-kaydırma P için $P \cdot (1,i,-1,-i) = -i \cdot (1,i,-1,-i)$, yani $\lambda = -i$ — i **DEĞİL**. Eşlenik kolon $(1,-i,-1,i)$ ise $\lambda = +i$ alır; bkz. Şekil 35.7.)

5. **(Ders 31 habercisi)** Bu derste sirkülantların özvektörlerinin birim kökleri olduğunu gördük. Bu özvektörleri kolon kolon bir matrise dizersen ne elde edersin? $n \times n$ bu matrisin adı ne, hangi dönüşümü temsil eder? Bir tahmin yaz — Ders 31 "sirkülant matrislerin özvektörleri: Fourier matrisi"ni işliyor. (Motor tanığı: özvektörleri kolon kolon dizince $F^{-1}CF = \text{diag}(\text{np.fft.fft}(c))$, yani F tüm sirkülantları köşegenleştirir — ortak özvektör tabanı.)

35.12 Sonraki Ders İçin Hazırlık

Ders 31: Sirkülant Matrislerin Özvektörleri — Fourier Matrisi. Bu dersin birim-kök özvektörlerini bir matrise dizince **Fourier matrisi** (F) çıkar; ayrık Fourier dönüşümü (DFT). Her sirkülant F'de köşegenleşir, evrişim çarpmaya döner (konvolüsyon teoremi), ve FFT bu yapıyı $O(n \log n)$'de hesaplar — sinyal işleme ve CNN'lerin matematiksel temeli.

> ⚠ Hazırlık
>
> Bu dersin Egzersiz 5'inin sorduğu soruyu zihninde tut: sirkülantların birim-kök özvektörlerini kolon kolon bir matrise dizersen hangi matris çıkar? Ders 31, bu **Fourier matrisini** (F) tanımlar ve her sirkülantın $F^{-1}CF = \text{diag}(\text{FFT}(c))$ ile köşegenleştiğini gösterir — köşegen değerleri tam FFT(c). Bu, evrişimi çarpmaya çeviren konvolüsyon teoreminin, FFT'nin ($O(n \log n)$) ve sinyal-işleme/CNN bloğunun (Ders 32) matematiksel kalbidir.

35.13 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Rank-1 tamamlama	$A = uv^\top$; $m+n-1$ girdi; tüm 2×2 det = 0	1m22
Bipartite graph	satır/sütun düğüm, girdi=kenar; döngüsüz $\iff$ tamamlanabilir	12m02
Cyclic shift P	sirkülant: ilk kolon tümünü belirler	28m21
C = polinom(P)	$c_0 I + c_1 P + c_2 P^2 + c_3 P^3$; $P^n = I$	32m45
Cyclic convolution	sirkülant çarpımı = döngüsel evrişim	36m48
Özvektörler = P'nin	C = polinom(P) $\rightarrow$ aynı özvektörler	45m01
Birim kökleri $\rightarrow$ Fourier	$\lambda(P) = 1,-1,i,-i = e^{2\pi ik/n}$	48m42

35.14 ML Bağlantıları Özeti

- **Sirkülant = evrişim = CNN:** sirkülant matrisle çarpmak döngüsel evrişim; CNN'in (Ders 32) ağırlık paylaşımı; fast.ai L15 im2col/conv2d aynı yapı.
- **Özvektörler Fourier → FFT:** tüm sirkülantlar Fourier'de köşegenleşir; evrişim çarpmaya döner (konvolüsyon teoremi); FFT $O(n \log n)$ (Ders 31).
- **Rank-1 tamamlama:** Ders 16 Netflix matris tamamlamasının kombinatoryal hali; bipartite graf döngüsü belirlenebilirliği verir.
- **Cebir ↔ graf:** rank-1 doldurulabilirlik saf grafik teorisine (döngü var mı?) indirgenir; öneri sistemleri bipartite graf düşünür.
- **Graf sinyal işleme:** sirkülant→Fourier fikri graf Laplacian özvektörlerine genelleşir (spektral GCN, NYU paralel, Ders 19/35).
- **Geriye köprü:** Ders 1 ($uv^\top$ dış-çarpım), Ders 16 (matris tamamlama/nuclear norm), Ders 4 (özvektör). Paralel: fast.ai L15 (convolution), NYU H6 (CNN).

! Kapanış

> *"…the eigenvectors of C are the same as the eigenvectors of P." — Strang, 45:01*

Tüm sirkülantlar tek bir özvektör tabanını (Fourier) paylaşır; bu, evrişimi çarpmaya çeviren konvolüsyon teoreminin ve tüm sinyal işlemenin kalbidir — sinyal/CNN bloğunun açılışı.

36 Sirkülant Matrislerin Özvektörleri — Fourier Matrisi

Toeplitz ve sirkülant evrişim, normal matris ailesi ve dünyanın en önemli kompleks matrisi

> **Bölüm bilgisi**
>
> Bu ders, Ders 30'un sirkülantlarının özvektörlerini somutlaştırır: **Fourier matrisi** F. Tüm sirkülantlar aynı özvektör matrisini paylaşır — bu, ayrık Fourier dönüşümü (DFT) ve FFT'nin temelidir. Strang'in Ders 31 videosu (≈52 dk) ve OCW Lecture 31 temel alınmıştır. Okuma süresi ≈34 dk. Önkoşul Ders 30 (sirkülant/cyclic shift P/birim kökleri), Ders 3 (ortogonal Q) ve Ders 4 (köşegenleştirme/özvektör matrisi).

36.1 Bu Derste Ne Var?

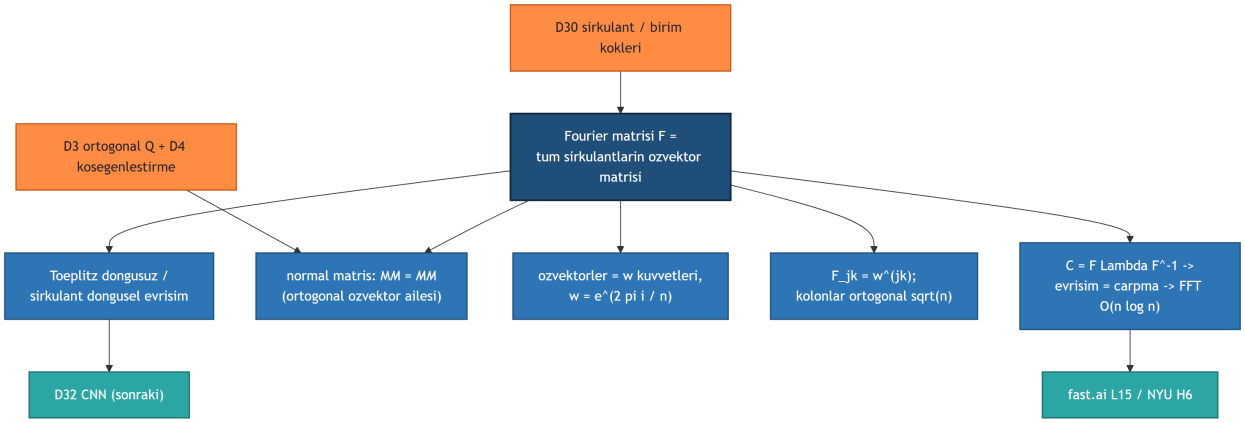
Bu ders **Fourier matrisi** F'yi beş adımda inşa eder: Toeplitz ile sirkülantı ayır, ailenin ortak adı olan normal matrisi tanımla, özvektörleri w kuvvetlerine indirge, F'yi kur ve sonunda her sirkülantı F'de köşegenleştir.

Beş sonuç:

1. **Toeplitz vs sirkülant:** sabit-köşegen Toeplitz = (döngüsüz) evrişim; sirkülant = döngüsel evrişim. ML matrisleri böyle özeldir (ağırlık paylaşımı).
2. **Normal matris:** $M^*M = MM^*$; ortogonal özvektörlü matrisler (simetrik, diyagonal, ortogonal, antisimetrik, sirkülant). Sirkülantlar **değişmeli** → normal.
3. **Özvektörler = w kuvvetleri:** $w = e^{2\pi i/n}$ (birim kökü); P'nin (ve tüm sirkülantların) özvektörleri.
4. **Fourier matrisi:** F = özvektör matrisi; $F_{jk} = w^{jk}$; kolonlar ortogonal (uzunluk $\sqrt{n}$), $F/\sqrt{n}$ ortonormal.
5. **Sonuç:** her sirkülant F'de köşegenleşir ($C = F\Lambda F^{-1}$); evrişim çarpmaya döner; FFT $O(n \log n)$.

> *"…Fourier matrix equals eigenvector matrix." — Strang, 43:39*

Şekil 36.1 dersin iskeletini gösterir: merkezdeki **Fourier matrisi F = tüm sirkülantların ortak özvektör matrisi** fikrinden beş dala ayrılır — Toeplitz döngüsüz / sirkülant döngüsel evrişim, normal matris testi $M^*M = MM^*$ (ortogonal özvektör ailesi), özvektörler = w kuvvetleri ($w = e^{2\pi i/n}$), $F_{jk} = w^{jk}$ kolonlar ortogonal uzunluk $\sqrt{n}$, ve $C = F\Lambda F^{-1}$ → evrişim frekansta çarpmaya döner → FFT $O(n \log n)$; köprü düğümleri D30 (sirkülant/birim kökleri), D3 ortogonal Q + D4 köşegenleştirme, D32 CNN (sonraki) ve fast.ai L15 / NYU H6 dalları önceki ve sonraki derslere bağlar.



Şekil 36.1: Ders 31 kavram haritasi: Fourier matrisi F = tum sirkulantlarin ortak ozvektor matrisi. Bes dal: Toeplitz dongusuz evrisim (ML/CNN) vs sirkulant dongusel evrisim (DFT), fark yalniz sarmalamada; normal matris testi M*M* = *MM* -> ortogonal ozvektor ailesi (sirkulantlar uye); ozvektorler = w kuvvetleri, w = e^(2 pi i / n); F_jk = w^(jk), kolonlar ortogonal uzunluk sqrt(n); C = F Lambda F^-1 -> evrisim frekansta carpmaya doner -> FFT O(n log n). Koprular: D30 sirkulant/birim kokleri, D3 ortogonal Q + D4 kosegenlestirme, D32 CNN (sonraki), fast.ai L15 / NYU H6.

Builder Notu — Dünyanın En Önemli Kompleks Matrisi

- **Fourier matrisi = dünyanın en önemli kompleks matrisi** (Strang); tüm sirkülantlar onda köşegenleşir → evrişim teoremi → FFT.
- **ML görüntü:** milyon-piksel feature; tam 3M×3M ağırlık imkansız → konvolüsyon (Toeplitz, ağırlık paylaşımı) + max pooling. CNN'in (Ders 32) temeli.
- **Normal matris** — ortogonal özvektörlü tüm ailenin adı; $M^*M = MM^*$ testi (Ders 3 ortogonal, Ders 4 özvektör matrisi).
- **Geriye köprü:** Ders 30 (sirkülant/P/birim kökleri), Ders 3 (ortogonal Q), Ders 4 (özvektör matrisi/köşegenleştirme), Ders 32 (CNN — sonraki). Paralel: fast.ai L15 (conv), NYU H6 (CNN).

Tek cümle: Tüm sirkülant matrisler aynı özvektör matrisini — Fourier matrisi F'yi ($F_{jk} = w^{jk}$, $w = e^{2\pi i/n}$) — paylaşır; bu matris tüm sirkülantları köşegenleştirir, evrişimi çarpmaya çevirir ve FFT'yi ($O(n \log n)$) mümkün kılar.

36.2 1. Toeplitz vs Sirkülant: Konvolüsyon

Sirkülantlar ayrık Fourier dönüşümüyle (DFT) yakından bağlı:

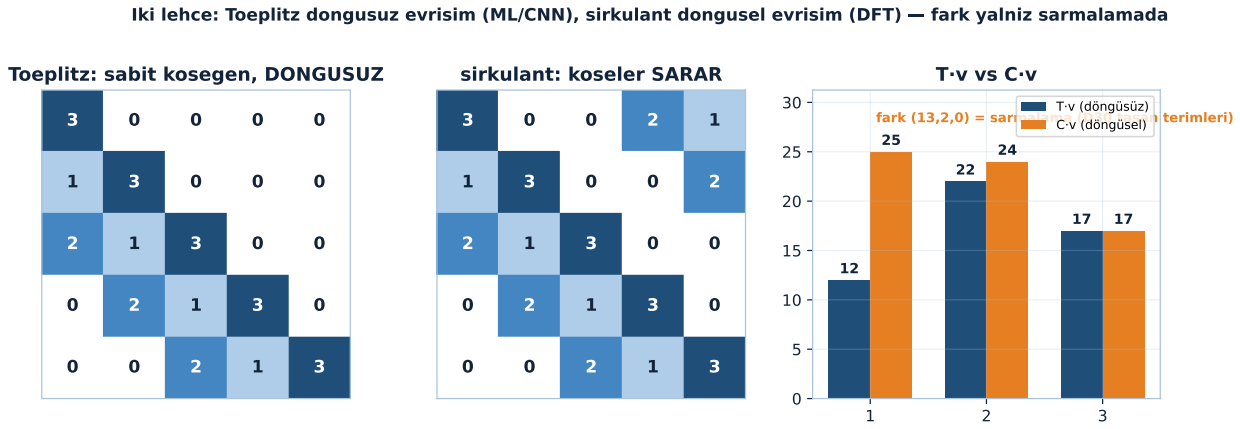
> *"…the discrete Fourier transform is… a very, very important algorithm in engineering…" — Strang, 2:49*

Sirkülant n×n matris sadece **n sayıyla** (ilk satır) tanımlanır, n^2 değil. Genel hâli: sabit-köşegenli ama **döngüsüz** matris. Adı:

"…math people would call it a Toeplitz matrix." — Strang, 9:16

$$\text{Toeplitz } T: \text{ sabit kosegen, dongusuz } \Rightarrow Tv = \text{(dongusuz) evrisim}$$

Sirkülant C ile çarpmak **döngüsel** (cyclic) evrişim; Toeplitz T ile çarpmak **döngüsüz** evrişim. İkisi de "linear shift/time invariant" (LSI/LTI) — sinyal işlemede filtre, evrişim. ML'de genelde Toeplitz (döngüsüz konvolüsyon) çıkar.



Şekil 36.2: İki lehçe: Toeplitz döngüsüz evrişim (ML/CNN), sirkülant döngüsel evrişim (DFT) — fark yalnız sarmalamada. Sol Toeplitz alt-üçgen sabit köşegen; orta sirkülant köşeler sarar; sağ T·v=(12,22,17) ile C·v=(25,24,17), fark (13,2,0) = D30'un taşan terimleri.

Şekil 36.2 iki lehçeyi yan yana koyar: solda Toeplitz alt-üçgen (sabit köşegen, döngüsüz), ortada sirkülant (köşeleri saran), sağda aynı v üzerine her ikisinin etkisi. Toeplitz çarpımı $T \cdot v = (12, 22, 17)$ döngüsüzdür (`linear_conv`'un ilk üç terimi); sirkülant çarpımı $C \cdot v = (25, 24, 17)$ döngüseldir. Fark $C \cdot v - T \cdot v = (13, 2, 0)$ tam olarak sarmalama (wrap-around) terimleridir — Ders 30'un döngüsel evrişimde başa sardığını gördüğümüz taşan 13 ve 2'sidir. İki dünya yalnızca bu sarmalamada ayrılır.

> **Builder Notu — Evrişimin İki Lehçesi**
>
> "Sirkülant = döngüsel evrişim, Toeplitz = döngüsüz evrişim" sinyal işlemenin matris dili. ML köprüsü: CNN'in konvolüsyon katmanı bir Toeplitz operasyonudur — aynı filtre tüm konumlara kaydırarak uygulanır (ağırlık paylaşımı). fast.ai L15 im2col bu Toeplitz yapısını matris çarpımına açar.

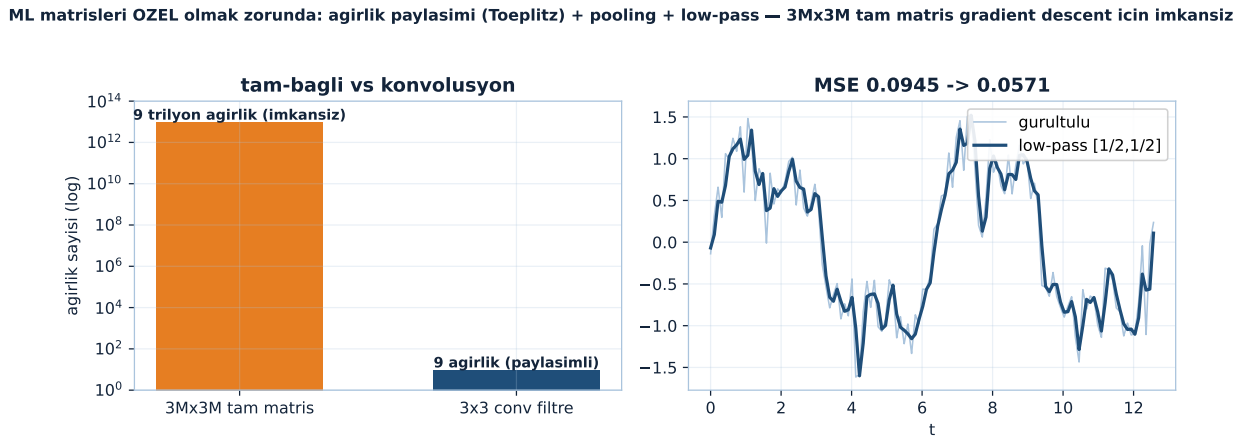
36.3 2. ML Görüntü: Neden Özel Matrisler?

Sirkülant/Toeplitz neden ML'de kritik? Görüntü = piksel; 1000×1000 görüntü = bir milyon özellik (renkte 3 milyon). Sıradan tam matrisle:

$$\text{tam matris: } 3\text{M} \times 3\text{M agirlik } \Rightarrow \text{ imkansiz}$$

3M×3M ağırlığı her katmanda hesaplamak imkansız — gradient descent bu kadar ağırlığı optimize edemez. Çözüm: ML matrisleri **özeldir** (sirkülant/Toeplitz gibi) — aynı operasyon her noktada (ağırlık paylaşımı).

Max pooling boyutu düşürür (9 pikseli 1'e indir, maksimumu al — doğrusal değil ama hızlı); **low-pass filtre** yüksek frekansı (gürültüyü) kırar (örn. [½, ½] komşu ortalama).



Şekil 36.3: ML matrisleri özel olmak zorunda: 3M×3M tam-bağlı matris 9 trilyon ağırlık (imkânsız) iken 3×3 konvolüsyon filtresi yalnız 9 paylaşımlı ağırlık; low-pass [1/2, 1/2] gürültüyü yumuşatır (MSE 0.0945 → 0.0571).

Şekil 36.3 ölçek sorununu somutlaştırır: solda 3M×3M tam-bağlı matris **9 trilyon ağırlık** (imkansız) iken 3×3 konvolüsyon filtresi yalnızca **9 paylaşımlı ağırlık** taşır (log-ölçek bu uçurumu sığdırır); sağda [½, ½] komşu-ortalama low-pass filtresi gürültülü sinyali yumuşatır — gürültü MSE'si **0.0945 → 0.0571** düşer (motor-tanıklı iyileşme). İkisi birlikte ML'in ölçeklenme sırrını gösterir: ağırlık paylaşımı (Toeplitz) parametre sayısını trilyonlardan binlere indirir, pooling boyutu azaltır, low-pass gürültü giderir.

> Builder Notu — Üç Milyona Üç Milyon İmkansızı
>
> "3M×3M imkansız → konvolüsyon + pooling" derin öğrenmenin ölçeklenme sırrı. ML köprüsü: tam-bağlı (fully-connected) katman milyar parametre olurdu; konvolüsyon ağırlık paylaşımıyla bunu binlere indirir (CNN'in zaferi, Ders 32). Max pooling boyut azaltır, low-pass filtre gürültü giderir — ikisi de görüntü işlemenin standart adımları.

36.4 3. Normal Matris: Ortogonal Özvektörlü Aile

Ortogonal özvektörlere sahip matrislerin tüm ailesinin adı:

> *"…a matrix of that form is a normal matrix." — Strang, 32:20*

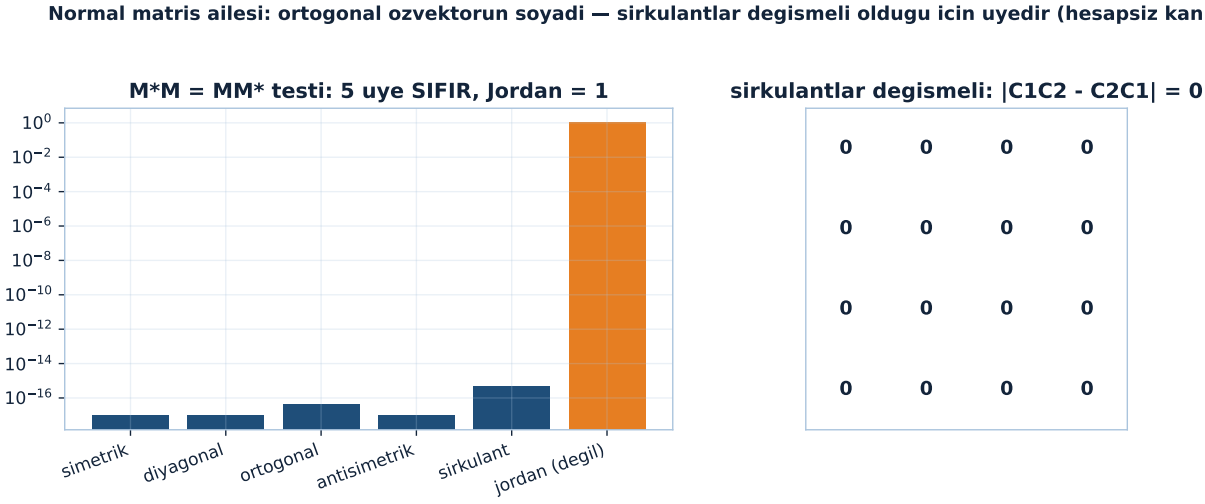
Normal matris: $M = Q\Lambda Q^*$ (ortogonal/üniter özvektörler Q, herhangi karmaşık özdeğerler Λ). Test: M, eşlenik-transpozuyla değişmeli (commute):

$$M^*M = MM^* \quad \text{(normal matris testi)}$$

Bu aile simetrik ($A^\top = A$), diyagonal, ortogonal (real λ veya $|\lambda|=1$), antisimetrik (sanal λ) ve **sirkülant** matrisleri kapsar. Sirkülantlar neden normal? Çünkü **değişmelidirler**:

"…circulant matrices commute. Any 2 circulant matrices commute." — Strang, 36:06

İki sirkülant $C_1C_2 = C_2C_1$ (ikisi de P'nin polinomu, polinomlar değişmeli). Dolayısıyla C ortogonal özvektörlere sahip — ayrıca hesaplamadan biliyoruz.



Şekil 36.4: Normal matris ailesi: M*M* = *MM* testi 5 üyede SIFIR (defect ≤ 4.44e-16), Jordan bloğunda 1.0 — sirkülantlar değişmeli olduğu için ailenin üyesidir (hesapsız kanıt).

Şekil 36.4 aileyi ve değişmeliliği bir arada doğrular: solda altı üyenin $M^*M = MM^*$ defect'i — simetrik, diyagonal, ortogonal, antisimetrik ve sirkülant beşi sıfırda (defect ≤ **4.44e-16**, log-ölçek tabanında), Jordan bloğu [[1,1],[0,1]] ise **1.0** ile turuncu olarak ayrışır (normal **değil** — kontrast). Sağda iki rastgele sirkülantın değişmeliliği: $|C_1C_2 - C_2C_1|$ tam sıfır matristir (motorda $C_1C_2 - C_2C_1 <$ 1e-12). İki sirkülant her zaman değişmeli olduğundan, sirkülantlar normaldir — ortogonal özvektörlere hesaplamadan sahip olduklarını biliriz.

> Builder Notu — Ortogonal Özvektörlülerin Soyadı
>
> "Normal matris = ortogonal özvektörlü aile, M*M*=*MM*" spektral teoremin (Ders 4) en geniş hâli: simetrik-ötesi, karmaşık özdeğerli ama hâlâ ortogonal özvektörlü. ML köprüsü: normal matrisler güvenli köşegenleştirme verir (kararlı, ortonormal taban); sirkülantların normal olması, Fourier tabanının (özvektörler) ortogonal olmasını garantiler — sinyal işlemenin sağlamlığı buradan.

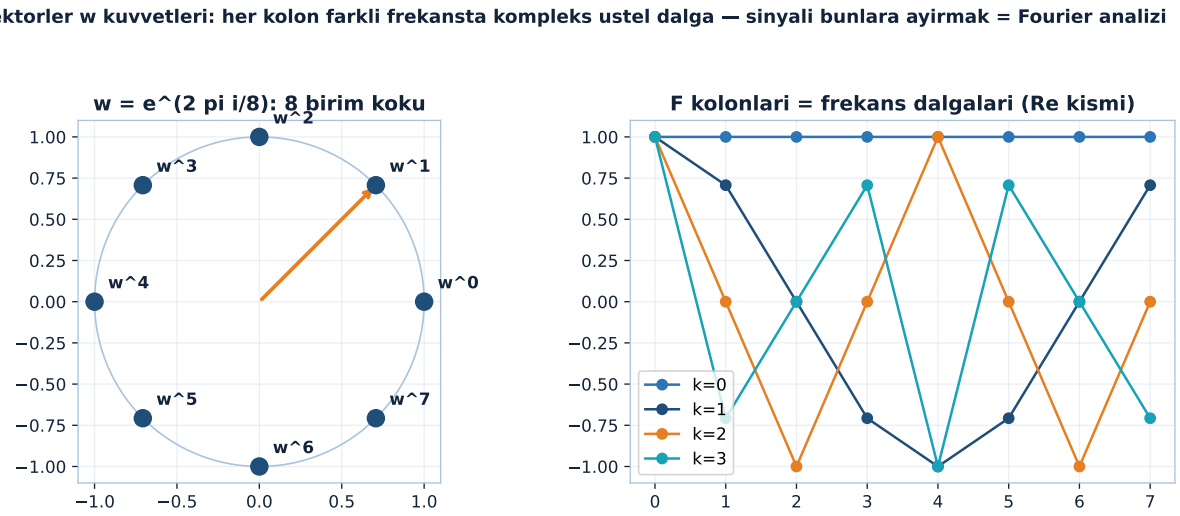
36.5 4. Özvektörler = w Kuvvetleri

C = P'nin polinomu olduğundan, C'nin özvektörleri = P'nin özvektörleri (Ders 30). P'nin özvektörleri çok özel — Fourier bağlantısı (çünkü periyodik):

"…everything is going to be powers of w. e to the 2 pi i over N…" — Strang, 41:59

$$w = e^{2\pi i/n} \quad \text{(birim cemberin 1/n'i)}$$

w, n'inci birim köküdür (karmaşık düzlemde dairenin 1/n'i kadar). λ=1 özvektörü (1,1,1,1); λ=−1 için (1,−1,1,−1); λ=i için (1, i, i^2, i^3); λ=−i için (1, −i, $(-i)^2$, $(-i)^3$). Her özvektörün bileşenleri w'nin kuvvetleri. P'yi (kaydırma) bir özvektörle çarpmak, onu w (veya bir kuvveti) ile ölçekler.



Şekil 36.5: Özvektörler w kuvvetleri: her kolon farklı frekansta kompleks üstel dalga — sinyali bunlara ayırmak = Fourier analizi. Solda n=8 birim çemberinde w = e^(2 pi i/8) kuvvetleri $w^{0..w}7$; sağda F8 kolonlarının reel kısmı k=0..3 frekans dalgaları.

Şekil 36.5 özvektörlerin geometrisini açar: solda $w = e^{2\pi i/8}$'in sekiz kuvveti $w^0, \ldots, w^7$ birim çember üzerinde eşit aralıklı sekiz noktadır (her biri $|\cdot| = 1$), w^1 turuncu okla vurgulanır — kaydırma P'nin özdeğer kümesi. Sağda F8 kolonlarının reel kısmı: k=0 sabit (DC bileşeni), k=1, 2, 3 giderek artan frekansta kosinüs dalgaları. Her kolon farklı frekansta bir kompleks üstel dalgadır; bir sinyali bu kolonlara ayırmak tam olarak Fourier analizidir.

> Builder Notu — Periyodiklik Fourier Doğurur
>
> "Özvektörler = w kuvvetleri, w = e^{2πi/n}" periyodikliğin Fourier'e dönüştüğü an. ML köprüsü: bu birim-kök özvektörleri sinüs/kosinüs dalgalarıdır (karmaşık üstel = frekans bileşenleri); bir sinyali bu özvektörlere ayırmak = Fourier analizi; CNN'lerin frekans-uzayı yorumu ve spektral filtreleme buradan.

36.6 5. Fourier Matrisi F

Tüm özvektörleri kolon kolon dizince **Fourier matrisi** F çıkar — ve bu, tüm sirkülantların özvektör matrisidir:

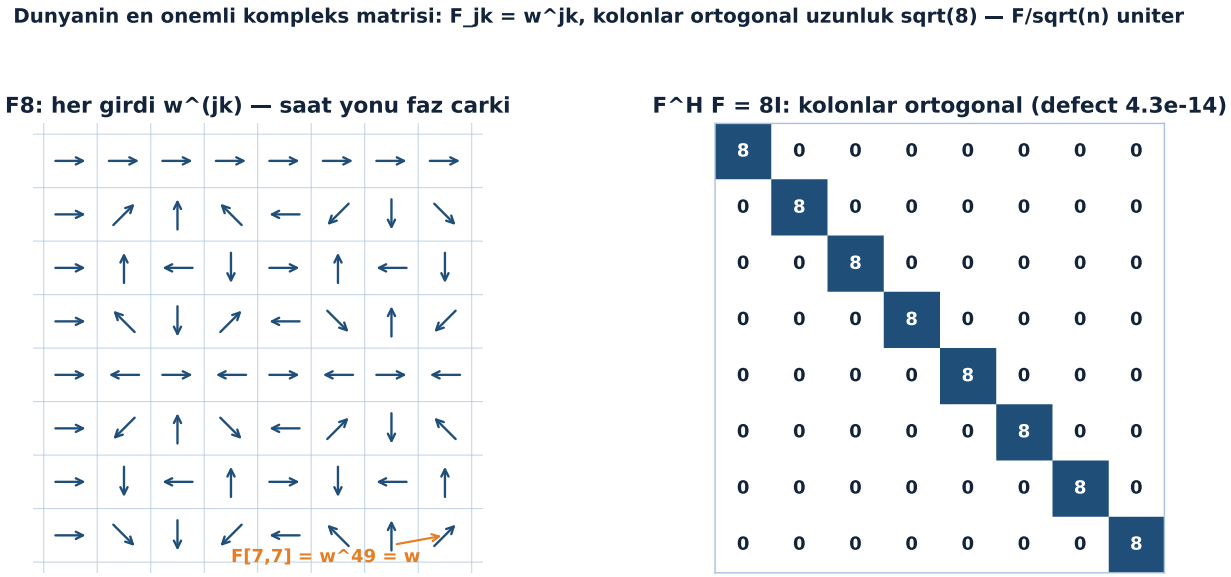
> *"…Fourier matrix equals eigenvector matrix." — Strang, 43:39*

$$F_{jk} = w^{jk}, \qquad w = e^{2\pi i/n}$$

(j, k = 0, 1, …, n−1; her girdi w'nin bir kuvveti, satır×sütun indisi.) İlk kolon hep 1; sonraki kolonlar w'nin artan kuvvetleri.

"…the most important complex matrix in the world…" — Strang, 44:06

F'nin kolonları **ortogonaldir** (uzunluk $\sqrt{n}$); $F/\sqrt{n}$ ortonormaldir (normal matrisin ortogonal özvektörleri, §3). n=8 için son kolon $w^0, w^7, w^{14}, \ldots, w^{49}$ ($w^{48} = 1$ olduğundan $w^{49} = w$).



Şekil 36.6: Dünyanın en önemli kompleks matrisi F8: her girdi F_jk = w^jk bir faz çarkı (sol), kolonlar ortogonal — F^H F = 8I (sağ, defect 4.3e-14), uzunluk √8, F/√n üniter.

Şekil 36.6 dersin amiral görselidir: solda F8'in her girdisi $F_{jk} = w^{jk}$ bir faz oku olarak çizilir — saat yönünde ilerleyen bir faz çarkı; köşe girdisi $F[7,7] = w^{49} = w$ ($w^{48} = 1$ olduğundan, Notion örneğinin motor-teyitli sonucu) turuncu okla işaretlenir. Sağda $|F^H F|$ ısı haritası tam $8I$'dır: köşegende 8, köşegen-dışında 0 — kolonlar ortogonaldir, ortogonallik defect'i n=8 için yalnızca **4.3e-14** (n=2 için 1.2e-16, n=4 için 4.7e-16). Kolonların uzunluğu $\sqrt{8}$, dolayısıyla $F/\sqrt{n}$ üniterdir.

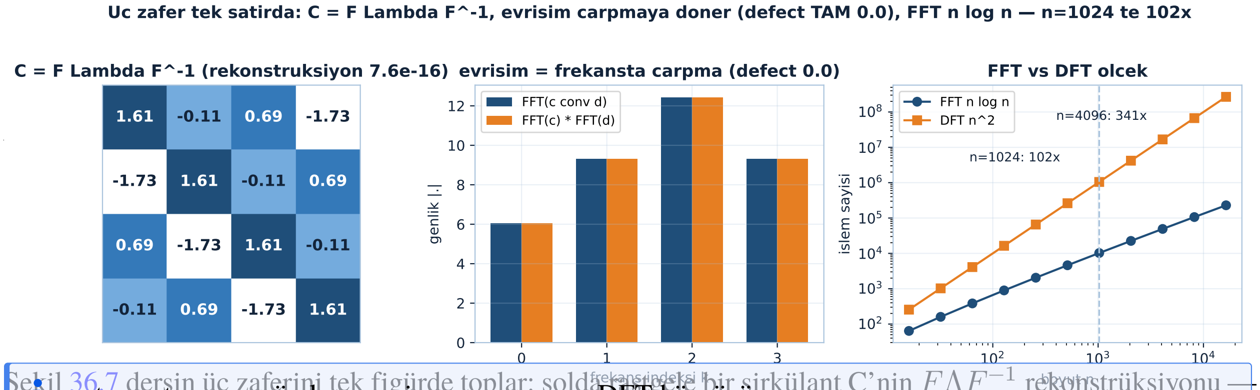
> Builder Notu — Bütün Frekanslar Tek Matriste
>
> "F = özvektör matrisi, $F_{jk} = w^{jk}$" sinyal işlemenin merkez nesnesi. ML köprüsü: F bir sinyali frekans bileşenlerine ayırır (DFT); kolonları farklı frekanslardaki karmaşık üstel dalgalar. Görüntü/ses sıkıştırma (JPEG/MP3), spektral filtreleme ve CNN'lerin Fourier-uzayı analizi hep F'ye dayanır.

36.7 6. Köşegenleştirme ve FFT

Tüm sirkülantlar aynı F'de köşegenleşir:

$$C = F\,\Lambda\,F^{-1}, \qquad \Lambda = \text{diag(c'nin Fourier donusumu)}$$

Özdeğerler Λ, ilk kolonun (c) **Fourier dönüşümü**dür (Fc). İşte konvolüsyon teoremi: C ile çarpmak (zaman uzayında evrişim) = F ile frekans uzayına git, Λ ile çarp (frekansta çarpma), F^{-1} ile geri dön. Evrişim → çarpma.



