

Temel MATLAB Eğitimi: 3. Gün

Metin Turgay

Selçuk Üniversitesi, Türkiye

14 Ocak 2026



metinturgay.net/matlab

1

İçerik

İçerik

- 1 Fonksiyonlar ve Modüler Yapı
- 2 Veri İçe Aktarma ve Temizleme
- 3 Görüntü İşleme Temelleri

İçerik

- 1 Fonksiyonlar ve Modüler Yapı
- 2 Veri İçerme Aktarma ve Temizleme
- 3 Görüntü İşleme Temelleri
- 4 İstatistiksel Analiz ve Modelleme

Script vs Fonksiyon

Özellik	Script (.m)	Fonksiyon (function)
Amaç	Hızlı denemeler, tekil analiz	Tekrarlanan işler, araçlar
Ömür	Kalıcı (Siz silene kadar)	Geçici (İşlem bitince ölür)
Akış	Manuel Değişiklik	Otomatik (Argüman ile)

Altın Kural

“Eğer bir kod bloğunu kopyalayıp başka bir yere yapıştırıyorsanız, o **mutlaka** bir fonksiyon olmalıdır.”

Yaklaşım 1: Script Dosyaları

Kötü Örnek (Script)

```
% analiz_script.m  
r = 5;  
alan = pi * r^2;  
disp(alan);
```

**Bu değişkenler sonsuza kadar hafızada kalır!*

- **Tanım:** Komutların sıralı çalıştığı dosyalardır.
- **Global Erişim:** İçindeki her değişken Workspace'e yerleşir.
- **Riskler:**
 - ✗ **Çakışma:** Başka koddan kalan i, j değerleri işi bozar.
 - ✗ **Kirlilik:** Gereksiz değişken yığını.
 - ✗ **Tekrar:** 100 dosya için 100 kere kopyala-yapıştır gerekir.

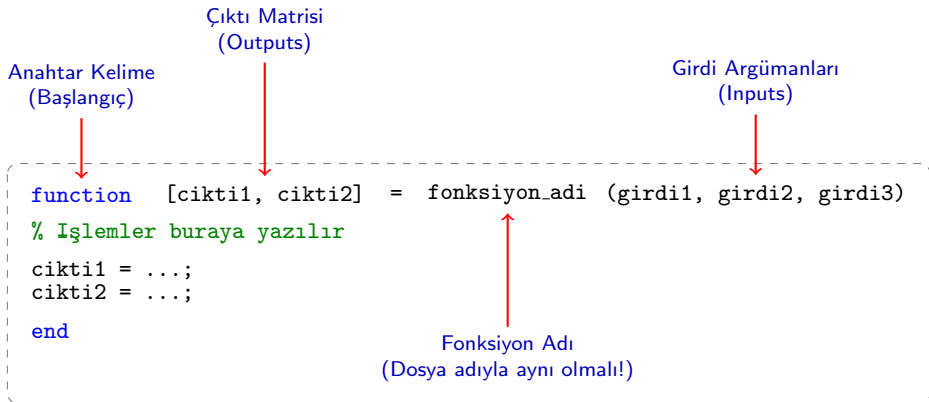
Yaklaşım 2: Fonksiyonlar

- **Tanım:** Kendi özel alanı (Scope) olan kod bloklarıdır.
- **İzolasyon:** Fonksiyon biter, değişkenler silinir.
- **Avantajlar:**
 - ✓ **Temiz Workspace:** Hafıza şişmez.
 - ✓ **Modülerlik:** Bir kez yaz, bin kez çağır.
 - ✓ **Güvenlik:** Dışarıdaki veri içeriye karışmaz.

İyi Örnek (Fonksiyon)

```
function s = daire(r)
    % r sadece burada yaşar
    s = pi * r^2;
end
```

Bir Fonksiyonun Anatomisi



Örnek Senaryo: 3 Farklı Deneyin Analizi (Script Yolu)

Görev

3 farklı veri seti için Ortalama ve Standart Sapma hesaplanacak.

Yöntem: Kopyala - Yapıştır (Copy-Paste)

```
A = [10 12 14];  
m1 = mean(A); s1 = std(A);  
  
B = [20 25 30];  
m2 = mean(A); % HATA! A değişmedi  
s2 = std(B);  
  
C = [5 8 2];  
m3 = mean(C);  
s3 = std(C);
```

Sonuç: Workspace Kaosu

- ✗ **Hata Riski:** Satır 5'te B yerine yanlışlıkla A'nın ortalaması alındı.
- ✗ **Kirlilik:** Hafızada A, B, C, m1, m2, m3, s1, s2, s3...
- ✗ **Yönetilemez:** 100 deney olsa ne yapacaktık?

>> senaryo1.m

Aynı Senaryo: Fonksiyon Yolu

Adım 1: Aracı İnşa Et

dosya: ıstatistikVer.m

```
function [m, s] = ıstatistikVer(veri)
% Bu kod sadece bir kez yazılır
m = mean(veri);
s = std(veri);
end
```

Adım 2: Aracı Kullan

main_ıstatistikVer.m (Ana Dosya)

```
deneyler = {[10 12 14], [20 25 30], [5 8 2]};

for i = 1:3
    [ort, sapma] = ıstatistikVer(deneyler{i});
    fprintf('Deney %d: %.2f \n', i, ort);
end
```

Sonuç: Tek bir satır kod ile 3 (veya 1000) deney analizi.

>> ıstatistikVer.m

>> main_ıstatistikVer.m

Detaylı Örnek: Veri Normalizasyonu

Bir makale için verilerinizi 0 ile 1 arasına sıkıştırmanız (normalize etmeniz) gerekiyor.

normalizeEt.m

```
function [yeni_veri] = normalizeEt(ham_veri)
en_kucuk = min(ham_veri);
en_buyuk = max(ham_veri);

% Formül: (x - min) / (max - min)
yeni_veri = (ham_veri - en_kucuk) ./ ...
(en_buyuk - en_kucuk);
end
```

Kazanım:

- `en_kucuk` ve `en_buyuk` değişkenleri iş bitince silinir.
- Ana kodunuzda değişken karmaşası olmaz.
- Formülü bir kez yazarsınız, her yerde kullanırsınız.

