

# MATLAB

---

## MATLAB Tarihçesi

1970'li yılların sonlarına doğru Amerika Birleşik Devletleri'nde Cleve Moler tarafından yazılmaya başlanan **MATLAB** programı, 1984 yılında kurulan MathWorks firmasının çatısı altında C dili kullanılarak gelişim sürecini devam ettirmiştir. **MATLAB**, MathWorks firmasının web sayfasında şu şekilde tanımlanmaktadır: **MATLAB** yüksek seviyeli bir programlama dilidir ve hesaplama yoğunluklu işlemlerin C, C++ ve Fortran gibi geleneksel programlama dillerinden daha hızlı bir şekilde gerçekleştirilemesine imkan sağlayan etkileşimli(interaktif) bir ortamdır.

**MATLAB**, başlangıçta matris manipülasyonlarını kolaylaşdıracak bir sayısal kütüphane (MATrix LABoratory - MATris LABoratuarı) olarak tasarlanmıştır ve halen matris işlemleri söz konusu olduğunda yardımına en çok başvurulan program olma özelliğini muhafaza etmektedir. Öte yandan **MATLAB**, matris işlemlerinin yanı sıra:

- ❖ Algoritma ve program geliştirebilmeye,
- ❖ Sayısal(nümerik) veya sembolik matematiksel hesaplamalar gerçekleştirebilmeye,
- ❖ 2 veya 3 boyutlu grafikler çizebilmeye,
- ❖ Modelleme ve simülasyon yapabilmeye,
- ❖ GUI (Graphical User Interface – Grafiksel Kullanıcı Arayüzü) tasarlayabilmeye de imkan sağlamaktadır ve resim işleme ve lineer cebir işlemleri içinde sıkılıkla kullanılmaktadır.
- ❖ Aynı zamanda, bünyesinde barındırdığı farklı mühendislik disiplinlerine yönelik yazılmış hazır araçlar(toolboxes) sayesinde zamanlı tasarruf sağlayarak kullanıcılarla büyük kolaylıklar sunmaktadır.
- ❖ **MATLAB** bütün bu özelliklerine ek olarak, hızlı prototip geliştirmeye de imkan sağlamaktadır.

## MATLAB Kullanım Alanları

**MATLAB**, matematik-istatistik, optimizasyon, Yapay Sinir Ağları (**YSA**), bulanık mantık, işaret ve görüntü işleme, kontrol tasarımları, yüneylem çalışmaları, tıbbi araştırmalar, finans ve uzay araştırmaları gibi çok çeşitli alanlarda kullanılmaktadır. **MATLAB**, kullanıcıya hızlı bir analiz ve tasarım ortamı sağlar.

- ❖ **MATLAB** programını C/C++ diline dönüştürebilir,
- ❖ 20. dereceden bir denklemin köklerini bulabilir,
- ❖  $100 \times 100$  boyutlu bir matrisin tersini alabilir,
- ❖ Bir elektrik motorunu gerçek zamanda kontrol edebilir,
- ❖ Bir aracın süspansiyon sisteminin benzetimini yapabilir,
- ❖ Mühendislik alanlarında karşılaşılan problemlere pratik ve hızlı sonuçlar sunabilir.

Bu nedenle Matlab, tüm dünyada binlerce endüstri, devlet ve akademik kurumlarda yaygınca kullanılmaktadır. Özellikle tüm üniversitelerde yaygın olarak kullanılmakta ve çeşitli derslerde ve kurslarda öğretilmektedir.

### **MATLAB Kullanan Şirketler:**

Dünyada Boeing, DaimlerChrysler, Motorola, NASA, Texas Instruments, Toyota ve Saab vd.,

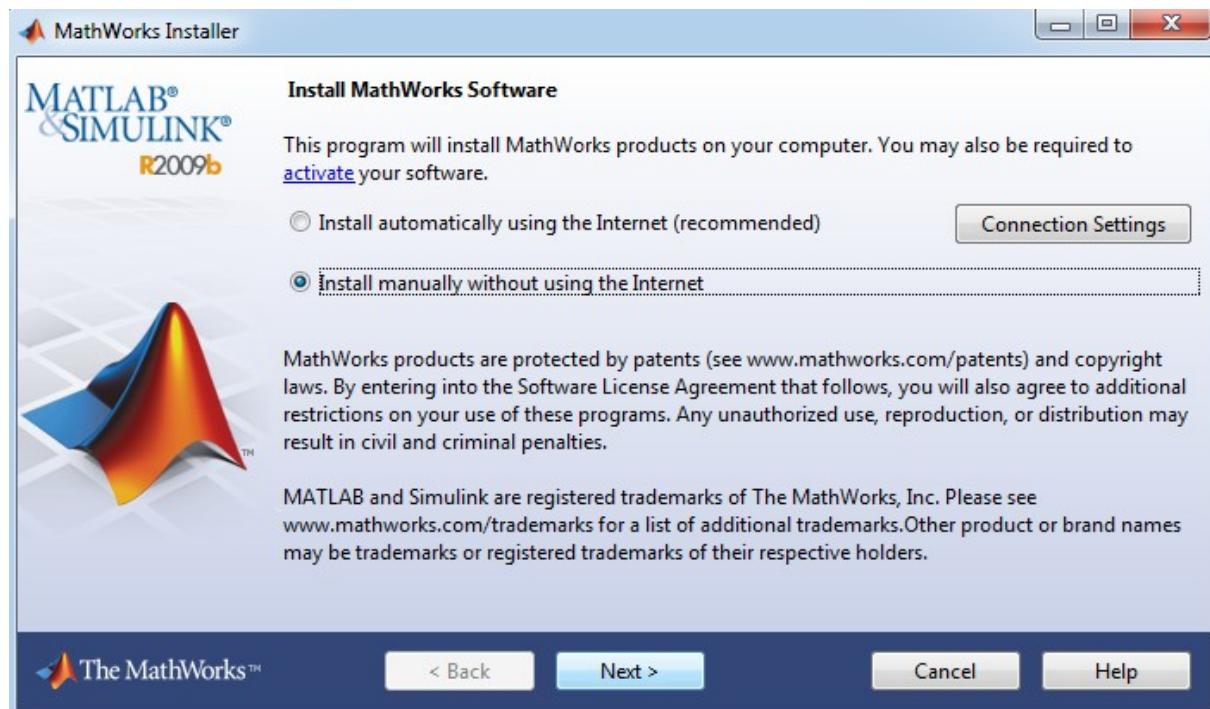
Ülkemizde ise Aselsan, Tofaş, Arçelik, Siemens, Alcatel, Garanti Bankası, Deniz Kuvvetleri, vd. sayılabilir.