Şekil 36.7: Üç zafer tek satırda: $C = F \Lambda F^{-1}$, evrişim çarpmaya döner (defect TAM 0.0), FFT $n \log n$ — $n = 1024$'te 102×. Sol C = F Lambda F^-1 rekonstrüksiyon defect'i 7.6e-16; orta FFT(c d) ile FFT(c)·FFT(d) barları üst üste (defect 0.0); sağ FFT n log n vs DFT n² log-log, n=1024'te 102×, n=4096'da 341×.

Şekil 36.7 dersin üç zaferini tek figürde toplar: solda 4×4 bir sirkülant C'nin $F \Lambda F^{-1}$ rekonstrüksiyonu — fark **7.6e-16** (köşegenleştirme kimliği tam), özdeğerler $\lambda =$ np.fft.fft(c) birebir (Ders 30'un köşegen=FFT(c) köprü-sayısının devamı). Ortada konvolüsyon teoremi: $|FFT(c \circledast d)|$ ile $|FFT(c) \cdot FFT(d)|$ barları üst üste oturur — evrişim frekansta çarpmaya döner, defect **tam 0.0** (n=4). Sağda FFT $n \log n$ ile DFT n^2 log-log eğrileri: n=1024'te oran **102×** (FFT 10.240 vs DFT 1.048.576), n=4096'da **341×** — n büyüdükçe açılan kazanç.

motor notu — w-yön konvansiyonu ve numpy DFT köprüsü

Strang tahtada $\lambda = Fc$ der; bizim motor kurulumumuzda (ilk-kolon c, aşağı-kaydırma P) özdeğerler $\lambda =$ np.fft.fft(c) olarak çıkar. Motor `circulant_reconstruct` defect'i tam bunu döndürür: $\lambda =$ np.fft.fft(c). Gerçek c'de Strang'in Fc'si ile numpy'ın `fft(c)`'si eşlenik kümedir (Fc = `conj(fft(c))`) — büyüklükler $|\lambda|$ aynı, yalnız w-yönü konvansiyonu (saat-yönü) farklı. İkinci tanık: numpy DFT köprüsü np.fft.fft$(v) = \text{conj}(F) \cdot v$ (w^{-jk} konvansiyonu). Pedagojik öz değişmez — Notion'ın cümlesine sadık kalırız, [motor notu] yalnızca konvansiyon dalını kaydeder.

Builder Notu — Üç Zafer Tek Satırda

"C = FΛF^{-1}, evrişim → çarpma, FFT O(n log n)" sinyal işlemenin üç zaferi tek satırda. ML köprüsü: büyük-çekirdek konvolüsyonlar FFT ile hızlandırılır (FFT-conv); ses modelleri (spektrogram), görüntü filtreleme ve bazı Transformer varyantları (FNet) Fourier dönüşümünü doğrudan kullanır. FFT, 20. yüzyılın en önemli algoritmalarından — sirkülant köşegenleştirmesinin meyvesi.

36.8 Bu Dersin Özeti

- **Toeplitz vs sirkülant:** Toeplitz (sabit köşegen, döngüsüz) = evrişim; sirkülant = döngüsel evrişim. ML matrisleri böyle özel (ağırlık paylaşımı).
- **ML görüntü:** milyon-piksel feature, 3M×3M tam matris imkansız → konvolüsyon + max pooling + low-pass filtre.
- **Normal matris:** $M^*M = MM^*$; ortogonal özvektörlü aile (simetrik/diyagonal/ortogonal/antisimetrik/sirkülant). Sirkülantlar değişmeli → normal.
- **Özvektörler:** $w = e^{2\pi i/n}$ kuvvetleri (birim kökleri); P'nin ve tüm sirkülantların özvektörleri.
- **Fourier matrisi:** F = özvektör matrisi; $F_{jk} = w^{jk}$; kolonlar ortogonal (uzunluk $\sqrt{n}$), $F/\sqrt{n}$ ortonormal.
- **Köşegenleştirme/FFT:** $C = F\Lambda F^{-1}$, \$ = \$ c'nin DFT'si; evrişim → çarpma (konvolüsyon teoremi); FFT $O(n \log n)$.

! Tek Bir Cümle

Tüm sirkülant matrisler aynı özvektör matrisini — Fourier matrisi F'yi ($F_{jk} = w^{jk}$, $w = e^{2\pi i/n}$) — paylaşır (çünkü normal ve değişmeli); bu matris her sirkülantı köşegenleştirir ($C = F\Lambda F^{-1}$), evrişimi çarpmaya çevirir (konvolüsyon teoremi) ve FFT'yi $O(n \log n)$ kılar.

36.9 Kontrol Soruları

i Soru 1 — Toeplitz ve sirkülant farkı, ML'de hangisi çıkar

İkisi de sabit-köşegenli (linear shift invariant). **Sirkülant** döngüseldir (köşeleri sarar) → döngüsel evrişim; **Toeplitz** döngüsüzdür → (sıradan) evrişim. ML'de genelde Toeplitz çıkar: CNN'in konvolüsyon katmanı aynı filtreyi tüm konumlara kaydırarak uygular (ağırlık paylaşımı), döngüsel değil.

i Soru 2 — Normal matris nedir, sirkülantlar neden normaldir

Normal matris ortogonal/üniter özvektörlere sahiptir; test: $M^*M = MM^*$ (eşlenik-transpozuyla değişmeli). Simetrik, diyagonal, ortogonal, antisimetrik ve sirkülant matrisleri kapsar. Sirkülantlar normaldir çünkü **değişmelidirler** ($C_1C_2 = C_2C_1$, ikisi de P'nin polinomu); dolayısıyla ortogonal özvektörlere sahip olduklarını hesaplamadan biliriz.

i Soru 3 — Fourier matrisi F'nin girdileri ve neden tüm sirkülantların özvektör matrisi

$F_{jk} = w^{jk}$, $w = e^{2\pi i/n}$ (n'inci birim kökü); her girdi w'nin satır×sütun kuvveti. Tüm sirkülantlar P'nin polinomu olduğundan hepsi P'nin özvektörlerini paylaşır; bu özvektörler w'nin kuvvetlerinden kurulu = F'nin kolonları. Kolonlar ortogonal (uzunluk $\sqrt{n}$); $F/\sqrt{n}$ üniter (normal matrisin ortogonal özvektörleri).

> **Soru 4 — C = FΛF^{-1} neden evrişim = çarpma verir ve FFT'nin rolü**
>
> C ile çarpmak (zaman uzayında evrişim) = F ile frekans uzayına git, Λ (diagonal, c'nin Fourier dönüşümü) ile çarp, F^{-1} ile dön. Diagonal çarpma basittir → evrişim frekansta çarpmaya döner (konvolüsyon teoremi). FFT, F·v çarpımını O(n^2) yerine $O(n \log n)$'de yapar (w kuvvetlerinin özyinelemeli yapısı); böylece evrişim/filtreleme çok hızlanır.

36.10 Egzersizler

1. **Birim kök.** n = 4 için $w = e^{2\pi i/4} = i$. w^0, w^1, w^2, w^3'ü hesapla (1, i, −1, −i). Bunların birim çember üzerinde 4 nokta olduğunu doğrula. (Motor tanığı: $w = i$ kuvvetleri $(1, i, -1, -i)$ — hepsi $|\cdot| = 1$.)
2. **Fourier matrisi.** n = 2 için F (2×2) yaz: $w = e^{2\pi i/2} = -1$. $F_{jk} = w^{jk}$ ile $F = [[1, 1], [1, -1]]$ olduğunu göster. Kolonlar ortogonal mi? Uzunlukları $\sqrt{2}$ mi? (Motor tanığı: $F(n{=}2) = [[1, 1], [1, -1]]$; kolonlar ortogonal, uzunluklar $\sqrt{2}$ — birebir.)
3. **Normal test.** Sirkülant $C = [[2, 1], [1, 2]]$ (simetrik). $C^*C = CC^*$ mı? (Simetrik olduğundan normal mi?) Antisimetrik $[[0, 1], [-1, 0]]$ normal mi (eşlenik-transpozla değişmeli mi)? (Motor tanığı: $C = [[2, 1], [1, 2]]$ defect 0 → normal; antisimetrik $[[0, 1], [-1, 0]]$ defect 0 → normal.)
4. **Konvolüsyon teoremi.** İki sirkülantın çarpımı = döngüsel evrişim (Ders 30). Bu Fourier'de neye karşılık gelir? (İpucu: $C = F\Lambda F^{-1}$, çarpım → Λ'ların çarpımı → frekansta nokta-çarpım.) (Motor tanığı: $FFT(c \circledast d) = FFT(c) \cdot FFT(d)$, defect **tam 0.0** n=4'te ve <1e-10 n=3 Notion örneğinde.)
5. **(Ders 32 habercisi)** Fourier/sirkülant fikrini 2 boyuta (görüntü) taşı. Bir görüntüye uygulanan konvolüsyon nasıl çalışır? CNN'ler bunu nasıl katman katman kullanır? Bir tahmin yaz — Ders 32 "ImageNet bir CNN'dir, evrişim kuralı"nı işliyor.

36.11 Sonraki Ders İçin Hazırlık

Ders 32: ImageNet bir CNN'dir, Evrişim Kuralı. Sirkülant/Fourier fikri 2B görüntüye genişler: konvolüsyonel sinir ağları (CNN). Evrişim kuralı (kaydır-çarp-topla), filtre/çekirdek, ağırlık paylaşımı; ImageNet'in CNN devrimini başlatması. fast.ai L15 (conv2d/im2col) ve NYU H6 (CNN) ile birebir köprü.

> **Hazırlık**
>
> Bu dersin Egzersiz 5'inin sorduğu soruyu zihninde tut: bu derste 1B sinyale uyguladığımız sirkülant/Fourier fikrini 2B görüntüye nasıl taşırsın? Ders 32, **konvolüsyonel sinir ağlarını** (CNN) işler — aynı filtre tüm görüntü konumlarına kaydırılarak uygulanır (Toeplitz ağırlık paylaşımı), max pooling boyutu düşürür, ve ImageNet'in CNN devrimi başlar. fast.ai L15 (conv2d/im2col) ve NYU H6 (CNN) bu dersle birebir köprü kurar.

36.12 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
DFT	sirkülant ↔ ayrık Fourier dönüşümü	2m49
Toeplitz	sabit köşegen, döngüsüz = konvolüsyon	9m16
Normal matris	$M^*M = MM^*$; ortogonal özvektör	32m20
Sirkülantlar değişmeli	$C_1C_2 = C_2C_1 \rightarrow$ normal	36m06
Özvektörler = w kuvvetleri	$w = e^{2\pi i/n}$ (birim kökü)	41m59
Fourier matrisi	F = özvektör matrisi; $F_{jk} = w^{jk}$	43m39
Köşegenleştirme/FFT	$C = F\Lambda F^{-1}$; evrişim → çarpma; $O(n \log n)$	44m06

36.13 ML Bağlantıları Özeti

- **Fourier matrisi = en önemli kompleks matris:** tüm sirkülantlar onda köşegenleşir → evrişim teoremi → FFT ($O(n \log n)$).
- **CNN = Toeplitz konvolüsyon:** aynı filtre tüm konumlara (ağırlık paylaşımı); 3M×3M tam matris yerine küçük filtre; fast.ai L15 im2col/conv2d.
- **Max pooling + low-pass:** boyut azaltma + gürültü giderme; görüntü işlemenin standart adımları.
- **FFT-conv:** büyük-çekirdek konvolüsyon FFT ile hızlanır; ses (spektrogram), FNet (Fourier Transformer) doğrudan F kullanır.
- **Normal matris/spektral:** ortogonal özvektörlü aile güvenli köşegenleştirme; graf sinyal işleme (spektral GCN, NYU paralel) Laplacian özvektörlerine genelleştirir.
- **Geriye köprü:** Ders 30 (sirkülant/P/birim kökleri), Ders 3 (ortogonal Q), Ders 4 (özvektör matrisi/köşegenleştirme). Paralel: fast.ai L15 (convolution), NYU H6 (CNN).

! Kapanış

> *"…the most important complex matrix in the world…" — Strang, 44:06*

Fourier matrisi tüm sirkülantların ortak özvektör tabanıdır; bu tek matris evrişimi çarpmaya çevirir, FFT'yi mümkün kılar ve sinyal işleme ile CNN'lerin köprüsüdür.

37 ImageNet bir CNN'dir, Evrişim Kuralı

AlexNet devrimi, evrişim = polinom çarpımı, özdeğerlerin çarpıldığı kural ve Kronecker ile iki boyut

> **Bölüm bilgisi**
>
> Bu ders sirkülant/Fourier'i (Ders 30-31) evrişime ve CNN'lere taşır. Strang'in Ders 32 videosu (≈47 dk) ve OCW Lecture 32 (≈32 dk okuma) temel alınmıştır. Önkoşul Ders 30-31 (sirkülant/Fourier/FFT) ve Ders 26 (sinir ağı yapısı).

37.1 Bu Derste Ne Var?

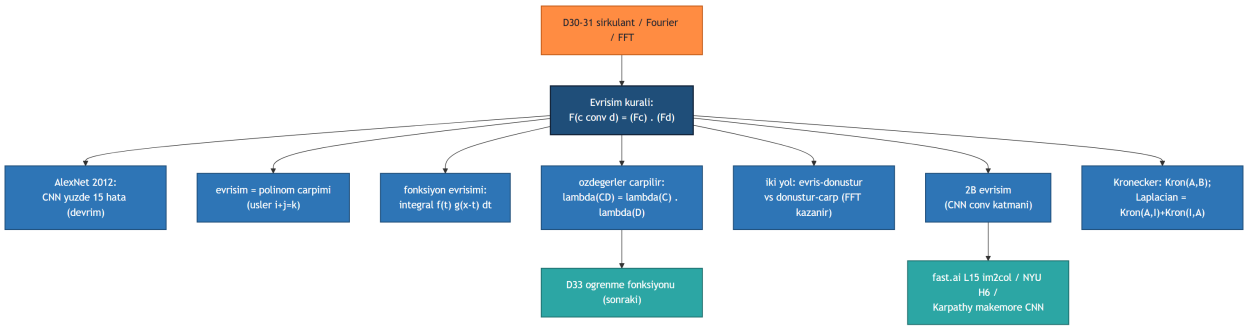
Ders 30-31'in sirkülant/Fourier fikrini evrişime ve görüntülere taşıyoruz. **Evrişim** (convolution) = polinom çarpımı; **evrişim kuralı** Fourier ile evrişimi çarpmaya çevirir; CNN'ler bu yapıyı görüntüye uygular.

Beş sonuç:

1. **ImageNet/AlexNet (2012):** Krizhevsky-Sutskever-Hinton; CNN %15 hata (2.'lik %26) — derin öğrenme devrimi. CNN az ağırlık (paylaşım).
2. **Evrişim tanımı:** $(c * d)_k = \sum_{i+j=k} c_i\, d_j$; polinom çarpımından gelir. Fonksiyon: $(f * g)(x) = \int f(t)\, g(x-t)\, dt$.
3. **Evrişim kuralı:** sirkülant CD'nin özdeğerleri = $\lambda(C) \odot \lambda(D)$ (bileşen-bileşen); $F(c \circledast d) = (Fc) \odot (Fd)$.
4. **FFT iki yol:** ya evriş→dönüştür ($O(n^2)$), ya ayrı-dönüştür→çarp ($2n \log n + n$). FFT ikincisini hızlı kılar.
5. **2B + Kronecker:** 2B evrişim çift-integral; Kronecker çarpımı (1B→2B), Laplacian = Kron(A,I)+Kron(I,A) (Kronecker toplamı).

> *"…the MATLAB command is Kron." — Strang, 38:55*

Şekil 37.1 dersin iskeletini gösterir: merkezdeki **evrişim kuralı** $F(c \circledast d) = (Fc) \odot (Fd)$ fikrinden yedi dala ayrılır — AlexNet 2012 (CNN %15 hata, ImageNet devrimi), evrişim = polinom çarpımı (üsler $i + j = k$), fonksiyon evrişimi $\int f(t)\, g(x-t)\, dt$, özdeğerler çarpılır $\lambda(CD) = \lambda(C) \odot \lambda(D)$, iki yol (evriş-dönüştür vs dönüştür-çarp, FFT kazanır), 2B evrişim (CNN conv katmanı) ve Kronecker Kron(A,B) + Laplacian = Kron(A,I)+Kron(I,A); köprü düğümleri D30-31 (sirkülant/Fourier/FFT), D33 öğrenme fonksiyonu (sonraki) ve fast.ai L15 im2col / NYU H6 / Karpathy makemore CNN dalları önceki ile sonraki derslere bağlar.



Şekil 37.1: Ders 32 kavram haritasi: evrisim kurali F(c conv d) = (Fc) . (Fd) sirkulantlari ortak Fourier tabaninda kosegenlestirir, kosegenler carpilir. Yedi dal: AlexNet 2012 CNN yuzde 15 hata (ImageNet devrimi); evrisim = polinom carpimi (usler i+j=k toplanir); fonksiyon evrisimi integral f(t) g(x-t) dt; ozdegerler carpilir lambda(CD) = lambda(C) . lambda(D); iki yol evris-sonra-donustur vs donustur-sonra-carp (FFT kazanir, buyuk-tamsayi/kriptografi); 2B evrisim = CNN conv katmani; Kronecker Kron(A,B) 1B yapiyi 2B ye tasir, Laplacian = Kron(A,I)+Kron(I,A). Koprular: D30-31 sirkulant/Fourier/FFT, D33 ogrenme fonksiyonu (sonraki), fast.ai L15 im2col / NYU H6 / Karpathy makemore CNN.

Builder Notu — Sinyal İşleme Derin Öğrenmeyle Buluşunca

- **CNN = ağırlık paylaşımı:** AlexNet 60M parametre, 5 conv + 3 tam-bağlı, max pooling, dropout; konvolüsyon tam matrisi küçük filtreye indirir.
- **Evrişim kuralı + FFT:** büyük evrişimleri F-dönüştür-çarp-geri-dön ile $O(n \log n)$'de yap; büyük-tamsayı çarpımı, ses, FFT-conv.
- **Kronecker = 2B yapı:** 1B matrislerden 2B (görüntü) matrisi; 2B Fourier = Kron(F,F), 2B Laplacian = Kronecker toplamı.
- **Geriye köprü:** Ders 30-31 (sirkülant/cyclic convolution/Fourier/FFT), Ders 4 (özdeğer çarpımı). Paralel: fast.ai L15 (conv2d/im2col), NYU H6 (CNN), Karpathy makemore CNN.

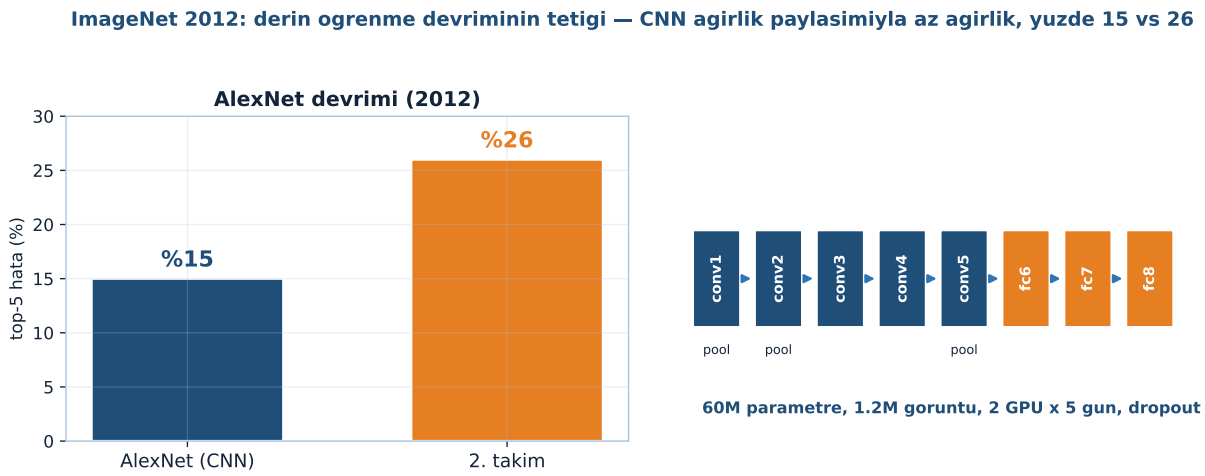
Tek cümle: Evrişim polinom çarpımıdır; evrişim kuralı (Fourier'de çarpmaya döner) FFT ile evrişimi hızlandırır; CNN'ler bu evrişimi (ağırlık paylaşımıyla) görüntüye uygular, 2B'ye geçiş Kronecker çarpımıyla yapılır.

37.2 1. ImageNet ve AlexNet (2012)

Derin öğrenmenin dönüm noktası: Krizhevsky-Sutskever-Hinton'ın 2012 ImageNet yarışması makalesi (AlexNet).

> *"We trained a large deep convolutional neural network." — Strang, 2:33*

1.2 milyon görüntü, 2 GPU'da 5 gün eğitim. Sonuç: top-5 test hatası **%15** (ikinci takım %26) — devrim niteliğinde. Ağ: 60 milyon parametre, 5 konvolüsyon katmanı + 3 tam-bağlı katman, max pooling (boyut indirme), dropout (aşırı-öğrenme önleme). Kritik: **CNN az ağırlık** kullanır — tam matris yerine sadece üst-satır (filtre) ağırlıkları (Ders 31 ağırlık paylaşımı).



Şekil 37.2: AlexNet'in 2012 ImageNet zaferi — derin öğrenme devriminin tetiği. Sol: top-5 hata, AlexNet (CNN) %15 (navy) vs ikinci takım %26 (turuncu); on bir puanlık fark bir CNN'in eseri. Sağ: AlexNet mimari şeridi — 8 katman zinciri, 5 evrişim katmanı (conv1..conv5, navy) + 3 tam bağlı katman (fc6..fc8, turuncu), conv1/conv2/conv5 sonrası max pool. 60M parametre, 1.2M görüntü, 2 GPU × 5 gün, dropout: ağırlık paylaşımı sayesinde derin ama az ağırlıklı.

Şekil 37.2 2012 sıçramasını iki panelde toplar: solda top-5 hata barı — AlexNet (CNN) **%15** (navy) ile ikinci takım **%26** (turuncu) arasındaki on bir puanlık fark tek bir CNN'in eseridir; sağda AlexNet mimari şeridi — sekiz katman zinciri, 5 evrişim katmanı (conv1..conv5, navy) + 3 tam-bağlı katman (fc6..fc8, turuncu), conv1/conv2/conv5 sonrası max pool, altta **60M parametre, 1.2M görüntü, 2 GPU × 5 gün, dropout**. Ağırlık paylaşımı sayesinde ağ derin ama az ağırlıklıdır.

> Builder Notu — On Beşe Karşı Yirmi Altı
>
> "AlexNet 2012 = derin öğrenme devriminin tetiği" ML tarihinin dönüm noktası. ML köprüsü: %15 vs %26 sıçraması (ve sonraki yıllarda hızla düşmesi) CNN'in görüntü tanımayı dönüştürdüğünü kanıtladı; dropout, ReLU, GPU eğitimi bu makaleyle yaygınlaştı. Sutskever sonra OpenAI'ı kurdu — bu makale modern AI'ın başlangıç noktalarından.

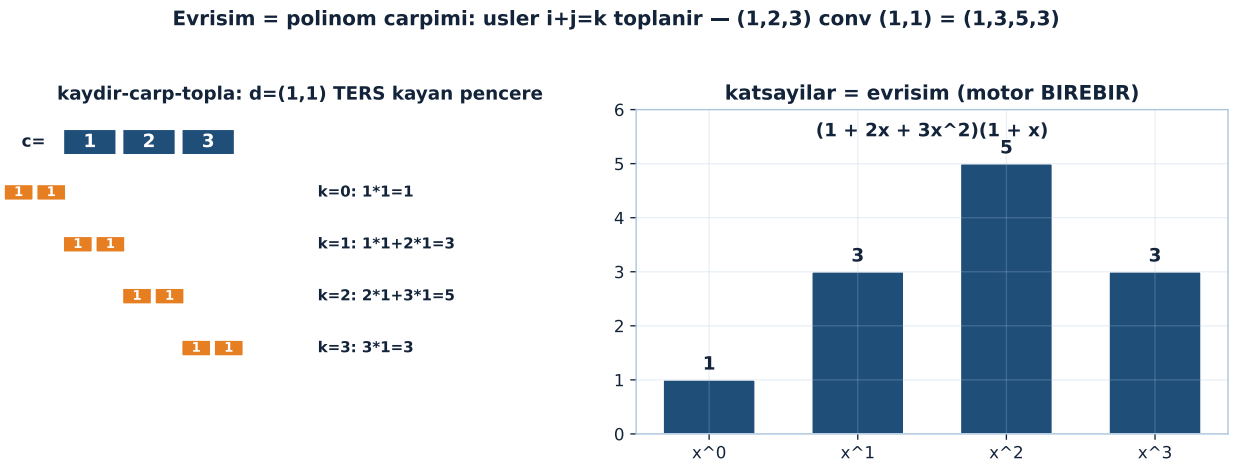
37.3 2. Evrişim Tanımı: Polinom Çarpımı

Evrişim (convolution) iki vektörü "kaydır-çarp-topla" ile birleştirir. k'ıncı bileşen, indisleri k'ya toplanan tüm çarpımların toplamı:

> *"…index i plus j adds to k." — Strang, 6:04*

$$(c * d)_k = \sum_{i+j=k} c_i \, d_j = \sum_i c_i \, d_{k-i}$$

Nereden gelir? **Polinom çarpımından.** c'yi ($c_0 + c_1 x + \ldots$) ve d'yi ($d_0 + d_1 x + \ldots$) polinom say, çarp; x^k'nin katsayısı tam bu toplamdır (üsler $i + j = k$ toplanır). Evrişim = polinom çarpımının katsayıları.



Şekil 37.3: Evrisim = polinom carpimi: usler i+j=k toplanir — (1,2,3) conv (1,1) = (1,3,5,3)

Şekil 37.3 tanımı iki açıdan somutlaştırır: solda kaydır-çarp-topla şeması — c=(1,2,3) üç navy kutu, altında d=(1,1) **ters çevrilmiş** turuncu kayan pencere dört konumda (k=0..3), her konumda örtüşen çarpımların toplamı sağda yazılı (k=0: 1, k=1: 3, k=2: 5, k=3: 3); sağda polinom karşılığı — $(1 + 2x + 3x^2)(1 + x)$ çarpımının katsayıları bar olarak **(1,3,5,3)** (x^0, x^1, x^2, x^3 üsleri), `poly_mult_coeffs` motoruyla birebir. Evrişim ile polinom çarpımı aynı sayıyı verir — üsler $i + j = k$ toplandığı için.

> Builder Notu — Kaydır Çarp Topla
>
> "Evrişim = polinom çarpımı" sinyal işlemenin cebirsel kökeni. ML köprüsü: CNN'deki konvolüsyon katmanı tam bu kaydır-çarp-topla; bir filtre (çekirdek) görüntü üzerinde kaydırılır, her konumda nokta-çarpım. fast.ai L15 im2col bu evrişimi büyük matris çarpımına açar (GPU verimliliği için).

37.4 3. Fonksiyon Evrişimi

Aynı fikir sürekli fonksiyonlarda — toplam yerine integral:

> *"…f star g at x." — Strang, 10:02*

$$(f * g)(x) = \int f(t)\, g(x - t)\, dt$$

Vektör evrişiminin sürekli karşılığı: $c_i \to f(t)$, $d_{k-i} \to g(x - t)$, toplam → integral. g, **ters çevrilip kaydırılır** $(x - t)$; t değiştikçe kayma miktarı değişir — bir filtrenin tam yaptığı şey. (Daire koymadım: bu sıradan/döngüsüz evrişim; periyodik fonksiyonlarda döngüsel olur.)

> Builder Notu — Ters Çevirip Kaydırmak
>
> "Fonksiyon evrişimi = ters çevir, kaydır, çarp, integre et" sinyal işlemenin sürekli dili. ML köprüsü: filtreleme (sinyal/görüntü), olasılıkta iki bağımsız değişkenin toplamının dağılımı (yoğunlukların evri-

şimi), ve sürekli CNN'ler/Gauss bulanıklaştırma hep bu integral. Ters-çevirme, evrişimi korelasyondan ayıran detaydır (CNN'ler aslında çapraz-korelasyon kullanır).

37.5 4. Evrişim Kuralı: Özdeğerler Çarpılır

Ders 30-31'i birleştir. İki sirkülant C, D çarpılınca CD yine sirkülanttır; köşegenleri $c \circledast d$ (döngüsel evrişim). CD'nin özdeğerleri ne? Sirkülantlar **değişmeli** ve aynı özvektörleri (Fourier) paylaştığından:

> *"…the eigenvalues just multiply." — Strang, 28:52*

$$\lambda(CD) = \lambda(C) \odot \lambda(D) \quad \text{(bilesen-bilesen carpim)}$$

Özdeğerler bileşen-bileşen (MATLAB `.*`, Hadamard çarpımı) çarpılır. Özdeğerler = ilk kolonun Fourier dönüşümü olduğundan (Ders 31), bu şu demek: $c \circledast d$'nin Fourier dönüşümü = (c'nin Fourier'i) $\odot$ (d'nin Fourier'i). Zaman uzayında evrişim → frekans uzayında çarpma. Şekil 37.4 bu kuralı ve aşağıdaki iki yolu (§5) tek figürde doğrular.

> Builder Notu — Köşegenler Çarpılır
>
> "Evrişimin özdeğerleri çarpılır" konvolüsyon teoreminin spektral ispatı: sirkülantlar ortak özvektör (Fourier) tabanında köşegenleşir, köşegenler (özdeğerler) çarpılır. ML köprüsü: bu, evrişimi frekans uzayında nokta-çarpıma çeviren temel teorem; ses (spektrogram), görüntü filtreleme ve FFT-tabanlı konvolüsyon hep buna dayanır.

37.6 5. İki Yol: Evrişim Kuralı + FFT

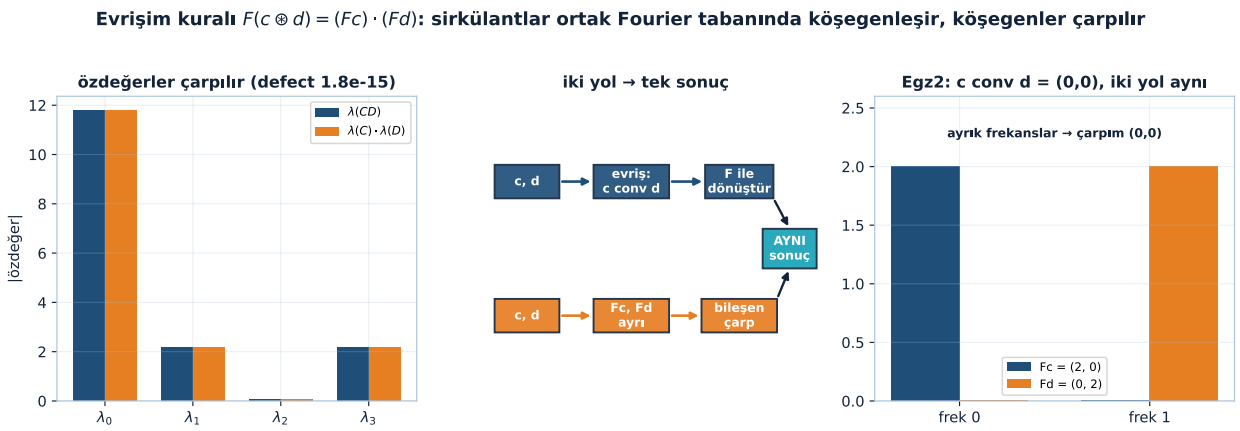
Evrişim kuralı (C kuralı) aynı sonuca iki yol verir:

> *"…Convolve then transform by F. Or transform separately by F." — Strang, 30:50*

$$F(c \circledast d) = (Fc) \odot (Fd)$$

Yol 1: önce evriş ($c \circledast d$), sonra Fourier'le dönüştür. **Yol 2:** her birini ayrı dönüştür (Fc, Fd), sonra bileşen-bileşen çarp. İkisi aynı sonuç. Neden önemli? Çünkü Fourier dönüşümü **FFT** ile çok hızlı ($O(n \log n)$). FFT'nin varlığı, Yol 2'yi çok cazip kılar.

Şekil 37.4 dersin amiral görselidir ve kuralı üç açıdan doğrular: solda rastgele c_4, d_4 (seed 32) için $\lambda(CD)$ (navy) ile $\lambda(C) \odot \lambda(D)$ (turuncu) barları üst üste oturur — ortak Fourier tabanında köşegenler bileşen-bileşen çarpılır, defect yalnızca **1.8e-15** (n=4; n=8 için <1e-10); ortada iki güzergah aynı sonuca varır — üst yol (navy) evriş yapıp F ile dönüştürür, alt yol (turuncu) önce ayrı dönüştürüp bileşen-bileşen çarpar, ikisi "AYNI sonuç"ta buluşur; sağda Egz2 sıfır-evrişim — $c = (1, 1)$ ile $d = (1, -1)$ ayrık frekanslarda yaşar, $Fc = (2, 0)$ ile $Fd = (0, 2)$ çakışmaz, çarpımları (0,0) olur ve $c \circledast d = (0, 0)$ çıkar (iki yol da aynı sıfırı verir — `conv_rule_defect` Strang F konvansiyonuyla D30 örneği (3,1,2)/(4,6,1) için de <1e-10).



Şekil 37.4: Evrişim kuralı $F(c \circledast d) = (Fc) \cdot (Fd)$ üç açıdan. Sol: rastgele c_4, d_4 (seed 32) için $\lambda(CD)$ ile $\lambda(C) \cdot \lambda(D)$ çubukları üst üste oturur — ortak Fourier tabanında köşegenler bileşen-bileşen çarpılır, defect 1.8e-15. Orta: iki güzergah aynı sonuca varır — üstte evriş yapıp dönüştür, altta önce dönüştürüp bileşen-bileşen çarp. Sağ: Egz2 sıfır-evrişim — $c = (1, 1)$ ve $d = (1, -1)$ ayrık frekanslarda yaşar, $Fc = (2, 0)$ ile $Fd = (0, 2)$ çakışmaz, çarpımları (0,0) olur ve $c \circledast d = (0, 0)$ çıkar.

> **Builder Notu — Aynı Hedefe İki Güzergah**
>
> "İki yol: evriş-sonra-dönüştür vs dönüştür-sonra-çarp" konvolüsyon teoreminin pratik gücü. ML köprüsü: bu seçim büyük-çekirdek konvolüsyonları hızlandırır — uzun filtrelerle FFT-conv (dönüştür-çarp-geri-dön) doğrudan evrişimden hızlıdır; ses işleme ve bazı CNN katmanları bunu kullanır.