>> normalizeEt.m

Uygulama: Normalizasyon Fonksiyonunu Kullanmak

main_normalizeEt.m (Ana Dosya)

```
% 1. Adım: Ham veriyi oluştur
ham_veri = [10, 50, 30, 80, 20];

% 2. Adım: Fonksiyonu çağır
norm_veri = normalizeEt(ham_veri);

% 3. Adım: Sonuçları karşılaştır
disp('Ham Veri:');
disp(ham_veri);
disp('Normalize Veri (0-1 Arası):');
disp(norm_veri);
```

Görsel Karşılaştırma

Eğer bu verileri çizerseniz:

```
subplot(2,1,1);
plot(ham_veri);
title('Orijinal (Y: 10-80)');

subplot(2,1,2);
plot(norm_veri, 'r');
title('Normalize (Y: 0-1)');
```

**Dikkat: Verinin şekli değişmez, sadece ölçeği (y eksenini) değişir.*

>> main_normalizeEt.m

Yerel (Local) vs. Global Değişkenler

Temel Kural

Fonksiyon içinde tanımlanan değişkenler
YERELDİR.

- 👉 Fonksiyon çalışırken değişkenler yaşar.
- 👉 Fonksiyon `end` ile bitince değişkenler **silinir.**

Büyük Avantaj

`i`, `j`, `temp` gibi geçici değişkenler
ana **Workspace**'inizi kirlilemez.

*"1000 satırlık bir kodda 'acaba x değişkenini daha önce kullanmış mıydım?'
diye endişelenmenize gerek kalmaz."*

Dosyasız, Tek Satırlık Fonksiyonlar

Tanım: Anonim Fonksiyonlar

Ayrı bir .m dosyası oluşturmadan, tek satırda matematiksel fonksiyon tanımlamanızı sağlar.

Söz Dizimi (Syntax)

```
f = @(girdi) islem;
```

% Örnek: $x^2 + \sin(x)$ fonksiyonu

```
f = @(x) x.^2 + sin(x);
```

% Kullanımı

```
sonuc = f(3);
```

Kullanım Alanları: Matematiksel formüller, optimizasyon, integraller.

Her küçük formül için dosya açmak yorucudur. @(x) yapısı bunu çözer.

Uygulama

Klasik yöntem ile dosya oluşturmak yerine, integrali doğrudan kodun içinde alabiliriz.

Kod Bloğu

```
% 0 ile pi arasında sin(x) integrali  
alan = integral(@(x) sin(x), 0, pi);  
  
disp(['Alan:' num2str(alan)]);
```

- ✓ Harici dosya gerekmez.
- ✓ Matematiksel notasyona çok yakındır: $\int_0^{\pi} \sin(x) dx$

Bu yöntem özellikle kağıt üzerindeki matematiği koda dökerken çok hız kazandırır.

Esnek Fonksiyonlar Yazmak (nargin)

Kullanıcının kaç adet girdi (argument) verdiğini **nargin** (Number of Arguments In) ile sayabiliriz.

Örnek: İki Sayıyı Çarp (Biri Opsiyonel)

```
function y = carp(a, b)
% Eğer kullanıcı 2'den az girdi veriyse:
if nargin < 2
b = 1; % b girilmezse varsayılan 1 olsun
end

y = a * b;
end
```

Kullanım:

- `carp(5, 4)` → Sonuç: 20
- `carp(5)` → Sonuç: 5 (Çünkü `b=1` kabul edildi)

>> carp.m

Temiz Kod: Yorum Satırları

Altın Kural

MATLAB'de % işareti yorum satırıdır. Kod çalışırken bu kısımlar yoksayılır.

KAÇINILMASI GEREKEN (Yetersiz; Anlamsız)

% BU KOD YANLIŞTIR:

```
a = 5; % a sayısı
```

(Değişkenin ne olduğu belli değil, yorum gereksiz.)

ÖNERİLEN (Açıklayıcı)

% BU KOD DOĞRUDUR:

```
esik_degeri = 5; % Sensör hassasiyet limiti (Volt)
```

(Değişken ismi anlamlı, yorum bağlam katıyor.)

Temiz Kod: İsimlendirme Standartları

Kodunuzu başkası (veya 6 ay sonra siz) okuduğunda bir roman gibi akmalıdır.

➔ **Anlamlı İsimler:** Değişken ismini gördüğünüzde ne işe yaradığını anlamalısınız.

✗ x, y, z (Matematiksel formüller hariç kaçının)

✓ zaman, basinc, sicaklik (Fiziksel anlamı var)

➔ **CamelCase Kullanımı:** Kelimeleri bitişik yazıp, yeni kelimeye büyük harfle başlama.

- ▶ veriokumafonksiyonu (Zor okunur)
- ▶ veriOkumaFonksiyonu (Standart)
- ▶ Veri_Okuma_Fonksiyonu (Alternatif: Snake_Case)

“Kodunuzu başkası okuduğunda (veya siz 1 yıl sonra okuduğunuzda) ne yaptığını anlamalı.
İsimlendirme en ucuz dokümantasyondur.”

Hata Yönetimi

Durum: 100 dosyanız var. 3. dosya bozuk. Kodunuz durmalı mı, yoksa bozuk olanı atlayıp devam mı etmeli?

Try-Catch: Amaç

Kodun beklenmedik bir hata yüzünden (örn: dosya silinmiş, internet kesilmiş) tamamen durmasını ("crash" olmasını) engellemek.

Güvenli Dosya Okuma Örneği

```
try
% Hata çıkma ihtimali olan kod
veri = readtable('olmayan_dosya.xlsx');

catch
% Hata oluşursa devreye giren "B Planı"
warning('Dosya yok, varsayılan veri yükleniyor.');
```

```
veri = zeros(10,1);
end
```

Not: 'warning' komutu kodu durdurmaz, sadece komut penceresinde sarı bir uyarı mesajı gösterir.

Uygulama Arası (2 Dakika)

Sıra Sizde!

(Süre: 2 Dakika)

Görev

Santigrat (C) dereceyi alıp Fahrenheit'a (F) çeviren C2F isiminde basit bir fonksiyon yazın.