### **MATLAB Kullanımında Temel Kurallar:**

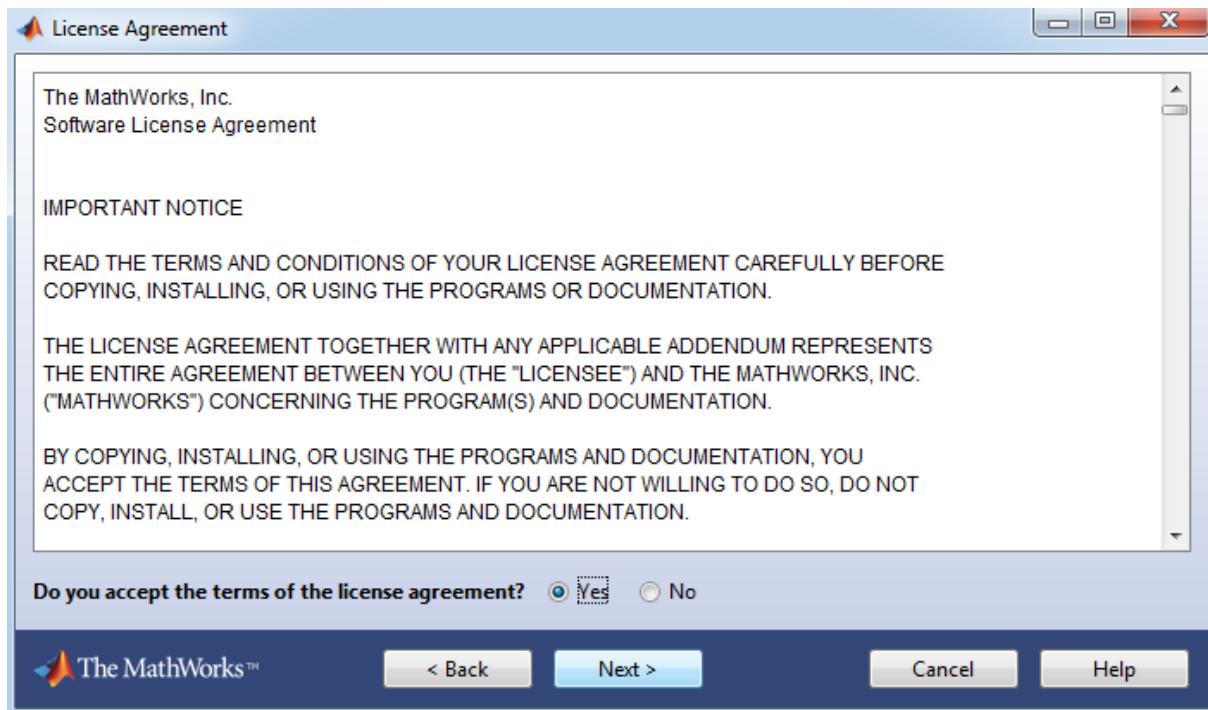
- ❖ Programın dili ve yardım bilgileri tamamen ingilizcedir.
- ❖ Komut temelli yazılımdır.
- ❖ Küçük-büyük harf ayrimı vardır. tot ve Tot farklı algılanır.
- ❖ » işaretи komut iletisidir (prompt).
- ❖ Komut satırında komutlar Enter tuşuna basılarak yürütülür.

### **MATLAB Kurulum**

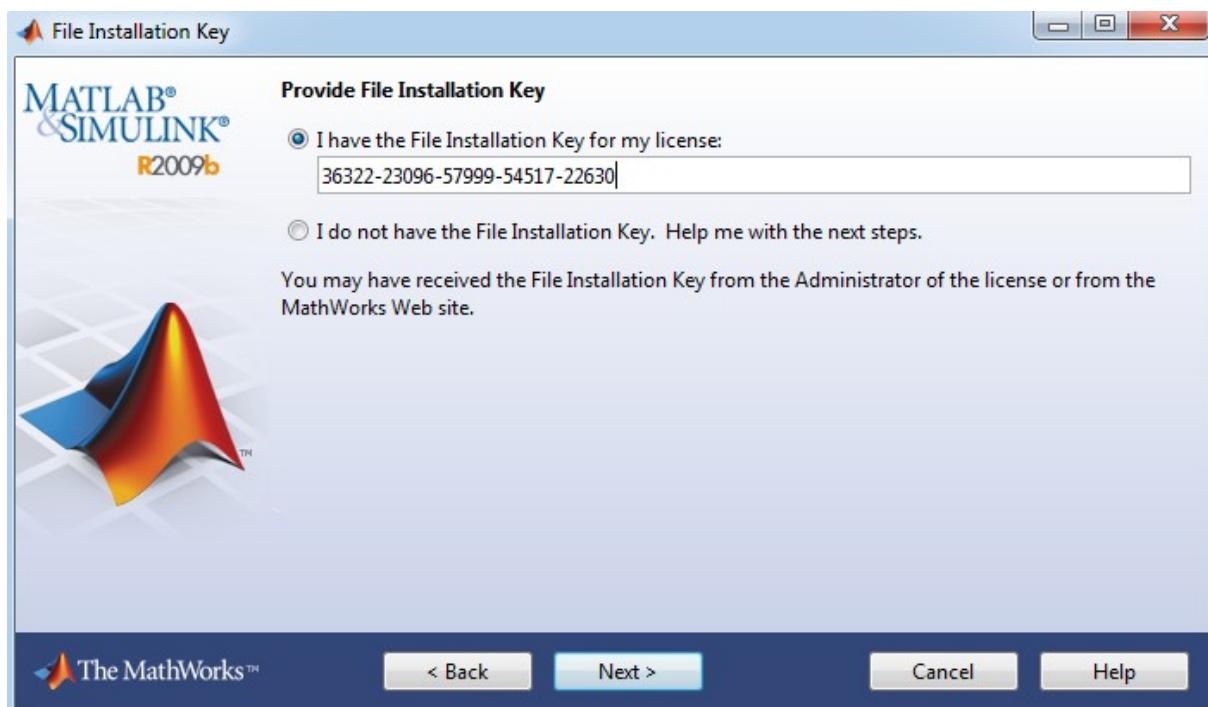
- 1- İlk olarak kurulum dosyası içerisindeki setup.exe dosyasını açınız.
- 2- Hoş geldiniz ekranında **Install without using the Internet** bölümünü seçikten sonra next butonuna tıklayıp kurulumu başlayın.



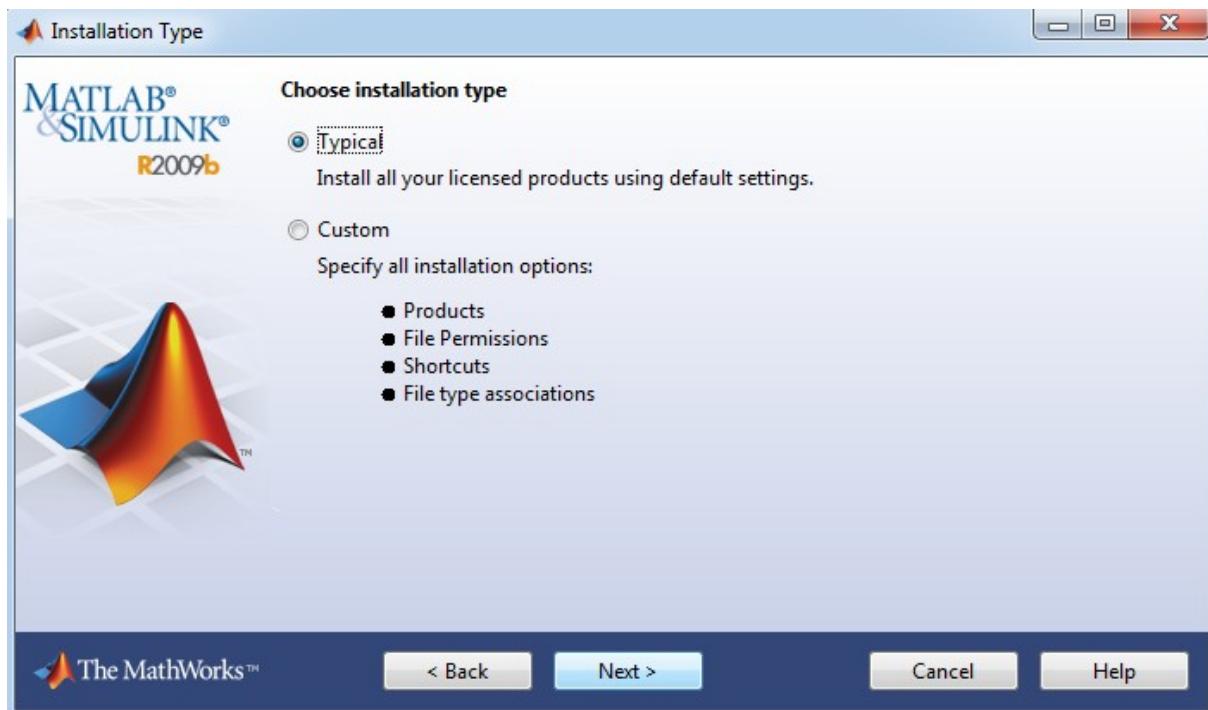
3- Lisans sözleşmesi ekranında sözleşmeyi kabul edip, next ile devam edin.