37.7 6. Maliyet: N² vs N log N

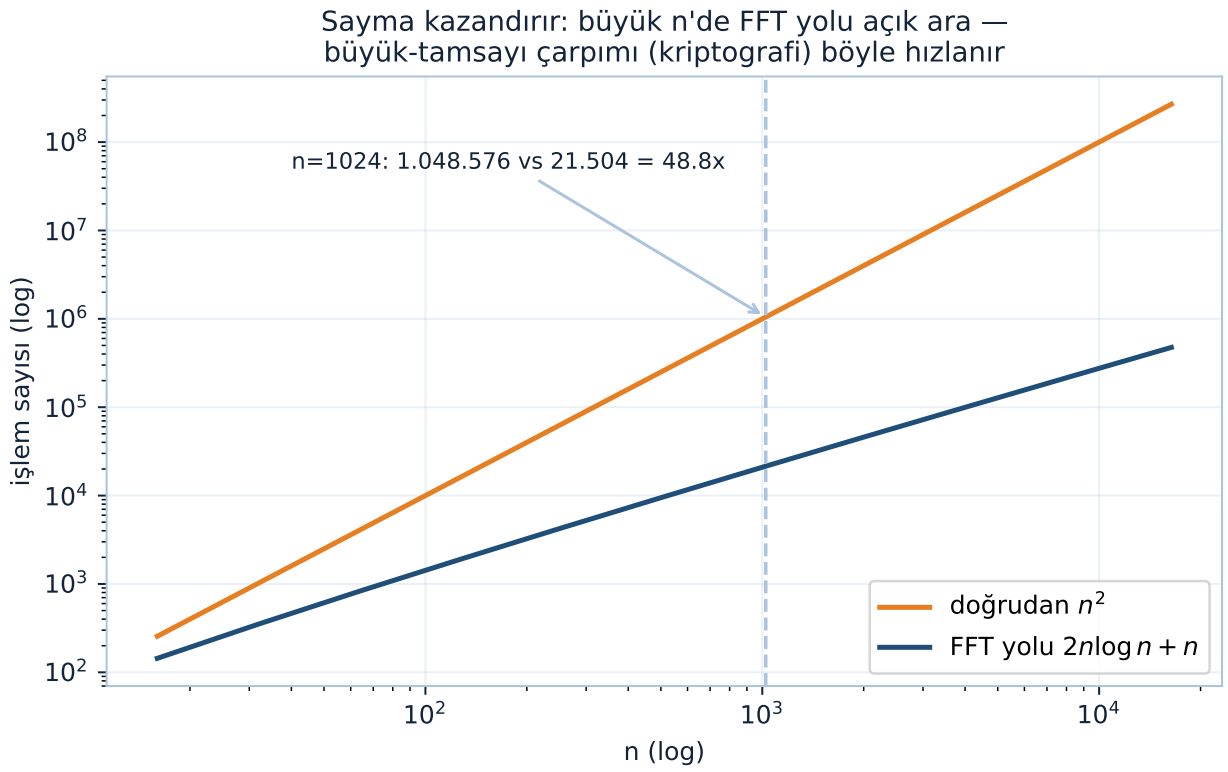
Hangi yol hızlı? Maliyet say. **Doğrudan evrişim:** her c her d'yi çarpar → $O(n^2)$ küçük çarpım. **FFT yolu:** iki Fourier dönüşümü $(2 \cdot n \log n)$ + bileşen-bileşen çarpım (n) = effektif $n \log n$:

$$\text{evris: } O(n^2) \qquad \text{FFT yolu: } 2(n \log n) + n$$

"…this is the fast way." — Strang, 35:32

Büyük n'de FFT yolu kazanır. Uygulama: iki büyük tamsayı çarpmak (kriptografide 128+ haneli) — ayrı ayrı FFT al, bileşen-bileşen çarp, geri dönüştür. Evrişim kuralı + FFT bunu mümkün kılar.

Şekil 37.5 iki yolun maliyetini log-log eksende açar: turuncu eğri doğrudan evrişim n^2, navy eğri FFT yolu $2n \log n + n$; n=1024'te gri kesik çizgi iki eğriyi ayırır — doğrudan yol **1.048.576** işlem ister, FFT yolu yalnızca **21.504**, oran **48.8×** (kabaca ~50×). Büyük n büyüdükçe makas açılır; büyük-tamsayı çarpımı (kriptografide 128+ haneli sayılar) tam bu nedenle FFT ile hızlanır.



Şekil 37.5: Sayma kazandırır: doğrudan evrişim $O(n^2)$, FFT yolu $2n \log n + n$. log-log eksende iki eğri açılır; n=1024'te doğrudan 1.048.576 işlem ister, FFT yolu yalnızca 21.504 — oran 48.8x (kabaca ~50x). Büyük-tamsayı çarpımı (kriptografi) tam bu nedenle FFT ile hızlanır.

Builder Notu — Sayma Kazandırır

"FFT yolu $O(n \log n)$, doğrudan $O(n^2)$" hesaplamalı kazanç. ML köprüsü: büyük-tamsayı çarpımı (kriptografi), polinom çarpımı, büyük-çekirdek konvolüsyon — hepsi FFT-conv ile hızlanır. Modern derin öğrenmede küçük çekirdekler (3×3) için doğrudan conv daha hızlı, ama uzun-bağlam/büyük-çekirdek modeller FFT'ye döner.

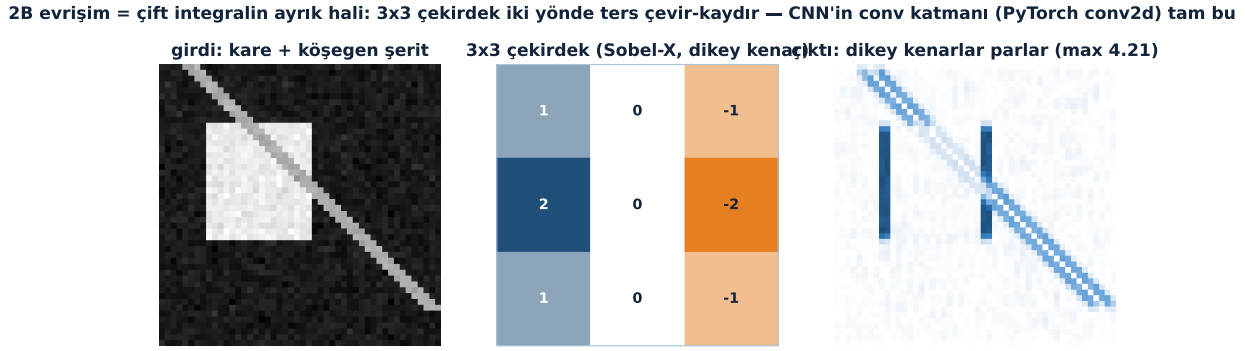
37.8 7. 2B Evrişim (Görüntü)

Görüntüler 2 boyutludur — evrişim çift integrale dönüşür:

> *"…what is a two dimensional convolution?" — Strang, 36:38*

$$(f * g)(x, y) = \iint f(t, u)\, g(x - t,\ y - u)\, dt\, du$$

Aynı imza: g'nin argümanı $(x - t, y - u)$ — iki yönde ters çevir-kaydır. Bir görüntüyü bir 2B çekirdekle (filtre) evirmek tam bu. CNN'lerin konvolüsyon katmanı 2B evrişimdir: küçük bir filtre (örn. 3×3) görüntü boyunca her iki yönde kaydırılır.



Şekil 37.6: 2B evrişim = çift integralin ayrık hali: 3x3 çekirdek iki yönde ters çevir-kaydır — CNN'in conv katmanı (PyTorch conv2d) tam bu

Şekil 37.6 2B evrişimi gerçek bir kenar-dedektörüyle gösterir: solda girdi görüntüsü (parlak kare + köşegen şerit), ortada 3×3 Sobel-X çekirdeği (dikey kenar dedektörü, DIVERGE ısı haritası), sağda `conv2d_full` çıktısı $|\text{kenar}|$ — dikey kenarlar parlar, max yanıt **4.21**. Çekirdek görüntü boyunca iki yönde ters çevrilip kaydırılır (çift-toplam); bu PyTorch `conv2d`'nin tam yaptığı işlemdir. (Sobel ayrıca **ayrılabilirdir** — kolon $(1, 2, 1) *$ satır $(1, 0, -1)$ evrişimi birebir, §8 Kronecker önizlemesi.)

Builder Notu — Çift İntegralin Filtresi

"2B evrişim = çift integral, iki yönde kaydır" CNN'in çekirdek operasyonu. ML köprüsü: PyTorch `conv2d` tam bu; 3×3 çekirdek tüm görüntü konumlarına uygulanır (ağırlık paylaşımı). fast.ai L15 im2col 2B evrişimi tek bir büyük matris çarpımına açar (GPU için); kenar/doku dedektörleri ilk CNN katmanlarının öğrendiği 2B filtrelerdir.

37.9 8. Kronecker Çarpımı ve Toplamı

1B'den 2B'ye geçişin matris aracı **Kronecker çarpımı**:

> *"…the MATLAB command is Kron." — Strang, 38:55*

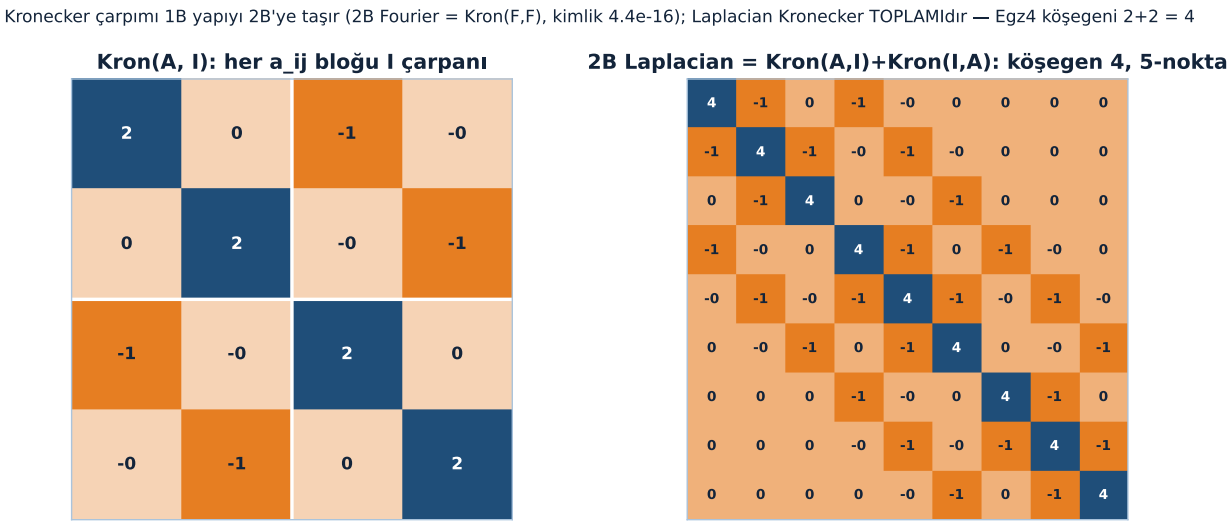
İki n×n matris A, B'den n^2×n^2 bir matris üretir; her a_{ij} bloğu B ile çarpılır:

$$\text{Kron}(A,B) = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \cdots & a_{nn}B \end{bmatrix} \quad (n^2 \times n^2)$$

Örnek: 2B Fourier dönüşümü = Kron(F, F). 2B Laplacian (her iki yönde −1, 2, −1 ikinci fark, 5-nokta şablon): tek Kronecker çarpımı değil, **Kronecker toplamı**:

$$\text{Laplacian} = \text{Kron}(A,I) + \text{Kron}(I,A)$$

Kron(A,I) x yönündeki ikinci türevi, Kron(I,A) y yönündekini verir; toplamı 2B Laplacian.



Şekil 37.7: Kronecker çarpımı 1B yapıyı 2B'ye taşır (2B Fourier = Kron(F,F), kimlik 4.4e-16); Laplacian Kronecker TOPLAMIdır — Egz4 köşegeni 2+2 = 4

Şekil 37.7 Kronecker yapısını iki ısı haritasıyla gösterir: solda Kron(A, I) 4×4 — A=[[2,−1],[−1,2]]'nin her a_{ij} girdisi 2×2 birim matrisle çarpılır (sol-üst blok 2I), beyaz çizgiler blok sınırlarını ayırır; sağda 2B Laplacian = Kron(A,I)+Kron(I,A) 9×9 (3×3 ızgara) — köşegende 4, iç satırlarda 5-nokta şablonu {4, −1×4}. Kronecker-vec kimliği $(B \otimes A) \cdot \text{vec}(X) = \text{vec}(AXB^\top)$ motorda **4.44e-16**; 2B Fourier = Kron(F,F) tanığı ($(F \otimes F)\text{vec}(X) = \text{vec}(FXF^\top)$, F simetrik) yine **4.44e-16**. Laplacian tek çarpım değil Kronecker **toplamıdır** — Egz4'ün köşegeni 2+2 = 4.

Builder Notu — Bloklarla İki Boyut Kurmak

"Kronecker çarpımı = 2B yapı, Kronecker toplamı = 2B Laplacian" çok-boyutlu operatörlerin matris dili. ML köprüsü: 2B/3B konvolüsyon, ayrılabilir (separable) filtreler (Kron(satır, sütun)), ve tensör işlemleri (PyTorch'ta `kron`) bu yapıyı kullanır; çözünürlük-uzayı yöntemleri (PDE çözücüler, fizik-bilgili ağlar) 2B Laplacian'ı Kronecker toplamıyla kurar.

37.10 Bu Dersin Özeti

- **ImageNet/AlexNet (2012):** Krizhevsky-Sutskever-Hinton; CNN %15 hata (devrim); 60M parametre, conv+pool+dropout; CNN az ağırlık (paylaşım).
- **Evrişim:** $(c * d)_k = \sum c_i\, d_{k-i}$ (polinom çarpımı); fonksiyon: $(f * g)(x) = \int f(t)\, g(x-t)\, dt$.
- **Evrişim kuralı:** sirkülant CD özdeğerleri = $\lambda(C) \odot \lambda(D)$; $F(c \circledast d) = (Fc) \odot (Fd)$. Evrişim $\rightarrow$ frekansta çarpma.
- **İki yol + FFT:** evriş-dönüştür ($O(n^2)$) vs ayrı-dönüştür-çarp ($2n \log n + n$); FFT ikinciyi hızlı kılar.
- **2B evrişim:** çift integral $\iint f(t,u)\, g(x-t, y-u)\, dt\, du$; CNN'in conv katmanı.
- **Kronecker:** çarpım Kron(A,B) ($n^2 \times n^2$, bloklar $a_{ij}B$) 2B yapı; Laplacian = Kron(A,I)+Kron(I,A) (Kronecker toplamı).

Tek Bir Cümle

Evrişim polinom çarpımıdır $(c * d)_k = \sum c_i\, d_{k-i}$; evrişim kuralı onu Fourier'de çarpmaya çevirir ($F(c \circledast d) = (Fc) \odot (Fd)$) ve FFT ile hızlandırır; CNN'ler bu evrişimi ağırlık paylaşımıyla görüntüye uygular (AlexNet), 2B'ye geçiş Kronecker çarpımıyla yapılır.

37.11 Kontrol Soruları

Soru 1 — AlexNet (2012) neden dönüm noktası ve CNN neden az ağırlık kullanır

AlexNet ImageNet 2012'de top-5 hatayı %15'e indirdi (ikinci takım %26) — CNN'in görüntü tanımayı dönüştürdüğünü kanıtladı. CNN az ağırlık kullanır çünkü **ağırlık paylaşımı**: aynı filtre (üst-satır ağırlıkları, sirkülant/Toeplitz) tüm konumlara uygulanır — tam n^2-ağırlık matrisi yerine küçük bir çekirdek. 60M parametre yine çoktu ama tam-bağlıdan çok azdı.

Soru 2 — Evrişim neden polinom çarpımıdır

c ve d'yi polinom katsayıları say ($c_0 + c_1 x + \ldots$, $d_0 + d_1 x + \ldots$). Çarpımda x^k'nin katsayısı, üsleri k'ya toplanan tüm $c_i d_j$ çarpımlarının toplamıdır ($i + j = k$). Bu tam $(c * d)_k = \sum_i c_i\, d_{k-i}$. Yani evrişim = polinom çarpımının katsayıları (kaydır-çarp-topla).

Soru 3 — Evrişim kuralı nedir ve neden FFT ile hızlıdır

$F(c \circledast d) = (Fc) \odot (Fd)$: döngüsel evrişimin Fourier dönüşümü = ayrı Fourier dönüşümlerinin bileşen-bileşen çarpımı (sirkülantlar ortak Fourier özvektörlerinde köşegenleşir, özdeğerler çarpılır). FFT ile hızlı: doğrudan evrişim $O(n^2)$; ama "iki kez FFT $(2n \log n)$ + bileşen-çarpım (n)" yolu $O(n \log n)$ — büyük n'de çok daha hızlı.

Soru 4 — Kronecker çarpımı ve toplamı nedir, 2B Laplacian hangisidir

Kronecker çarpımı Kron(A,B): iki n×n matristen $n^2 \times n^2$ matris, her a_{ij} bloğu B ile çarpılır — 1B'den 2B yapıya geçiş (örn. 2B Fourier = Kron(F,F)). 2B Laplacian tek çarpım değil **Kronecker toplamıdır**: Kron(A,I) + Kron(I,A) — biri x yönündeki ikinci türevi, diğeri y yönündekini verir, toplamı 5-nokta şablonu.

37.12 Egzersizler

1. **Evrişim hesabı.** $(1, 2, 3)$ ile $(1, 1)$ vektörlerini evir (sıradan, döngüsüz). Polinom $(1+2x+3x^2)(1+x)$ çarpımıyla doğrula. (Motor tanığı: $(1, 2, 3) * (1, 1) = (1, 3, 5, 3)$, `poly_mult_coeffs` ile birebir; D30 örneği $(3, 1, 2) * (4, 6, 1) = (12, 22, 17, 13, 2)$ de tutarlı.)
2. **Konvolüsyon teoremi.** n=2'de c=(1,1), d=(1,−1). Döngüsel evrişim $c \circledast d$'yi doğrudan hesapla. Sonra Fc, Fd al (F=[[1,1],[1,−1]]), bileşen-bileşen çarp, F^{-1} ile geri dön; aynı sonucu bulduğunu göster. (Motor tanığı: $c \circledast d = (0, 0)$ doğrudan; Fourier yolu $Fc = (2, 0)$, $Fd = (0, 2) \to \odot = (0, 0) \to F^{-1} \to$ (0,0). İki yol aynı — ve örnek özel: c, d ayrık frekanslarda yaşar, çarpımları sıfırlanır.)
3. **Maliyet.** n = 1024 için doğrudan evrişim (n^2) ve FFT yolu $(2n \log_2 n + n)$ kaç işlem? FFT kaç kat hızlı (kabaca)? (Motor tanığı: doğrudan **1.048.576** vs FFT yolu $2 1024 10 + 1024 = $ **21.504** → oran **48.8×**, kabaca ~50×.)
4. **Kronecker.** A = [[2,−1],[−1,2]] (1B Laplacian, 2×2). Kron(A, I_2) ve Kron(I_2, A)'yı (4×4) yaz. Toplamlarının köşegeninin 4 olduğunu (2+2) göster. (Motor tanığı: Kron(A,I_2)+Kron(I_2,A) köşegeni **(4,4,4,4)**; 3×3 ızgara 2B Laplacian 9×9, iç satır 5-nokta {4, −1×4}.)
5. **(Ders 33 habercisi)** CNN'ler evrişimi katman katman uygular; ama bir sinir ağının "öğrendiği fonksiyon" tam olarak ne, hangi fonksiyonları temsil edebilir? Bir tahmin yaz — Ders 33 "sinir ağları ve öğrenme fonksiyonu"nu (Ders 26'nın derinleşmesi) işliyor.

37.13 Sonraki Ders İçin Hazırlık

Ders 33: Sinir Ağları ve Öğrenme Fonksiyonu. Ders 26'nın sinir-ağı yapısını derinleştirir: öğrenme fonksiyonu $F(x, v)$'nin ifade gücü, hangi fonksiyonları öğrenebildiği, ve eğitim (SGD + backprop, Ders 25/27) ile yapının birleşimi. Derin öğrenme bloğunun kapanışına doğru.

Hazırlık

Bu dersin Egzersiz 5'inin sorduğu soruyu zihninde tut: CNN'ler evrişimi katman katman uygularken, bir sinir ağının "öğrendiği fonksiyon" tam olarak ne ve hangi fonksiyonları temsil edebilir? Ders 33,

öğrenme fonksiyonu $F(x, v)$'yi (Ders 26'nın sinir-ağı yapısının derinleşmesi) işler — ifade gücü, temsil edebildiği fonksiyon sınıfı, ve eğitim (SGD + backprop, Ders 25/27) ile birleşimi. Derin öğrenme bloğunun kapanışına yaklaşıyoruz.

37.14 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
ImageNet/AlexNet	CNN %15 hata (2012); 60M param; ağırlık paylaşımı	2m33
Evrişim	$(c * d)_k = \sum c_i\, d_{k-i}$; polinom çarpımı	6m04
Fonksiyon evrişimi	$(f * g)(x) = \int f(t)\, g(x-t)\, dt$	10m02
Evrişim kuralı	$\lambda(CD) = \lambda(C) \odot \lambda(D)$; $F(c \circledast d) = (Fc) \odot (Fd)$	28m52
İki yol	evriş-dönüştür vs ayrı-dönüştür-çarp	30m50
Maliyet	doğrudan $O(n^2)$ vs FFT yolu $O(n \log n)$	35m32
2B evrişim	$\iint f(t,u)\, g(x-t, y-u)\, dt\, du$	36m38
Kronecker	Kron(A,B) $n^2 \times n^2$; Laplacian = Kron(A,I)+Kron(I,A)	38m55

37.15 ML Bağlantıları Özeti

- **CNN = ağırlık paylaşımı:** AlexNet (2012) derin öğrenme devrimi; konvolüsyon tam matrisi küçük filtreye indirir; dropout/max pooling/ReLU yaygınlaştı.
- **Evrişim = polinom çarpımı = kaydır-çarp-topla:** PyTorch conv2d; fast.ai L15 im2col evrişimi matris çarpımına açar (GPU).
- **Konvolüsyon teoremi + FFT:** $F(c \circledast d) = (Fc) \odot (Fd)$; büyük-çekirdek konvolüsyon FFT-conv ile $O(n \log n)$; ses (spektrogram), FNet.
- **2B evrişim:** çift integral; CNN'in conv katmanı; kenar/doku dedektörleri ilk katman filtreleri.
- **Kronecker:** 1B→2B (Kron(F,F) 2B Fourier); ayrılabilir filtreler; 2B Laplacian = Kronecker toplamı (PDE çözücüler, PINN).
- **Geriye köprü:** Ders 30-31 (sirkülant/cyclic convolution/Fourier/FFT), Ders 4 (özdeğer çarpımı). Paralel: fast.ai L15 (conv2d/im2col), NYU H6 (CNN), Karpathy makemore CNN bölümü.

! Kapanış

> *"…the MATLAB command is Kron." — Strang, 38:55*

Evrişim sirkülant/Fourier dünyasından doğar (convolution teoremi + FFT), CNN'ler onu ağırlık paylaşımıyla görüntüye uygular, ve Kronecker çarpımı tüm bunu 2B'ye taşır — sinyal işleme ile derin öğrenmenin buluştuğu nokta.

38 Sinir Ağları ve Öğrenme Fonksiyonu

İki değişken kümesi F(x,v), dört kayıp fonksiyonu ve uzaklıklardan konum kurtarma

> **Bölüm bilgisi**
>
> Bu ders **iki ayrı konuyu** birleştirir: önce Ders 26'nın sinir-ağı yapısını öğrenme fonksiyonu $F(x, v)$ olarak yeniden kurar (kitapta section 7.1), sonra uzaklıklardan konum kurtaran uzaklık matrislerine geçer (section 4.9). Strang'in Ders 33 videosu (≈56 dk) ve OCW Lecture 33 temel alınmıştır. Okuma süresi ≈30 dk. Önkoşul Ders 26 (sinir-ağı yapısı, ilk hâli), Ders 25 (SGD/finite-sum kayıp) ve Ders 5-6 (PSD/SVD).

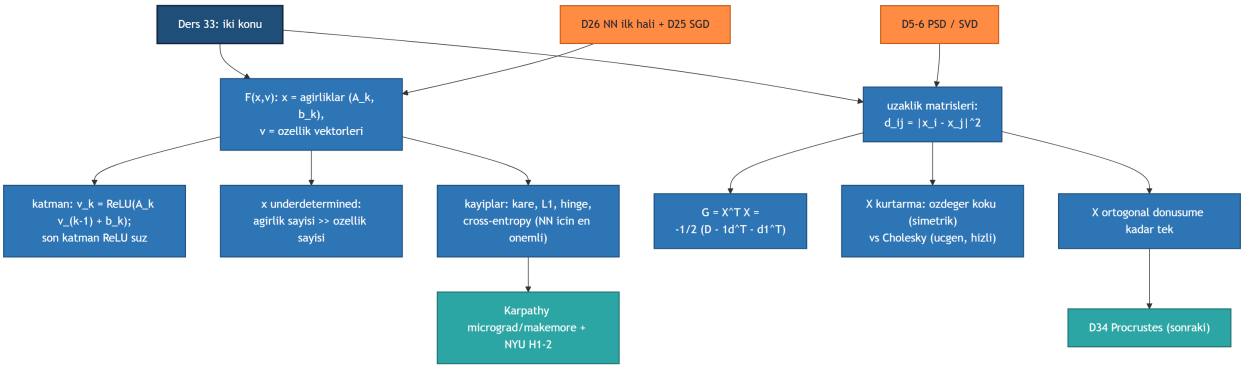
38.1 Bu Derste Ne Var?

Strang bu derste **iki konu** işliyor. İlki Ders 26'nın **revize edilmiş** sinir-ağı yapısı: öğrenme fonksiyonu $F(x, v)$ — iki değişken kümesi. İkincisi **uzaklık matrisleri**: noktalar arası uzaklıklardan konumları geri kurma (Ders 34 Procrustes bloğunun girişi).

Beş sonuç:

1. **Öğrenme fonksiyonu** $F(x, v)$: x = ağırlıklar (A_k, b_k), v = özellik vektörleri (eğitim verisi). SGD x'i optimize eder; v veriden gelir, optimizasyona dahil değil.
2. **Ağ yapısı:** her katman $v_k = \text{ReLU}(A_k v_{k-1} + b_k)$; son katman ReLU'suz, $v_\ell = A_\ell v_{\ell-1} + b_\ell$.
3. x **underdetermined:** ağırlık sayısı çoğu zaman özellik sayısını **çok aşar** — çok sayıda çözüm (eksik-belirlenmiş sistem).
4. **Kayıp fonksiyonu** $L(x)$: tüm örnekler üzerinden toplam; kare (L2), L1 (Lasso), hinge (± 1 sınıflama), **cross-entropy** (sinir ağları için en önemli).
5. **Uzaklık matrisleri:** $d_{ij} = \|x_i - x_j\|^2$; $G = X^T X$'i anahtar özdeşlikten bul; X'i G'den kurtar — özdeğer kökü $Q\Lambda^{1/2}Q^T$ veya Cholesky $D^{1/2}L^T$; X ortogonal dönüşüme kadar tektir.

Şekil 38.1 dersin iki konulu iskeletini gösterir: merkezdeki **Ders 33: iki konu** düğümünden A ve B dalları ayrılır. Dal A öğrenme fonksiyonu $F(x, v)$ — x = ağırlıklar (A_k, b_k) optimize edilir, v = özellik vektörleri veriden akar; katman $v_k = \text{ReLU}(A_k v_{k-1} + b_k)$, son katman ReLU'suz; x underdetermined (ağırlık sayısı $\gg$ özellik sayısı); kayıplar kare/L1/hinge/cross-entropy (NN için en önemli). Dal B uzaklık matrisleri — $d_{ij} = \|x_i - x_j\|^2$, merkezleme özdeşliği $G = X^T X = -\frac{1}{2}(D - \mathbf{1}d^T - d\mathbf{1}^T)$, X kurtarma (özdeğer kökü simetrik vs Cholesky üçgen-hızlı), X ortogonal dönüşüme kadar tek. Köprü düğümleri D26 (NN ilk hâli) + D25 (SGD), D5-6 (PSD/SVD), D34 Procrustes (sonraki) ve Karpathy micrograd/makemore + NYU H1-2 önceki/sonraki bloklara bağlar.



Şekil 38.1: Ders 33 kavram haritasi: iki konu tek derste. Dal A ogrenme fonksiyonu F(x,v) - x = agirliklar (A_k, b_k) optimize edilir, v = ozellik vektorleri veriden akar; katman v_k = ReLU(A_k v_(k-1) + b_k), son katman ReLU suz; x underdetermined (agirlik sayisi » ozellik sayisi, sonsuz sifir-kayip cozum); kayiplar kare/L1/hinge/cross-entropy (NN icin en onemli cross-entropy). Dal B uzaklik matrisleri - d_ij = |x_i - x_j|^2; merkezleme ozdesligi G = X^T X = -1/2 (D - 1d^T - d1^T); X kurtarma ozdeger koku (simetrik) vs Cholesky (ucgen, hizli); X ortogonal donusume kadar tek. Koprular: D26 NN ilk hali + D25 SGD, D5-6 PSD/SVD, D34 Procrustes (sonraki), Karpathy micrograd/makemore + NYU H1-2.

Builder Notu — Ağın İki Yüzü

- $F(x, v)$ **ayrımı = param vs veri:** ağırlıklar x optimize edilir (PyTorch `requires_grad=True`), özellikler v edilmez. Her framework bu ayrımı yapar — bu, Strang'ın sinir ağına en temiz bakışı.
- **Kayıp seçimi göreve bağlı:** regresyon → kare/L1, sınıflama → cross-entropy. Loss, F'in çıktısını gerçek etikete bağlar.
- **Uzaklık matrisi = konum kurtarma:** sensör ağı, NMR molekül şekli, manifold öğrenme — hepsi "uzaklıklardan konum" problemi. Cevap saf lineer cebir (Gram matrisi + PSD kök).
- **Geriye köprü:** Ders 26 (sinir ağı yapısı, ilk hali), Ders 25 (SGD/finite-sum loss), Ders 5-6 (PSD/SVD), Ders 18 (parametre sayımı). Paralel: Karpathy micrograd/makemore F, fast.ai Learner, NYU H1-2.

Tek cümle: Bir sinir ağı, ağırlıklar x ile veri v'nin iki-değişkenli öğrenme fonksiyonu $F(x, v)$'dir (katmanlı ReLU + kayıp); uzaklık matrisi problemi ise tersini yapar — uzaklıklardan Gram matrisi $G = X^T X$ üzerinden konumları (PSD kök ile, ortogonal dönüşüme kadar) geri kurar.

38.2 1. Öğrenme Fonksiyonu F(x,v): İki Değişken Kümesi

Strang, Ders 26'daki sinir-ağı bölümünü (kitapta section 7.1) yeniden yazdı. Anahtar fikir: ağ, **iki ayrı değişken kümesinin** fonksiyonudur.

> *"I now think of this as a function of two sets of variables, x and v." — Strang, 2:05*

$$F(x, v)$$

- x = **ağırlıklar**: çarptığın matrisler A_k ve eklediğin bias vektörleri b_k. Gradyan inişiyle bunları optimize edersin.
- v = **özellik vektörleri**: eğitim verisinden gelen örnek vektörler (v_0 = ham girdi, görüntüyse tüm pikseller). Bunlar **verilidir**, optimizasyonun parçası değildir.

Ders 25'teki gibi: SGD ile mini-batch boyutu 1 ise tek örnek, B ise B örnek, tüm epoch ise hepsi birden işlenir. Optimize edilen hep x (tüm A_k, b_k); v veriden gelir. Bu x/v ayrımı Ders 26'da net değildi — Strang burada düzeltti.

Şekil 38.2 (sonraki bölümde) bu x/v ayrımını ağ şeması üzerinde gösterir: navy kutular optimize edilen ağırlıkları (x), turuncu akış veriden gelen özellikleri (v) işaretler.

> Builder Notu — Parametre ile Verinin Ayrılığı
>
> "Öğrenme fonksiyonu = ağırlıkların ve verinin ortak fonksiyonu" sinir ağının en temiz tanımı. ML köprüsü: PyTorch'ta ağırlıklar `nn.Parameter` (requires_grad=True), veri `Tensor` (grad'sız) — tam bu x/v ayrımı. Karpathy micrograd'da `Value` ağırlıkları ile girdi aynı grafikte ama yalnız ağırlıklar güncellenir; Strang'ın x/v ayrımı bunun matematiksel ifadesidir.

38.3 2. Ağın Yapısı: Katman Katman ReLU

Ağ, girdi v_0'dan başlar ve katman katman ilerler. Her katman **iki adım**: önce lineer (A_k ile çarp, b_k ekle), sonra nonlineer (ReLU her bileşene):

> *"You take the linear step using the first weights… Then you take a nonlinear step, and that gives you v1." — Strang, 4:09*

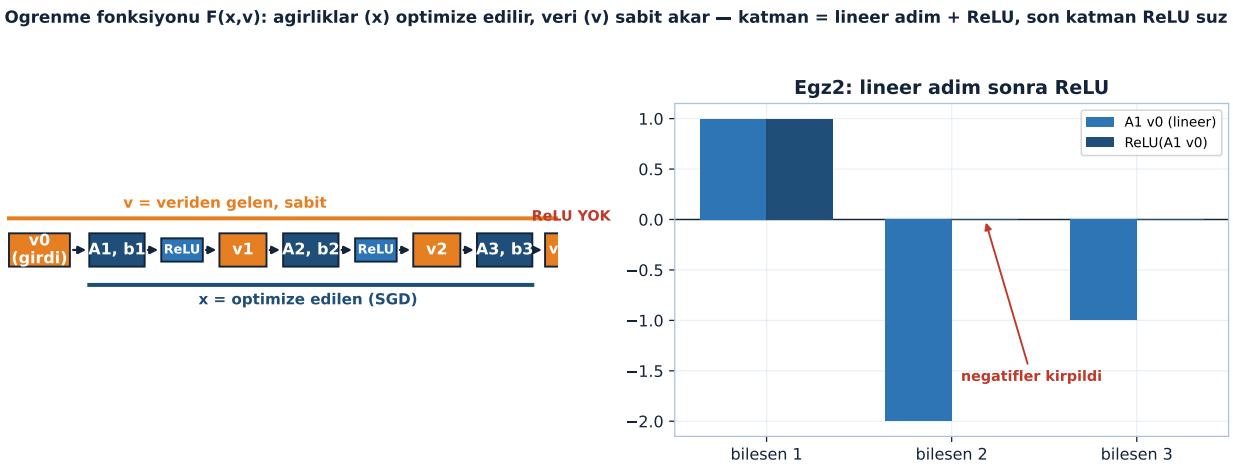
$$v_k = \text{ReLU}(A_k v_{k-1} + b_k), \quad k = 1, \ldots, \ell$$

Son katmanda ReLU **yok** — yalnız lineer adım (ve çoğu zaman bias bile olmadan):

$$v_\ell = A_\ell v_{\ell-1} + b_\ell$$

Burada v_0 girdi (bir eğitim örneği), v_ℓ son katmanın çıktısı. Her gizli katman farklı sayıda nöron (bileşen) içerebilir — A_k matrisinin boyutu bunu belirler. Strang vurguluyor: elle çizilen bir resim, makine çizimine kıyasla zayıf kalır, ama yapı budur — lineer-sonra-ReLU zinciri.

Şekil 38.2 ağ yapısını ve x/v ayrımını birlikte gösterir: solda girdi v_0'dan başlayan lineer-sonra-ReLU zinciri — navy kutular ($A_1, b_1, A_2, b_2, A_3, b_3$) optimize edilen ağırlıkları (x, alttaki navy çizgi "SGD ile optimize edilen"), turuncu kutular ($v_0, v_1, v_2, v_{\text{son}}$) veriden akan özellikleri (v, üstteki turuncu çizgi "veriden gelen, sabit") kapsar; son katmanda kırmızı "ReLU YOK" vurgusu yer alır. Sağda Egz2 somutu: $A_1 v_0 = (1, -2, -1)$ lineer adım (steel barlar) ve ReLU sonrası $(1, 0, 0)$ (navy barlar) yan yana — ikinci ve üçüncü bileşendeki negatifler kırpılır (motor-tanıklı), "negatifler kirpildi" annotation'ı bunu işaretler.