Dönüşüm Formülü

$$F = C \times 1.8 + 32$$

İpucu: Fonksiyonunuzu `function f = C2F(c)` satırı ile başlatmayı unutmayın.



Uygulama Çözümü: C2F Fonksiyonu

C2F.m (Fonksiyon Dosyası)

```
function f = C2F(c)
% C: Santigrat girdisi
% f: Fahrenheit çıktısı

f = c * 1.8 + 32;

end
```

**Dosya adının C2F.m olduğundan emin olun.*

Test Edelim (Command Window)

```
>> sicaklik = 100;
>> sonuc = C2F(sicaklik)

sonuc =
212

>> C2F(0) % Donma noktası
ans =
32
```

Kritik Nokta

Fonksiyonun içinde `input` veya `disp` komutlarını kullanmadık. Fonksiyon sadece değer **alır** ve değer **döndürür**.

>> C2F.m

Neden Modülerlik?

Modüler kodlama, sadece “şık” olduğu için değil, hayatta kalmak için gereklidir.

Hata Ayıklama (Debugging) Kolaylaşır

- ▶ Tüm koda bakmak yerine, sadece hata veren fonksiyona (*parçaya*) odaklanırsınız.

İşbirliği Mümkündür

- ▶ Bir kişi `analizYap.m` dosyasını yazarken, diğeri `grafikCiz.m` dosyasını yazar. Kimse kimsenin kodunu bozmaz.

Kod Tekrarı Önlenir (DRY Prensipleri)

- ▶ Aynı formülü 50 yere kopyalamak yerine, fonksiyonu 50 kere çağırırsınız. Formül değişirse tek bir yeri düzeltmek yeterlidir.

DRY: Don't Repeat Yourself (Kendini Tekrar Etme). Yazılım dünyasının en temel kuralıdır.

Bölüm Sonu Özeti: Cebimizde Neler Var?

Temel Kavramlar

- 1 **Dosya Yapısı:** Fonksiyonlar, dosya adı ile fonksiyon adının aynı olduğu .m dosyalarıdır.
- 2 **Akış Mantığı:**
$$\text{input} \longrightarrow \boxed{\text{FUNCTION}} \longrightarrow \text{output}$$
- 3 **Pratik Matematik:** Basit formüller ve integraller için dosya açmaya gerek yoktur; Anonim Fonksiyonlar (@) kullanılır.
- 4 **Güvenlik:** Fonksiyon içindeki değişkenler **Yereldir (Local)**. İş bitince silinirler, böylece Workspace temiz ve güvenli kalır.

Sırada: Verileri MATLAB'e nasıl aktarırız?

MATLAB Hangi Verilerle Konuşur?

MATLAB, akademik dünyadaki hemen hemen her veri formatını okuyabilir.

- ➡ **Excel Dosyaları:** `.xlsx`, `.xls`
 - ▶ En yaygın ham veri formatı.
- ➡ **Metin Tabanlı:** `.csv` (Comma Separated), `.txt`
 - ▶ Cihazlardan alınan ham loglar genelde bu formattadır.
- ➡ **MATLAB Verisi:** `.mat`
 - ▶ MATLAB'ın kendi binary formatıdır. Büyük verileri (Big Data) saniyeler içinde yükler.
- ➡ **Görüntü/Medya:** `.jpg`, `.png`, `.tif`
 - ▶ Görüntü işleme projeleri için.

Karşılaştırma: readtable vs. xlsread

✗ xlsread (Eski Yöntem)

- Sadece sayıları alır, metinleri kaybeder veya ayırır.
- Sütun başlıklarını (Header) okumaz.
- Karışık veri tiplerinde (Sayı + Metin) hata yapmaya meyillidir.

MathWorks artık önermiyor.

✓ readtable (Modern)

- Veriyi **Table** formatında, Excel'deki gibi bütün olarak alır.
- Sütun isimlerini korur (T.Sıcaklık gibi erişim sağlar).
- Eksik verileri ve metinleri akıllıca yönetir.

Tavsiye Edilen

Özet

Excel veya CSV okuyacaksanız, komutunuz her zaman **readtable** olmalıdır.

readtable Kullanımı: Farkı Görelim

Eski (xlsread)

```
[sayi, yazı] = xlsread('veri.xlsx');  
disp(sayı);
```

Çıktı (Matris)

```
ans =  
12.5000    0.0500  
13.1000    0.0400  
14.2000    0.0600
```

✗ Başlıklar kayboldu!
Hangi sütun neydi?

Yeni (readtable)

```
T = readtable('veri.xlsx');  
disp(head(T, 3));
```

Çıktı (Tablo)

Zaman	Voltaj
12.5	0.05
13.1	0.04
14.2	0.06

✓ Sütun isimleri korundu!

Standart Kullanım

```
T = readtable('deney_verileri.xlsx');  
disp(head(T,3)); % İlk 3 satırı göster
```

*Table veri tipi, sütun isimlerini koruduğu için kodun okunabilirliğini muazzam artırır. 1. sütun demek yerine T.Zaman diyebiliyoruz.

Veriye Erişim (Dot Notation)

Tablo formatındaki verinin içinden belirli bir sütunu çekip almak için **Nokta** (.) operatörü kullanılır.

Söz Dizimi (Syntax)

TabloAdi.SutunAdi

Tablomuz (T)

Time	Temp
0.1	24.5
0.2	24.8
...	...

↑
Başlıklar Önemli!

Kod Örnekleri

```
% Time sütununu al, 'zaman'a ata  
zaman = T.Time;
```

```
% Temp sütununu al, 'sicaklik'a ata  
sicaklik = T.Temp;
```

Dikkat: Boşluk Karakteri

MATLAB değişken isimlerinde boşluk kabul etmez. Excel'den gelen sütun isimlerinde boşluk varsa, otomatik olarak **alt çizgiye** dönüştürülür.