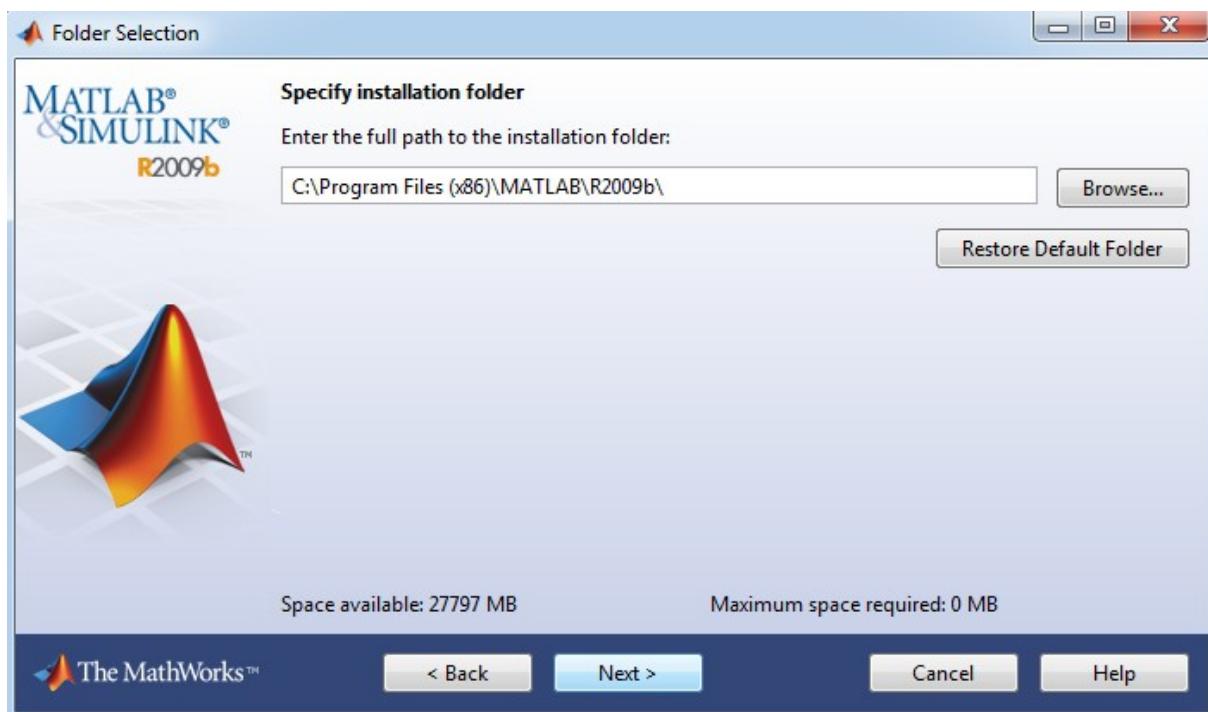
4- Dosya sunucusunda yer alan Matlab2013a\_Lisans\_key.txt dosyası içindeki key'i girdikten sonra next dierek devam edin.



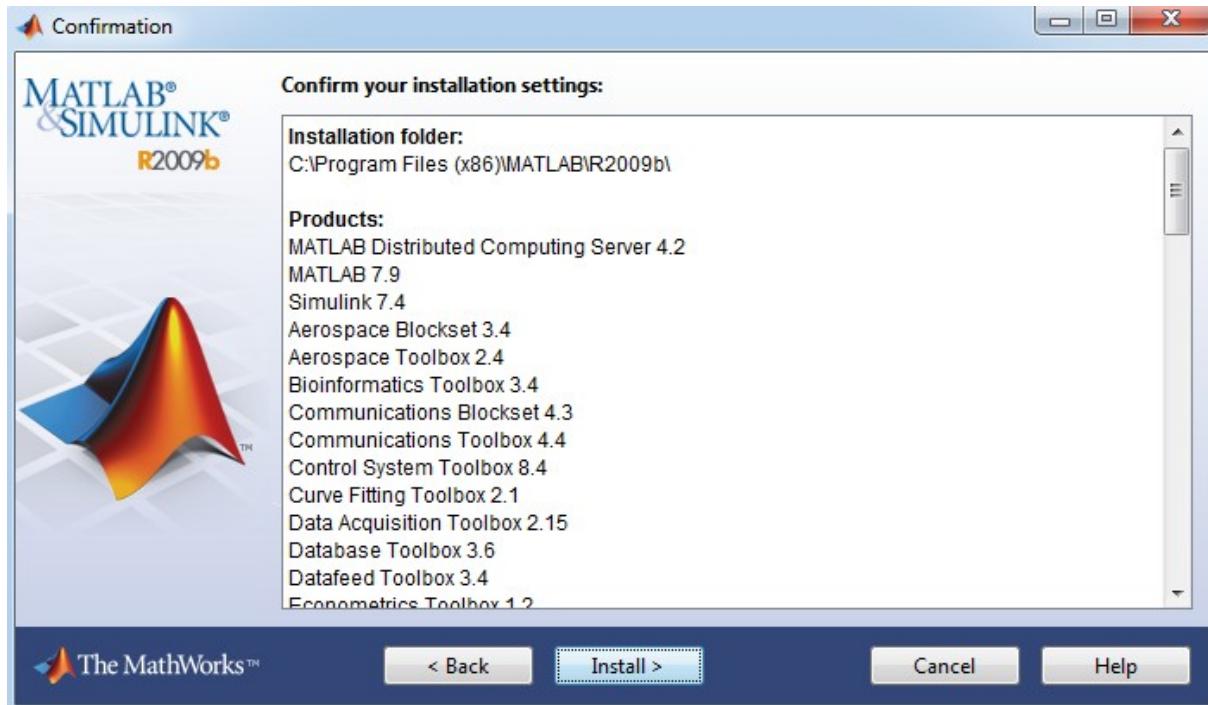
- 5- Bu kısımda karşımıza gelen ekrandan Typical kurulum seçeneğini seçip next dierek devam ediyoruz.