Şekil 38.2: Öğrenme fonksiyonu F(x,v): ağırlıklar (x) SGD ile optimize edilir, veri (v) sabit akar — katman = lineer adım + ReLU, son katman ReLU'suz. Sağda Egz2 somut: $A_1 v_0 = (1, -2, -1)$ lineer adım, ReLU sonrası (1, 0, 0) — negatif bileşenler kırpılır.

> **Builder Notu — Lineer Adım Artı Kırpma**
>
> "Katman = lineer adım + ReLU" derin ağın yapı taşı. ML köprüsü: bu tam bir MLP'nin (çok-katmanlı algılayıcı) ileri geçişi; PyTorch'ta `nn.Linear` (A_k, b_k) + `nn.ReLU()` ardışık. Son katmanda ReLU olmaması kritik: regresyonda ham çıktı, sınıflamada softmax/logit gerekir — ReLU çıktıyı kırparak bilgiyi yok ederdi. Karpathy makemore'daki MLP tam bu zincir.

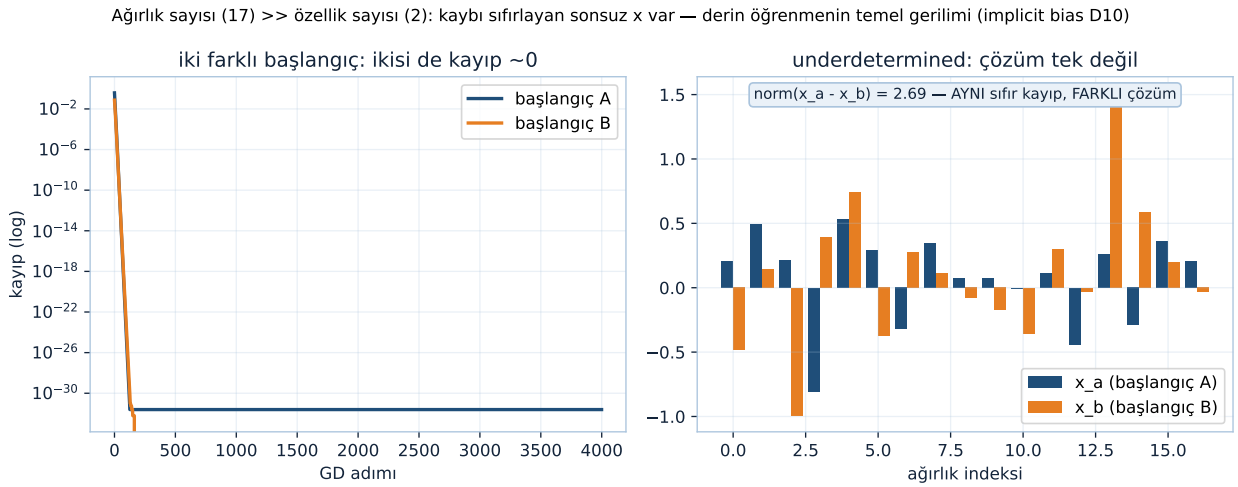
38.4 3. x Underdetermined: Ağırlıklar Özelliklerden Çok

Uygulamada kritik bir gözlem: ağırlık sayısı (A_k, b_k'deki tüm sayılar) çoğu zaman özellik sayısını (v'lerin bileşenleri) **çok aşar**.

> *"…the number of x's exceeds, and often far exceeds, the number of v's." — Strang, 8:12*

Sonuç: x **eksik-belirlenmiş** (underdetermined) — kaybı sıfırlayan tek bir x yoktur, sonsuz çözüm vardır. Strang bunu "interesting and unexpected" diye niteliyor. Ama bu, derin öğrenmenin kalbinde: aşırı-parametreli ağlar eğitim verisini tam ezberleyebilecek kadar serbestliğe sahiptir, yine de iyi genelleyebilir.

Şekil 38.3 eksik-belirlenmişliğin canlı sayısal kanıtıdır: aynı tek-örnek ağ (17 ağırlık) iki farklı seed ile eğitilir. Solda iki kayıp eğrisi (semilogy) — her iki başlangıç da kaybı ~ 0'a indirir (motorda **2.5e-32** ve **0.0**). Sağda 17 ağırlık bileşeninin yan yana barı: çözüm vektörleri belirgin biçimde **farklıdır** — $\|x_a - x_b\| = 2.69$ (tam değer **2.686**). Aynı sıfır kayıp, farklı çözüm; kaybı sıfırlayan sonsuz x'in canlı örneği. Bu, ağırlık sayısının (17) özellik sayısını (2) çok aşmasının doğrudan sonucu.



Şekil 38.3: Ağırlık sayısı (17) » özellik sayısı (2): kaybı sıfırlayan sonsuz x var — derin öğrenmenin temel gerilimi (implicit bias D10).

Builder Notu — Çözüm Bolluğu Sorun Değil

"Ağırlık sayısı örnek sayısı = eksik-belirlenmiş" modern derin öğrenmenin temel gerilimi. ML köprüsü: bugünün ağları milyonlarca-milyarlarca parametre, eğitim örneğinden kat kat fazla. Çok çözüm varsa SGD hangisini seçer? Genelde küçük-normlu / düz minimum (implicit regularization) — Ders 10 (implicit bias) + Ders 22-23 (gradyan inişi yolu) bunu hazırladı. Klasik istatistik "çok parametre = aşırı-öğrenme" derdi; derin öğrenme bu sezgiyi kırdı.

38.5 4. Kayıp Fonksiyonu L(x): Kare, L1, Hinge, Cross-Entropy

x'i, kaybı $L(x)$ minimize ederek seçersin. Kayıp, tüm örnekler üzerinden bir toplamdır (Ders 25 finite-sum yapısı):

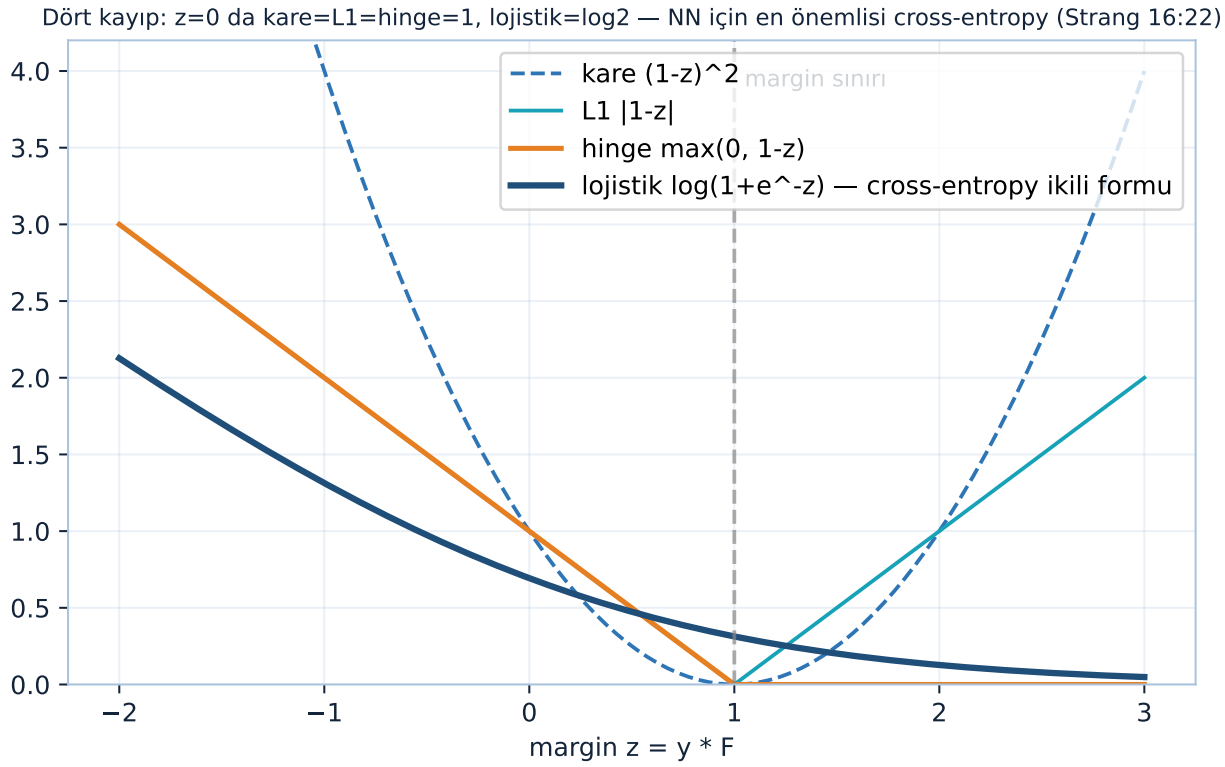
> *"…L is often a finite sum over all of F." — Strang, 12:18*

$$L(x) = \sum_{i=1}^{N} \ell(F(x, v_i),\, y_i)$$

Tam-ölçek gradyan inişi tüm i için hesaplar (pahalı); SGD bunlardan 1 ya da b tanesini (örn. 32, 128) seçer. Strang dört kayıp fonksiyonu sıralıyor:

- **Kare kayıp (L2):** $\sum_i \|F(x, v_i) - y_i\|^2$ — en bilineni; regresyon için.
- **L1 kayıp:** hata mutlak değerlerinin toplamı; Lasso gibi problemlerde geçer, "ama derin öğrenmede değil" (Strang).
- **Hinge kayıp:** ± 1 sınıflama problemleri için (SVM'in kaybı).
- **Cross-entropy (çapraz-entropi):** sinir ağları için **en çok kullanılan** kayıp.

> *"…the most important for neural nets, is cross-entropy loss." — Strang, 16:22*



Şekil 38.4: Dört kayıp eğrisi (margin z = y·F): z=0'da kare = L1 = hinge = 1, lojistik = log2 ≈ 0.693. Hinge z≥1'de tam sıfır (ceza yok), lojistik hep azalan ama asla tam sıfır olmaz — NN için en önemlisi cross-entropy'nin ikili formu lojistik kayıp (Strang 16:22).

Şekil 38.4 dört kaybı tek panelde margin $z = y \cdot F$ ekseninde karşılaştırır: $z = 0$'da kare = L1 = hinge = 1, lojistik = $\log 2 \approx 0.693$ (motor-dürüst). Hinge ($\max(0, 1 - z)$, turuncu) $z \geq 1$'de tam sıfırdır — margin aşıldıktan sonra ceza yok; lojistik ($\log(1 + e^{-z})$, navy — cross-entropy'nin ikili formu) hep azalır ama **asla tam sıfır olmaz**, doğru sınıflanmış örneklere bile küçük gradyan verir. Gri kesik çizgi $z = 1$ margin sınırını işaretler. Ayrı bir tanık olarak motorun çok-sınıf cross-entropy'si: logits $(2.0, 1.0, 0.1)$, target=0 → CE = **0.4170** ($= -\log \text{softmax}[0]$); yanlış sınıfa CE daha büyük çıkar.

> Builder Notu — Görev Kaybını Seçer
>
> "Kayıp seçimi göreve göre" pratik bir karar. ML köprüsü: regresyon → kare kayıp (MSE); sınıflama → cross-entropy (PyTorch `nn.CrossEntropyLoss`; Karpathy makemore'un negatif log-olabilirlik kaybı tam budur). Hinge → SVM. L1 → seyreklik/özellik seçimi (Lasso). Strang'ın "L1 derin öğrenmede değil" notu yerinde — DL'de cross-entropy hâkimdir çünkü olasılık dağılımlarını doğal biçimde karşılaştırır.

38.6 5. Uzaklık Matrisleri: Uzaklıklardan Konumlar

Dersin ikinci yarısı tamamen yeni bir problem (kitapta section 4.9):

> *"We have a bunch of points in space… we know the distances between the points… what's their position?" — Strang, 18:25*

Uzaklık-kare matrisi D verili: d_{ij} = noktalar i ve j arası uzaklığın karesi. Soru: noktaların konumları x_i ne? Uygulamalar: sensör ağları (radar/seyahat süresiyle uzaklık ölç), NMR ile molekül şekli, ve makine öğrenmesinde manifold öğrenme (yüksek-boyutlu noktalar düşük-boyutlu eğri bir yüzeyde toplanır).

$$d_{ij} = \|x_i - x_j\|^2$$

Konumlar **tek değildir**: tüm noktaları öteleyebilir (translation) ya da katı döndürebilirsin (rigid rotation) — ikili uzaklıklar değişmez. Ağırlık merkezini (centroid) orijine koyarak en azından ötelemeyi sabitlersin.

> *"…positions are not unique, but I can come closer by saying, put the centroid at the origin." — Strang, 30:49*

Şekil 38.5 (sonraki bölümde) bu problemi 3-4-5 üçgeni üzerinde somutlaştırır: üç noktanın uzaklık-kare matrisi D'den iç-çarpım matrisi G kurtarılır.

> Builder Notu — Uzaklıktan Geometriye
>
> "Uzaklıklardan konum = ters problem" geometrik bir kurtarma işi. ML köprüsü: bu, **çok-boyutlu ölçekleme (MDS)** ve manifold öğrenmenin (Isomap, t-SNE'nin atası) çekirdeğidir — yüksek-boyutlu veri düşük-boyutlu bir manifolda oturuyorsa, yalnız ikili uzaklıklardan o düşük-boyutlu gömmeyi kurtarırsın. Strang'ın deyişiyle bir geometricinin "manifold" diyeceği eğri yüzey; onu lineerleştirip boyutu düşürürsün (Ders 7 PCA ile akraba).

38.7 6. Anahtar Özdeşlik: Gram Matrisi G = X^TX

Uzaklığın karesini aç (Strang 34:57):

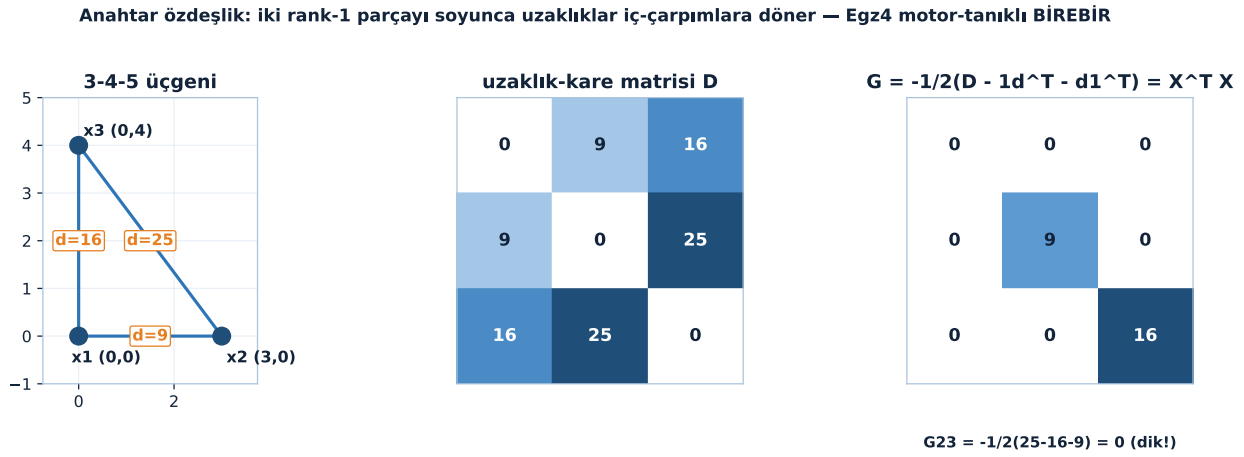
$$d_{ij} = \|x_i - x_j\|^2 = x_i \cdot x_i - 2\, x_i \cdot x_j + x_j \cdot x_j$$

Sağdaki üç terim: $x_i \cdot x_i$ (yalnız satıra bağlı → rank-1 parça), $x_j \cdot x_j$ (yalnız sütuna bağlı → rank-1 parça) ve $-2\, x_i \cdot x_j$ (çapraz çarpım = aradığımız iç-çarpımlar). İç-çarpım matrisini **Gram matrisi** $G = X^T X$ olarak tanımla ($G_{ij} = x_i \cdot x_j$). Köşegen değerlerini $d = (x_i \cdot x_i)$ vektörüyle gösterip özdeşliği matris dilinde yaz:

> *"…minus 1/2 of the D matrix plus 1/2 of the 1's times the d… plus 1/2 of the d times the 1's." — Strang, 39:10*

$$G = X^T X = -\tfrac{1}{2}(D - \mathbf{1}\, d^T - d\, \mathbf{1}^T)$$

Burada **1** = hepsi-bir vektörü, d = köşegen $(x_i \cdot x_i)$ vektörü. İki rank-1 parça ($\mathbf{1}d^T$ ve $d\mathbf{1}^T$) köşegen katkılarını temizler, geriye saf iç-çarpımlar (G) kalır. D verili olduğundan G'yi doğrudan hesaplarsın.



Şekil 38.5: Anahtar özdeşlik: iki rank-1 parçayı soyunca uzaklıklar iç-çarpımlara döner — Egz4 motor-tanıklı BİREBİR. Sol panel 3-4-5 üçgeni, orta panel uzaklık-kare matrisi D, sağ panel Gram matrisi G = X^TX.

Şekil 38.5 özdeşliği 3-4-5 üçgeni üzerinde motor-tanıklı doğrular: solda üç nokta $x_1 = (0,0)$, $x_2 = (3,0)$, $x_3 = (0,4)$ düzlemde, kenar uzaklık-kareleri $d = 9$, $d = 16$, $d = 25$ (kareler!) ile etiketli. Ortada uzaklık-kare matrisi $D = [[0,9,16],[9,0,25],[16,25,0]]$. Sağda $G = -\frac{1}{2}(D - \mathbf{1}d^T - d\mathbf{1}^T) = \mathrm{diag}(0,9,16)$ — köşegen $(0,9,16)$ tam olarak $x_i \cdot x_i$ gerçek iç-çarpımlarıdır, yani G birebir X^TX. Özellikle $G_{23} = -\frac{1}{2}(25-16-9) = 0$: x_2 ve x_3 dik kenarlar olduğundan iç-çarpımları sıfır (dik!). Özdeşlik, $d = (0,9,16)$ köşegeniyle uzaklıkları iç-çarpımlara çevirir.

> **Builder Notu — İki Rank-Bir Parçayı Soyunca**
>
> "Uzaklık → Gram matrisi tek özdeşlikle" problemin can alıcı adımı. ML köprüsü: bu, çift-merkezleme (double centering) adımıdır — uzaklık-kare matrisinden iç-çarpım (kernel/Gram) matrisine geçiş; kernel

yöntemleriyle akraba çünkü G bir Gram/kernel matrisidir. [tahmin] Klasik MDS literatürü bu dönüşüme "Gower / çift-merkezleme" der, ama Strang bu adı kullanmadı — özdeşliği terim terim elle türetti.

38.8 7. X'i Kurtarmak: Özdeğer Kökü vs Cholesky

Artık $G = X^T X$ biliniyor; X'i bul. G simetrik ve **pozitif yarı-tanımlı** (PSD). X'i bulmak = G'nin bir "karekökünü" bulmaktır. X tek değildir: herhangi ortogonal Q için QX de işe yarar (çünkü $(QX)^T(QX) = X^T Q^T Q X = X^T X$). İki yol var:

> *"…one way is to use eigenvalues… the other way would be to use elimination on X transpose X."*
> *— Strang, 49:24*

Yol 1 — Özdeğer kökü. Spektral teorem: $G = Q\Lambda Q^T$ (PSD olduğundan tüm $\lambda \geq 0$). Aynı özvektörleri tut, özdeğerlerin karekökünü al:

$$G = Q\Lambda Q^T \;\Rightarrow\; X = Q\,\Lambda^{1/2}\,Q^T$$

Bu X simetrik PSD'dir; $X^T X = Q\Lambda^{1/2}Q^T Q\Lambda^{1/2}Q^T = Q\Lambda Q^T = G$.

Yol 2 — Cholesky (eleme). Simetrik PSD matriste eleme, pozitif pivotlarla LDL^T verir (alt-üçgen × köşegen × üst-üçgen, simetride $U = L^T$):

$$G = LDL^T \;\Rightarrow\; X = D^{1/2}L^T$$

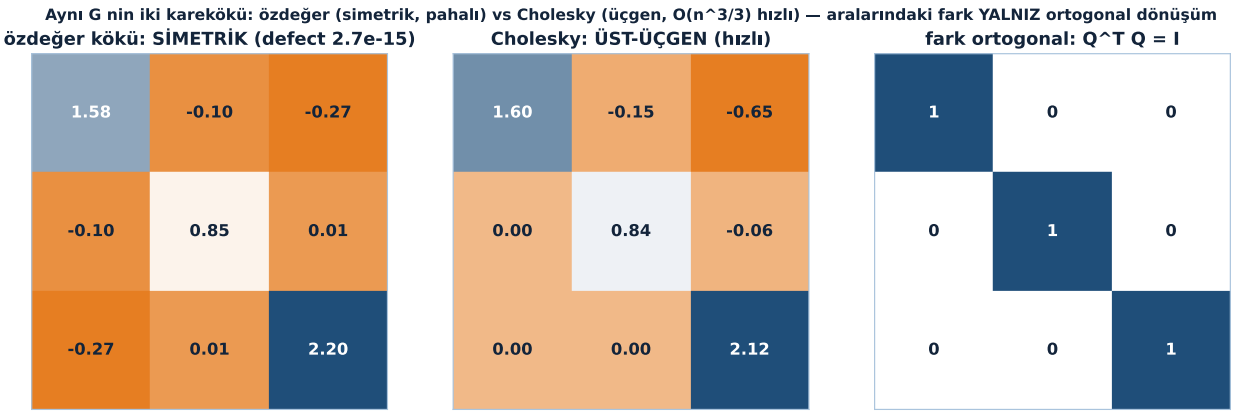
Bu X üçgenseldir; $X^T X = LD^{1/2}D^{1/2}L^T = LDL^T = G$.

> *"…this is the Cholesky Factorization… much faster to compute than the eigenvalue square root."*
> *— Strang, 53:31*

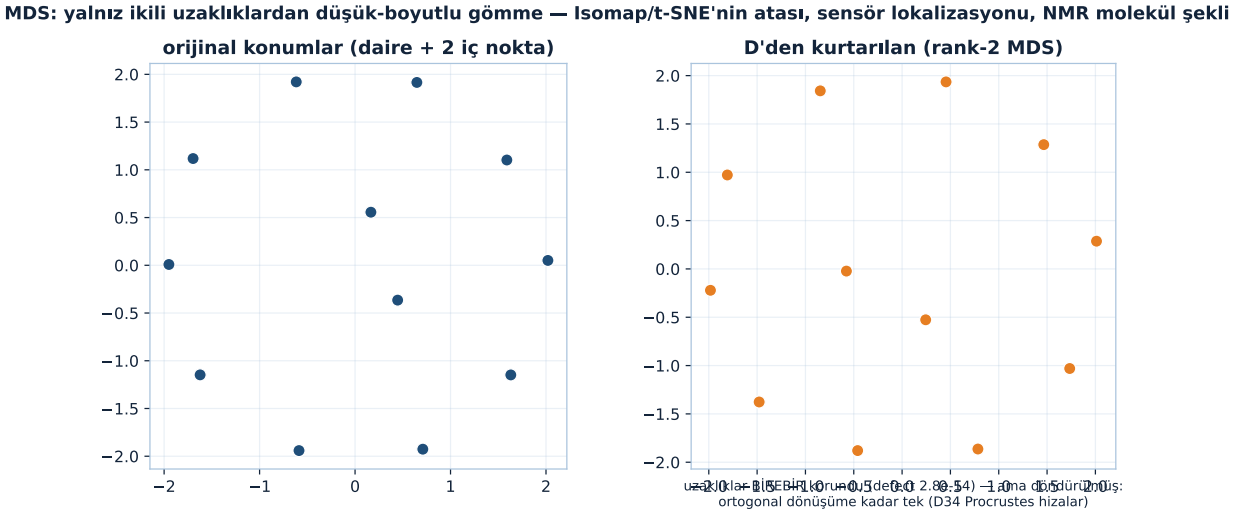
Özdeğer kökü simetrik bir X verir (daha pahalı, özçözüm gerekir); Cholesky üçgen bir X verir (çok daha hızlı). İkisi de geçerli — aralarındaki fark yalnız bir ortogonal dönüşümdür.

Şekil 38.6 aynı G'nin iki karekökünü yan yana koyar: solda özdeğer kökü $X_{\text{eig}} = Q\Lambda^{1/2}Q^T$ **simetriktir** ($X^T X - G$ defect **2.7e-15**, özçözüm gerekir, pahalı), ortada Cholesky kökü X_{chol} **üst-üçgendir** ($O(n^3/3)$, çok daha hızlı). Sağda ikisini bağlayan $Q = X_{\text{eig}} \cdot X_{\text{chol}}^{-1}$ için $|Q^T Q| = I$ — birim matris (motor-tanıklı): iki kök arasındaki fark **yalnızca bir ortogonal dönüşümdür**. "X ortogonal dönüşüme kadar tek" cümlesinin sayısal kanıtı budur.

> Builder Notu — Bir Matrisin İki Karekökü
>
> "PSD matrisin iki karekökü: simetrik (özdeğer) vs üçgen (Cholesky)" temel bir hesaplama seçimi. ML köprüsü: Cholesky her yerde — Gauss süreçleri (GP), Kalman filtresi, ikinci-derece optimizasyon ve çok-değişkenli normalden örnek üretme hep PSD-kök ister; Cholesky $O(n^3/3)$ ile en ucuzudur. Özdeğer kökü ise PCA/whitening'de tercih edilir. Ders 5-6 (PSD, SVD) bu kökleri zaten kurmuştu; burada uzaklık probleminin son adımı oldu.



Şekil 38.6: Aynı G nin iki karekökü: özdeğer (simetrik, pahalı) vs Cholesky (üçgen, O(n^3/3) hızlı) — aralarındaki fark YALNIZ ortogonal dönüşüm. Özdeğer kökü X_eig simetrik (X^TX-G defect 2.7e-15), Cholesky kökü X_chol üst-üçgen ve çok daha hızlı; ikisini bağlayan $Q = X_{eig} \cdot X_{chol}^{-1}$ ortogonaldir ($Q^TQ = I$).



Şekil 38.7: MDS: yalnızca ikili uzaklıklardan düşük-boyutlu gömme — Isomap/t-SNE'nin atası, sensör lokalizasyonu, NMR molekül şekli. 12 noktalı 2B bulut → D → G → rank-2 özdeğer gömmesi: uzaklıklar BİREBİR korunur (defect 2.8e-14), ama kurtarılan konum döndürülmüş görünür (ortogonal dönüşüme kadar tek — D34 Procrustes hizalar).

Şekil 38.7 özdeğer kökünü gerçek bir kurtarma probleminde çalıştırır ve D34'e köprü kurar: 12 noktalı 2B bulut → uzaklık-kare matrisi D → Gram G → rank-2 özdeğer gömmesi. Solda orijinal konumlar (daire + 2 iç nokta), sağda yalnız D'den kurtarılan konumlar. Uzaklıklar **birebir korunur** (D-koruma defekti **2.8e-14**), ama kurtarılan bulut **döndürülmüş** görünür: ham kurtarma orijinalle çakışmaz, çünkü X ancak ortogonal dönüşüme kadar tektir. İşte sonraki dersin (D34 Procrustes) sorusu — döndürülmüş gömmeyi orijinale en iyi ortogonal Q ile nasıl hizalarsın.

38.9 Bu Dersin Özeti

- **Öğrenme fonksiyonu F(x,v):** x = ağırlıklar (A_k, b_k), v = özellik vektörleri; SGD x'i optimize eder, v veriden gelir (Ders 26'nın revize hali).
- **Ağ yapısı:** $v_k = \text{ReLU}(A_k v_{k-1} + b_k)$; son katman ReLU'suz ($v_\ell = A_\ell v_{\ell-1} + b_\ell$).
- **x underdetermined:** #ağırlık #özellik → sonsuz çözüm (aşırı-parametreli ağların temeli, implicit bias Ders 10).
- **Kayıp L(x):** finite-sum; kare (L2), L1 (Lasso), hinge (± 1 sınıflama), cross-entropy (NN için en önemli).
- **Uzaklık matrisleri:** $d_{ij} = \|x_i - x_j\|^2$; $G = X^T X = -\frac{1}{2}(D - \mathbf{1}d^T - d\mathbf{1}^T)$; X'i G'den özdeğer kökü ($Q\Lambda^{1/2}Q^T$) veya Cholesky ($D^{1/2}L^T$) ile bul; X ortogonal dönüşüme kadar tektir.

! Tek Bir Cümle

Bir sinir ağı, ağırlıklar x ve veri v'nin iki-değişkenli öğrenme fonksiyonu $F(x, v)$'dir (katmanlı ReLU + cross-entropy gibi bir kayıp); uzaklık matrisi problemi ise tersini yapar — uzaklık-kare matrisi D'den Gram matrisi $G = X^T X = -\frac{1}{2}(D - \mathbf{1}d^T - d\mathbf{1}^T)$'yi çıkarıp X'i (PSD kök ile, ortogonal dönüşüme kadar) geri kurar.

38.10 Kontrol Soruları

i Soru 1 — Öğrenme fonksiyonu F(x,v)'de x ve v nedir; neden ayrı tutulur

x = ağırlıklar (tüm A_k matrisleri ve b_k bias vektörleri) — gradyan inişiyle **optimize edilen** değişkenler. v = özellik vektörleri (eğitim örnekleri, v_0 = ham girdi) — **veriden gelen, sabit** değişkenler, optimize edilmez. Ayrım kritik: SGD yalnız x'i seçer; v eğitim setinden örneklenir. Bu, "ağırlık güncelle, veriyi güncelleme" ayrımının matematiksel ifadesidir.

i Soru 2 — Neden son katmanda ReLU uygulanmaz

ReLU çıktıyı $[0, \infty)$ aralığına kırpar (negatifleri sıfırlar). Son katman çıktısı sonucu doğrudan verir: regresyonda negatif değerler gerekebilir, sınıflamada softmax/logit ham (kırpılmamış) skorlar ister. Son katmanda ReLU olsaydı bu bilgiyi yok ederdi. Bu yüzden son katman yalnız lineer: $v_\ell = A_\ell v_{\ell-1} + b_\ell$.

i Soru 3 — Uzaklık matrisi D'den Gram matrisi G=X^{T}X'e nasıl ve neden geçilir

Uzaklığın karesini aç: $d_{ij} = x_i \cdot x_i - 2\, x_i \cdot x_j + x_j \cdot x_j$. İlk ve son terim yalnız satıra/sütuna bağlı (iki rank-1 parça); orta terim $-2\, x_i \cdot x_j$ aradığımız iç-çarpımları taşır. Köşegeni $d = (x_i \cdot x_i)$ ile gösterip iki rank-1 parçayı çıkarınca özdeşlik: $G = -\frac{1}{2}(D - \mathbf{1}d^T - d\mathbf{1}^T)$. Böylece verili D'den iç-çarpım (Gram) matrisini doğrudan elde edersin.

i Soru 4 — G=X^{T}X'ten X'i bulmanın iki yolu nedir; farkları

(1) **Özdeğer kökü:** $G = Q\Lambda Q^T$, sonra $X = Q\Lambda^{1/2}Q^T$ — simetrik bir X, ama özçözüm gerektirir (pahalı). (2) **Cholesky:** $G = LDL^T$, sonra $X = D^{1/2}L^T$ — üçgen bir X, çok daha hızlı. İkisi de $X^TX = G$ sağlar. X tek değildir: herhangi ortogonal Q için QX de çözümdür (uzaklıklar rotasyonda değişmez), yani X ancak ortogonal dönüşüme kadar belirlidir.

38.11 Egzersizler

1. **Parametre sayımı.** İki katmanlı bir ağ: $v_0 \in \mathbb{R}^2$, A_1 boyutu 3×2 + $b_1 \in \mathbb{R}^3$, sonra A_2 boyutu 1×3 + $b_2 \in \mathbb{R}^1$. Ağırlık (x) toplam sayısını ve girdi özelliği (v_0) bileşen sayısını say. Hangisi büyük — sistem underdetermined mı? (Motor tanığı: A_1 3×2 + b_1 3 + A_2 1×3 + b_2 1 = **13 ağırlık** vs v_0 = **2 özellik** → 13 2, underdetermined.)
2. **ReLU ileri geçiş.** $A_1 = [[1,0],[0,1],[1,1]]$, $b_1 = 0$, $v_0 = (1,-2)$. Lineer adım A_1v_0'ı hesapla, sonra $v_1 = \text{ReLU}(A_1v_0)$'ı bul (her bileşene $\max(0,\cdot)$). (Motor tanığı: lineer adım $A_1v_0 = (1,-2,-1)$; ReLU sonrası $v_1 = (1,0,0)$ — negatifler kırpılır.)
3. **Kayıp eşleştirme.** Üç görev için uygun kaybı seç: (a) ev fiyatı tahmini, (b) kedi/köpek ikili sınıflama, (c) 10-sınıf rakam tanıma. Seçenekler: kare kayıp (L2), cross-entropy. Her eşleştirmeyi bir cümleyle gerekçelendir.
4. **Uzaklık → Gram.** Düzlemde 3 nokta: $x_1 = (0,0)$, $x_2 = (3,0)$, $x_3 = (0,4)$. Uzaklık-kare matrisi D'yi ($d_{ij} = \|x_i - x_j\|^2$) yaz. Sonra köşegen $d = (0,9,16)$ ile $G = -\frac{1}{2}(D - \mathbf{1}d^T - d\mathbf{1}^T)$'yi hesapla; G'nin gerçek iç-çarpımlarla ($x_i \cdot x_j$) eşleştiğini doğrula. (Motor tanığı: $D = [[0,9,16],[9,0,25],[16,25,0]]$; $G = \text{diag}(0,9,16) = X^TX$; $G_{23} = -\frac{1}{2}(25 - 16 - 9) = 0$, dik kenarlar.)
5. **(Ders 34 habercisi)** İki nokta kümen var; biri diğerinin döndürülüp ötelenmiş (ve hafif gürültülü) hali. En iyi ortogonal hizalamayı (rotasyon Q) nasıl bulursun? Bir tahmin yaz — Ders 34 "Procrustes problemi"ni (SVD ile en iyi Q) işliyor.

38.12 Sonraki Ders İçin Hazırlık

Ders 34: Uzaklık Matrisleri, Procrustes Problemi. Bu dersin uzaklık-matrisi çözümünü tamamlar ve **Procrustes problemine** geçer: iki nokta kümesini en iyi ortogonal dönüşümle (rotasyon) hizalama. Çözüm SVD'den gelir (en iyi $Q = UV^T$). Geometrik kurtarma + hizalama bloğunun devamı — sinyal/CNN/graf bloğunun ortası.

Hazırlık

Bu dersin Egzersiz 5'inin sorduğu soruyu zihninde tut: iki nokta kümesi birbirinin döndürülüp ötelenmiş hâliyse, en iyi ortogonal hizalamayı (rotasyon Q) nasıl bulursun? Şekil 38.7'in gösterdiği gibi — uzaklıklardan kurtarılan konum döndürülmüş görünür; Ders 34 **Procrustes problemini** işler: SVD ile en iyi ortogonal $Q = UV^T$. Bu derste açtığımız "uzaklıklardan konum" çözümünün hizalama ile tamamlanmasıdır.