Excel: “Isil Deger” → MATLAB: T.Isil_Deger

Uygulama: Tabloyu Tanımak ve Değiştirmek

Adım 1: Excel Dosyası Yarat

```
% Sanal bir veri seti kuralım  
Isim = {'Ali'; 'Ayşe'; 'Fatma'};  
Yas = [25; 30; 22];  
Maas = [5000; 7000; 6000];  
  
T_raw = table(Isim, Yas, Maas);  
  
% Excel olarak kaydet  
writetable(T_raw, 'calisanlar.xlsx');
```

**Bu kod çalışınca klasörde 'calisanlar.xlsx' oluşur.*

```
>> tabloIslemleri.m
```

Adım 2: Oku, İncele, Değiştir

```
% 1. Dosyayı Oku  
T = readtable('calisanlar.xlsx');  
  
% 2. Tabloyu Özetle (Min, Max, vs.)  
summary(T);  
  
% 3. Boyutları Öğren  
disp(['Satır Sayısı: ', num2str(height(T))]);  
  
% 4. Veri Değiştir (Ayşe'ye zam yap)  
T.Maas(2) = 7500;
```

Kritik Komutlar:

- `height(T)`: Satır sayısı.
- `width(T)`: Sütun sayısı.
- `summary(T)`: İstatistiksel özet.
- `T.Properties.VariableNames`: Sütun isimlerini listeler.

Kirli Veri Nedir?

Tanım

Kirli veri, analize uygun olmayan, hatalı veya eksik veridir.

✗ Eksik Değerler (NaN):

(Not a Number) Sensör okuyamadı veya veri girilmedi.

✗ Hatalı Okumalar:

Oda sıcaklığının -999 veya 0 çıkması.

✗ Aykırı Değerler (Outliers):

İstatistiksel olarak imkansız sapmalar.

Örnek Tablo

Zaman	Sıcaklık
10:00	24.5
10:05	NaN
10:10	25.1
10:15	-999

Eksik Veriyi Tespit Etme (ismissing)

Analize başlamadan önce verinin sağlığını (check-up) kontrol etmeliyiz.

Kod: Tarama Yap

```
% 1. Nerede eksik var? (Harita)
eksik_var_mi = ismissing(T);

% 2. Toplam kaç eksik var?
toplam = sum(sum(eksik_var_mi));

disp(['Eksik Sayısı: ' ...
      num2str(toplam)]);
```

“Analize başlamadan önce verinizin sağlığını kontrol etmelisiniz. NaN değerleri ortalama, standart sapma gibi istatistiksel hesaplamaları bozar.”

Nasıl Çalışır?

`ismissing` komutu, orijinal tablonun boyutunda **Mantıksal (Logical)** bir matris üretir.

5	→	0
NaN		1
8		0

1'lerin toplamı eksik sayısını verir.

Yöntem 1: Eksik Veriyi Silme (rmmissing)

Akademik Yaklaşım

Veri setiniz yeterince büyükse, silmek en güvenilir bilimsel yöntemdir. *"Yanlış veri üretmektense veri kaybetmek iyidir."*

10	→ rmmissing →	10
NaN		12
12		15
15		

Kod

```
% İçinde NaN olan SATIRLARI sil  
T_temiz = rmmissing(T);  
  
% Boyut kontrolü  
disp(height(T_temiz));
```

Not: Tüm satır silinir, sadece hücre silinmez.

Yöntem 2: Eksik Veriyi Doldurma (fillmissing)

Veri setiniz küçükse veya zaman serisi analizi yapıyorsanız (zamanı koparmamak için), eksikleri tahmin ederek doldurursunuz.

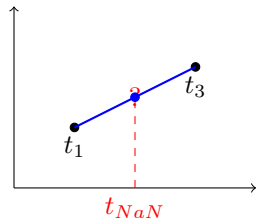
Enterpolasyon (Interpolation)

Bir önceki ve bir sonraki değere bakarak ara değeri tahmin etme işlemidir.

Kod: Linear Doldurma

```
% Boşlukları çizgi çekerek doldur  
T_dolu = fillmissing(T, 'linear');
```

```
% Diğer Yöntemler:  
% 'previous' (Öncekini kopyala)  
% 'nearest' (En yakını al)
```



Mavi nokta, lineer yöntemle üretilmiştir.

Uygulama: Kirli Veriyle Başa Çıkmak

1. Sorun: Karışık Hatalar

Verimizde hem boşluklar (NaN) hem de saçma değerler (-999) var.

2. Çözüm: NaN'a Çevir

Önce hatalı okumaları NaN yaparak sorunu tek tipe indiriyoruz.

```
% 55 veya < -30 olanları NaN yap  
idx = T.Sicaklik > 55 | T.Sicaklik < -30;  
T.Sicaklik(idx) = NaN;
```

3. Temizlik Yöntemini Seç

A) Silme (Veri Bol ise)

```
T_sil = rmmissing(T);  
% 5 satırdan 3 satıra düştük
```

B) Doldurma (Veri Kıymetliyse)

```
T_dolu = fillmissing(T, 'linear');  
% 2. ve 4. satırlar tahmin edildi
```

Sonuç

Artık elimizde mean veya polyfit yapabileceğimiz temiz bir matris var.

```
>> veriTemizlemeDemo.m
```

Mantıksal İndeksleme (Logical Indexing) - TEORİ

Önem Derecesi: ★★★★★

MATLAB'ın en güçlü ve en sık kullanılan özelliğidir. Döngü (`for`) kullanmadan veri seçmenizi sağlar.

Mantık

Verinin **pozisyonuna** (adresine) göre değil, **içeriğine** (değerine) göre seçim yapmaktır.

1. Soru ($A > 5$)

2	8
6	3

2. Cevap (Maske)

0	1
1	0

(MATLAB bunu üretir)

3. Seçim ($A(A > 5)$)

8
6

(1 olanları getirir)

$A(A > 5) \rightarrow$ "A matrisinde, 5'ten büyük olan yerleri getir."

Uygulama: Voltaj Filtreleme

Bir tablodan (Excel verisi) belirli satırları çekip almak için **Mantıksal İndeksleme** kullanılır.