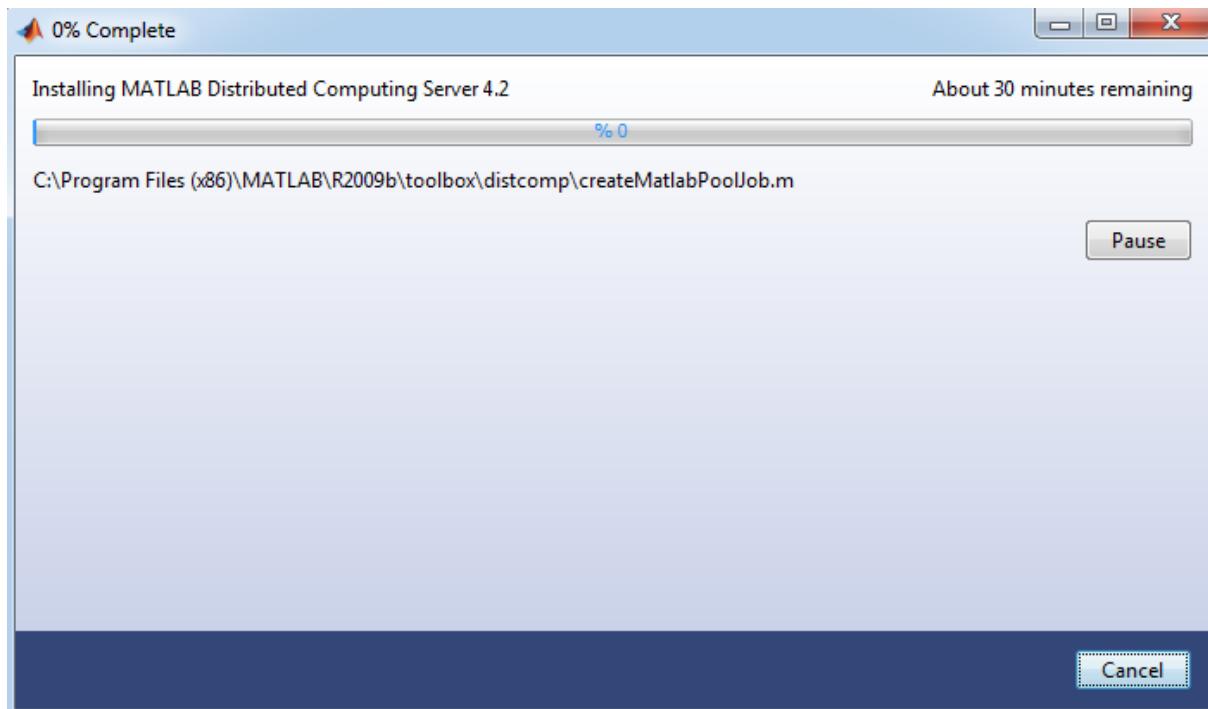
- 6- Karşımıza gelen yeni ekranda hiçbir değişiklik yapmadan direk next seçeneğini tıklıyoruz.
- 7- Browse'u tıklayıp dosya sunucusundan edindiğimiz License.dat dosyasının yolunu gösterip next ile devam ediyoruz.



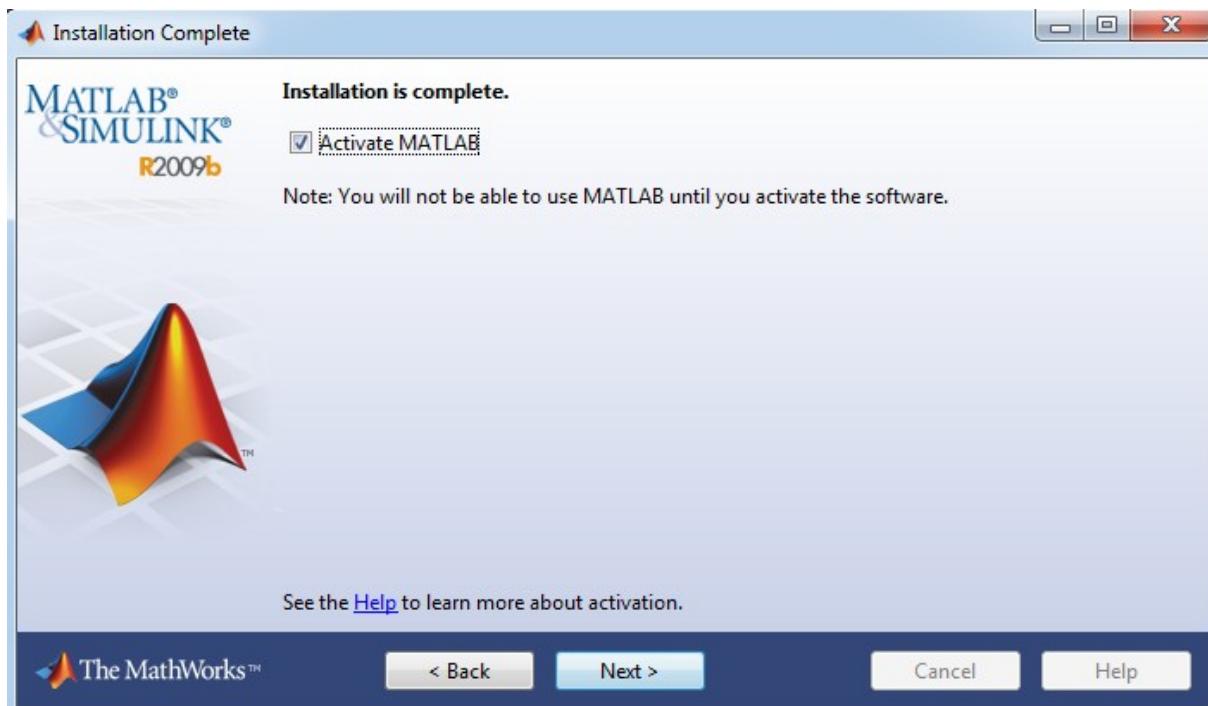
- 8- Kurulum bileşenleri otomatik seçildiği için karşımıza gelen yeni ekranda hiçbir değişiklik yapmadan install seçeneği ile devam ediyoruz.



- 9- Kurulumun tamamlanmasını bekliyoruz. İşlem tamamlandıktan sonra karşımıza gelen ilk ekranda next seçeneği ile devam ediyoruz.



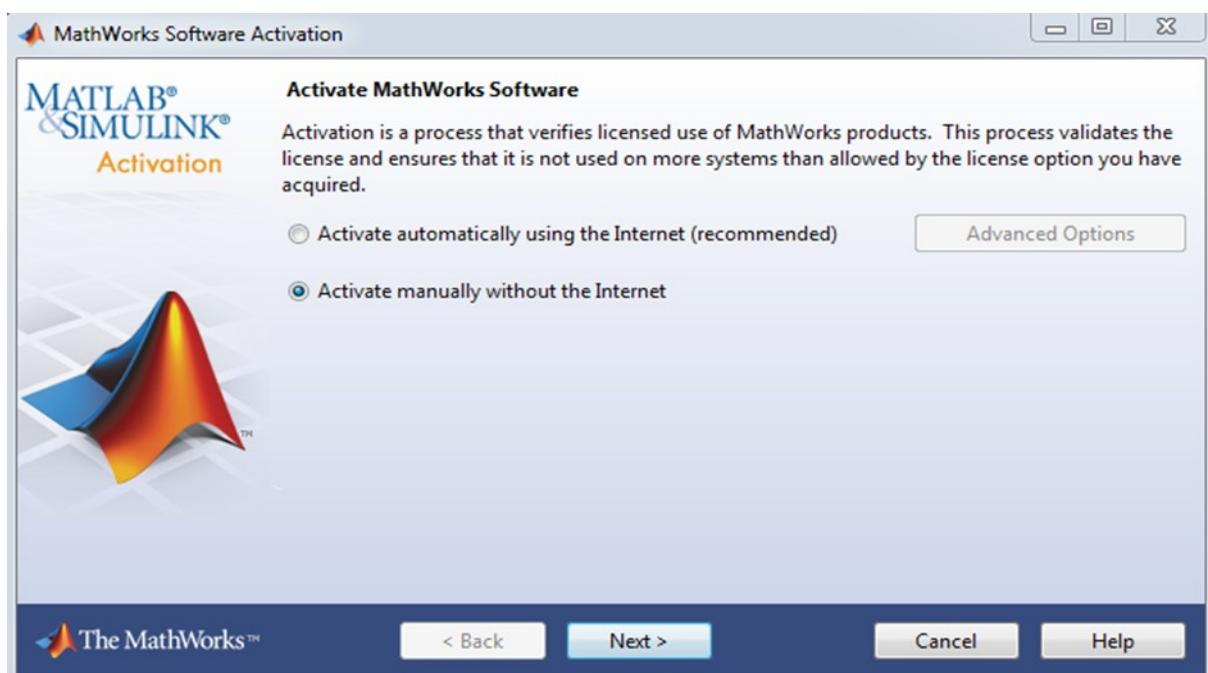
10- "Installation is complete" yazısını gördükten sonra Finish'e tıklayıp **MATLAB'ı** çalıştırabiliriz.