38.13 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Öğrenme fonksiyonu	$F(x, v)$; x = ağırlıklar, v = özellik vektörleri	2m05
Ağ katmanı	$v_k = \text{ReLU}(A_k v_{k-1} + b_k)$; son katman ReLU'suz	4m09
x underdetermined	#ağırlık #özellik → sonsuz çözüm	8m12
Kayıp fonksiyonları	kare / L1 / hinge / cross-entropy (NN: cross-entropy)	16m22
Uzaklık problemi	$d_{ij} = \|x_i - x_j\|^2$; uzaklıktan konum	18m25
Anahtar özdeşlik	$G = X^T X = -\frac{1}{2}(D - \mathbf{1}d^T - d\mathbf{1}^T)$	39m10
Özdeğer kökü	$X = Q\Lambda^{1/2}Q^T$ (simetrik, pahalı)	49m24
Cholesky	$X = D^{1/2}L^T$ (üçgen, hızlı)	53m31

38.14 ML Bağlantıları Özeti

- **F(x,v) = param vs veri:** ağırlıklar optimize edilir, veri sabit; PyTorch `nn.Parameter` vs `Tensor`; Karpathy micrograd/makemore'un öğrenme fonksiyonu.
- **Katmanlı ReLU:** MLP ileri geçişi (`nn.Linear` + `nn.ReLU`); son katman ham çıktı (regresyon) ya da softmax öncesi logit (sınıflama).
- **Underdetermined ağlar:** aşırı-parametreli derin öğrenme; implicit bias SGD'yi düz/küçük-normlu minimuma çeker (Ders 10).
- **Kayıplar:** cross-entropy (sınıflama, makemore NLL), kare (regresyon/MSE), hinge (SVM), L1 (Lasso/seyreklik).
- **Uzaklık matrisi → MDS/manifold:** Isomap, t-SNE'nin atası; sensör lokalizasyonu, NMR molekül şekli; Gram matrisi = kernel matrisi.
- **PSD kök (Cholesky / özdeğer):** Gauss süreçleri, Kalman, çok-değişkenli normalden örnekleme; Cholesky $O(n^3/3)$ en ucuz kök. Ders 5-6 (PSD/SVD) temeli.
- **Geriye köprü:** Ders 26 (NN ilk hali), Ders 25 (SGD/finite-sum kayıp), Ders 5-6 (PSD/SVD), Ders 18 (parametre sayımı), Ders 10 (implicit bias). Paralel: Karpathy micrograd/makemore F, fast.ai Learner, NYU H1-2.

! Kapanış

> *"…most important was to get the structure of a neural net straight, separating the v's, the sample vectors, from the x's, the weights." — Strang, 55:35*

Bu ders iki yüzü gösterir: ileri yönde ağ, ağırlık (x) ve veriyi (v) $F(x, v)$'de birleştirir (öğrenme); ters yönde uzaklık matrisi, ölçülen uzaklıklardan geometriyi (konumları) PSD kökle geri kurar — derin öğrenme ile klasik geometrinin aynı lineer cebir diline yaslandığı yer.

39 Uzaklık Matrisleri, Procrustes Problemi

Üçgen eşitsizliği testi, Frobenius normu ve tek SVD ile en iyi ortogonal hizalama

> **Bölüm bilgisi**
>
> Bu ders kısa bir **coda**: Ders 33'ün uzaklık-matrisi problemini geçerlilik testiyle kapatır, sonra Procrustes problemini (en iyi ortogonal hizalama) tek bir SVD ile çözer. Strang'in Ders 34 videosu (≈29 dk — bu blokun kısa dersi) ve OCW Lecture 34 temel alınmıştır. Okuma süresi ≈22 dk. Önkoşul Ders 33 (uzaklık/Gram matrisleri) ve Ders 6-8 (SVD/Eckart-Young/Frobenius normu).

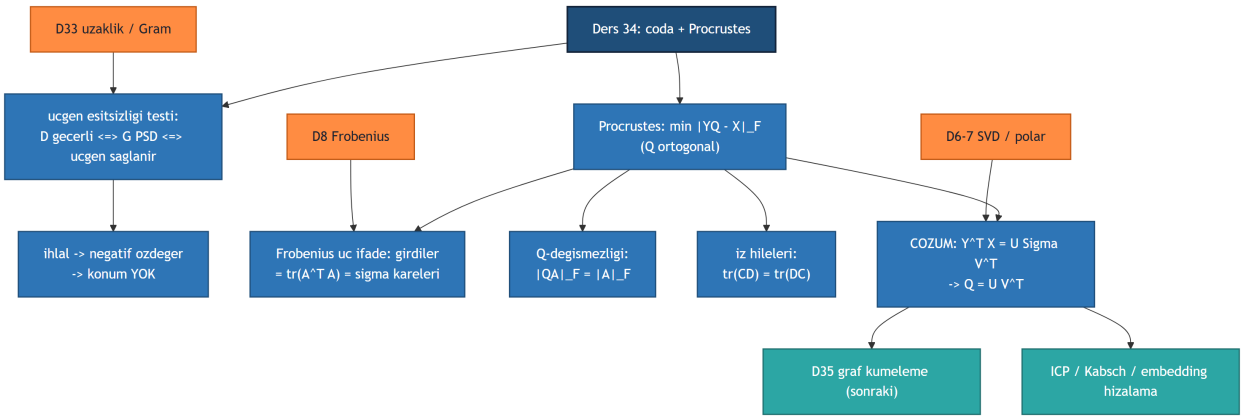
39.1 Bu Derste Ne Var?

Ders **iki parça**. İlki Ders 33'ün uzaklık-matrisi probleminin **codası**: her uzaklık matrisi geçerli mi? İkincisi **Procrustes problemi**: iki vektör kümesini en iyi ortogonal dönüşümle (rotasyon) hizalama — çözümü şaşırtıcı sadelikte: $Q = UV^T$.

Beş sonuç:

1. **Üçgen eşitsizliği testi:** Bir uzaklık matrisi D ancak üçgen eşitsizliğini sağlıyorsa geçerli konum verir. İhlal edilirse Gram matrisi G **PSD çıkmaz** (negatif özdeğer) — konum bulunamaz ($\Longleftrightarrow$ koşulu).
2. **Procrustes problemi:** İki vektör kümesi X ve Y; ortogonal Q ile YQ'yu X'e en yakın yap: $\min_{Q^TQ=I} \|YQ - X\|_F^2$.
3. **Frobenius normu, üç ifade:** $\|A\|_F^2 = \sum_{i,j} a_{ij}^2 = \text{tr}(A^TA) = \sum_i \sigma_i^2$.
4. Q **değişmezliği:** ortogonal Q ile çarpmak Frobenius normunu (ve tekil değerleri) **değiştirmez**: $\|QA\|_F = \|A\|_F$.
5. **Çözüm:** Y^TX'in SVD'sini al, $Y^TX = U\Sigma V^T$; en iyi $Q = UV^T$. İspat iz (trace) özellikleriyle gelir: $\text{tr}(CD) = \text{tr}(DC)$.

Şekil 39.1 dersin codalı iskeletini gösterir: merkezdeki **Ders 34: coda + Procrustes** düğümünden iki dal ayrılır. Dal A üçgen eşitsizliği testi — D geçerli $\Longleftrightarrow$ G PSD $\Longleftrightarrow$ üçgen sağlanır; ihlal → negatif özdeğer → konum YOK (10 boyut da kurtarmaz). Dal B Procrustes problemi — min $\|YQ - X\|_F$, Q ortogonal (uydu/şekil hizalama). Alt-dallar üç araç verir: Frobenius üç ifade (girdiler = $\text{tr}(A^TA)$ = σ kareleri); Q-değişmezliği $\|QA\|_F = \|A\|_F$; iz hileleri $\text{tr}(CD) = \text{tr}(DC)$. Hepsi tek bir çözümde toplanır: $Y^TX = U\Sigma V^T \Rightarrow Q = UV^T$ (tek SVD, iterasyon yok). Köprü düğümleri D33 (uzaklık/Gram), D6-7 (SVD/polar), D8 (Frobenius), D35 graf kümeleme (sonraki) ve ICP/Kabsch/embedding hizalama önceki/sonraki bloklara bağlar.



Şekil 39.1: Ders 34 kavram haritasi: D33 codasi + Procrustes. Dal A ucgen esitsizligi testi - D gecerli <=> G PSD <=> ucgen saglanir; ihlal -> negatif ozdeger -> konum YOK (10 boyut da kurtarmaz). Dal B Procrustes problemi - min |YQ - X|_F, Q ortogonal (uydu/sekil hizalama). Alt-dallar: Frobenius uc ifade (girdiler = tr(A^T A) = sigma kareleri); Q-degismezligi |QA|_F = |A|_F (sigma lar korunur); iz hileleri tr(CD) = tr(DC) (dikdortgende bile, sifir-disi ozdegerler ortak); COZUM Y^T X = U Sigma V^T -> Q = U V^T (tek SVD, iterasyon yok, kapali-form = global optimum). Koprular: D33 uzaklik/Gram, D6-7 SVD/polar, D8 Frobenius, D35 graf kumeleme (sonraki); ICP/Kabsch/embedding hizalama.

Builder Notu — Geometri Sayılardan Çıkar

- **Procrustes = en iyi rotasyon, kapalı-form:** iki nokta kümesini hizalamanın çözümü tek SVD. Optimizasyon gerekmez — lineer cebir doğrudan verir.
- **ML köprüsü:** nokta-bulutu hizalama (ICP/Kabsch, moleküler RMSD), şekil analizi ve diller-arası word embedding hizalama hep ortogonal Procrustes çözer.
- **Frobenius + iz:** veri biliminde matris "büyüklüğü" ölçüsü; iz hileleri ($\text{tr}(CD) = \text{tr}(DC)$) birçok türev/optimizasyon ispatını taşır.
- **Tarihsel not:** Strang bu derste "derin öğrenme gerçekten çalışıyor mu?" sorusunu açtı — bir uzman e-postası görüşünü değiştirmiş; başarılı ağ yapıları deneyle ortaya çıkar, baştan verili değil (Pazartesi'ye bıraktı).
- **Geriye köprü:** Ders 33 (uzaklık/Gram), Ders 6-7 (SVD/Eckart-Young), Ders 8 (Frobenius normu). Paralel: fast.ai embedding hizalama, NYU manifold.

Bu ders iki bağımsız sonucu yan yana koyar: bir uzaklık matrisinin geçerlilik testi (üçgen eşitsizliği $\iff G$ PSD) ile iki vektör kümesini hizalayan en iyi rotasyon. İkisini de okurken aynı SVD'nin farklı yüzlerini ara.

39.2 1. Üçgen Eşitsizliği: Ne Zaman Konum Bulunamaz?

Ders 33'te uzaklık matrisinden konumları kurtardık. Ama soru: her D geçerli mi? Strang bir karşı-örnekle başlıyor:

> *"…by the triangle inequality, the distance from x1 to x3 could not be more than 2. And when we*

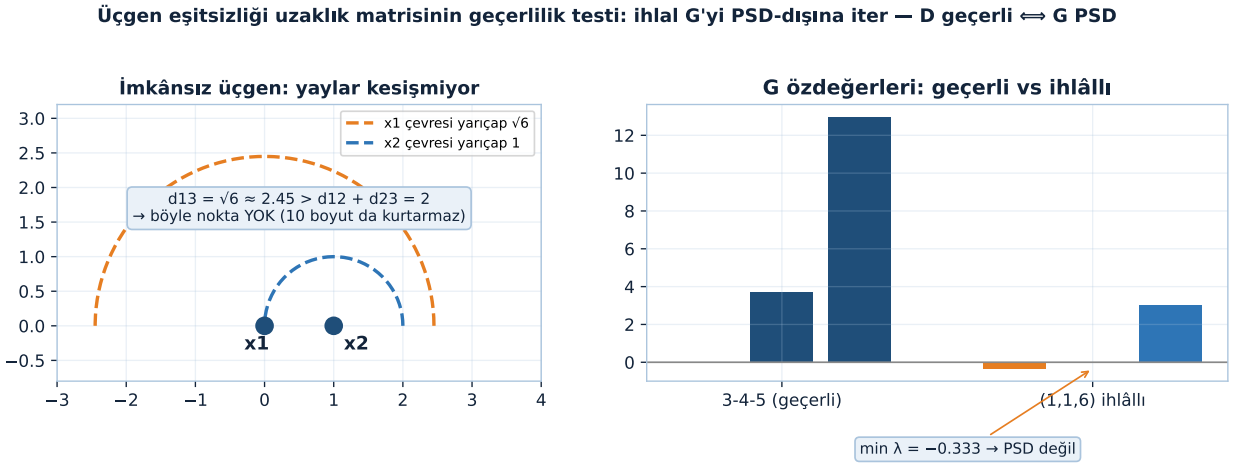
square it, it could not be more than 4. And here it's 6." — Strang, 0:00

Üç nokta; $d_{12} = 1$, $d_{23} = 1$ olsun. Üçgen eşitsizliği gereği $d_{13} \leq 2$, yani $d_{13}^2 \leq 4$. Ama D'de $d_{13}^2 = 6$ yazıyorsa — imkânsız. 10 boyuta çıkmak da kurtarmaz (üçgen eşitsizliği boyuttan bağımsızdır).

Ne ters gider? Gram matrisi $G = X^T X$, D'den türetilir ve **PSD olmalıdır** (çünkü her $X^T X$ PSD'dir). Üçgen eşitsizliği ihlal edilirse:

"…the matrix G that comes out of that equation will turn out not to be positive definite." — Strang, 4:03

G negatif özdeğer kazanır → PSD değil → karekök (X) yok. Güzel sonuç (Strang): D **geçerli konum verir** $\iff$ **üçgen eşitsizliği sağlanır** $\iff$ G **PSD'dir.**



Şekil 39.2: Üçgen eşitsizliği uzaklık matrisinin geçerlilik testidir: ihlal G'yi PSD-dışına iter — D geçerli $\iff$ G PSD. Strang örneği D=(0,1,6/1,0,1/6,1,0) çift-merkezlenmiş G'nin en küçük özdeğerini –0.333'e iter (PSD değil, konum yok); 3-4-5 üçgeni geçerli (tüm özdeğerler ≥0).

Şekil 39.2 üçgen eşitsizliğinin geçerlilik testini iki yandan motor-tanıklı gösterir: solda imkânsız üçgen — x_1 ve x_2 arasındaki uzaklık 1, ama D'nin istediği $d_{13} = \sqrt{6} \approx 2.45$ için x_1 çevresinde yarıçap $\sqrt{6}$ yay (turuncu kesik) ile x_2 çevresinde yarıçap 1 yay (steel kesik) **kesişmiyor** — $d_{13} > d_{12} + d_{23} = 2$ olduğundan böyle bir nokta yoktur (10 boyut da kurtarmaz). Sağda iki D'nin çift-merkezlenmiş G özdeğerleri yan yana: 3-4-5 üçgeni (navy, hepsi ≥ 0) geçerli; Strang'ın bozuk örneği $(1, 1, 6)$ ise en küçük özdeğeri -0.333'e iter (turuncu bar, min $\lambda = -0.3333$) — PSD değil, konum yok. Negatif özdeğer ihlalin doğrudan imzasıdır.

Builder Notu — Geçersiz Uzaklığın İmzası

"Üçgen eşitsizliği = uzaklık matrisinin geçerlilik testi" temiz bir karakterizasyon. ML köprüsü: bir benzerlik/uzaklık matrisinin gerçek bir gömmeden gelip gelmediğini PSD testiyle anlarsın — kernel yöntemlerinde "kernel geçerli mi" (Mercer koşulu) tam bu PSD kontrolüdür. Gürültülü ölçümlerde G'nin küçük negatif özdeğerlerini sıfırlayıp en yakın PSD'ye yansıtırsın (Ders 7 Eckart-Young ile akraba).

39.3 2. Procrustes Problemi: En İyi Ortogonal Hizalama

İkinci problem — adı Yunan mitinden gelir; Procrustes misafirini yatağına uydurmak için gerer ya da keserdi:

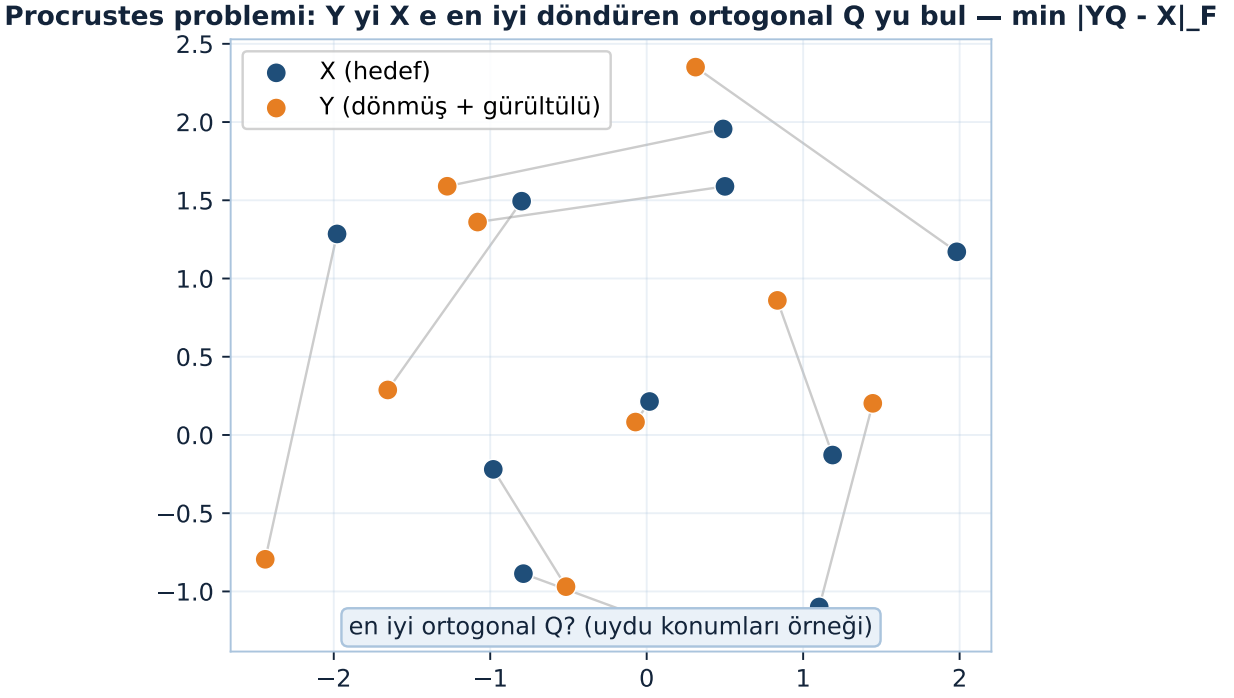
> *"…Procrustes adjusted the length of the visitor to fit the bed." — Strang, 8:06*

Problem: iki vektör kümen var — $X = (x_1, \dots, x_n)$ ve $Y = (y_1, \dots, y_n)$. Örnek: uydu konumlarının iki farklı hesabı; biri diğerinden kısmen döndürülmüş, üstüne yuvarlama hatası var. En iyi **ortogonal** Q'yu bul ki YQ, X'e olabildiğince yakın olsun:

> *"…over all orthogonal Q's I want to minimize YQ minus X in the Frobenius norm." — Strang, 12:07*

$$\min_{Q^TQ=I} \|YQ - X\|_F^2$$

Eğer X ve Y ortonormal tabanlar olsaydı tam eşitlik elde edilirdi (Q ile biri diğerine tam dönerdi); değiller, bu yüzden "en iyi"yi ararız. Bu, en küçük kareler'in ortogonal-kısıtlı halidir.



Şekil 39.3: Procrustes problemi: iki nokta bulutu — X (hedef, navy) ve Y (X'in döndürülmüş + gürültülü hali, turuncu). Eşleşen noktalar ince gri çizgilerle bağlı. Soru: Y'yi X'e en iyi oturtan ortogonal Q nedir? (uydu konumları örneği) — min |YQ - X|_F.

Şekil 39.3 problemi somutlaştırır: navy noktalar hedef kümesi X, turuncu noktalar X'in döndürülmüş ve hafif gürültülü hali Y (motorda noise $= 0.08$, gerçek rotasyon $\theta = 0.9$). Eşleşen noktalar ince gri çizgilerle bağlı — her çizgi Y'nin bir noktasını X'teki karşılığına bağlar, hizasızlığı görünür kılar. Soru tam ortada:

Y'yi X'e en iyi oturtan ortogonal Q nedir? Uydu konumları örneği bunun pratik kalbidir — iki farklı hesap aynı geometriyi taşır, yalnız bir rotasyon (ve gürültü) farkıyla. Aradığımız min $\|YQ - X\|_F$'i çözen Q.

> **Builder Notu — Yatak ile Misafir**
>
> "Procrustes = kısıtlı en küçük kareler (Q ortogonal)" pratik bir hizalama problemi. ML köprüsü: nokta-bulutu kaydı (point cloud registration), şekil eşleme ve diller-arası kelime gömme hizalaması (X=İngilizce, Y=Türkçe embedding; Q diller arası köprü) hep bunu çözer. Kabsch algoritması (moleküler yapı RMSD) ve ICP'nin (iterative closest point) çekirdeği ortogonal Procrustes'tir.

39.4 3. Frobenius Normu: Üç İfade

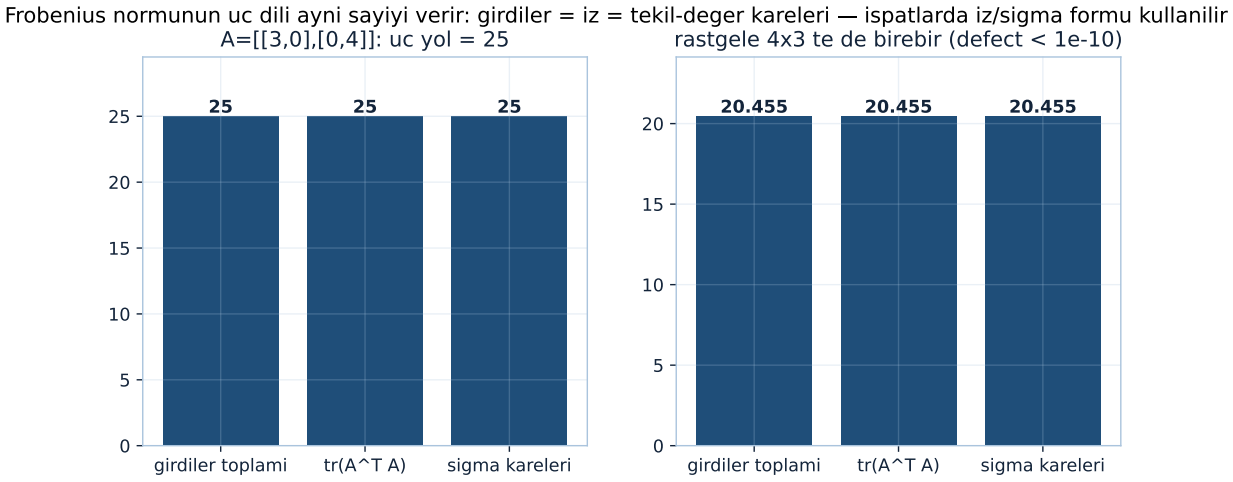
Procrustes'i çözmek için Frobenius normunu iyi tanımak gerekir. Üç eşdeğer ifadesi var:

> *"…three nice expressions for the Frobenius norm." — Strang, 16:12*

$$\|A\|_F^2 = \sum_{i,j} a_{ij}^2 = \text{tr}(A^T A) = \text{tr}(AA^T) = \sum_i \sigma_i^2$$

- **Girdiler:** matrisi uzun bir vektör gibi gör, tüm a_{ij}^2'leri topla.
- **İz (trace):** $A^T A$ (veya AA^T) köşegen toplamı = tüm sütunların kare uzunlukları toplamı.
- **Tekil değerler:** σ_i^2 toplamı — çünkü σ_i^2, $A^T A$'nın özdeğerleridir ve iz = özdeğerlerin toplamıdır.

İlk ifade yazması hantal; iz ve tekil-değer biçimleri çok daha kullanışlıdır (özellikle ispatlarda).



Şekil 39.4: Frobenius normunun üç dili aynı sayıyı verir: girdiler toplamı = iz = tekil-değer kareleri. Sol — A=[[3,0],[0,4]] için üçü de 25; sağ — rastgele 4×3 matriste de defect < 1e-10. İspatlarda iz/sigma formu kullanılır.

Şekil 39.4 üç ifadenin birebir aynı sayıyı verdiğini motor-tanıklı gösterir: solda Egzersiz 2'nin matrisi $A = [[3, 0], [0, 4]]$ için üç yol da **25** çıkar — girdiler toplamı ($3^2 + 4^2$), $\text{tr}(A^T A)$ ve $\sum \sigma_i^2$ ($\sigma = (4, 3)$); üç bar tıpatıp aynı yükseklikte. Sağda rastgele bir 4×3 matris için üç yol yine birebir eşittir (defect $< 10^{-10}$)

— eşitlik özel bir matrise değil, normun tanımına bağlı. İlk ifade (girdiler) hesaplaması hantal kalır; iz ve tekil-değer biçimleri ispatlarda kullanılan formdur, çünkü tr ve σ cebirsel manipülasyona açıktır.

> **Builder Notu — Aynı Büyüklüğün Üç Dili**
>
> "Frobenius normu = $\sqrt{(\Sigma\sigma_i^2)}$ = $\sqrt{\text{tr}(A^TA)}$" — üç dil aynı sayıyı verir. ML köprüsü: ağırlık matrisi büyüklüğü (L2/weight decay aslında Frobenius normunun karesidir), gradyan normu izleme ve düşük-rank yaklaşım hatası (Eckart-Young: atılan σ'ların kareleri) hep Frobenius'tur. PyTorch `torch.norm(A, 'fro')`.

39.5 4. Q Değişmezliği: Ortogonal Çarpım Normu Korur

Procrustes'in anahtarı: ortogonal Q ile çarpmak Frobenius normunu değiştirmez.

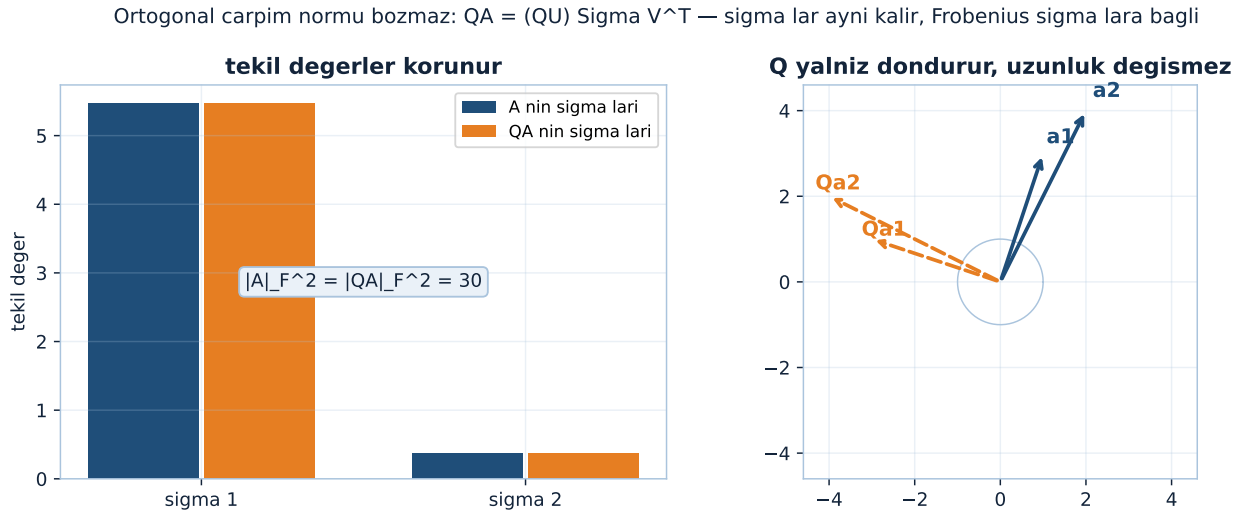
> *"…the length of Q times any vector squared is the same as the vector squared." — Strang, 18:12*

$$\|QA\|_F = \|A\|_F$$

İki kanıt: (1) QA'nın her sütunu Q ile çarpılır; Q vektör uzunluğunu korur ($\|Qv\| = \|v\|$), dolayısıyla sütun-sütun kare toplam değişmez. (2) SVD ile: $A = U\Sigma V^T$ ise $QA = (QU)\Sigma V^T$ — yalnız ilk ortogonal faktör değişir, Σ (tekil değerler) aynı kalır:

> *"…QA will have the SVD QU sigma V transpose. …I didn't change the sigmas." — Strang, 18:12*

Frobenius normu σ'lara bağlı olduğundan Q onu değiştiremez. (Aynısı diğer taraf için de geçerli: AQ' de tekil değerleri korur.)



Şekil 39.5: Ortogonal çarpım normu bozmaz: A=[[1,2],[3,4]] ve 90° rotasyon Q için A ile QA'nın tekil değerleri birebir aynı (sol), dolayısıyla $\|A\|^2\_F = \|QA\|^2\_F = 30$; Q yalnızca kolonları döndürür, uzunluklarını değiştirmez (sağ).

Şekil 39.5 Q-değişmezliğini iki yandan gösterir: solda Egzersiz 3'ün matrisi $A = [[1, 2], [3, 4]]$ ve 90° rotasyon Q için A ile QA'nın tekil değerleri **birebir aynı** (navy/turuncu barlar üst üste; tekil değer farkı **8.9e-16**), dolayısıyla $\|A\|_F^2 = \|QA\|_F^2 = 30$ (defect **0.0**). Sağda birim çember üzerinde A'nın kolonları (navy oklar) ve QA'nın kolonları (turuncu kesik oklar) — Q her kolonu 90° döndürür ama **uzunluğunu aynen korur**; oklar farklı yöne bakar, eşit uzundur. İki yan aynı mesajı söyler: $QA = (QU)\Sigma V^T$ ayrışımında yalnız ortogonal faktör değişir, σ'lar sabit kalır, Frobenius normu σ'lara bağlı olduğundan değişmez.

> Builder Notu — Döndürmek Normu Bozmaz
>
> "Ortogonal çarpım tekil değerleri ve Frobenius normunu korur" — SVD'nin temel simetrisi. ML köprüsü: ortogonal/üniter dönüşümler enerjiyi (bilgiyi) korur; bu yüzden ortogonal başlatma (orthogonal init), normalleştirici akışlar (normalizing flows) ve dönüşüme-değişmez kayıplar bu özelliğe yaslanır. Procrustes'te bu özellik, problemi bir iz (trace) maksimizasyonuna indirgememizi sağlar.

39.6 5. İz (Trace) Hileleri

Çözüm ize dayanır, o yüzden birkaç iz özdeşliği gerekir. İz = köşegen toplamı = özdeğerlerin toplamı.

> *"…the trace of A transpose B is equal to the trace of B transpose A." — Strang, 20:14*

$$\text{tr}(A^T B) = \text{tr}(B^T A) = \text{tr}(BA^T)$$

Çünkü transpoz köşegeni değiştirmez (iz, transpoz altında sabittir). Dahası, **döngüsel özellik** — çarpım sırasını çevirebilirsin:

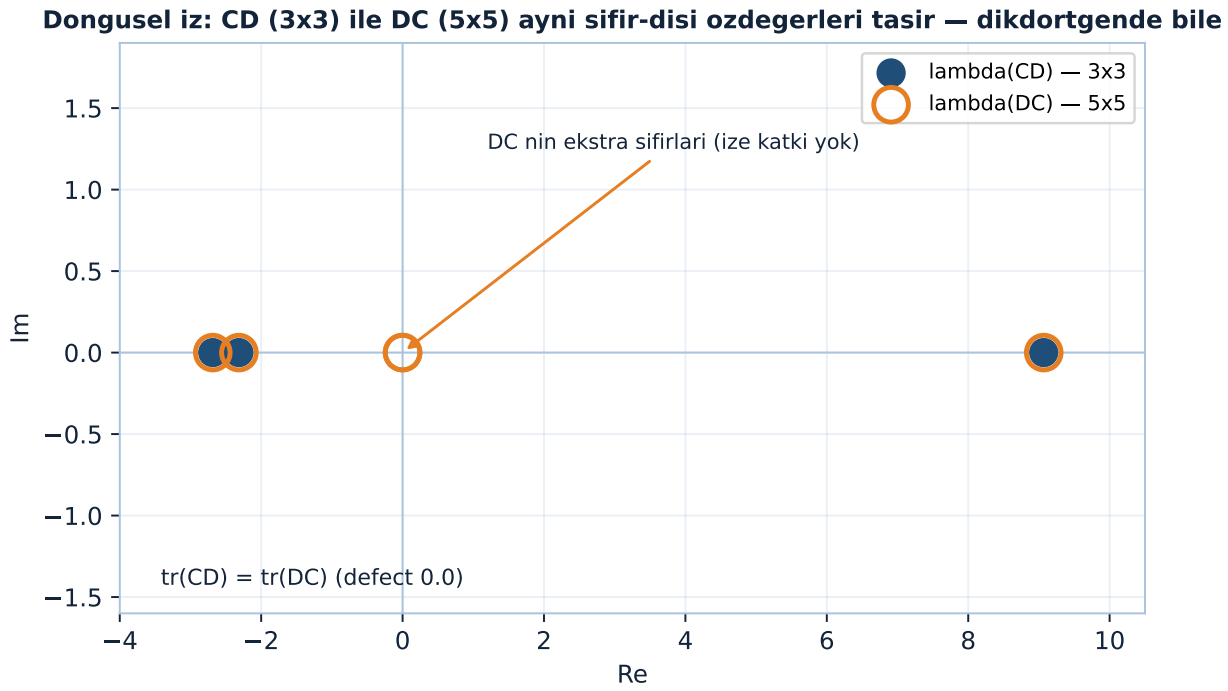
$$\text{tr}(CD) = \text{tr}(DC)$$

Neden? CD ile DC aynı sıfırdan-farklı özdeğerlere sahiptir (dikdörtgen olsalar bile fazladan sıfır özdeğerler ize katkı yapmaz), iz = özdeğer toplamı olduğundan eşittirler.

> *"…How are the eigenvalues of CD related to the eigenvalues of DC? They're the same." — Strang, 22:14*

Şekil 39.6 döngüsel iz özelliğini dikdörtgen matrislerde bile motor-tanıklı doğrular: C bir 3×5, D bir 5×3 matris olsun. $\lambda(CD)$ (3 navy nokta, 3×3) karmaşık düzlemde işaretlenir; $\lambda(DC)$ (5 turuncu halka, 5×5) bu üç özdeğeri **aynen** taşır — üç turuncu halka navy noktaların tam üstüne oturur — artı orijinde **2 ekstra sıfır** özdeğer, ki bunlar ize katkı yapmaz. Sonuç: $\text{tr}(CD) = \text{tr}(DC)$, defect **0.0**. İz = özdeğer toplamı olduğundan, sıfır-dışı özdeğerlerin ortaklığı izlerin eşitliğini doğrudan verir; çarpım sırasını çevirmek izi değiştirmez — Procrustes ispatının taşıyıcı kuralı.

> Builder Notu — Köşegen Toplamının Hileleri
>
> "İz döngüseldir: tr(CD)=tr(DC)" hesaplamada altın kuraldır. ML köprüsü: matris türevleri ($\partial\,\text{tr}(AX)/\partial X = A^T$), Frobenius normu = $\text{tr}(A^T A)$ ve birçok kayıp/regularizasyon ize çevrilir; iz-döngüsel hilesi ifadeleri optimize edilebilir biçime sokar. Bu, "matris kalkülüsü"nün (Ders 15-16) günlük aletidir.



Şekil 39.6: Döngüsel iz: CD (3×3) ile DC (5×5) aynı sıfır-dışı özdeğerleri taşır — dikdörtgende bile. λ(CD) = {9.07, −2.31, −2.68}; DC bu üçünü aynen taşır + orijinde 2 ekstra sıfır (ize katkı yok). tr(CD) = tr(DC), defect 0.0.