Senaryo

Elimizde binlerce satırlık veri var. Biz sadece **Voltaj değeri 12'den büyük** olan deneyleri analiz etmek istiyoruz.

```
% 1. Adım: Filtreyi Oluştur (Soru Sor)
filtre = T.Voltaj > 12;
% Sonuç: [0; 1; 1; 0...] gibi bir vektör
```

```
% 2. Adım: Filtreyi Uygula
% Söz Dizimi: T(satirlar, sutunlar)
T_yuksek = T(filtre, :);
```

Söz Dizimi (Syntax)

T(filtre, :)

- **filtre:** Koşulu sağlayan satırları (1'leri) seç.
- **iki nokta (:):** O satıra ait tüm sütunları (Zaman Sıcaklık vs.) **>> filtreleme.m**

Çoklu Koşul ile Filtreleme (&, |)

Gerçek hayatta genelde tek bir kriter yetmez. Birden fazla koşulu birleştirmek için **Mantıksal Operatörler** kullanılır.

Operatörler

& (VE): Her iki koşul da sağlanmalı.

| (VEYA): En az biri sağlansa yeter.

İpucu: Tablolarla çalışırken tek (&) ve tek (|) kullanın.

Senaryo

"Voltajı 12'den yüksek **VE** Sıcaklığı 50'den düşük olan verileri getir."

```
% Parantez kullanmak okumayı kolaylaştırır  
T_filtreli = T( ...  
T.Voltaj > 12 & T.Sicaklik < 50, ...  
: );
```

Kritik Hata: && Kullanmayın!

MATLAB'de **&&** operatörü sadece tekil sayılar (scalar) içindir. Tablo veya dizi filtrelerken **mutlaka tek &** kullanılmalıdır. Aksi halde hata alırsınız.

>> cokuFiltre.m

Veriyi Sıralama (sortrows)

Verilerin zaman sırasına göre veya büyüklüğe göre dizilmesi analiz için kritiktir.

Komut: sortrows

```
% 'Zaman' sütununa göre  
% küçükten büyüğe sırala  
T_sirali = sortrows(T, 'Zaman');  
  
% TERS Sıralama (Büyükten küçüğe)  
T_ters = sortrows(T, 'Zaman', ...  
'descend');
```

Önce (Karışık)

10:05	24
10:01	22
10:03	23

↓ sortrows

Sonra (Düzenli)

10:01	22
10:03	23
10:05	24

>> veriSiralama.m

Veriyi Dışa Aktarma (writetable)

Analizimiz bitti, temizlenmiş ve filtrelenmiş veriyi raporlamak için kaydetmeliyiz.

Söz Dizimi

```
writetable(TabloAdi, 'DosyaAdi.uzanti')
```

Örnek Kullanım

```
% Excel Olarak Kaydet  
writetable(T_filtreli, 'Temizlenmis_Veri.xlsx');
```

```
% CSV (Metin) Olarak Kaydet  
writetable(T_filtreli, 'Analiz_Sonuc.csv');
```

Altın Kural: Ham Veriye Dokunma!

İşlenmiş veriyi asla orijinal dosyanın üzerine (Overwrite) kaydetmeyin. Her zaman yeni bir isim verin (örn: _v2, _processed). Orijinal veri kutsaldır.

Bölüm Sonu Özeti: Veri Döngüsü

Bu bölümde bir veri bilimcisinin günlük rutinini öğrendik:

- ➡ **İçe Aktar (Import):**
`readtable` kullanarak Excel'i akıllı tabloya çevir.
- ➡ **Tanı (Diagnose):**
`summary` ve `ismissing` ile verinin röntgenini çek.
- ➡ **Temizle (Clean):**
`rmmmissing` (Sil) veya `fillmissing` (Doldur) ile eksikleri gider.
- ➡ **Filtrele (Filter):**
Mantıksal operatörler (`>`, `<`, `&`) ile sadece ilgilendiğin veriyi seç.
Örn: `T(T.Hiz > 50, :)`

Görüntü Nedir?

Kavram

Dijital dünyada “Resim” diye bir şey yoktur. Gördüğünüz her kare, sayılardan oluşan devasa bir **MATRİS**'tir.

Motto

Pixel = Sayı

- **0** → Siyah (Karanlık)
- **255** → Beyaz (Parlak)
(*uint8 formatında*)

Bilgisayarın Gördüğü

10	30	55
45	120	150
80	200	250

Okuma ve Gösterme (imread, imshow)

MATLAB'de dış dünyadan bir görüntüyü içeri almak için `imread`, ekrana basmak için `imshow` kullanılır.

Gri Resim Okuma

```
% MATLAB'in hafızasından 'Kameraman' resmini oku
img = imread('cameraman.tif');

% Boyutuna bak
disp(size(img));
% Sonuç: 256 256 (Sadece Satır ve Sütun)

% Ekrana bas
imshow(img);
```

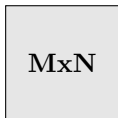
Hayati Uyarı (!)

`imread` komutunun sonuna **noktalı virgül (;)** koymayı ASLA unutmayın!
Aksi halde yüzlerce sayı ekrana dökülür ve MATLAB kilitlenir. >> `goruntuGiris.m`

Görüntü Türleri: Gri vs. Renkli (RGB)

1. Gri Seviye (Grayscale)

Tek bir katmandan (matristen) oluşur. Her pikselde sadece parlaklık (0-255) bilgisi vardır.



2 Boyutlu

```
img = imread('cameraman.tif');  
size(img) -> [256 256]
```

2. Renkli (RGB)

Üç katmanlıdır. **Kırmızı**, **Yeşil** ve **Mavi** matrisleri üst üstedir.



3 Boyutlu

```
img = imread('peppers.png');  
size(img) -> [384 512 3]
```

Not: Görüntü işleme algoritmaları genellikle Gri Seviye üzerinde çalışır.

Griye Dönüştürme (rgb2gray)

Neden Griye Çeviriyoruz?

Nesne tespiti veya kenar bulma gibi işlemlerde **renk** genellikle gereksiz bir bilgidir (gürültüdür). Önemli olan **ışık yoğunluğudur** (Intensity).

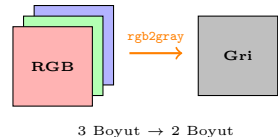
- 🔊 **Maliyet Kazanımı:** 3 katmanlı matrisi işlemek yerine tek katmanlı işlemek işlem yükünü 3'te 1'e indirir.
- 🔊 **Odaklanma:** Algoritma "rengi" değil, "şekli" görür.