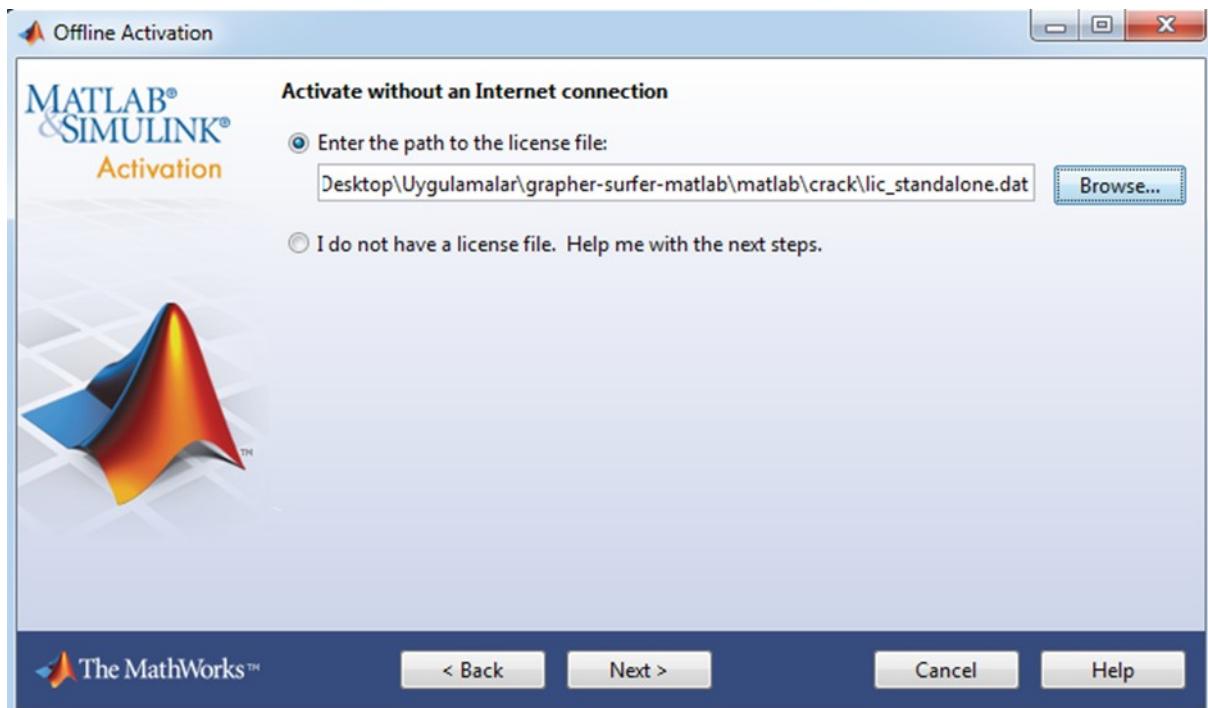
## MatLab Aktivasyonu

**MATLAB'ı** sorunsuz bir şekilde kullanabilmek için öncelikle aktivasyon işlemini gerçekleştirmemiz gereklidir. Bunun için aşağıdaki işlemleri sırasıyla gerçekleştiriyoruz.

- 1- Aktivasyon işlemini başlattıktan sonra karşımıza gelen ilk ekranda "Aktivate manually without internet" seçeneğini işaretleyip next tıklıyoruz.



- 2- Daha sonra karşımıza gelen ekranda gerekli olan “lic\_standalone.dat” dosyasının yolunu gösteriyoruz ve next ile devam ediyoruz.

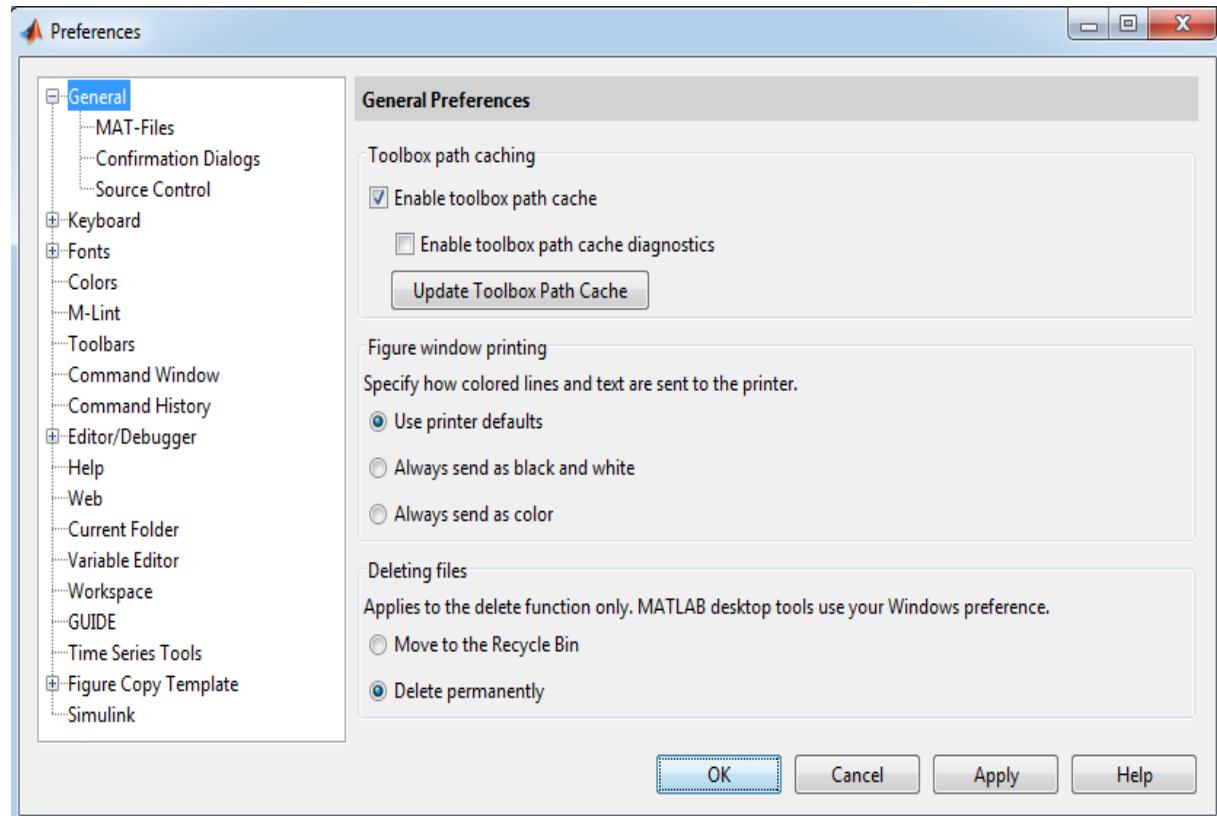


- 3- Aktivasyon işlemini tamamladıktan sonra Finish diyerek **MATLAB** programını çalıştırıp kullanmaya başlayabiliriz.