39.7 6. Çözüm: Q = UV^T

Frobenius normunu açıp Q değişmezliğini ve iz hilelerini kullanınca problem, bir izi maksimize etmeye iner. Cevap şaşırtıcı sadelikte:

> *"…Compute Y transpose X. Compute its SVD, and use the orthogonal matrices from the SVD."*
> *— Strang, 26:15*

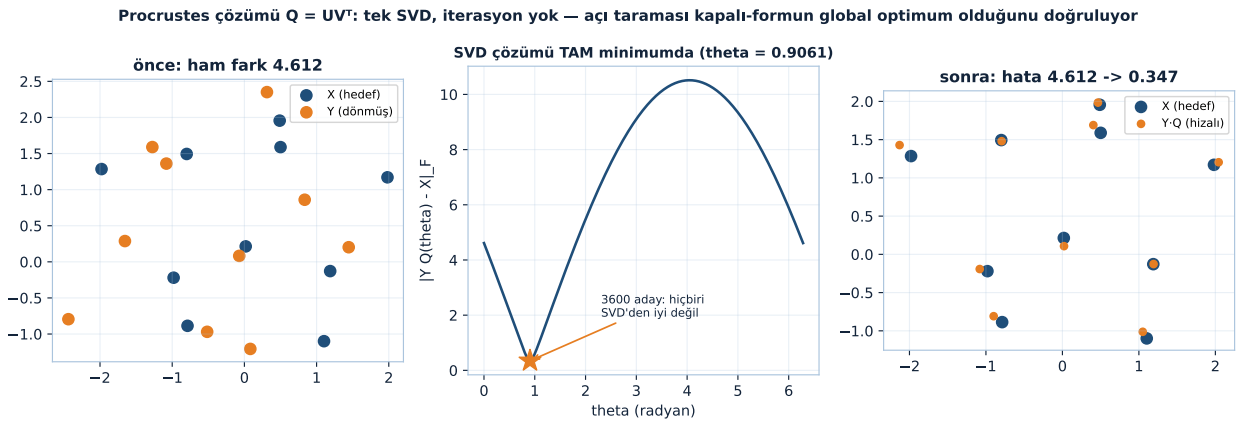
$$Y^T X = U\Sigma V^T \quad \Longrightarrow \quad Q_{\text{opt}} = UV^T$$

Yani: verilen X, Y için $Y^T X$ matrisini kur (iki kümenin tüm iç-çarpımları), SVD'sini al, ortogonal faktörleri çarp. $Q = UV^T$ en iyi ortogonal hizalamadır.

> *"…the best Q is— Da dun da duh. …It is UV transpose." — Strang, 26:15*

Σ (tekil değerler) düşer; geriye yalnız ortogonal U ve V kalır — çarpımları da ortogonaldir (iki ortogonalin çarpımı). Eğer Y zaten X'e tam dönüyorsa $Y^T X$ ortogonal çıkar ve UV^T tam o dönüşü verir.

Şekil 39.7 dersin **flagship** kanıtıdır: kapalı-form çözüm $Q = UV^T$ gerçekten global optimumu verir. Solda hizalama öncesi X (navy) ve Y (turuncu) üst üste — ham fark $|Y - X| = $ **4.612**. Ortada açı taraması: 3600 farklı θ için $\|YQ(\theta) - X\|_F$ çizilir (navy eğri); SVD çözümünün açısı $ = $ **0.9061** turuncu yıldızla işaretli ve eğrinin **tam minimumunda** oturur — 3600 adayın hiçbiri SVD çözümünden iyi değil. Bu, kapalı-formun iterasyonsuz biçimde global optimum olduğunun doğrudan tanığıdır (brute-force tarama ile kapalı-form



Şekil 39.7: Procrustes çözümü Q = UV^T: tek SVD, iterasyon yok. Sol panel hizalama öncesi ham farkı (‖Y − X‖ = 4.612) gösterir; orta panel 3600 açı adayını tarar ve SVD çözümünün (turuncu yıldız, theta = 0.9061) TAM minimumda oturduğunu — hiçbir Q(theta) ondan iyi değil — kanıtlar; sağ panel hizalama sonrası X ile Y·Q'nun üst üste düştüğünü (hata 4.612 → 0.347) gösterir. Açı taraması kapalı-formun global optimum olduğunu doğrular.

çakışır). Sağda hizalama sonrası X (navy) ile $Y \cdot Q$ (küçük turuncu) neredeyse üst üste — hata \$4.612 →\$ **0.347**'ye iner (kalan, eklenen gürültünün payı). Tek bir SVD, hiçbir optimizasyon döngüsü olmadan, açı-uzayını tarayan bir aramanın bulduğu optimumu verir.

> Builder Notu — Tek SVD ile En İyi Dönüş
>
> "En iyi ortogonal Q = UV^T (Y^TX'in SVD'sinden)" kapalı-form bir mücevher. ML köprüsü: bu formül ortogonal Procrustes'in standart çözümüdür — şekil hizalama, embedding hizalama (örn. çok-dilli kelime vektörleri, MUSE) ve poz tahmini hep bunu kullanır. SVD'nin "en yakın ortogonal matrisi verme" gücüyle (polar ayrışım, Ders 6) aynı kökten gelir.

39.8 Bu Dersin Özeti

- **Üçgen eşitsizliği:** D geçerli konum verir $\iff$ üçgen eşitsizliği sağlanır $\iff$ $G = X^TX$ PSD'dir (ihlalde G negatif özdeğer kazanır, X bulunamaz).
- **Procrustes problemi:** ortogonal Q üzerinden min $\|YQ - X\|_F^2$ — iki vektör kümesini en iyi rotasyonla hizala.
- **Frobenius normu:** $\|A\|_F^2 = \sum a_{ij}^2 = \text{tr}(A^TA) = \text{tr}(AA^T) = \sum \sigma_i^2$ (üç eşdeğer ifade).
- **Q değişmezliği:** $\|QA\|_F = \|A\|_F$ (ortogonal çarpım tekil değerleri ve Frobenius normunu korur).
- **İz hileleri:** $\text{tr}(A^TB) = \text{tr}(B^TA)$, $\text{tr}(CD) = \text{tr}(DC)$ (döngüsel; $\lambda(CD) = \lambda(DC)$ sıfırdan-farklı özdeğerler).
- **Çözüm:** $Y^TX = U\Sigma V^T \Rightarrow Q = UV^T$ — en iyi ortogonal hizalama, tek bir SVD ile.

! Tek Bir Cümle

Bir uzaklık matrisi ancak üçgen eşitsizliğini sağlarsa konuma çevrilebilir (yoksa $G = X^T X$ PSD olmaz); Procrustes problemi ise iki vektör kümesini en iyi hizalayan ortogonal matrisi tek bir SVD ile bulur — $Y^T X = U\Sigma V^T \Rightarrow Q = UV^T$.

39.9 Kontrol Soruları

i Soru 1 — Üçgen eşitsizliği ihlal edilirse ne olur; bu G'yi nasıl etkiler

Üçgen eşitsizliği (örn. $d_{13} \leq d_{12} + d_{23}$) ihlal edilirse, D'den türetilen Gram matrisi $G = X^T X$ **pozitif yarı-tanımlı çıkmaz** — negatif özdeğer kazanır. PSD olmayan bir matrisin $X^T X$ biçiminde karekökü yoktur, dolayısıyla bu uzaklıkları sağlayan konumlar bulunamaz. Tam karakterizasyon: D geçerli konum verir $\iff$ üçgen eşitsizliği sağlanır $\iff$ G PSD'dir.

i Soru 2 — Procrustes problemi nedir; neden genelde tam eşitlik olmaz

İki vektör kümesi X ve Y verili; amaç ortogonal Q ile $\min_Q \|YQ - X\|_F^2$'yi çözmek (Y'yi X'e en iyi döndürmek). Tam eşitlik ancak X ve Y her ikisi de ortonormal tabansa ve biri diğerinin tam rotasyonuysa olur. Pratikte iki küme yuvarlama hatası/gürültü içerir ya da tam ortonormal değildir, bu yüzden "tam" değil "en iyi" hizalamayı ararız.

i Soru 3 — Ortogonal Q neden Frobenius normunu değiştirmez

İki nedenle. (1) Q her vektörün uzunluğunu korur ($\|Qv\| = \|v\|$); Frobenius normu sütunların kare uzunlukları toplamı olduğundan QA ile A aynı normda. (2) SVD ile: $A = U\Sigma V^T$ ise $QA = (QU)\Sigma V^T$ — yalnız ilk ortogonal faktör değişir, tekil değerler Σ aynı kalır. Frobenius normu $\sqrt{\sum \sigma_i^2}$ olduğundan değişmez.

i Soru 4 — Procrustes çözümü nedir; nasıl hesaplanır

İki kümenin iç-çarpım matrisi $Y^T X$'i kur, SVD'sini al: $Y^T X = U\Sigma V^T$. En iyi ortogonal matris $Q = UV^T$'dir. Σ (tekil değerler) çözümden düşer; yalnız ortogonal faktörler U ve V kullanılır. Bu kapalı-form çözüm optimizasyon iterasyonu gerektirmez — tek bir SVD yeterlidir.

39.10 Egzersizler

1. **Üçgen testi.** Üç nokta için uzaklıklar $d_{12} = 1$, $d_{23} = 1$, $d_{13} = 3$ (kareler 1, 1, 9). Üçgen eşitsizliği sağlanıyor mu? Bu D'den geçerli konumlar bulunabilir mi — Gram matrisi G PSD olur mu (sezgisel açıkla)? (Motor tanığı: $d_{13} = 3 > d_{12} + d_{23} = 2 \rightarrow$ İHLAL; çift-merkezlenmiş G'nin en küçük özdeğeri **−0.8333** $\rightarrow$ PSD değil, konum yok. Sınır durumu $d_{13}^2 = 4$ (tam eşitlik): min $\lambda \approx 0$ — üç nokta bir çizgiye çöker, doğrusal dizilim.)

2. **Frobenius üç yol.** $A = [[3,0],[0,4]]$. $\|A\|_F^2$'i üç yolla hesapla: (a) girdilerin kareleri toplamı, (b) tr(A^TA), (c) $\sum \sigma_i^2$ (bu köşegen matriste σ'lar nedir?). Üçü de eşit mi? (Motor tanığı: girdiler $= tr(A^T A) = $\sum$_i^2 = $ **25**; $\sigma = (4,3)$.)
3. **Q değişmezliği.** $A = [[1,2],[3,4]]$, $Q = [[0,-1],[1,0]]$ (90° rotasyon). $\|A\|_F^2$ ve $\|QA\|_F^2$'i hesapla; eşit olduklarını göster. (Motor tanığı: $|A|_F^2 = |QA|_F^2 = $ **30** (defect 0.0); tekil değerler birebir korunur (fark 8.9e-16).)
4. **Procrustes.** $X = [[1,0],[0,1]]$ (birim), $Y = [[0,1],[-1,0]]$ (90° dönmüş). Y^TX'i kur, SVD'sinden $Q = UV^T$'yi bul; Q'nun Y'yi X'e götürdüğünü ($YQ \approx X$) doğrula. (Motor tanığı: $Q = UV^T$ ortogonal; Y ortogonal olduğundan $YQ = X$ **TAM** eşitlik.)
5. **(Ders 35 habercisi)** Bir sosyal ağı (graf) iki kümeye ayırmak istiyorsun. Hangi matris ve hangi özvektör işe yarar? Bir tahmin yaz — Ders 35 "graflarda kümeler bulmak"ı (graf Laplacian'ı, Fiedler vektörü, spektral kümeleme) işliyor.

39.11 Sonraki Ders İçin Hazırlık

Ders 35: Graflarda Kümeler Bulmak. Sinyal/CNN/graf bloğunun graf kısmı: bir grafı iki (veya k) kümeye ayırma (graph partitioning/clustering). Araç graf Laplacian'ı ($L = D - A$) ve onun ikinci en küçük özvektörü (Fiedler vektörü) — spektral kümeleme. Lineer cebir, ağ yapısını kümelere çevirir; Ders 19'un maxmin/özvektör fikriyle akraba.

> ⚠ Hazırlık
>
> Bu dersin Egzersiz 5'inin sorduğu soruyu zihninde tut: bir grafı iki kümeye ayırmak için hangi matris ve hangi özvektör işe yarar? Ders 35 **graf Laplacian'ını** ($L = D - A$) ve onun ikinci en küçük özvektörünü (Fiedler vektörü) işler — spektral kümeleme. Bu derste gördüğümüz "SVD/özdeğerlerden geometri çıkarma" fikri orada graf yapısına taşınır: özvektör, düğümleri iki kümeye en doğal biçimde böler.

39.12 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Üçgen eşitsizliği	D geçerli $\iff G = X^TX$ PSD; ihlalde negatif özdeğer	0m00
Procrustes	$\min_Q \|YQ - X\|_F^2$ (Q ortogonal)	12m07
Frobenius normu	$\|A\|_F^2 = \sum a_{ij}^2 = \text{tr}(A^TA) = \sum \sigma_i^2$	16m12
Q değişmezliği	$\|QA\|_F = \|A\|_F$ (σ korunur)	18m12
İz döngüsel	$\text{tr}(CD) = \text{tr}(DC)$; $\lambda(CD) = \lambda(DC)$	22m14
Procrustes çözümü	$Y^TX = U\Sigma V^T \Rightarrow Q = UV^T$	26m15

39.13 ML Bağlantıları Özeti

- **Üçgen eşitsizliği / PSD:** geçerli kernel/uzaklık testi (Mercer koşulu); gürültülü G'yi en yakın PSD'ye yansıtma (Eckart-Young).
- **Procrustes:** nokta-bulutu kaydı (ICP, Kabsch/RMSD), şekil hizalama, çok-dilli kelime gömme hizalama (MUSE), poz/oryantasyon tahmini.
- **Frobenius + iz:** L2/weight decay (= $\|W\|_F^2$), gradyan normu izleme, düşük-rank yaklaşım hatası; PyTorch `torch.norm(·,'fro')`.
- **Q değişmezliği:** ortogonal başlatma, normalleştirici akışlar, enerji koruyan dönüşümler bu özelliğe dayanır.
- **Çözüm $Q=UV^T$:** SVD'nin "en yakın ortogonal matrisi verme" gücü (polar ayrışım, Ders 6) ile aynı kök.
- **Geriye köprü:** Ders 33 (uzaklık/Gram), Ders 6-7 (SVD/Eckart-Young/polar), Ders 8 (Frobenius), Ders 15-16 (matris kalkülüsü/iz). Paralel: fast.ai embedding hizalama, NYU manifold öğrenme.

! Kapanış

> *"…the best Q is— …It is UV transpose." — Strang, 26:15*

Procrustes problemi "iki şekli en iyi nasıl üst üste bindiririm" sorusuna SVD ile tek-satırlık yanıt verir; üçgen eşitsizliği codası ise geometrinin (uzaklıkların) ne zaman tutarlı olduğunu söyler — ikisi birlikte PSD ve SVD'nin ham sayılardan geometri çıkarma gücünü gösterir.

40 Graflarda Kümeler Bulmak

k-means alternasyonu, graf Laplacian'ı ve Fiedler vektörüyle spektral kümeleme

> **i** Bölüm bilgisi
>
> Sinyal/CNN/graf bloğunu kapatan ders: bir grafı bölmenin görünür kombinatoryal sorusunu sürekli bir özvektör problemine çeviren spektral kümeleme. Strang'in Ders 35 videosu (≈35 dk) ve OCW Lecture 35 temel alınmıştır. Okuma süresi ≈24 dk. Önkoşul Ders 4-6 (spektral teorem/PSD), Ders 19 (özvektör/maxmin) ve Ders 30-32 (sirkülant/Fourier/Kronecker — graf Laplacian'ı bu dünyaların graf hali).

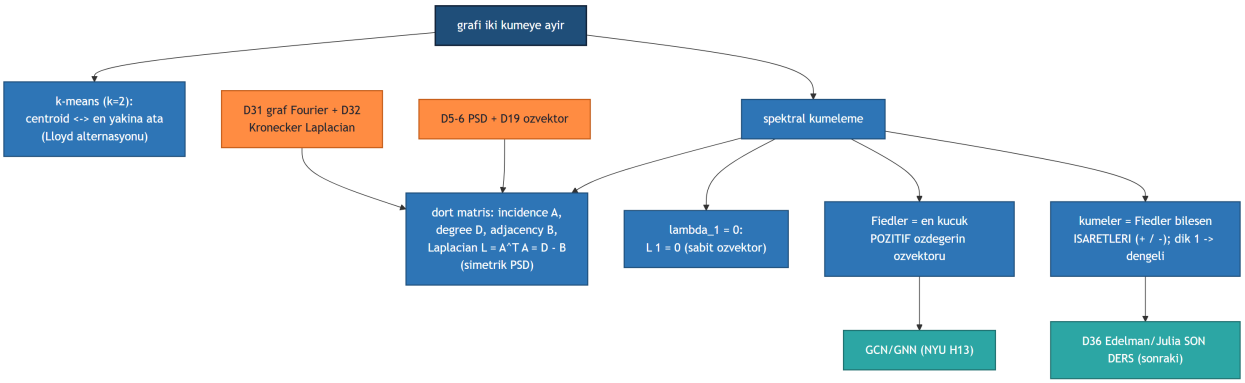
40.1 Bu Derste Ne Var?

Bloğun graf kısmı: büyük bir grafı anlamlı kümelere ayırmak. İki nokta kümesine bölmek için üç yaklaşım var; en zarifi **spektral kümeleme** — graf Laplacian'ının **Fiedler vektörü** kümeleri doğrudan verir.

Beş sonuç:

1. **Graf kümeleme problemi:** Grafı iki (≈eşit boyutlu) kümeye ayır. Hedef: min $\sum_{a_i \in A} \|a_i - x\|^2 + \sum_{b_i \in B} \|b_i - y\|^2$ (artı boyut dengesi).
2. **k-means (k=2):** Kümeler verilince centroid'leri (x, y) bul $\leftrightarrow$ centroid'ler verilince her nokta en yakın merkeze; yakınsayana dek **alternasyon**.
3. **Spektral kümeleme:** "spektral" = özdeğer. Graf Laplacian'ı $L = A^T A = D - B$ (incidence^T × incidence = derece − komşuluk); simetrik PSD.
4. $\lambda_1 = 0$ **+ Fiedler:** L tekildir, $L\mathbf{1} = 0$ (özvektör tüm-birler). **Fiedler vektörü** = en küçük **pozitif** özdeğerin özvektörü.
5. **Kümeler Fiedler işaretlerinden:** Fiedler vektörünün **pozitif** bileşenleri bir küme, **negatif** bileşenleri öteki. (Fiedler $\perp \mathbf{1}$ olduğundan bileşenleri 0'a toplanır.)

Şekil 40.1 dersin iskeletini gösterir: merkezdeki **grafi iki kümeye ayır** düğümünden iki yol çıkar. İlk yol k-means (k=2) — centroid ile en yakına ata arasında Lloyd alternasyonu. İkinci yol spektral kümeleme; bu dal grafın matrislerini kullanır. Spektral dalın alt-dalları dört anahtar matrisi (incidence A, degree D, adjacency B, Laplacian $L = A^T A = D - B$, simetrik PSD), $\lambda_1 = 0$ özdeşliğini ($L\mathbf{1} = 0$, sabit özvektör), Fiedler tanımını (en küçük pozitif özdeğerin özvektörü) ve son adımı (kümeler Fiedler bileşen işaretlerinden, $\perp \mathbf{1}$ olduğundan dengeli) gösterir. Köprü düğümleri dersi öncesine ve sonrasına bağlar: D5-6 PSD + D19 özvektör; D31 graf Fourier + D32 Kronecker Laplacian; GCN/GNN (NYU H13); ve D36 Edelman/Julia, kursun SON dersi (sonraki).



Şekil 40.1: Ders 35 kavram haritasi: merkez bir grafi iki kumeye ayir. Iki yaklasim: k-means (k=2) centroid ile en yakina ata arasinda Lloyd alternasyonu; spektral kumeleme grafin matrislerini kullanir. Spektral dal: dort matris incidence A, degree D, adjacency B, Laplacian L = A^T A = D - B (simetrik PSD); lambda_1 = 0 cunku L 1 = 0 (sabit ozvektor); Fiedler = en kucuk POZITIF ozdegerin ozvektoru; kumeler Fiedler bilesen isaretlerinden (+ / -) okunur, dik 1 oldugundan dengeli boluner. Koprular: D5-6 PSD + D19 ozvektor; D31 graf Fourier + D32 Kronecker Laplacian; GCN/GNN (NYU H13); D36 Edelman/Julia SON DERS (sonraki).

Builder Notu — Kombinatoriği Özdeğere Çevirmek

- **Graf Laplacian'ı = lineer cebir × graf teorisi köprüsü:** dört anahtar matris — incidence (m×n), degree (köşegen), adjacency, Laplacian ($L = D - B = A^T A$).
- **Spektral kümeleme = özvektörle kümeleme:** graf yapısını özdeğer/özvektöre çevirir; GNN/GCN'lerin (graf sinir ağları) matematiksel atası.
- **k-means = en temel denetimsiz öğrenme:** centroid alternasyonu (Lloyd algoritması); yakınsama teorisi zayıf ama pratikte çok kullanılır.
- **Geriye köprü:** Ders 5-6 (PSD/SVD), Ders 19 (maxmin/özvektör), Ders 30-31 (sirkülant/Fourier — graf Laplacian'ı döngüsel grafta sirkülant). Paralel: NYU H13 spektral GCN (Bresson, laplacian-smoothness/ChebNet), fast.ai embedding.

Bir grafı bölmek görünüşte kombinatoryal bir tercih (hangi düğüm hangi tarafta) gibidir; bu dersin numarası onu sürekli bir özvektör problemine çevirir. Okurken aynı $L = D - B$ matrisinin iki yüzünü ara: incidence çarpımı ($A^T A$) onu PSD yapar, derece-eksi-komşuluk biçimi ise $L\mathbf{1} = 0$'ı görünür kılar.

40.2 1. Graf Kümeleme Problemi

Büyük bir grafın düğümlerini iki kümeye ayırmak istiyoruz — iki makul-eşit parça arasında iyi bir "cut" (kesim) bulan bir algoritma.

> *"…clustering for graphs. …you've got a giant graph. And then the job is to make some sense out of it." — Strang, 0:00*

Düğümleri A (merkez x) ve B (merkez y) olarak böl; amaç toplam uzaklık-karesini en aza indirmek:

$$\min \sum_{a_i \in A} \|a_i - x\|^2 + \sum_{b_i \in B} \|b_i - y\|^2$$

A ∪ B = tüm düğümler, A ∩ B = ∅. Ek koşul: |A| ≈ |B| (dengeli kümeler — yoksa "bir düğüm + gerisi" gibi değersiz bir kesim çıkar).

> *"…I probably want to impose some condition that the number of a's is reasonably close to the number of b's." — Strang, 4:03*

> Builder Notu — İyi Kesim Dengeli Kesimdir
>
> "Graf kümeleme = dengeli iyi kesim" temel bir ağ analizi problemi. ML köprüsü: topluluk tespiti (community detection) sosyal ağlarda, görüntü segmentasyonu (normalized cut), öneri sistemlerinde kullanıcı/ürün gruplama. Denge koşulu kritik — saf min-cut tek bir düğümü ayırarak hile yapabilir, bu yüzden dengeli kesim aranır.

40.3 2. k-means Algoritması (Centroid Alternasyonu)

İlk yöntem **k-means** (burada k=2: A ve B'ye bölme). İki adım arasında dönüşümlü çalışır.

Adım 1 — Kümeler verili → centroid'ler. A kümesi verilince en iyi x, A'nın **centroid'idir** (ağırlık merkezi): noktaların ortalaması.

> *"…X is the centroid of the a's. …it's the sum of the a's… divided by the number of a's." — Strang, 6:07*

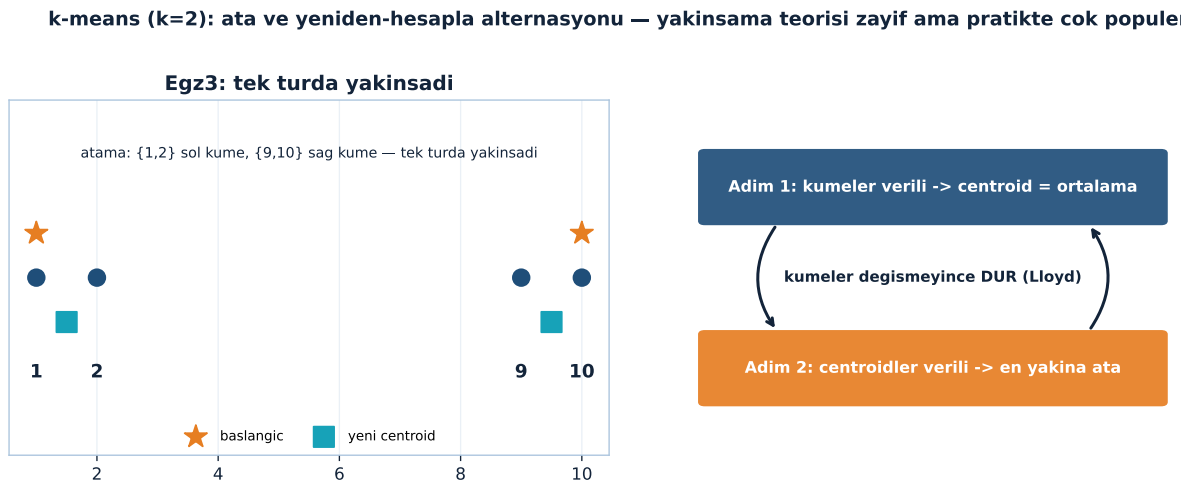
$$x = \frac{1}{|A|} \sum_{a_i \in A} a_i, \qquad y = \frac{1}{|B|} \sum_{b_i \in B} b_i$$

Adım 2 — Centroid'ler verili → kümeler. Her düğümü, x ile y'den **hangisine yakınsa** o kümeye ata.

> *"…each node goes with the closer of x and y." — Strang, 10:20*

Sonra Adım 1'e dön (yeni centroid'ler), tekrar Adım 2… Kümeler değişmeyince algoritma yakınsamıştır. Strang: düzgün bir yakınsama/yakınsama-hızı teorisi yok, ama çok popüler bir yöntem.

Şekil 40.2 Lloyd alternasyonunu hem somut bir örnekte hem şema olarak gösterir. Solda Egzersiz 3'ün durumu: doğru üzerindeki noktalar $\{1, 2, 9, 10\}$ (navy), başlangıç centroidleri $x = 1$ ve $y = 10$ (turuncu yıldız). Tek turda atama $\{1, 2\}$'yi sol kümeye, $\{9, 10\}$'u sağ kümeye gönderir; yeni centroidler ortalamalardır — **1.5** ve **9.5** (teal kareler). Bir sonraki atama hiçbir düğümü değiştirmez, yani algoritma **tek turda yakınsadı**. Sağdaki şema iki adımı kapatır: navy kutu "kümeler verili → centroid = ortalama", turuncu kutu "centroidler verili → en yakına ata"; çift ok döngüyü, ortadaki not "kümeler değişmeyince DUR (Lloyd)" durdurma koşulunu gösterir. Yakınsama teorisi zayıf olsa da pratikte bu basit alternasyon her yerde kullanılır.



Şekil 40.2: k-means (k=2): ata ve yeniden-hesapla alternasyonu — yakinsama teorisi zayif ama pratikte cok populer. Sol: Egz3 {1,2,9,10}, baslangic centroidleri 1 ve 10 -> kumeler {1,2} ve {9,10}, yeni centroidler 1.5 ve 9.5, tek turda yakinsadi. Sag: Lloyd alternasyonu — Adim 1 (kumeler -> centroid = ortalama) ve Adim 2 (centroidler -> en yakina ata) doneru, kumeler degismeyince durur.

> **Builder Notu — Ata ve Yeniden Hesapla**
>
> "k-means = ata ↔ yeniden-hesapla alternasyonu" denetimsiz öğrenmenin klasiğidir (Lloyd algoritması). ML köprüsü: kümeleme (müşteri segmentasyonu, görüntü renk nicemleme), vektör nicemleme (VQ) ve embedding kümeleme. Zayıflıkları: yerel minimuma takılır (başlangıca duyarlı; k-means++ ile iyileştirilir), küme sayısı k önceden verilmelidir. EM algoritmasının sert-atama (hard assignment) halidir.

40.4 3. Spektral Kümeleme ve Spektral Teorem

İkinci yöntem **spektral kümeleme**. Önce: "spektral" ne demek?

> *"…spectral clustering is using the eigenvalues of some matrix." — Strang, 12:20*

Spektrum = bir matrisin **özdeğerleri**. Spektral kümeleme, grafa bağlı bir matrisin özdeğer/özvektörlerini kullanır. Dayanağı **spektral teorem**: simetrik bir S için özdeğerler gerçeldir, özvektörler ortogonaldir — $\lambda = 5$ dört kez tekrarlasa bile, ona karşılık dört bağımsız ve ortogonal özvektör bulunur (yalnız simetrik matrislerde garanti).

> *"…for a symmetric matrix S, the eigenvalues are real, and the eigenvectors are orthogonal." — Strang, 14:21*

> **Builder Notu — Adını Özdeğerden Alır**
>
> "Spektral = özdeğer dünyası" — ad buradan gelir. ML köprüsü: spektral kümeleme, spektral graf teorisi ve PCA (Ders 7) hep "doğru matrisin özvektörleri yapıyı açığa çıkarır" fikrini paylaşır. Spektral teoremin

ortogonal-özvektör garantisi (Ders 4-5) bu yöntemleri sağlam kılar — kümeleri ortogonal eksenlere yansıtırsın.

40.5 4. Graf Laplacian'ı: Dört Anahtar Matris

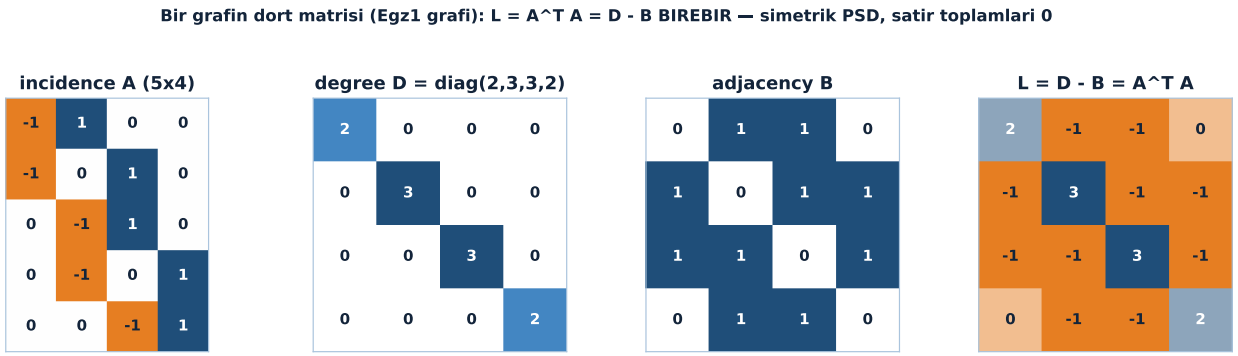
Spektral kümeleme **graf Laplacian'ı** ile başlar — lineer cebirin graf teorisine en önemli köprüsü. Her grafın dört temel matrisi vardır:

- **Incidence (geliş) matrisi A:** m×n (kenar × düğüm); her satır bir kenar, başladığı düğümde −1, bittiği düğümde +1.
- **Degree (derece) matrisi D:** n×n köşegen; her düğümün kenar sayısı.
- **Adjacency (komşuluk) matrisi B:** n×n; köşegen 0, bağlı düğüm çiftinde 1.
- **Laplacian L:** iki eşdeğer biçim —

$$L = A^T A = D - B$$

"…this Laplacian is symmetric, positive, semi-definite." — Strang, 16:25

A incidence matrisi olduğundan $A^T A$ simetrik PSD'dir; aynı matris $D - B$'ye (derece eksi komşuluk) eşittir. Köşegende dereceler, köşegen-dışında bağlı çiftlerde −1 oturur.



Şekil 40.3: Bir grafın dört matrisi (Egz1 grafı): L = A^TA = D − B BİREBİR — simetrik PSD, satır toplamları 0.

Şekil 40.3 Egzersiz 1 grafının (kenarlar (1-2),(1-3),(2-3),(2-4),(3-4)) dört matrisini yan yana koyar. Soldaki incidence A (5×4) her satırda bir kenarı kodlar — başlangıçta −1, bitişte +1 (turuncu/navy diverging renk). Derece matrisi $D = \text{diag}(2, 3, 3, 2)$ köşegende düğüm derecelerini taşır; adjacency B bağlı çiftleri işaretler. Sağdaki Laplacian $L = D - B = A^T A$ iki yoldan da **birebir** aynı çıkar: köşegende dereceler, köşegen-dışında bağlı çiftlerde −1; her satır toplamı sıfır. $A^T A$ biçimi onu simetrik PSD yapar, $D - B$ biçimi $L\mathbf{1} = 0$'ı görünür kılar — bir sonraki bölümün başlangıç noktası.

Builder Notu — Bir Grafın Dört Matrisi

"L = A^TA = D − B" graf teorisinin lineer cebir kalbidir. ML köprüsü: graf sinir ağları (GCN) tam bu Laplacian'ı (ya da normalize edilmiş halini) kullanır — komşu bilgisini toplama (message passing) bir

Laplacian çarpımıdır. Spektral graf teorisi; ağ analizi (PageRank akrabası), kümeleme ve yarı-denetimli öğrenmenin temelidir.

40.6 5. $\lambda_1 = 0$ ve Fiedler Vektörü

L PSD'dir, ama tekil mi? $Lx = 0$'ın sıfırdan farklı çözümü var mı?

> *"…the vector of all 1's will be a solution to Lx equals 0." — Strang, 20:32*

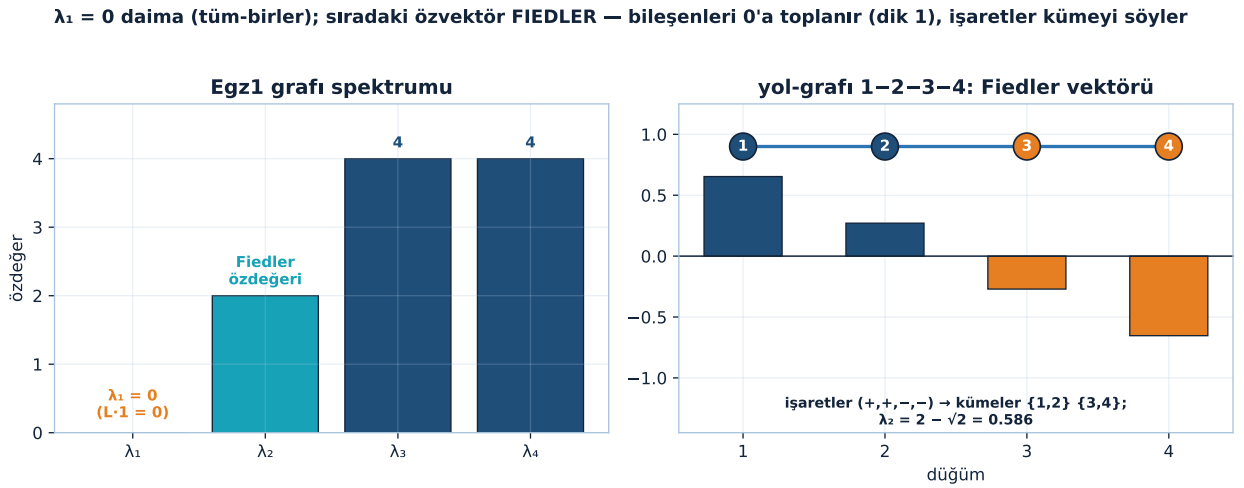
Evet: tüm-birler vektörü $(1, 1, \dots, 1)$ her zaman $Lx = 0$ sağlar (her satırın toplamı sıfır — derece eksi komşu sayısı). Yani:

$$L\mathbf{1} = 0, \qquad \lambda_1 = 0$$

ve karşılık gelen özvektör sabittir $(C, C, \dots, C)$. Bu, her bağlı grafta olur (boş-uzay boyutu 1). **Graf kümelemenin fikri**: bir sonraki özvektöre bak — en küçük **pozitif** özdeğerin (λ_2) özvektörü:

> *"…this is called the Fiedler vector, named after the Czech mathematician." — Strang, 22:38*

Bu **Fiedler vektörü** (Çek matematikçi Fiedler'den). λ_2 ve özvektörü, grafın bağlantı yapısını taşır.