Dönüşüm Kodu

```
% Renkli bir resim yükle (Peppers)
img_renkli = imread('peppers.png');

% Griye çevir (Boyut düşürme)
img_gri = rgb2gray(img_renkli);

% Yan yana göster
montage({img_renkli, img_gri});
```



>> griDonusum.m

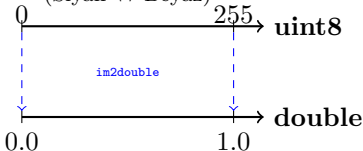
Veri Tipleri: uint8 vs. double

Sorun: Dil Uyuşmazlığı

Görüntü dosyaları hafıza tasarrufu için **uint8** (0-255 Tamsayı) kullanır. Ancak MATLAB matematiği **double** (Küsuratlı Sayı) sever.

Dönüşüm Mantığı

(Siyah ↔ Beyaz)



>> veriTipleri.m

Neden Dönüştürmeliyiz?

uint8 matematikte hata yapabilir:

- $200 + 100 = 255$ (Taşma/Clipping)
- $10/3 = 3$ (Yuvarlama Hatası)

Çözüm Kodu

```
img = imread('resim.jpg');  
% 0-1 aralığına sıkıştır  
img_dbl = im2double(img);  
% Artık çarpma yapabiliriz  
parlak = img_dbl * 1.5;
```

Görüntü Analizi: Histogram (imhist)

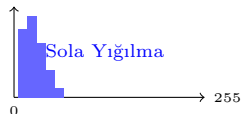
Bir resmin röntgenini çekmek gibidir. Piksellerin parlaklık dağılımını gösterir.

Ne İşe Yarar?

- Resim çok mu **karanlık**?
- Çok mu **parlak**?
- Kontrast bozuk mu?

Kural:

- Grafik **Sola** yığılıysa → **Karanlık**
- Grafik **Sağa** yığılıysa → **Parlak**



Temel İşlem 1: Eşikleme (Binarization)

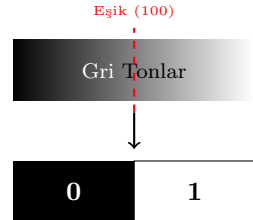
Amaç

Resmi sadece **Siyah (0)** ve **Beyaz (1)** haline getirmektir. Nesne tespiti (Object Detection) için en kritik adımdır.

Kod Örnekleri

```
% 1. Otomatik (Otsu Yöntemi)  
% En iyi eşik değerini kendi bulur  
bw = imbinarize(img_gri);  
  
% 2. Manuel (Elle Ayarlama)  
% 100'den büyükse 1, değilse 0  
bw = img_gri > 100;
```

Sonuç "Logical" (Mantıksal) veri tipindedir. Sadece 0 ve 1 barındırır.



>> histogramVeEsikleme.m

Temel İşlem 2: İyileştirme (Filtering)

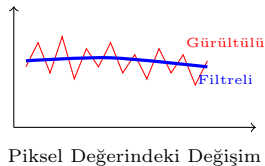
Mantık: Neden Bulanıklaştırıyoruz?

Bulanıklaştırma (Blur), resimdeki ani değişimleri (cızırtı, gürültü, leke) yumuşatarak yok eder. Bir nevi **“Resmi Ütülemek”** gibidir.

Gaussian Blur (Yumuşatma)

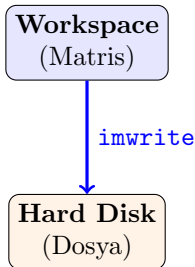
```
% img: Orjinal (Gürültülü)  
% 2: Bulanıklık Şiddeti (Sigma)  
  
img_blur = imgaussfilt(img, 2);  
  
% Farkı görmek için yan yana koy  
montage({img, img_blur});
```

Sigma değeri (burada 2) ne kadar artarsa resim o kadar bulanıklaşır ama gürültü o kadar azalır.



Sonuçları Kaydetme (imwrite)

İşlediğimiz matrisi, tekrar resim dosyası olarak diske yazmak için `imwrite` kullanılır.



Kod

```
% JPEG Olarak Kaydet  
imwrite(img_blur, 'sonuc.jpg');  
  
% PNG Olarak Kaydet (Kayıpsız)  
imwrite(img_blur, 'sonuc.png');  
  
% TIF Olarak Kaydet (Yüksek Kalite)  
imwrite(img_blur, 'sonuc.tif');
```

Otomatik Algılama

MATLAB, yazdığımız dosya uzantısına (`.jpg`, `.png`) bakarak formatı otomatik ayarlar. Ekstra ayar yapmanıza gerek yoktur.

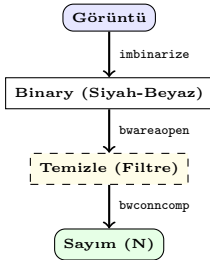
`>> filtreVeKayit.m`

Mini Proje: Bozuk Para Sayma

Görev

Kamera görüntüsündeki madeni paraları otomatik sayan bir algoritma tasarlayın.

Çözüm: Bozuk Para Sayma



>> paraSayma.m

Çözüm Kodu

```
% 1. Oku
img = imread('coins.png');

% 2. Eşikleme (Siyah-Beyaz Yap)
bw = imbinarize(img);

% 3. Delikleri Doldur (Simit olmasın)
bw = imfill(bw, 'holes');

% 4. Gürültü Temizliği (KRITİK)
% 100 pikselden küçük noktaları sil
bw = bwareaopen(bw, 100);

% 5. Say
sonuc = bwconncomp(bw);
disp(['Para Sayısı: ' ...
      num2str(sonuc.NumObjects)]):
```

Akademik İstatistiğe Giriş

Amaç Nedir?

Bilimsel çalışmalarda ham veriye bakarak karar veremeyiz. İstatistik, veriyi “**Özetlemek**” (Tanımlamak) ve o veriden genel geçer “**Sonuçlar Çıkarmak**” (Kestirim) için kullandığımız dildir.

1. Tanımlayıcı (Descriptive)

- “Elimde ne var?”
- Ortalama, Standart Sapma
- Geçmişi anlatır.