## **MatLab Tercihler(Preferences)**

**MATLAB** ile ilgili bütün düzenlemeleri ve ayarları yapabileceğimiz bölümdür. Bu bölümde ulaşırken MatLab’ın Başlat Menüsündeki(Start) Preferences seçeneğini seçiyoruz. Karşımıza aşağıdaki gibi bir ekran gelmektedir. Bu ekrandan Genel(General), Klavye(Keyboard), Yazı(Fonts), Renk(Colors), Toolbars(Araç Çubuğu), Komut Penceresi(Command Window), Editor, Web ve Simulink ayarlarının yanı sıra daha pek çok ayarlamayı yapabilmekteyiz. Bu ayarlardan işimize yarama ihtimali fazla olan Fonts yani Yazı ayarlarını aşağıda göstereceğiz.



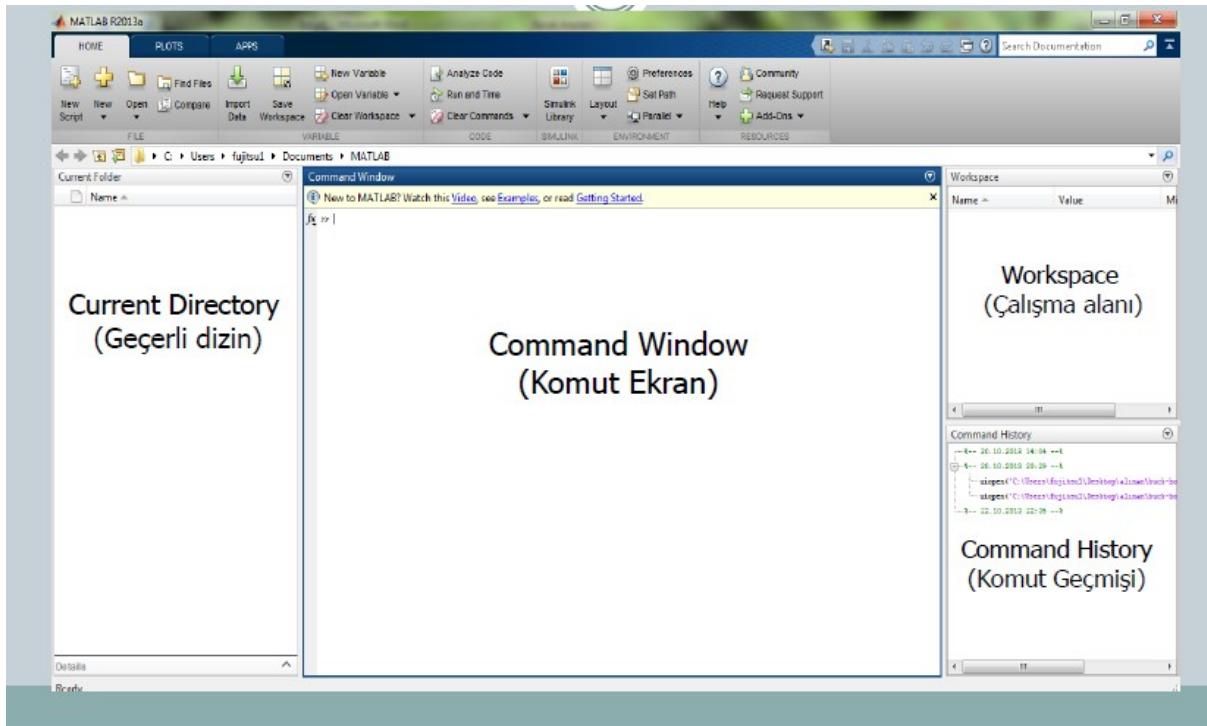
### **Programlama Ortamının Yazı Tipinin, Yazı Biçiminin ve Yazı Boyutunun Değiştirilmesi**

**MATLAB** programlama ortamının varsayılan yazı tipi **Monospaced**, yazı biçimi **Plain** ve yazı boyutu **12**'dir. Kullanıcılar yazı karakterleri üzerinde istedikleri değişiklikleri yapabilme yetkisine sahiptirler ve bu işlem için aşağıdaki adımları sırasıyla takip ederler.

- 1- **MATLAB'** in Başlat menüsünü(Start) ve sonrasında Preferences seçeneğini tıklıyoruz.
- 2- **Fonts** kategorisini seçili hale getiriyoruz.
- 3- Aşağı açılır menü oklarını kullanarak önce yazı tipini, daha sonra yazı biçimini ve en sonunda yazı boyutunu belirliyoruz.
- 4- İşlemleri sırasıyla gerçekleştirdikten sonra **OK** düğmesini tıklayarak işlemi sonlandırıyoruz.

## **MATLAB Çalışma Penceresi**

**MATLAB** programını çalıştırduğumızda karşımıza gelen ekran. Çalışma Alanı(Workspace), Komut Penceresi (Command Window), Çalışma Dizini(Current Folder), Komut Geçmişİ(Command History) ve Metin Düzenleyici(Editor) gibi alanların bulunduğu bir ekrandır. Bu bölümler sayesinde birçok işlem yapabilmekteyiz. Aşağıda bu bölümler tek tek incelenecektir.



## **Komut Penceresi(Command Window)**

Kullanıcının **MATLAB** ile etkileşim kurduğu ve en fazla kullandığı ana bileşendir. **MATLAB** yorumlayıcısı (interpreter), kullanıcıdan gelecek komutları kabul etmeye hazır olduğunun kullanıcıya bildirebilmek amacıyla komut penceresi üzerinde >> simgesini kullanır. Komut penceresinin bu simgeyi barındıran satırına **Komut Satırı(Command Prompt)** adı verilir. Komut penceresi, istenildiğinde üzerinde aritmetik işlemler gerçekleştirebilecek bir hesap makinesi ve yine istenildiğinde **MATLAB** komutlarını **MATLAB** yorumlayıcısına iletecek bir ara yüz vazifesi görür. Komut penceresi sayesinde gerçekleştirilebilecek bu iki ayrı işlev aşağıdaki örnekler sayesinde daha iyi anlaşılabılır. Örneğin  $2*2^3+14/7$  matematiksel işleminin sonucu, bu ifade komut satırına yazıldıkten sonra **ENTER** tuşuna basılarak öğrenilebilir. \* operatörünün **MATLAB** için çarpma, + operatörünün **MATLAB** için toplama ve / operatörünün **MATLAB** için bölme operatörünü olduğunu, ayrıca herhangi bir değişkene atanmayan sonuçların özel bir **MATLAB** değişkeni içerisinde (**ans**) saklandığını daha sonraki bölümlerimizde detaylı olarak öğreneceğiz.

```
>>2*23+14/7
```

```
ans =
```

```
48
```

Ya da **MATLAB'**ın kullanıma-hazır fonksiyonları sayesinde komut penceresi kullanılarak aşağıdaki gibi bir **MATLAB** deyimi ile 1 ile 1000 arasında rastgele bir tam sayı üretilebilir.

```
>>round(1+999*rand(1))  
ans =  
815
```

**İPUCU:** Yukarıdaki 2 örnekte de ENTER tuşuna basmadan önce satır sonunda noktalı virgül (;) karakterini kullanmayarak işlem sonucunun ekranda görüntülenmesini sağladık. Öte yandan, özellikle program yazarken, birçok sonucun ya da yapılan değişken atamalarının ekranda gereksiz yere görüntülenmesi istenmez. Komut penceresinde ya da MATLAB program dosyalarında satır sonuna konulacak olan noktalı virgül(;) karakteri sayesinde bu durumun önüne geçilmiş olur. Programcı ise, ekranda görüntülenmeyen bu sonuçların kaydına gerekiğin de çalışma alanı penceresi üzerinden ulaşabilir.