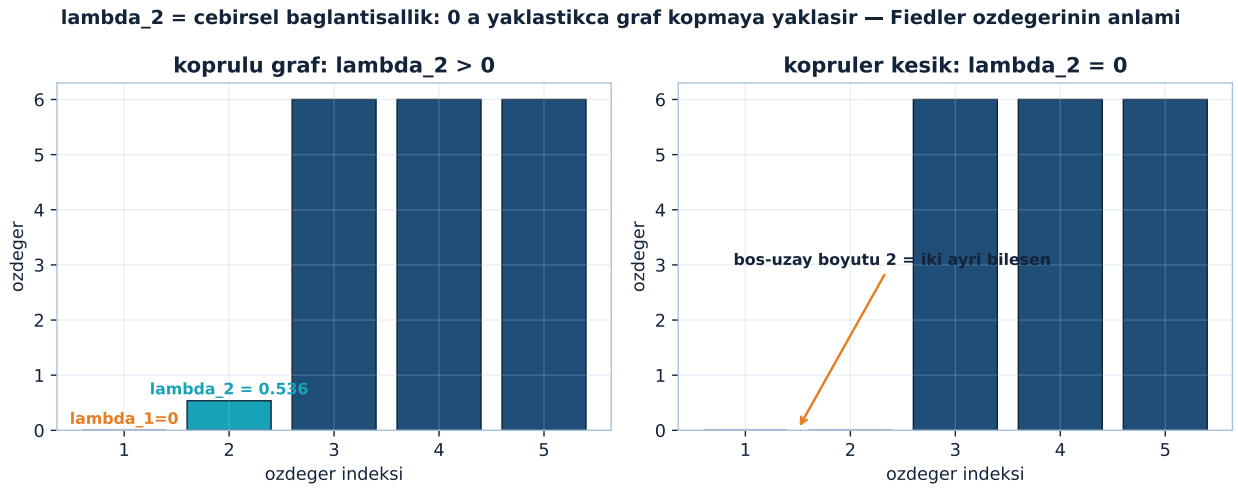


Şekil 40.4: λ_1 = 0 daima (tüm-birler özvektörü); sıradaki özvektör FIEDLER — bileşenleri 0'a toplanır (1'e dik), işaretleri kümeyi söyler. Sol: Egz1 grafının spektrumu (0, 2, 4, 4) — λ_2 = 2 Fiedler özdeğeri. Sağ: yol-grafı 1–2–3–4'te Fiedler işaretleri (+,+,−,−) → kümeler {1,2} ve {3,4}, $\lambda_2 = 2 - \sqrt{2} =$ 0.586.

Şekil 40.4 $\lambda_1 = 0$'dan Fiedler'e geçişi iki grafta gösterir. Solda Egzersiz 1 grafının spektrumu: özdeğerler $(0, 2, 4, 4)$. İlk bar (turuncu) $\lambda_1 = 0$ — tüm-birler özvektörü, $L\mathbf{1} = 0$; ikinci bar (teal) $\lambda_2 = 2$, Fiedler özdeğeri; kalan ikisi (navy) $\lambda_3 = \lambda_4 = 4$. Sağda yol-grafı 1–2–3–4: üstteki şerit dört düğümü çizgisel bağlar, düğüm renkleri Fiedler işaretine göredir (pozitif navy / negatif turuncu); alttaki bar Fiedler bileşenlerini gösterir. İşaretler **(+,+,−,−)** çıkar — yani kümeler $\{1, 2\}$ ve $\{3, 4\}$, yolu tam ortasından keser. Fiedler vektörü $\mathbf{1}$'e dik olduğundan bileşenleri 0'a toplanır; $\lambda_2 = 2 - \sqrt{2} \approx$ **0.586** (bilinen kapalı-form, motorla birebir).

> **Builder Notu — Sıfırdan Sonraki İlk Özvektör**
>
> “λ_1=0 (sabit özvektör), λ_2 = Fiedler” graf Laplacian’ının imzasıdır. ML köprüsü: λ_2 (cebirsel bağlantısallık, algebraic connectivity) grafın ne kadar “iyi bağlı” olduğunu ölçer — 0’a yakınsa graf neredeyse iki parçaya ayrık demektir. Fiedler vektörü, manifold öğrenmede (Laplacian eigenmaps) ve GCN’de en düşük-frekanslı graf modudur (Ders 31 Fourier ile akraba: Laplacian özvektörleri = graf Fourier tabanı).

λ_2’nin “cebirsel bağlantısallık” olduğunu somutlaştırmak için, planted (iki-topluluk) grafı köprülü ve köprüsüz haliyle yan yana koyalım:



Şekil 40.5: λ_2 = cebirsel bağlantısallık: köprülü grafta $\lambda_2 = 0.536 > 0$; köprüler kesilince $\lambda_1 = \lambda_2 = 0$ (boş-uzay boyutu 2 = iki ayrı bileşen). Sıfıra yaklaşan λ_2, grafın kopmaya yaklaştığını söyler — Fiedler özdeğerinin anlamı.

Şekil 40.5 λ_2’nin grafın “ne kadar bağlı” olduğunu nasıl ölçtüğünü doğrudan gösterir. Solda iki-topluluk grafının ilk beş özdeğeri: $\lambda_1 = 0$ (turuncu), $\lambda_2 = \mathbf{0.536}$ (teal) — köprü kenarları zayıf olduğundan λ_2 küçük ama **pozitif** kalır; graf tek parçadır. Sağda aynı graftan iki köprü kenarı kesilmiştir: şimdi $\lambda_1 = \lambda_2 = \mathbf{0}$ (iki turuncu bar) — boş-uzay boyutu **2**’ye çıkar, çünkü graf iki ayrı bileşene düşmüştür. Bu, “λ_2 bağlantıyı ölçer” tanığının çekirdeğidir: λ_2 sıfıra yaklaştıkça graf kopmaya yaklaşır, sıfıra ulaştığında kopmuştur.

40.7 6. Neden “Laplacian”? (Laplace Denklemi)

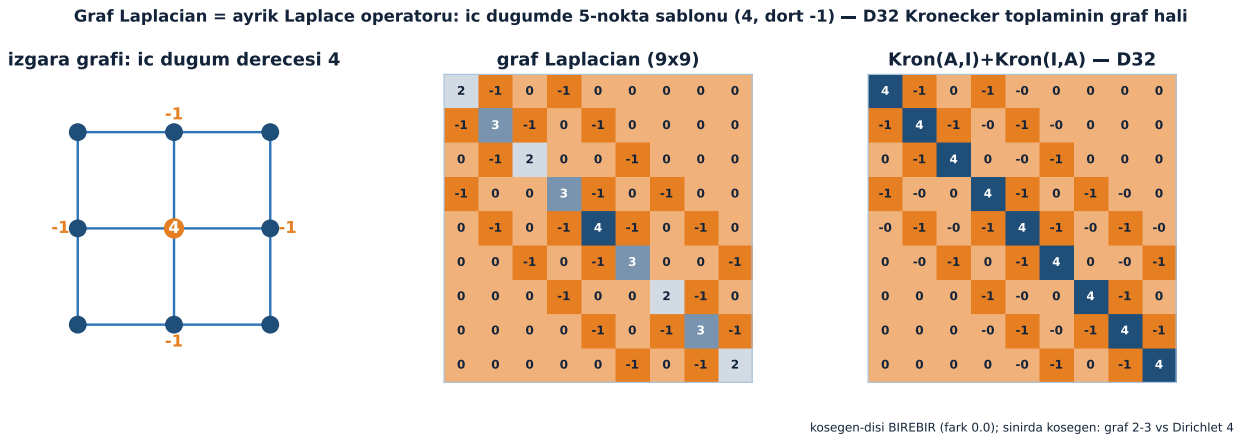
İsim nereden geliyor? Laplace’ın diferansiyel denkleminden.

> *“…it connects to Laplace’s finite difference equation.” — Strang, 24:38*

Bir kare-ızgara (grid) grafında iç düğümün derecesi 4’tür; L’nin tipik satırı: köşegende **4**, dört komşuda **–1**. Bu, ikinci türevlerin ayrık (finite difference) halidir:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \;\longrightarrow\; Lu = 0$$

İkinci x-türevi (−1, 2, −1) ile değişir, ikinci y-türevi (−1, 2, −1) ile; toplamı 5-nokta şablonu (köşegende 4, dört komşuda −1). Yani graf Laplacian'ı = **ayrık Laplace operatörü**.



Şekil 40.6: Graf Laplacian = ayrık Laplace operatörü: iç düğümde 5-nokta şablonu (4, dört −1) — D32 Kronecker toplamının graf hali. Sol: 3×3 ızgara grafında iç (merkez) düğümün derecesi 4, dört komşusuna −1. Orta: ızgara graf Laplacian'ı L_g (9×9). Sağ: Kron(A, I) + Kron(I, A) (D32). Köşegen-dışı BİREBİR (fark 0.0); fark yalnız sınır köşegeninde (graf derecesi 2–3 vs Dirichlet 4).

Şekil 40.6 graf Laplacian'ının ayrık Laplace operatörü olduğunu üç panelde gösterir. Solda 3×3 ızgara grafı: merkez (iç) düğüm derecesi **4** (turuncu), dört komşusuna **−1** etiketli — tam 5-nokta şablonu. Ortada bu grafın Laplacian'ı L_g (9×9 heatmap), sağda Ders 32'nin Kronecker toplamı Kron(A, I) + Kron(I, A). İki matris **köşegen-dışında BİREBİR** aynıdır — fark **0.0** (motor tanığı); iç düğüm satırı her ikisinde de aynı (köşegende 4, dört komşuda -1) şablonunu taşır. Tek fark sınır köşegenindedir: graf Laplacian'ı sınır düğümün gerçek derecesini (2 veya 3) köşegene yazarken, Dirichlet ayrık-Laplace (Kron toplamı) köşegeni hep 4 tutar. Yani köprü dürüst-etiketlidir — gövde (köşegen-dışı + iç düğümler) tıpatıp aynı, ayrılık yalnızca sınır düzeltmesinde.

> Builder Notu — Izgarada Laplace ile Buluşma
>
> "Graf Laplacian'ı = ayrık Laplace operatörü (−1,2,−1)" sürekli ile ayrık dünyanın buluştuğu yer. ML köprüsü: bu bağlantı graf üzerinde difüzyon (heat kernel), graf sinyal işleme ve fizik-bilgili ağların (PINN) PDE çözücüleriyle akrabadır. Ders 32'deki 2B Laplacian = Kron(A,I)+Kron(I,A) tam bu 5-nokta şablonun Kronecker halidir — burada gördüğümüz gibi, köşegen-dışı birebir, fark yalnız sınırda.

40.8 7. Kümeleri Fiedler İşaretlerinden Bulmak

Son adım: Fiedler vektörü kümeleri nasıl verir?

> *"…the positive components of x… and there are negative components… And those are the two clusters." — Strang, 28:46*

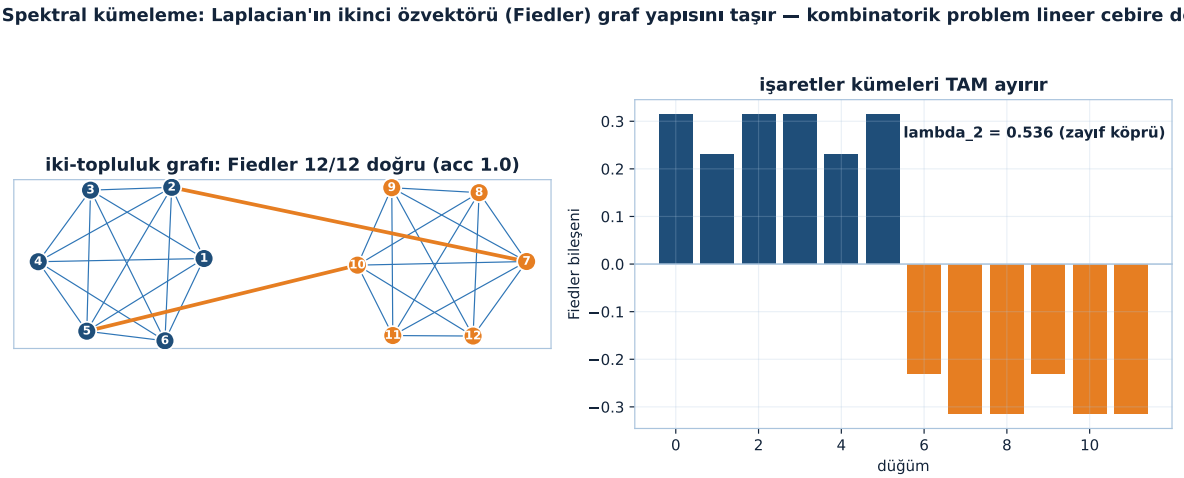
Fiedler vektörünün **pozitif** bileşenleri bir kümeye, **negatif** bileşenleri öteki kümeye gider. Her düğümün Fiedler bileşeninin işaretine bakarsın — artılar A, eksiler B.

Neden mantıklı? Fiedler vektörü, sabit $(1, \ldots, 1)$ özvektörüne **ortogonaldir** (simetrik matrisin farklı özdeğerli özvektörleri ortogonaldir):

> *"…to be orthogonal to this guy means that your components add to 0." — Strang, 30:46*

$$x_2 \perp \mathbf{1} \;\Rightarrow\; \sum_i (x_2)_i = 0$$

Yani pozitif ve negatif bileşenler aynı toplam büyüklüğe sahiptir → doğal olarak ~dengeli iki küme. Daha çok küme istersen ilk k özvektörü kullanırsın ($\lambda_1 = 0$'ı atlayıp sonrakileri).



Şekil 40.7: Spektral kümeleme (iki-topluluk grafı): Fiedler vektörünün işaretleri 12/12 düğümü doğru kümeler (acc 1.0); zayıf köprülerle λ_2 = 0.536, köprüsüz iki bileşende λ_2 = 0 — Laplacian'ın ikinci özvektörü graf yapısını taşır, kombinatorik kümeleme problemi lineer cebire döner.

Şekil 40.7 dersin **flagship** sonucudur: Fiedler işaretleri gerçek kümeleri tam ayırır. Solda iki-topluluk grafı — iki yoğun (tam-bağlı) altıdüğüm bloğu, aralarında iki zayıf köprü kenarı (turuncu kalın); düğüm renkleri Fiedler işaretine göredir (pozitif navy / negatif turuncu). On iki düğümün hepsi doğru kümeye düşer: **acc = 1.0**, yani $12/12$ düğüm doğru sınıflanır. Sağda Fiedler bileşenleri bar olarak çizilir; işaretler temiz biçimde iki gruba ayrılır — ilk altı düğüm bir işarette, son altı düğüm tersinde, sıfır çizgisinin iki yanında. Köprüler zayıf olduğundan $\lambda_2 = \mathbf{0.536}$ küçük ama pozitiftir; köprüler tümüyle kesilse graf iki bileşene düşer ve $\lambda_2 = 0$ olurdu (önceki Şekil 40.5'nin sonucu). Görünüşte kombinatoryal olan "hangi düğüm hangi tarafta" sorusu, tek bir özvektörün işaretine indirgenir.

Builder Notu — İşaretler Kümeleri Söyler

"Kümeler = Fiedler bileşen işaretleri" zarif bir hile. ML köprüsü: bu, spektral kümelemenin özüdür — Laplacian'ın düşük özvektörleri düğümleri düşük-boyutlu bir uzaya gömer, orada işaret (ya da k-means) kümeleri ayırır. scikit-learn `SpectralClustering` tam bunu yapar; görüntü segmentasyonu (normalized cut) ve topluluk tespitinde standarttır. $\perp(1,\ldots,1)$ koşulu dengeli kesimi otomatik teşvik eder.

40.9 Bu Dersin Özeti

- **Graf kümeleme:** grafı 2 dengeli kümeye böl; min $\sum \|a_i - x\|^2 + \sum \|b_i - y\|^2$ ($|A| \approx |B|$).
- **k-means (k=2):** centroid hesapla $\leftrightarrow$ en yakına yeniden ata, yakınsayana dek alternasyon (Lloyd algoritması).
- **Graf Laplacian:** $L = A^T A = D - B$ (incidence/degree/adjacency); simetrik PSD. Dört anahtar graf matrisi.
- **λ_1=0:** $L\mathbf{1} = 0$, özvektör sabit $(1, \dots, 1)$; Fiedler = en küçük pozitif özdeğerin (λ_2) özvektörü.
- **Spektral kümeleme:** Fiedler bileşen işaretleri iki kümeyi verir ($\perp (1, \dots, 1) \rightarrow$ dengeli). Daha çok küme için ilk k özvektör.
- **Neden Laplacian:** grid'de 5-nokta şablon (köşegen 4, dört −1) = ayrık Laplace operatörü.

! Tek Bir Cümle

Bir grafı iki kümeye ayırmak için ya k-means (centroid alternasyonu) ya da spektral kümeleme kullanılır; spektral yöntem graf Laplacian'ı $L = D - B = A^T A$'nın Fiedler vektörünü ($\lambda_1 = 0$'dan sonraki en küçük pozitif özdeğer) hesaplayıp düğümleri bu vektörün bileşen işaretlerine göre ($\perp (1, \dots, 1)$ olduğundan dengeli) böler.

40.10 Kontrol Soruları

i Soru 1 — Graf kümeleme problemi nedir; denge koşulu neden gerekir

Grafın düğümlerini iki kümeye (A, B) ayırıp toplam uzaklık-karesini en aza indirmek: min $\sum_{a_i} \|a_i - x\|^2 + \sum_{b_i} \|b_i - y\|^2$, burada x ve y küme merkezleri. Denge koşulu ($|A| \approx |B|$) gereklidir çünkü olmazsa çözüm "tek bir düğüm A, geri kalan hepsi B" gibi dejenere bir kesime kayar — bu kümeleme açısından değersizdir.

i Soru 2 — k-means'in iki adımı nedir; ne zaman durur

Adım 1: kümeler verili → her kümenin centroid'ini (noktaların ortalaması) merkez olarak hesapla. Adım 2: centroid'ler verili → her düğümü en yakın centroid'in kümesine yeniden ata. İki adım dönüşümlü tekrarlanır; kümeler bir turda değişmezse algoritma yakınsamıştır (durur). Yerel minimuma takılabilir ve başlangıca duyarlıdır.

i Soru 3 — Graf Laplacian'ı nedir; neden λ_1=0

$L = A^T A = D - B$ — A incidence, D derece (köşegen), B komşuluk matrisi; L simetrik ve pozitif yarı-tanımlıdır. $\lambda_1 = 0$'dır çünkü tüm-birler vektörü $(1, \dots, 1)$ her zaman $L\mathbf{1} = 0$ sağlar: L'nin her satırı, o düğümün derecesi eksi komşu sayısı = 0'a toplanır. Sabit vektör sıfır özdeğerli özvektördür (bağlı grafta boş-uzay boyutu 1).

> **Soru 4 — Fiedler vektörü kümeleri nasıl verir; sonuç neden dengeli olur**
>
> Fiedler vektörü = en küçük **pozitif** özdeğerin (λ_2) özvektörü. Bileşenlerinin işaretine bakarsın: pozitif bileşenli düğümler bir küme, negatif bileşenliler öteki küme. Sonuç dengeye yatkındır çünkü Fiedler vektörü sabit $(1, \ldots, 1)$ özvektörüne ortogonaldir → bileşenleri 0'a toplanır ($\sum_i (x_2)_i = 0$), yani pozitif ve negatif kütleler eşittir.

40.11 Egzersizler

1. **Küçük Laplacian.** 4 düğümlü graf; kenarlar (1-2), (1-3), (2-3), (2-4), (3-4). Derece matrisi D, komşuluk matrisi B ve Laplacian $L = D - B$'yi yaz. L'nin her satır toplamının 0 olduğunu doğrula. (Motor tanığı: $D = \text{diag}(2, 3, 3, 2)$; $L = A^T A$ **birebir** (incidence kimliği); satır toplamları 0; spektrum $\lambda(L) = (0, 2, 4, 4)$.)
2. **λ_1=0.** Yukarıdaki L için $(1, 1, 1, 1)$ vektörünün $L\mathbf{1} = 0$ sağladığını göster. Bu neden her grafta olur (satır toplamı argümanı)? (Motor tanığı: $L\mathbf{1} = 0$; $\lambda_1 \approx 0$, $\lambda_2 = 2 > 0$ — graf bağlı.)
3. **k-means elle.** Doğru üzerindeki noktalar $\{1, 2, 9, 10\}$. Başlangıç centroidleri $x = 1$, $y = 10$. Bir tam k-means turu çalıştır (önce atama, sonra centroid güncelleme); ortaya çıkan iki küme ve yeni centroidler ne olur? (Motor tanığı: atama → kümeler $\{1, 2\}$ ve $\{9, 10\}$; yeni centroidler **1.5** ve **9.5**; tek turda yakınsadı.)
4. **Fiedler işareti.** Bir yol-grafı 1–2–3–4'ün Fiedler vektörü kabaca $(-, -, +, +)$ işaretlidir. Hangi iki kümeyi önerir? Bu, grafın yapısına göre mantıklı mı? (Motor tanığı: işaretler $(+, +, -, -)$ (eşdeğer $-, -, +, +$) → kümeler $\{1, 2\}$ ve $\{3, 4\}$, yolu ortadan keser; Fiedler $\perp \mathbf{1}$ (toplam 0); $\lambda_2 = 2 - \sqrt{2} \approx$ **0.5858**.)
5. **(Ders 36 habercisi)** Backprop'u (Ders 27) bir kez daha göreceğiz — ama bu kez Julia diliyle, "lineer cebir olarak temiz". Edelman'ın yaklaşımı backprop'u neyi sadeleştirerek anlatabilir? Bir tahmin yaz — Ders 36 (son ders) Alan Edelman + Julia dilini (otomatik türev) işliyor.

40.12 Sonraki Ders

Ders 36: Alan Edelman ve Julia Dili (SON DERS). Konuk Prof. Alan Edelman, backpropagation'a (Ders 27) Julia diliyle yeni ve temiz bir bakış sunar — otomatik türevin lineer cebir olarak açık ifadesi. Kursun kapanışı; Phase 2'nin "aynı yere giden yollar" izleğinin (Karpathy micrograd + fast.ai + NYU + Strang matris-zinciri) olası beşinci bakışı.

> **Hazırlık**
>
> Bu dersin Egzersiz 5'inin sorduğu soruyu zihninde tut: backprop'u (Ders 27) Julia diliyle, "lineer cebir olarak temiz" yeniden görsek neyi sadeleşir? Ders 36 Alan Edelman + Julia dilini (otomatik türev) işler — kursu kapatan SON derstir. Bu derste graf yapısını özvektöre çevirdiğimiz gibi, orada da türev zincirini matris çarpımına çeviren bir bakış göreceğiz.

40.13 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Strang (dk)
Graf kümeleme	$\min \sum\|a_i - x\|^2 + \sum\|b_i - y\|^2$ ($\|A\| \approx \|B\|$)	2m02
k-means (k=2)	centroid $\leftrightarrow$ en yakına ata, alternasyon	6m07
Graf Laplacian	$L = A^T A = D - B$ (simetrik PSD)	16m25
$\lambda_1 = 0$	$L\mathbf{1} = 0$, özvektör $(1, \dots, 1)$	20m32
Fiedler vektörü	λ_2 özvektörü (en küçük pozitif özdeğer)	22m38
Spektral kümeleme	Fiedler bileşen işaretleri $\rightarrow$ 2 küme	28m46

40.14 ML Bağlantıları Özeti

- **Graf Laplacian:** GCN/GNN'in çekirdeği (message passing = Laplacian çarpımı); spektral graf teorisinin temeli.
- **Spektral kümeleme:** scikit-learn `SpectralClustering`, normalized cut (görüntü segmentasyonu), topluluk tespiti.
- **k-means:** en temel denetimsiz öğrenme (Lloyd); vektör nicemleme, müşteri segmentasyonu; EM'in sert-atama hali.
- **Fiedler / λ_2:** cebirsel bağlantısallık; Laplacian eigenmaps (manifold öğrenme); graf Fourier tabanı (Ders 31 ile akraba).
- **Laplacian = ayrık Laplace:** heat kernel/difüzyon, fizik-bilgili ağlar (PINN), Ders 32'nin Kronecker 2B Laplacian'ı.
- **Geriye köprü:** Ders 5-6 (PSD/SVD), Ders 19 (özvektör/maxmin), Ders 30-32 (sirkülant/Fourier/Kronecker). Paralel: NYU H13 spektral GCN, fast.ai embedding.

! Kapanış

> *"…those two sets of components are your… two clusters in spectral clustering." — Strang, 30:46*

Bir grafı bölmek görünüşte kombinatoryal bir problemdir; spektral kümeleme onu lineer cebire çevirir — Laplacian'ın ikinci özvektörünün (Fiedler) işaretleri kümeyi verir. Graf teorisiyle özdeğerlerin buluştuğu, Phase 2'nin sinyal/CNN/graf bloğunu kapatan nokta.

41 Alan Edelman ve Julia Dili — Son Ders

Otomatik türevin üçüncü yolu, dual sayılar ve backprop'un beşinci tanığı: alt-üçgen çözüm

> **Bölüm bilgisi**
>
> Kursun **son dersi** ve konuk dersi: konuşmacı **Prof. Alan Edelman** (Julia dilinin yaratıcılarından), Strang ev sahibidir (yalnız kapanış vedasında konuşur). Edelman, backpropagation'a (Ders 27) Julia diliyle yeni ve temiz bir bakış sunar: otomatik türev (AD) ne sembolik ne sayısaldır; forward-mode'da dual sayılarla, reverse-mode'da $(I-L)^{-1}$ alt-üçgen çözümle çalışır. Konuk: Alan Edelman Ders 36 videosu (≈38 dk) ve OCW Lecture 36 temel alınmıştır. Okuma süresi ≈28 dk. Önkoşul Ders 27 (backprop = ters-mod AD, dört tanık) ve Ders 21 (Newton/Babylonian karekök). Bu ders Phase 2'nin "aynı yere giden yollar" izleğini backprop'un **beşinci** tanığıyla kapatır.

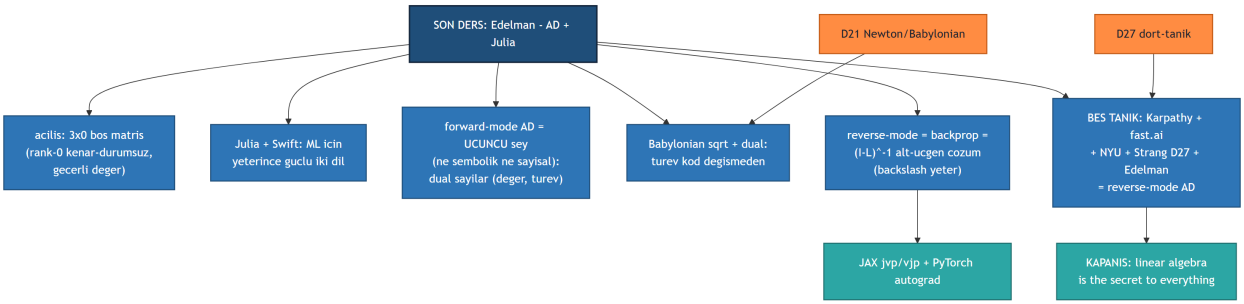
41.1 Bu Derste Ne Var?

Kursun **son dersi** — konuk Prof. **Alan Edelman** (Julia dilinin yaratıcılarından). Ana mesaj: **otomatik türev** (automatic differentiation, AD) makine öğrenmesinin iş atıdır ve **lineer cebir her şeyin sırrıdır**. Backprop'a Phase 2'nin **beşinci** bakışı burada.

Altı sonuç:

1. **Açılış jimnastiği:** satır rankı = kolon rankı ispatı sıfır matriste de çalışır (3×0 boş matris kavramı).
2. **Julia neden?** Google'ın blogu: ML için yeterince güçlü iki dil — **Julia ve Swift** (Python/Java/C++/Rust elendi). "Dil matematiksel anlamda önemlidir."
3. **Forward-mode AD = üçüncü bir şey:** ne sembolik türev (kalkülüs tablosu) ne sayısal türev (sonlu fark). Üçüncü bir şey: **dual sayılar** $D = (\text{deger},\ \text{turev})$.
4. **Dual sayılar + 8 satır Julia:** tip tanımı + toplam/bölme kuralı (operator overloading). Babylonian $\sqrt{x}$ algoritmasını dual sayıya uygula — türev **kod yeniden yazılmadan** kendiliğinden çıkar.
5. **Reverse-mode AD = backprop = $(I-L)^{-1}$ üçgen çözüm:** skaler sinir ağının türevini lineer cebire yaz $\rightarrow dx = (I-L)^{-1} D\, dP$. Backprop aslında bir **alt-üçgen sistem çözümü**; "backslash zaten yapar".
6. **Beşinci tanık:** Edelman'ın $(I-L)^{-1}$ bakışı, Ders 27'nin dört tanığına (Karpathy micrograd + fast.ai + NYU Jacobian + Strang matris-zinciri) eklenir.

Şekil 41.1 son dersin iskeletini gösterir: merkezde Edelman'ın **AD + Julia** dersi, etrafında altı dal. Açılış 3×0 boş matrisle (rank-0 kenar-durum değil, geçerli değer) başlar; Julia + Swift ML için yeterince güçlü iki dildir; forward-mode AD üçüncü bir şeydir (dual sayılar, değer-türev çifti); Babylonian karekök + dual sayı türevi kod değişmeden verir; reverse-mode = backprop = $(I-L)^{-1}$ alt-üçgen çözümdür; ve beş tanık (Karpathy +



Şekil 41.1: Ders 36 kavram haritasi (KURSUN SON DERSI, konuk Alan Edelman): merkez Edelman in AD + Julia dersi. Acilis 3x0 bos matris (rank-0 kenar-durum DEGIL gecerli deger). Julia + Swift ML icin yeterince guclu iki dil. Forward-mode AD ucuncu sey (ne sembolik ne sayisal): dual sayilar (deger, turev). Babylonian sqrt + dual: turev kod degismeden cikar. Reverse-mode = backprop = (I-L)^-1 alt-ucgen cozum (backslash yeter). BES TANIK: Karpathy + fast.ai + NYU + Strang D27 + Edelman = ayni reverse-mode AD. Koprular: D27 dort-tanik, D21 Newton/Babylonian, JAX jvp/vjp + PyTorch autograd. Kapanis: linear algebra is the secret to everything.

fast.ai + NYU + Strang D27 + Edelman) aynı reverse-mode AD'yi söyler. Köprü düğümleri dersi öncesine ve sonrasına bağlar: D27 dört-tanık, D21 Newton/Babylonian, JAX jvp/vjp + PyTorch autograd ve kursu kapatan motto — "linear algebra is the secret to everything".

Builder Notu — Son Sözü Konuk Söyler

- **AD = üçüncü şey:** dual sayılar otomatik türevin kalbi (forward-mode); PyTorch autograd / JAX bunun (çoğunlukla reverse-mode) endüstri hali. Sembolik değil (şişme yok), sayısal değil (h seçimi/yuvarlama yok).
- **Backprop = üçgen çözüm:** $(I - L)^{-1}$ geri-yerine-koyma (back-substitution) = backprop; lineer cebir "geri"yi zaten içerir, tekerleği yeniden icat etme.
- **Julia:** tip + multiple dispatch ile "kodu yeniden yazmadan" türev (sıfır-maliyet soyutlama); kısa assembler = hızlı kod.
- **Geriye köprü:** Ders 27 (backprop/reverse-mode AD — dört tanık), Ders 21 (Newton/Babylonian $\sqrt{}$), Ders 6 (SVD), Ders 5 (PSD). Paralel: Karpathy micrograd `backward()`, fast.ai manuel backprop, NYU vJp.

Bu kursun son dersi bir konuk dersidir; Strang sınıfı Edelman'a bırakır ve yalnız sonda veda eder. Okurken her bölümün sahibinin Edelman olduğunu, Strang'ın yalnız ev sahibi olduğunu unutma. Asıl ödül §6-7'de: backprop aslında lineer cebirin zaten bildiği bir alt-üçgen çözümdür, ve bu Phase 2'nin dört tanığına eklenen beşinci bakıştır.

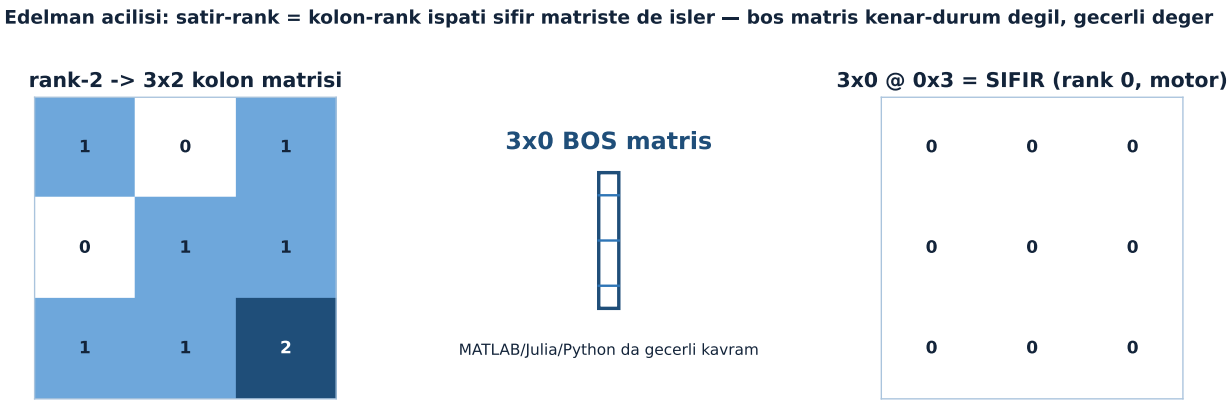
41.2 1. Açılış: Sıfır Matris ve Rank

Edelman derse Strang'ın 18.06'daki bir ispatıyla başlıyor: satır rankı = kolon rankı. Strang sınıftan çıktıktan sonra Edelman'ın aklına takılan soru: bu ispat **sıfır matris** için de işler mi?

> *"…would that work for the zero matrix?" — Edelman, 0:00*

Bir matrisin lineer bağımsız kolonlarını ayrı bir matrise koyarsan: tam-rank 3×3 → 3×3; rank-2 → 3×2; rank-1 → 3×1; **rank-0 (sıfır matris) → 3×0 boş matris**. Sonra 3×0 çarpı 0×3 = 3×3 sıfır matrisi verir. İspat hiçbir özel kenar-durum gerektirmeden işler.

> *"…a 3 by 0 empty matrix. And that's a concept that exists in MATLAB, and in Julia, and in Python…" — Edelman, 2:04*



Şekil 41.2: Edelman açılışı: satır-rank = kolon-rank ispatı sıfır matriste de işler — boş matris kenar-durum değil, geçerli değer (motor: 3×0 @ 0×3 = 3×3 sıfır, rank 0).