Örn: Bu sınıfın not ortalaması 60’tır.

2. Çıkarımsal (Inferential)

- “Gelecekte ne olacak?”
- Regresyon, Hipotez Testleri
- Geneli tahmin eder.

Örn: Gelecek yıl giriş puanları artacak.

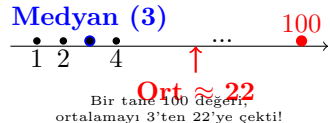
Temel İstatistik Komutları

Veriyi anlamlandırmak için kullanılan “Muhteşem Dörtlü”:

- **mean(x): Aritmetik Ortalama**
- **median(x): Ortanca Değer**
(*Aykırı değerlere dirençlidir*)
- **std(x): Standart Sapma**
(*Risk / Dalgalanma ölçüsü*)
- **min(x) / max(x): Uç Değerler**

Mean vs. Median

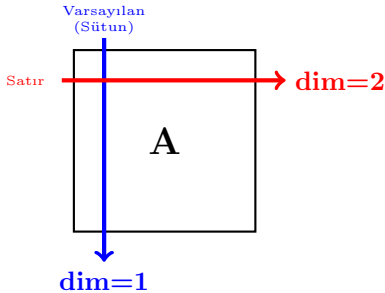
Veride “Uçuk Kaçık” (Outlier) değerler varsa Ortalama yalan söyler, Medyan doğruyu söyler.



>> temelIstatistik.m

Matrislerde Boyut Farkı (DİKKAT!)

MATLAB fonksiyonları (mean, sum, min, max) varsayılan olarak sütun bazlı çalışır. Yönü değiştirmek sizin elinizdedir.



Kod Kullanımı

```
A = [10 20;  
30 40];
```

```
% 1. Sütun Ortalaması (Mavi)
```

```
m1 = mean(A, 1);
```

```
% Sonuç: [20 30]
```

```
% 2. Satır Ortalaması (Kırmızı)
```

```
m2 = mean(A, 2);
```

```
% Sonuç: [15; 35]
```

Altın Kural

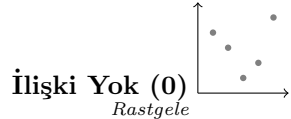
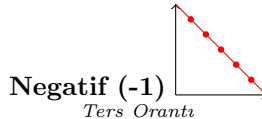
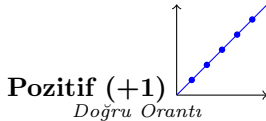
Satır satır işlem yapmak istiyorsanız (örneğin her öğrencinin kendi not ortalaması), komutun sonuna mutlaka **,2** eklemelisiniz.

Korelasyon Analizi (corrcoef)

İki değişken arasındaki ilişkinin yönünü ve gücünü ölçer. Sonuç -1 ile +1 arasındadır.

Kod

```
R_matris = corrcoef(T.Hiz, T.Yakit);  
R = R_matris(1,2); % Korelasyon katsayısını çek
```



```
>> istatistikAnaliz.m
```

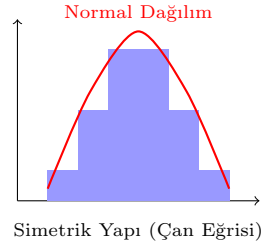
Dağılımı Görmek (histogram)

Verinin karakterini anlamak için ilk soru şudur: “**Verim Normal (Gaussian) Dağılıma uyuyor mu?**”

Kod: Histogram

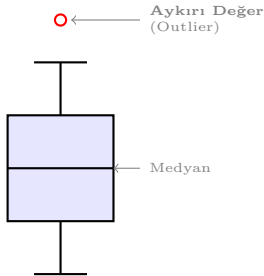
```
% Veri setindeki dağılımı çiz  
% 20: Çubuk (Bin) sayısı  
  
histogram(T.HataPayi, 20);  
  
title('Hata Dağılımı');  
xlabel('Hata Değeri');  
ylabel('Frekans');
```

İpucu: Çubuk sayısı artarsa detay artar, azalırsa genel hatlar görünür.



Aykırı Değer Tespiti (boxchart)

Veri setindeki "Çürük Elma"ları (Outliers) bulmanın en bilimsel yolu Kutu Grafiği çizmektir.



Kod: boxchart (Modern)

```
% Standart MATLAB komutudur.  
% (Ek paket gerektirmez)
```

```
boxchart(T.Degerler);
```

```
ylabel('Ölçüm Sonuçları');  
title('Aykırı Değer Analizi');
```

Nasıl Okunur?

Kutunun bıyıkları (Whiskers) dışına taşan **daireler**, istatistiksel olarak "Aykırı Değer"dir. Bunlar genellikle hatalı ölçümdür.

```
>> dagilimAnalizi.m
```

Eğri Uydurma (Curve Fitting) Nedir?

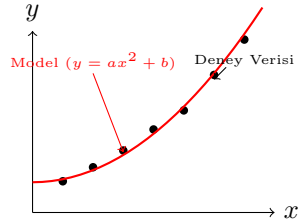
Senaryo

Elinizde deneyden gelen "Gürültülü" (hatalı, titrek) veriler var. Siz bu noktaların arkasındaki **Asıl Fiziksel Yasayı** (Formülü) bulmak istiyorsunuz.

Amaç:

- Veri noktalarının üzerinden (veya en yakınından) geçen en pürüzsüz çizgiyi çizmektir.
- Geleceği tahmin etmektir (Extrapolation).

[Image of linear regression scatter plot]



Polinom Uydurma (polyfit)

En küçük kareler yöntemini (Least Squares) kullanarak veriye en uygun polinom katsayılarını bulur.

Komut Yapısı

```
p = polyfit(x, y, n);
```

Parametreler:

- **x**, **y**: Veri setleri (Vektör olmalı).
- **n**: Polinom Derecesi.
 - ▶ **n=1**: Doğru ($y = ax + b$)
 - ▶ **n=2**: Parabol ($y = ax^2 + bx + c$)
 - ▶ **n=3**: Kübik Eğri...

```
>> egriUydurma.m
```

Çıktı (p)

Sonuç bir **Katsayılar Vektörüdür**.