**İPUCU:** İstenildiğinde komut penceresini temizlemek için **clc** adlı MATLAB komutundan faydalananır.

## Çalışma Alanı(*Workspace*)

Yazdığımız programlar içerisinde veya komut penceresi üzerinde tanımladığımız bütün değişkenlerin kaydının tutulduğu yer çalışma alanı penceresidir. Bu pencere özellikle programlarımızdaki mantıksal hataları ayıklarken bize oldukça faydalayan bir penceredir.

**İPUCU:** Komut penceresinde yeni bir program çalıştırmadan hemen önce veya istedigimiz her an, halihazırda çalışma alanında tanımlı olan ve düşük bir ihtimalle de olsa çakışmalara sebep olabilecek tüm değişkenleri çalışma alanı penceresinden temizlemek için **clear** adlı MATLAB komutundan faydalanzıız.

## Çalışma Dizini(*Current Folder*)

Kullanıcı tarafından yazılan ve bir dosya içerisinde saklanan **MATLAB** programlarının komut penceresinden çağrılabilmesi, bu program dosyalarının içerisinde bulunduğu dizinin çalışma dizini(mevcut dizin) olarak **MATLAB'**e tanıtılmış olması ile mümkündür. **MATLAB**, ancak ve ancak çalışma dizini olarak tanımlanmış dizin altındaki program dosyalarına komut penceresi üzerinden ulaşır. Çalışma dizini penceresi sayesinde gerekli dizinin yolu istenildiğinde değiştirilebilir.

## **Komut Geçmişi(Command History)**

Kullanıcının komut penceresi üzerinde çalıştığı bütün komutların/programların kaydının tutulduğu yer komut geçmişi penceresidir. Bu pencerede listelenen herhangi bir komut çift tıklanarak ya da komut penceresi üzerine fare yardımıyla sürüklenebilir.

**İPUCU:** Komut geçmişi penceresi, istenildiğinde fare ile sağ tıklanarak ve “**Clear Command History**” seçeneği kullanılarak tamamen temizlenebilir.

**İPUCU:** Komut penceresi aktif pencere iken geçmişte kullandığımız komutları yukarı ve aşağı ok tuşları ile de ekrana getirebiliriz. Örneğin son çalıştığımız komuta, eğer gerekiyorsa, baştan yazmak yerine yukarı ok tuşu ile çabucak ulaşabiliriz. İmlecini, sol ve sağ ok tuşları yardımıyla da yapılan herhangi bir yanlışlığı düzeltmek amacıyla istenilen pozisyon'a götürebiliriz.

## **Metin Düzenleyici(Editor)**

Herhangi bir metin düzenleyicisi kullanılarak **MATLAB** programları yazılabılır ve çalıştırılabilir. Zira dosyalarda saklanan **MATLAB** programları çok kabaca bir problemi çözmeye yönelik, alt alta yazılmış anlamlı **MATLAB** metinlerinden oluşan düzyazı metinleridir. Ama **MATLAB'**ın bizlere programlama ortamı içerisinde sunduğu kendi metin düzenleyicisi oldukça fonksiyoneldir. Bu düzenleyici anahtar kelimeleri tanır ve renklendirir. Yapılan yazılım hatalarına yönelik dinamik uyarılar verir. Bir blok/bölge içerisindeki komutları otomatik olarak içeri girintiler. Yazdığımız programlardaki mantıksal hataları bulabilmek için kesme noktaları kullanabilememize ve böylece hata ayıklamaya (debugging) imkan tanır.

**İPUCU:** **MATLAB** programlama ortamındaki bütün pencereler **MATLAB'**ın pencere başlıklarını sürükle-bırak özelliği sayesinde ekranın herhangi bir yerinde konumlandırılabilirler ve dolayısıyla bu ortam kişiselleştirilebilir bir ortamdır. Bu pencerelerden herhangi birisini yanlışlıkla kapattığımızda veya ortamı ilk açılıştaki haline getirmek istediğimizde **Desktop Menüsü** > **Desktop Layout** > **Default** yolunu takip etmemiz yeterlidir.

## **MATLAB Yardım Komutları**

Yardımcı komutlar başlığı ile söz edilen komutlar **MATLAB** komut satırında kullanılabilen sistem komutları, ekran ve hafıza ile ilgili komutlardır. **MATLAB** programı büyük/küçük harf duyarlıdır (case sensitive). Bu nedenle komutlar küçük harfler ile kullanılmalı, değişken tanımlamalarında da küçük ve büyük harf değişkenlerin farklı rakamları isimlendirmek için kullanılabileceği dikkat edilmelidir.

**dir:** DOS işletim sisteminde kullanıldığı gibi aktif dizin içerisindeki dosyaların listesini görmek için kullanılan komuttur.

**pwd:** Aktif dizini görüntülemek için kullanılan komuttur.

**clc:** DOS işletim sistemindeki CLS (CLear Screen) komutu ile aynı işlevi gören komuttur. Ekranı temizlemek için kullanılır.

**whos:** MATLAB programında o ana kadar tanımlanmış tüm değişkenleri görüntüler.

**clear:** MATLAB programında o ana kadar tanımlanmış tüm değişkenleri hafızadan silmek için kullanılan komuttur.

**cd:** MATLAB programı komut satırından aktif dizini değiştirmek için kullanılan komuttur.

**help:** MATLAB programında bir komuta ait yardım istenildiğinde komut satırında yardım alınacak komut ile birlikte kullanılır.

**doc:** MATLAB yardım pencersinde daha detaylı ve örnekli yardım dosyasını gösterir.

**lookfor:** MATLAB programı içerisinde genel bir arama yapmak için kullanılan komuttur.

## **MATLAB Özel Karakterler ve Yazım Notasyonları**

MATLAB'da ifadeler formunda veya MATLAB kontrolünü kullanabilmeniz için özel karakterlere sahiptir. Bu özel karakterler köşeli parantez, normal parantez, nokta, üç nokta, noktalı virgül, yüzde işaretti, ünlem işaretti, iki nokta ve tırnak işaretleridir. Aşağıda bu özel karakterlerin kullanım şekilleri sırayla gösterilecek ve örnekler yapılacaktır.

**[ ]-Köşeli Parantez:** Vektörleri ve matrisleri biçimlendirmek için kullanılır. Örneğin [5.45 9.3 sqrt(-6)] elemanları boşluklarla ayrılmış üç elemanlı bir vektördür. Bunun yanında [5 6 45;12 91 2], üç satır ve üç sütundan ibaret 3x3 lük bir matrisi gösterir. Burada noktalı virgül (;) matris oluşturulmasında bir satırı sona erdirip bunu izleyen satırın olmasını sağlar. A=[ ] biçiminde bir bildirim boş bir matrisi A değiştirecektir.

**( )-Normal Parantez:** MATLAB da normal parantez çeşitli çalışmalarla çeşitli yerlerde kullanılır. En önemlisi; aritmetik deyimlerin üstünlüğünü belirmek ve fonksiyon argümanlarını kapatmak için kullanılır. Eğer X ve Y bir vektör ise X(Y) de bir vektördür ve [X(Y(1)),X(Y(2)),...,X(Y(N))] şeklinde ifade edilir.

**.-Nokta:** Kesir ayırma işaretidir. 1452/10, 1.452 ve .1452e1 aynı anlama gelir. Bu işaret aynı zamanda eleman elemana çarpma ve bölme işleminde de kullanılabilir. Buna göre kullanış biçimleri .\* , .^ , ./ , .\ veya .' şeklinde edilir.