Şekil 41.2 Edelman'ın açılış jimnastiğini üç panelde gösterir. Solda rank-2 bir örnek matris: bağımsız kolonlarını ayrı bir matrise koyduğunda 3×2 bir kolon matrisi çıkar. Ortada asıl mesele — **3×0 boş matris**: rank-0 (sıfır matris) durumunun temiz ifadesi, MATLAB/Julia/Python'da geçerli bir kavram. Sağda motor tanığı: bu 3×0 boş matrisi 0×3 ile çarpınca **3×3 sıfır matris** çıkar, rank **0**. Satır-rank = kolon-rank ispatı hiçbir özel kenar-durum gerektirmeden, sıfır matriste de işler — iyi bir tip sistemi sıfırı bir istisna değil, geçerli bir değer sayar.

> **Builder Notu — Sıfırın da Hakkı Var**
>
> "Boş matris (3×0) = rank-0 durumunun temiz ifadesi" kenar-durum disiplinidir. ML köprüsü: boyut-tutarlı boş tensörler (PyTorch `torch.empty(3,0)`) batch/maskeleme kodunda kenar durumlarını özel-kılıf olmadan halleder — iyi bir tip sistemi sıfırı bir istisna değil, geçerli bir değer sayar.

41.3 2. Julia Neden Önemli?

Edelman'ın ilk tezi: **dil matematiksel anlamda önemlidir** — doğru dil, bir algoritmayı tahtadan koda çevirmekten fazlasını yapabilir.

> *"…the language matters in a mathematical sense." — Edelman, 8:13*

Kanıt olarak Google'ın blogunu gösteriyor: ML için yeterince güçlü dilleri eleye eleye — teknik yeterlilikte Python ve Java düşer; kullanılabilirlikte C++ ve Rust düşer — geriye **Julia ve Swift** kalır.

> *"…there really are two languages that are powerful enough to do machine learning…" — Edelman, 4:08*

İkinci tezi (ve tüm kursun ruhu): **lineer cebir her şeyin temelidir** — "her ders lineer cebirle başlamalı".

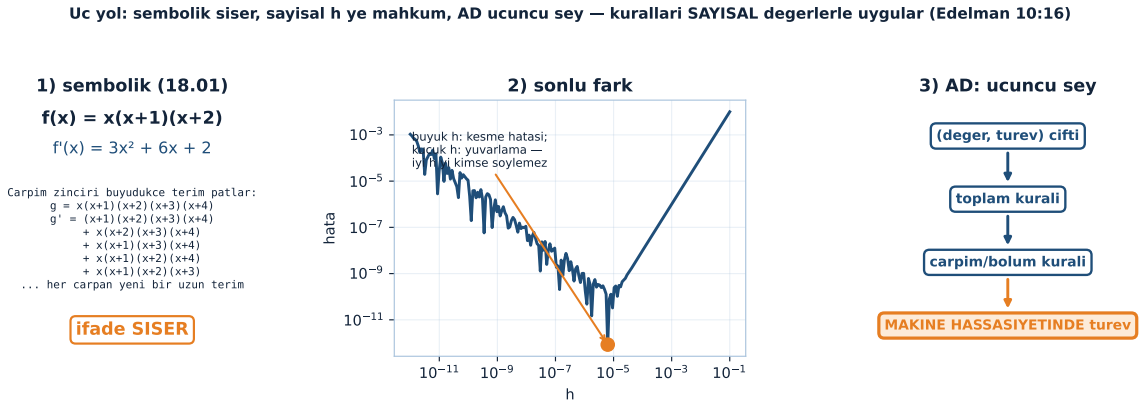
> **Builder Notu — Dil Matematiği Taşır**
>
> "Doğru dil matematiği taşır" performans + ifade gücü argümanı. ML köprüsü: JAX (XLA derleme), Julia (LLVM + multiple dispatch), PyTorch 2.0 (`torch.compile`) hep "yüksek-seviye matematik + düşük-seviye hız" peşindedir. Edelman'ın asıl noktası: tip sistemi ve dispatch, türev/Jacobian gibi şeyleri kodu yeniden yazmadan verir — bu bir dil özelliğidir, eklenti kütüphane değil.

41.4 3. Forward-Mode AD: Üçüncü Bir Şey

Edelman'ın itirafı: yıllarca otomatik türevi önemsememiş — "kalkülüsü bilgisayara öğretmek" sanmış. Meğer AD **üçüncü bir şeymiş** — ne biri ne öteki:

> *"…automatic differentiation was neither the first nor the second thing… there's actually a third thing." — Edelman, 10:16*

- **Birinci (sembolik):** kalkülüs tablosu (zincir/çarpım/bölüm kuralı), 18.01'in sembolik türevi. Şişer — devasa ifadeler, büyük katsayılar.
- **İkinci (sayısal):** sonlu fark, dy/dx ≈ (f(x+h)−f(x))/h. h çok büyükse kesme (truncation) hatası, çok küçükse yuvarlama (round-off) hatası; "iyi h"yi kimse söylemez.
- **Üçüncü (AD):** ne sembolik ne sayısal. Her adımda toplam/bölüm kuralını **sayısal değerlerle** uygular — sembolik şişme yok, h seçimi yok.



Şekil 41.3: Üç yol: sembolik şişer, sayısal h'ye mahkûm, AD üçüncü şey — kuralları SAYISAL değerlerle uygular (Edelman 10:16). Sonlu farkta minimum civarı: büyük h kesme, küçük h yuvarlama hatası.

Şekil 41.3 türev almanın üç yolunu yan yana koyar. Solda **sembolik** (18.01): $f(x) = x(x+1)(x+2)$ türevi küçük çarpanlarda temiz görünür, ama çarpım zinciri büyüdükçe her çarpan yeni bir uzun terim doğurur — "ifade SİŞER". Ortada **sayısal** (sonlu fark): x^3'ün türev hatasının h'ye göre loglog eğrisi U biçimlidir; büyük h'de kesme, küçük h'de yuvarlama hatası baskındır, en iyi h'yi kimse önceden söyleyemez. Sağda **AD = üçüncü şey**: (değer, türev) çiftiyle başlayıp toplam, çarpım/bölüm kurallarını **sayısal değerlerle** uygulayarak makine hassasiyetinde türev verir — ne sembolik şişme ne h seçimi.

Builder Notu — Ne Sembolik Ne Sayısal

"AD = üçüncü yol (ne sembolik ne sonlu-fark)" derin öğrenmenin temelini anlamanın anahtarı. ML köprüsü: bu, "neden PyTorch/JAX sonlu fark kullanmaz" sorusunun cevabı — AD makine hassasiyetinde **tam** türev verir (sonlu farkın yaklaşıklık hatası yok) ve sembolik patlama olmadan. Edelman: ML = büyük bir optimizasyon = türev al; AD bu işin iş atı.

41.5 4. Dual Sayılar: 8 Satır Julia

Nasıl? **Dual sayılar** ile. Bir dual sayı, bir (değer, türev) çiftidir — bir asır öncesine giden bir kavram:

$$D = (f,\ f')$$

8 satır Julia: 3 satır D tipini tanımlar (bir çift float; "sayı" gibi davranır), 5 satır **toplam kuralı** ile **bölüm kuralını** operatör aşırı-yüklemeyle (operator overloading) öğretir:

$$(f, f') + (g, g') = (f + g,\ f' + g')$$

$$\frac{(f, f')}{(g, g')} = \left(\frac{f}{g},\ \frac{gf' - fg'}{g^2}\right)$$

Şimdi sihir: Babylonian karekök algoritmasını (Newton: $t_{k+1} = \frac{1}{2}(t_k + x/t_k)$) **dual sayıya** uygula. Kodu hiç değiştirmeden türev **kendiliğinden** çıkar:

> *"…I'm getting magically the right answer without ever [typing the derivative]." — Edelman, 18:21*

Hiç $\frac{1}{2}x^{-1/2}$ yazmadı, hiç sonlu fark almadı — yine de $\sqrt{x}$'in türevini ($\frac{1}{2}/\sqrt{x}$) tam buldu. Bu sonucu (Şekil 41.4) bir sonraki bölümde işin nasıl yürüdüğünü gösterdikten sonra göreceğiz.

Builder Notu — Sekiz Satırlık Sihir

"Dual sayı = (değer, türev) çifti + operatör aşırı-yükleme" forward-mode AD'nin tüm sırrı. ML köprüsü: bu, JAX'in `jvp` (Jacobian-vektör çarpımı) ve forward-mode autodiff'inin çekirdeği; ileri geçişte her sayının yanında türevini taşırsın. ε^2=0 olan ikili sayı cebiriyle (a+bε) aynı: bir f fonksiyonu için f(a+bε) = f(a) + b·f'(a)·ε. Kodu yeniden yazmadan türev — bilgisayar cebiri zarafeti.

41.6 5. Forward-Mode Nasıl Çalışır?

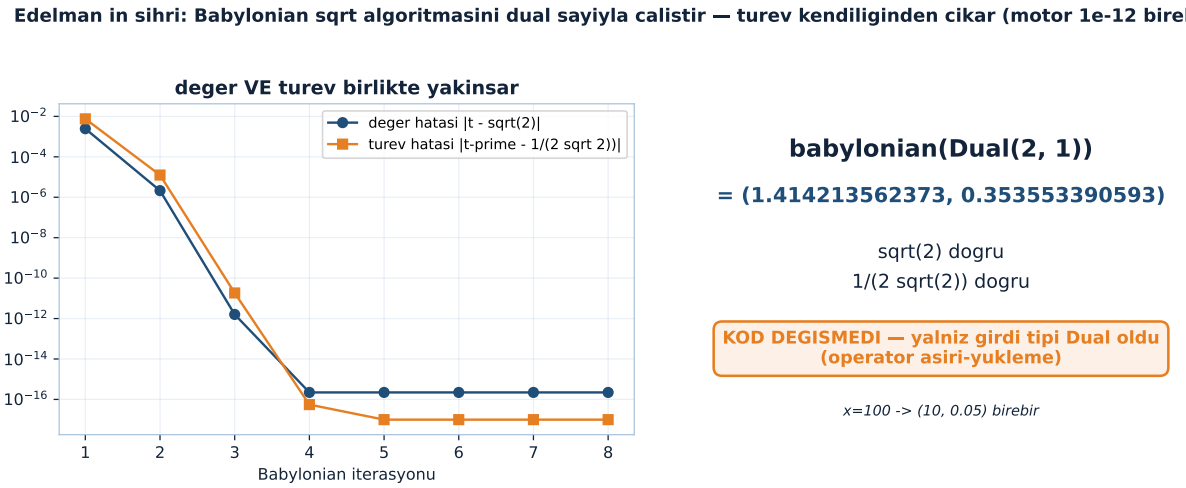
Sihir nasıl oluyor? **Kodun her satırının türevini al.** Babylonian algoritmasının her satırının x'e göre türevini yazsan (toplam/bölüm kuralıyla), bu türev algoritması — yalnız +, −, ×, ÷ kullanarak — gerçek türeve yakınsar:

> *"…if you just take the derivatives back to your variable of every line of your code, then you can get a derivative out." — Edelman, 22:27*

Bu ne sembolik (18.01) ne sayısaldır (sonlu fark) — her adımda kuralları **sayısal değerlerle** uygular. Elle yazmak yerine: dual sayı tipi + operatör aşırı-yükleme, derleyicinin (Julia'nın) bunu otomatik yapmasını sağlar. Eski Fortran'da kaynak-kaynak çevirici (source-to-source) vardı; Julia'da tip + dispatch yeter.

> *"…you just give the rules and you let the computer do it." — Edelman, 24:29*

Bonus: dual sayıda Babylonian'ın **assembler'ı kısadır** = hızlı kod (Python'da "ekranlar dolusu" olurdu).



Şekil 41.4: Edelman'ın sihri: Babylonian sqrt algoritmasını dual sayıyla çalıştır — türev kendiliğinden çıkar (motor 1e-12 birebir). Sol: iterasyon arttıkça değer hatası ve türev hatası BİRLİKTE makine hassasiyetine düşer. Sağ: girdi tipi `Dual` olunca aynı kod hem √2 = 1.414213562373 hem türev 1/(2√2) = 0.353553390593 üretir — algoritma satırı hiç değişmedi.

Şekil 41.4 bu dersin **flagship** sonucudur: Babylonian karekök algoritmasını dual sayıyla çalıştırınca türev kendiliğinden çıkar. Solda değer hatası (navy daire) ve türev hatası (turuncu kare) iterasyon arttıkça **birlikte** makine hassasiyetine düşer — yani algoritma hem $\sqrt{x}$'e hem türevine aynı anda yakınsar. Sağda sonuç kartı: `babylonian(Dual(2, 1))` çağrısı $(1.414213562373,\ 0.353553390593)$ verir — ilk bileşen $\sqrt{2}$, ikinci bileşen $1/(2\sqrt{2})$, ikisi de doğru. Anahtar nokta turuncu kutuda: **KOD DEĞİŞMEDİ** — yalnız girdi tipi `Dual` oldu (operatör aşırı-yükleme). $x = 100$ girdisi de aynı kodla $(10, 0.05)$ verir (motor 1e-12 birebir); algoritma satırına hiç dokunulmadı, yalnız sayının tipi değişti.

> **Builder Notu — Her Satırın Türevini Taşı**
>
> "Forward-mode = her satırın türevini taşı, derleyici otomatikleştir" sıfır-maliyet soyutlamanın gösterisi. ML köprüsü: forward-mode AD, az girdi–çok çıktı (veya Jacobian-vektör çarpımı) durumunda verimlidir; çok-girdi (milyonlarca ağırlık, tek skaler kayıp) durumunda reverse-mode (backprop) tercih edilir — sıradaki bölüm. JAX ikisini de verir (`jvp` / `vjp`).

41.7 6. Reverse-Mode AD = Backprop = (I–L)$^{-1}$ Üçgen Çözüm

Edelman'ın final mücevheri: sinir ağı backprop'unun lineer-cebir özü. Skaler bir ağ düşün: ağırlık/bias w_i, b_i ve

$$x_{i+1} = h(w_i x_i + b_i), \qquad h = \text{ReLU} = \max(0, t)$$

sonunda kayıp $\frac{1}{2}(y - x_m)^2$. Anahtar satırın türevini al; $\delta_i = h'(w_i x_i + b_i)$ olsun:

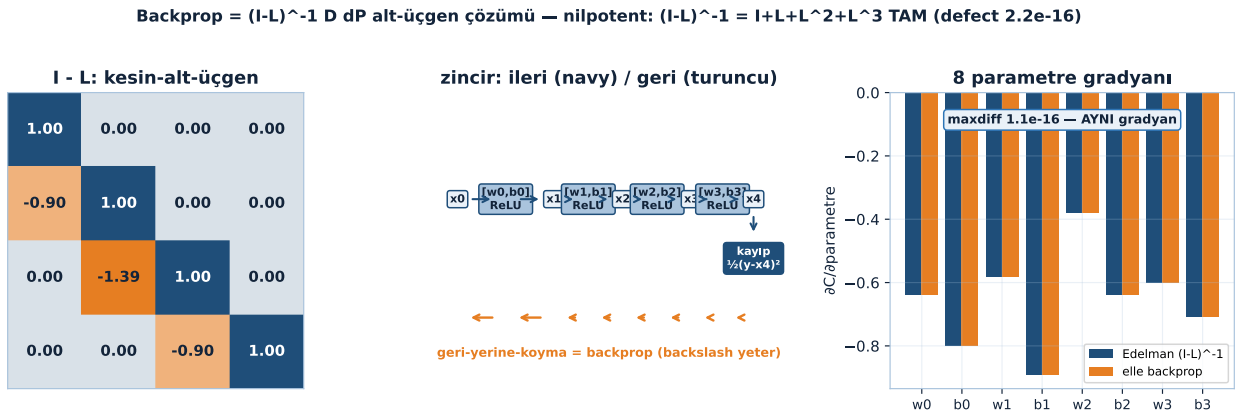
$$dx_{i+1} = \delta_i \left(x_i\, dw_i + w_i\, dx_i + db_i\right)$$

Tüm bu perturbasyonları **lineer cebir** olarak yaz. Sonuç: bir köşegen matris (D), bir alt-üçgen matris (L) ile

$$dx = D\,dP + L\,dx \implies (I - L)\,dx = D\,dP \implies dx = (I - L)^{-1} D\,dP$$

> *"…you don't need to do your own backpropagation. Because a simple backslash will do it for you." — Edelman, 36:34*

İşte can alıcı nokta: **(I–L) alt-üçgendir; onu çözmek = geri-yerine-koyma (back-substitution) = tam olarak backprop.** Lineer cebir "geri çözüm"ü zaten içerir; kendi backprop'unu elle yazmana gerek yok — bir \ (backslash) yeter. Yalnız son elemanın türevini istiyorsan e_n vektörüyle çekersin.



Şekil 41.5: Backprop = (I-L)^-1 D dP alt-üçgen çözümü: lineer cebir geri'yi zaten içerir — backslash yeter (Egz4: nilpotent, (I-L)^-1 = I+L+L^{2+L}3 TAM). Sol I-L kesin-alt-üçgen; orta zincir ileri (navy) / geri (turuncu); sağ Edelman (I-L)^-1 = elle backprop maxdiff 1.1e-16 — AYNI gradyan.

Şekil 41.5 dersin **ikinci flagship'i**: backprop'un $(I - L)^{-1} D\,dP$ alt-üçgen çözümü olduğunu üç panelde gösterir. Solda $I - L$ matrisi (4×4 heatmap, diverging renk) **kesin-alt-üçgendir** — köşegen 1, köşegen-altında katman bağımlılığı, üst üçgen sıfır; çünkü katman i'nin türevi yalnız önceki katmanlara bağlıdır. Ortada zincir şeması: $x_0 \to [w_0, b_0]\text{ReLU} \to x_1 \to \ldots \to x_4 \to$ kayıp; navy oklar ileri geçiş, alttaki turuncu oklar **geri-yerine-koyma = backprop (backslash yeter)**. Sağda asıl tanık: Edelman'ın $(I - L)^{-1}$ çözümü (navy) ile elle backprop (turuncu) 8 parametrenin her birinde **aynı gradyanı** verir — maxdiff **1.1e-16**, yani makine sıfırı. Nilpotent köşesi: $L^4 = 0$ olduğundan $(I - L)^{-1} = I + L + L^2 + L^3$ **tam** olur (defect 2.2e-16, Egz4).

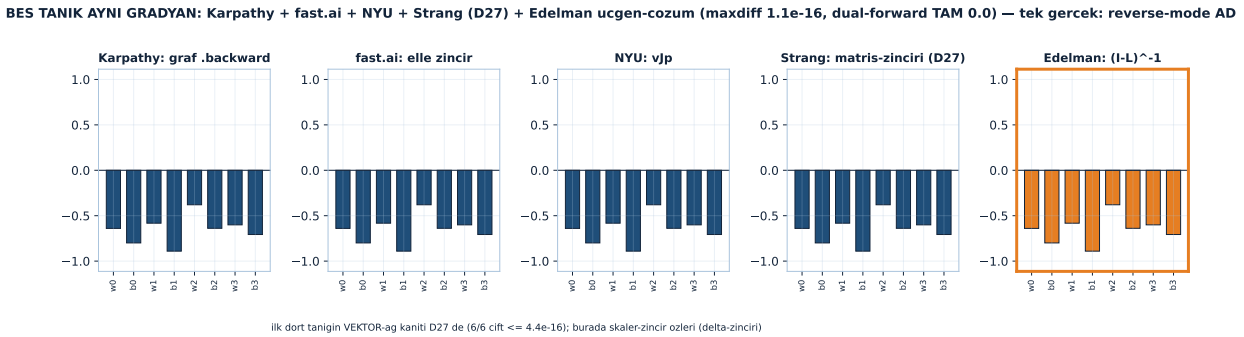
> **Builder Notu — Backprop Dediğin Üçgen Çözüm**
>
> "Backprop = $(I–L)^{-1}$ alt-üçgen çözüm" Edelman'ın devrimci yeniden-çerçevelemesi. ML köprüsü: zincir kuralının ardışık (katman katman) yapısı tam bir alt-üçgen bağımlılıktır; ileri geçiş L'yi kurar, geri geçiş $(I - L)^{-1}$'i back-substitution ile çözer. PyTorch/JAX'in reverse-mode autograd'ı tam bu üçgen çözümün otomatik halidir — Ders 27'nin matris-zinciri görüşünün en saf ifadesi.

41.8 7. Beşinci Tanık: Backprop'a Beş Bakış

Ders 27'de Phase 2'nin "aynı yere giden yollar" tezini kurmuştuk: backpropagation = reverse-mode otomatik türev, dört farklı kursta dört farklı dille anlatıldı. Edelman bugün **beşinci** bakışı ekliyor:

#	Tanık / Kurs	Backprop'a bakış
1	Karpathy — Zero-to-Hero	micrograd: hesap grafiğinde geri yayılım; her düğüm yerel gradyanı zincirler
2	fast.ai — Howard	elle backprop: zincir kuralını katman katman tek tek yaz
3	NYU — LeCun	Jacobian / vektör-Jacobian çarpımı (vJp)
4	Strang — 18.065 Ders 27	matris-zinciri kuralı: reverse mode (soldan çarpım) daha ucuz
5	Edelman — 18.065 Ders 36	backprop = $(I–L)^{-1}$ alt-üçgen sistem çözümü (back-substitution)

Beşi de aynı matematiği söyler: **reverse-mode AD**. Beş bakış birlikte, "tek bir gerçeğin beş dili" — Phase 2'nin bütün omurgası.



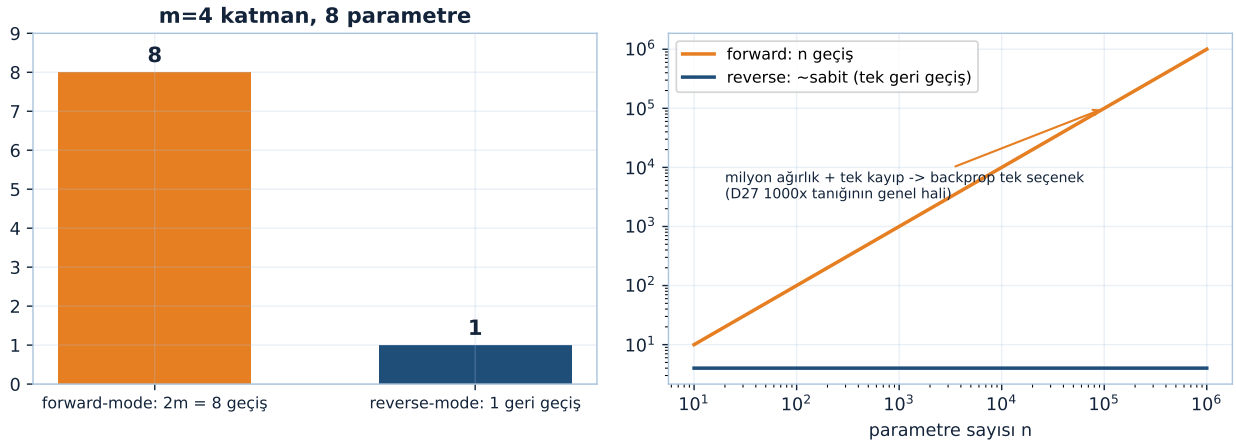
Şekil 41.6: BEŞ TANIK AYNI GRADYAN: Karpathy + fast.ai + NYU + Strang (D27) + Edelman üçgen-çözüm (maxdiff 1.1e-16, dual-forward TAM 0.0) — tek gerçek: reverse-mode AD. İlk dört tanığın vektör-ağ kanıtı D27'de (6/6 çift ≤ 4.4e-16); burada skaler-zincir özleri.

Şekil 41.6 kursun **sentez flagship'i**: beş tanığın aynı 8-parametreli gradyanı, aynı ölçekte beş panel. Karpathy (graf `.backward`), fast.ai (elle zincir), NYU (vJp) ve Strang (matris-zinciri, D27) — bu dördü skaler özde

aynı δ-zincirini hesaplar (panel başlıkları her birinin pedagojisini taşır); beşinci panel, turuncu çerçeveyle vurgulanan **Edelman:** $(I - L)^{-1}$. Beş bar grafiği üst üste konsa fark görünmez: Edelman üçgen-çözümü = elle backprop **maxdiff 1.1e-16**, dual-forward ile **TAM 0.0**. Dürüstlük notu altta: ilk dört tanığın vektör-ağ kanıtı Ders 27'de verildi (6/6 çift ≤ 4.4e-16); burada gösterilen skaler-zincir özleridir. Beş dil, tek gerçek: **reverse-mode AD**.

> **Builder Notu — Beş Dil Tek Gerçek**
>
> "Beş tanık, tek gerçek: reverse-mode AD" Phase 2'nin sentez anıdır. ML köprüsü: bir kavramı beş farklı çerçeveden görmek (hesap grafiği, elle zincir, Jacobian, matris-zinciri, üçgen çözüm) onu "ezber" olmaktan çıkarıp "anlaşılmış" kılar — Karpathy'nin "kendin inşa et" felsefesinin özü. Edelman'ın katkısı belki en derini: backprop yeni bir algoritma değil, lineer cebirin zaten bildiği bir alt-üçgen çözümdür.

Şekil 41.7 ise maliyet dersini kapatır (Egz3): m=4 katmanlı skaler ağda forward-mode dual türev her parametre için ayrı geçiş ister (2m=8 geçiş, turuncu), reverse-mode tek geri geçişle yeter (navy). Sağda parametre sayısı n büyüdükçe forward maliyeti n ile doğrusal artar, reverse ~sabit kalır — milyon ağırlık + tek kayıpta backprop tek seçenektir (Ders 27'nin 1000× tanığının genel hali).



Şekil 41.7: Egz3: az girdi-çok çıktı -> forward; çok girdi-tek kayıp -> reverse. Derin öğrenme bu yüzden backprop kullanır. Sol: m=4 katmanlı zincirde forward-mode 2m=8 ayrı geçiş gerektirirken reverse-mode tek geri geçiş yeter. Sağ: parametre sayısı n büyüdükçe forward maliyeti n ile doğrusal artar, reverse ~sabit kalır — milyon ağırlık + tek kayıpta backprop tek seçenektir (D27'nin 1000x tanığının genel hali).

41.9 Bu Dersin Özeti

- **Açılış:** rank (satır=kolon) ispatı sıfır matriste de işler (3×0 boş matris).
- **Julia:** ML için iki güçlü dil (Julia + Swift); "dil matematiksel olarak önemli"; lineer cebir her şeyin temeli.

- **Forward-mode AD:** üçüncü şey (ne sembolik ne sayısal); dual sayılar D=(değer, türev), 8 satır Julia, türev kod yeniden yazılmadan çıkar.
- **Reverse-mode AD = backprop:** skaler ağ → $dx = (I - L)^{-1} D \cdot dP$; backprop = alt-üçgen çözüm (back-substitution), "backslash zaten yapar".
- **Beşinci tanık:** Edelman'ın $(I - L)^{-1}$ bakışı + Karpathy / fast.ai / NYU / Strang = backprop'a beş bakış, tek gerçek (reverse-mode AD).
- **Kapanış mottosu:** "linear algebra is the secret to everything."

! Tek Bir Cümle

Otomatik türev ne sembolik ne sayısaldır — forward-mode'da dual sayılarla, reverse-mode'da $(I-L)^{-1}$ alt-üçgen çözümle (= backprop) türevi kod yeniden yazılmadan üretir; ve Edelman'ın bu üçgen-çözüm bakışı, Karpathy / fast.ai / NYU / Strang'ın yanında backprop'a beşinci tanıktır — hepsinin söylediği tek gerçek: reverse-mode AD.

41.10 Kontrol Soruları

i Soru 1 — Otomatik türev neden 'üçüncü bir şey'dir; sembolik ve sayısaldan farkı ne

Sembolik türev (kalkülüs tablosu, 18.01) ifadeleri devasa büyütür (şişme). Sayısal türev (sonlu fark (f(x+h)−f(x))/h) h seçimine duyarlıdır — büyük h kesme hatası, küçük h yuvarlama hatası. AD ise üçüncü bir şeydir: her temel işlemde toplam/çarpım/bölüm kuralını **sayısal değerlerle** uygular. Sonuç makine hassasiyetinde tam türevdir — ne sembolik şişme ne de h kaynaklı yaklaşıklık hatası vardır.

i Soru 2 — Dual sayı nedir; √ türevi nasıl kod yeniden yazılmadan elde edilir

Dual sayı bir (değer, türev) çiftidir: D=(f, f). Operatörleri aşırı-yükleyerek (toplam: bileşen-bileşen; bölüm: (f/g, (gf −fg)/g^2)) bu çiftler sayı gibi davranır. Babylonian karekök algoritmasını sayılar yerine **dual sayılarla** çalıştırınca, her işlem değeri yanında türevini de taşır; algoritma sonunda (√x, ½/√x) çiftini verir. Kod hiç değişmedi — yalnız girdi tipi dual sayıya çevrildi.

i Soru 3 — Edelman'a göre backprop neyle aynıdır; (I−L) neden alt-üçgendir

Backprop, $(I - L)^{-1} D\, dP$ ile aynıdır — yani alt-üçgen bir $(I - L)$ sisteminin geri-yerine-koyma (back-substitution) çözümü. L alt-üçgendir çünkü ağdaki katman bağımlılığı ardışıktır: katman i'nin türevi yalnız **önceki** katmanlara bağlıdır (ileriye değil). Bu yüzden "geri çözüm" lineer cebirde zaten vardır; ayrı bir backprop kodu yazmaya gerek yoktur (bir backslash yeter).

i Soru 4 — 'Beş tanık' hangileridir; ortak gerçek nedir

(1) Karpathy micrograd (hesap grafiğinde .backward()), (2) fast.ai elle backprop (zincir kuralı katman katman), (3) NYU Jacobian / vektör-Jacobian çarpımı, (4) Strang matris-zinciri kuralı (Ders 27), (5) Edelman $(I - L)^{-1}$ alt-üçgen çözümü (Ders 36). Beşinin de söylediği tek gerçek: **reverse-mode otomatik türev**. Aynı matematik, beş farklı dil.

41.11 Egzersizler

1. **Dual sayı aritmetiği.** D=(değer, türev). (a) (3, 1) + (2, 5) toplamını bul. (b) (3, 1) · (2, 5) çarpımını, çarpım kuralı (fg, f g + fg) ile hesapla. (Burada (3,1), x=3 noktasında f=x, f =1 anlamına gelir.) (Motor tanığı: toplam $(3,1)+(2,5)=$ **(5,6)**; çarpım $(3,1)\cdot(2,5)=$ **(6,17)** (1·2+3·5=17); bölüm $(3,1)/(2,5)=$ **(1.5, –3.25)**.)
2. **Dual ile türev.** f(x)=x^2 fonksiyonunu dual sayıyla hesapla: girdi (3, 1) (yani x=3, dx=1). Çarpım kuralıyla x·x dual çarpımını yap; sonucun (9, 6) çıktığını ve 6 = $2x|_{x\ 3}$ olduğunu doğrula. (Motor tanığı: $(3,1)\cdot(3,1)=$ **(9,6)**; $6=2x|_{x=3}$ ✓.)
3. **Forward vs reverse maliyeti.** n girdili, tek skaler çıktılı (kayıp) bir fonksiyon için: forward-mode AD kaç ileri geçiş gerektirir, reverse-mode kaç? Sinir ağı eğitiminde (milyonlarca ağırlık, tek kayıp) neden reverse-mode (backprop) tercih edilir? (Motor tanığı: m=4 katmanda forward-mode dual **2m = 8 ayrı geçiş** (her parametre için bir), reverse/backprop **tek geri geçiş**; milyon parametrede fark milyon kat — Ders 27'nin 1000× tanığının kardeşi.)
4. **$(I–L)^{-1}$ üçgen.** 3-katmanlı bir zincirde L kesin-alt-üçgen 3×3 olsun. (I–L)'nin de alt-üçgen olduğunu ve $(I–L)^{-1}$ x = b sisteminin geri-yerine-koymayla (back-substitution) O(n^2) işlemde çözüldüğünü açıkla. Bu neden "backprop bedava" anlamına gelir? (Motor tanığı: kesin-alt-üçgen L nilpotenttir (L^4=0); bu yüzden $(I-L)^{-1}=I+L+L^2+L^3$ **tam** olur (defect 2.2e-16); elle üçgen-çözüm O(n^2) = np.linalg.solve birebir.)
5. **(Kurs sentezi — kapanış)** Phase 2 boyunca backprop'u beş kez gördün: Karpathy, fast.ai, NYU, Strang (D27), Edelman (D36). Hangi bakış sana en çok "tıkladı" ve neden beşi de aynı reverse-mode AD'yi anlatıyor? Kısa bir paragraf yaz — bu, kursun kapanış refleksiyonudur.

41.12 Kurs Kapanışı (Ders 36 = Son Ders)

Bu, MIT 18.065'in **son dersi**. Edelman'ın ve Strang'ın veda sözleri:

> *"…linear algebra is the secret to everything. That's the big message." — Edelman, 36:34*

> *"…I hope you guys enjoyed it. I certainly enjoyed it, as you could tell." — Strang, kapanış*

34 kayıtlı ders (OCW Lec 28-29 kayıtsız lab) tamamlandı: kolon uzayından (Ders 1) backprop'un üçgen-çözüm bakışına (Ders 36); SVD, Eckart-Young, gradyan inişi, sinir ağları, sirkülant/Fourier, evrişim, graf Laplacian. Phase 2'nin altıncı kursu burada kapanır.

41.13 Anahtar Kavramlar (Cheat Sheet)

Kavram	Formül / Fikir	Edelman (dk)
Sıfır matris / rank	rank-0 → 3×0 boş matris; ispat kenar-durumsuz işler	0m00
Julia (Edelman tezi)	ML için Julia + Swift; "dil matematiksel olarak önemli"	4m08
Forward-mode AD	üçüncü şey; ne sembolik ne sayısal	10m16

Kavram	Formül / Fikir	Edelman (dk)
Dual sayılar	D=(değer, türev); 8 satır Julia, türev bedava	14m19
Reverse-mode / backprop	$dx = (I - L)^{-1} D \cdot dP$; alt-üçgen çözüm	34m33
Kapanış	"linear algebra is the secret to everything"	36m34

41.14 ML Bağlantıları Özeti

- **Forward-mode AD / dual sayılar:** JAX `jvp`, ileri-mod autodiff; az girdi–çok çıktı durumunda verimli.
- **Reverse-mode AD = backprop:** PyTorch/JAX autograd; çok girdi–tek kayıp; özü $(I - L)^{-1}$ alt-üçgen çözüm.
- **Julia:** multiple dispatch + tip = sıfır-maliyet türev; yüksek-seviye matematik + düşük-seviye hız.
- **ML = optimizasyon = türev:** AD bu işin iş atı; "ML aslında tek bir min/maks problemi".
- **Beşinci tanık:** backprop'a beş bakış (Karpathy micrograd + fast.ai + NYU vJp + Strang matris-zinciri + Edelman $(I - L)^{-1}$) = reverse-mode AD.
- **Geriye köprü:** Ders 27 (dört tanık), Ders 21 (Newton/Babylonian $\sqrt{\ }$), Ders 5-6 (PSD/SVD). Paralel: Karpathy micrograd, fast.ai manuel backprop, NYU vJp, JAX/PyTorch autograd.

! Kapanış

> *"…linear algebra is the secret to everything." — Edelman, 36:34*

18.065 burada kapanır: kolon uzayından (Ders 1) Eckart-Young'a, gradyan inişinden backprop'a, sirkülanttan graf Laplacian'ına — ve son sözde hepsi tek bir cümleye iner: lineer cebir her şeyin sırrı. Phase 2'nin altıncı kursu tamam; backprop'un beşinci tanığıyla "aynı yere giden yollar" izleği kapandı.