Eğer $n = 2$ ise:

$$p = [a, \quad b, \quad c]$$

(Sırasıyla x^2 , x^1 , x^0 'ın katsayıları)

Katsayıları Anlamak: Çıkan Sayılar Ne Anlama Geliyor?

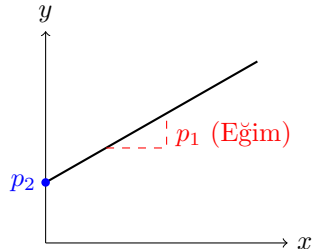
Eğer 1. dereceden (Lineer) bir model kurduysanız ($n = 1$), MATLAB size 2 tane sayı verir.

Matematiksel Karşılık

$$y = p_1x + p_2$$

Bu sayılar nedir?

- p_1 (**Eğim/Slope**): Değişim hızıdır. (Örn: İvme)
- p_2 (**Kesişim/Intercept**): Başlangıç değeridir. ($x = 0$ iken y 'nin değeri)



MATLAB Kodu

```
p = polyfit(zaman, hiz, 1);  
disp(['İvme: ' num2str(p(1))]);
```

Model Değerlerini Hesaplama (polyval)

Katsayıları (p) bulduk. Şimdi bu formülü kullanarak x değerlerine karşılık gelen teorik y değerlerini hesaplamalıyız.

✗ Zor Yol (Elle)

Polinom derecesi arttıkça bunu yazmak işkenceye dönüşür:

```
y = p(1)*x^3 + p(2)*x^2 ...  
+ p(3)*x + p(4);
```

✓ Kolay Yol (polyval)

MATLAB dereceye bakmaksızın hesabı otomatik yapar:

% p: Katsayılar

% x: Giriş verisi

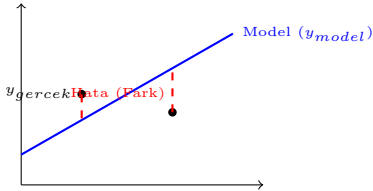
```
y_model = polyval(p, x);
```

*“Artık elimizde hem gerçek (deney) verileri, hem de teorik (polyval) verileri var.
Bunları üst üste çizdirip kıyaslayabiliriz.”*

>> regresyonAnaliz.m

Hata Hesabı: Modelim Ne Kadar İyi? (RMSE)

Modeli kurduk ama ne kadar güvenilir? Bunu ölçmenin endüstri standardı ****RMSE**** (Kök Ortalama Kare Hata) değeridir.



[Image of RMSE formula visual explanation]

Mantık:

- 1 Farkları bul ($y - y_{model}$)
- 2 Karesini al (Negatifler gitsin)
- 3 Ortalamasını al ve kökünü çek.

```
>> hataAnalizi.m
```

MATLAB Kodu

```
% 1. Hata Vektörü  
hata = y_gercek - y_model;  
  
% 2. RMSE Hesabı  
RMSE = sqrt(mean(hata.^2));  
  
disp(['Hata Payı: ', ...  
      num2str(RMSE)]);
```

Yorum

RMSE değeri **0'a ne kadar yakınsa**, model o kadar mükemmeldir.
(RMSE birimi, verinin birimiyle aynıdır.)

CASE STUDY: Yay-Kütle Deneyi Simülasyonu

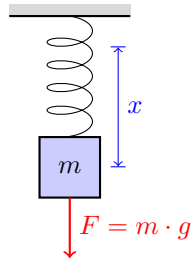
Senaryo

Bir yayın ucuna farklı ağırlıklar asıp ne kadar uzadığını cetvelle ölçtük. Amacımız bu verileri kullanarak yayın sertliğini yani **Yay Sabiti** (k) değerini bulmak.

Fiziksel Model (Hooke Yasası)

$$F = k \cdot x$$

- F : Kuvvet (Newton)
- x : Uzama Miktarı (cm veya m)
- k : Yay Sabiti (Aradığımız şey!)



Adım 1: Veriyi Hazırla ve Dönüştür

Önce elimizdeki ham veriyi (Kütle) fiziksel anlamlı veriye (Kuvvet) çevirmeliyiz.

MATLAB Kodu

```
% 1. Deney Verileri
kutle = [1, 2, 3, 4, 5];          % kg
uzama = [0.9, 2.1, 2.9, 4.2, 5.1]; % cm

% 2. Fiziksel Dönüşüm (Newton 2. Yasası)
% F = m * g (Yerçekimi ivmesi g = 9.81)

kuvvet = kutle * 9.81;

% Sonuçları görelim
disp(table(kutle', uzama', kuvvet', ...
'VariableNames', 'Kutle_kg', 'Uzama_cm', 'Kuvvet_N'));
```

Not: "uzama" verisi cetvelle ölçüldüğü için biraz hatalı (gürültülü) olabilir.

Adım 2: Modelleme (k sabitini bulma)

Fizik formülü $F = k \cdot x$ ile Doğru Denklemi $y = p_1 \cdot x + p_2$ aynı yapıdadır. Yani doğrunun **Eğimi** (p_1), bize **Yay Sabitini** (k) verir.

Çözüm Kodu

```
% Lineer Regresyon (1. Derece)
% X eksenini: Uzama, Y eksenini: Kuvvet

p = polyfit(uzama, kuvvet, 1);

% İlk katsayı eğimdir
yay_sabiti = p(1);

disp(['Hesaplanan Yay Sabiti (k): ', ...
      num2str(yay_sabiti), ' N/cm']);
```

>> yayDeneyi.m

Bölüm 4 Özeti: İstatistik ve Modelleme

Veriyi "Konuşturmak" için kullandığımız araçlar:

- ➡ **Tanı (Diagnosis):**
mean, median, std ile verinin karakterini anladık.
- ➡ **Görselleştir (Visualize):**
histogram ile dağılımı, boxchart ile aykırı değerleri yakaladık.
- ➡ **Modelle (Fit):**
polyfit ile gürültülü veriden matematiksel formül çıkardık.
- ➡ **Tahmin Et (Predict):**
polyval ile modelimizi kullanarak geleceği tahmin ettik.

TEŞEKKÜRLER