**...-Üç Nokta:** Bir tek satırı sıyrımayan ifadelerin devam ettiğini belirmek için kullanılır.

**; -Noktalı Virgül:** Bu işaret bir bildirimde elde edilen sonuçların programın içrası sonunda akranda görüntülenmesini engellemek için kullanılır. Aynı zamanda matris

oluşturulmasında bir satırı sona erdirip bunu izleyen satırın olmasını sağladığını da söylemiştir.

**%-Yüzde İşareti:** Açıklama yapılacak zaman ilgili satırlara bu işaretle başlanarak yapılır. **% işaretü** ile başlayan **MATLAB** tarafından dikkate alınmaz ve icraya da sokulmaz.

**!-Ünlem işaretü:** Herhangi bir yazı DOS işlemi görür. Böylece **MATLAB** içerisinde iken DOS komutları ile çalışılabilir.

**:-İki Nokta:** Bu işaret sütun işaretü olarak kullanılır ve her yerde kullanılabilir. Örneğin; J:K J'den başlayarak birer birer K ya kadar artan bir dizi oluşturur ve [J, J+1,...,K] ile aynı anlama gelir.

**'-Tırnak işaretü:** matrislerin transpozunun alınmasını sağlar. M', M matrisinin karmaşık eşlenik transpozonu M'. ise eşlenik olmayan transpozunu sonuçlandırır.

Aşağıda anlatılan özel karakterler ile ilgili örnek gösterilmiştir.

```
>>A=[1 2;3 4];           %2x2lik bir matris tanımlaması.  
>> A*A/A             %A matrisi üzerinde aritmetik işlemler yapılır.  
ans =  
    1   2  
    3   4  
>> A*A-A^2  
ans =  
    0   0  
    1   0  
>> a1=A(1)            %A matrisinin ilk elemanı bir değişkene atanır.  
a1 =  
    1
```

## **MATLAB Değişkenler ve Değişken Tanımlama Kuralları**

Bilgi saklamak amacıyla kullanılan programlama yapısına değişken denir. Değişkenler, **MATLAB** deyimleri içerisinde, bünyelerinde sakladıkları sayısal değerleri temsil amacıyla kullanılırlar. Diğer bütün programlama dillerinde olduğu gibi **MATLAB'de** de değişken tanımlarken dikkat edilmesi gereken kurallar vardır:

- ❖ Değişken isimleri daima bir harf ile başlamalıdır. İlk harfi başka harfler, rakamlar ve alt çizgiler(\_) izleyebilir. Örneğin **1sayı** uygun bir değişken ismi değilken, **sayı\_1** kullanılabilir bir değişken adıdır.
- ❖ Değişken isimleri noktalama işaretleri ve boşluk barındırmamalıdır. **Sayı.1** ve **sayı 2** şeklinde tanımlanan değişken adları bu kural ihlaline iki güzel örnektir.

- ❖ Değişken isimleri içerisinde Türkçemize özgü küçük veya büyük “ç, ğ, ı, ö, ş, ü” karakterleri kullanılmamalıdır. Komut penceresinde **sayı1** adında bir değişken tanımlamaya çalışmak **MATLAB** tarafından hatanın yeri de işaret edilerek aşağıdaki gibi bir hata mesajı oluşturulmasına sebep olur.

```
>> sayı1=3
??? sayı1=3
```

|  
*Error: The input character is not valid in MATLAB statements or expressions.*

**İPUCU:** Bizim tarafımızdan yazılan bir bilgisayar programının bir başkası tarafından anlaşılmasını kolaylaştırmak için programlarımıza açıklayıcı satırlar eklemeliyiz ve anlamlı değişken ve fonksiyon isimleri kullanmalıyız. Değişkenler ve daha sonra göreceğimiz fonksiyonlar muhakkak kullanım amaçları dikkate alınarak ve anlamalı bir şekilde isimlendirilmelidirler. Örneğin kullanıcidan klavye yoluyla alınacak ve ortalamaları hesaplanacak üç sayı **a, b, c** veya **x, y, z** gibi jenerik değişken isimleriyle değil **sayı1, sayı2, sayı3** gibi anlamlı isimlerle temsil edilmelidir. Bir zorunluluk olmasa bile yazılan bir programın anlaşılı bilirliğini artttırmak adına değişken ve fonksiyonlar adlandırılırken bazı notasyonlardan faydalanjır.

**Deve ve Paskal Notasyonları:** Yukarıdaki ipucunda anlatılanlara imkan sağlayan notasyonlar deve ve paskal notasyonlarıdır. Bu göstereceğimiz notasyonlar dünyada genel kabul görmüşlerdir ve diğerlerine göre daha fazla tercih edilirler. **Deve notasyonu** değişken tanımlamalarında, **Paskal notasyonu** ise dosya isimlendirmelerinde (düzyazı ve fonksiyon m-dosyaları) kullanılırlar. **Deve Notasyonu** değişken adı birden fazla değişken içeriyorsa ilk kelime hariç diğer bütün kelimelerin ilk karakterlerinin büyük harfle yazıldığı bir notasyondur. **Deve Notasyonu** ile tanımlanmış değişken isimlerine birkaç örnek olarak **sayı1, enKucuk, bankaHesapNo** ve **basamakSayısı** verilebilir.

Dosya isimlendirmelerde kullandığımız **Paskal Notasyonunun Deve Notasyonundan** tek farkı dosya adını oluşturan ilk kelimenin ilk karakterinin de büyük harfle yazılıyor olmasıdır. **Paskal Notasyonu** ile tanımlanmış dosya isimlerine örnek olarak **AsalSayiMi.m, OrtalamayıHesapla.m** ve **ToplayarakCarpma.m** verilebilir. **.m**, daha sonra da öğreneceğimiz üzere **MATLAB** programlama dosyalarının uzantısıdır.

## **MATLAB Sayılar ve Sabitler**

**MATLAB** de negatif bir sayıyı temsil için o sayının önünde (-) işaretı vardır. Bir sayının önüne (+) işareti koyulması veya hiçbir işaret koyulmaması o sayının pozitif olduğunu delilidir. Kayan noktalı (ondalık) sayıarda ayrıc olarak nokta(.) ve 10 sayısının kuvvetini temsilen e harfi kullanılır. Karmaşık sayıların sanal kısımları **MATLAB** de son takı olarak i harfi kullanır. Bu bilgiler ışığında **MATLAB** yorumlayıcısının anlayacağı birkaç

rakama örnek olarak **6**, **-175**, **3.1416**, **2.54e2**, **7i** veya **5+7i** verilebilir. Aşağıda **MATLAB'** da kullanılan bazı sabitler ve kullanım amaçları açıklanmıştır.

**Ans:** Bu değişken bir deyim tarafından hesaplanan fakat bir değişken ismi altında saklanmayan değerleri saklamak için kullanılır.

**inf ( $\infty$ ):** Bu kelime **MATLAB'** ta sonsuz değeri için atanmış bir değişkendir ve sıfıra bölme işlemlerinde ortaya çıkar. Eğer sıfıra bölme işlemi görüntülenmek istenirse bir uyarı mesajı alınır ve sonuç  $\infty$  işaretini şeklinde görüntülenir veya basılır.

**NaN:** Bu değer Not-a-Number (rakam değil) anlamına gelir ve sıfır bölümü sıfır bölümünde olduğu gibi tanımlanmamış deyimlerde ortaya çıkar.

**Eps:** Bu değer fonksiyon, kullanılmakta olan bilgisayar için floating point (virgüllü sayılar) tamlığını içerir. Bu epsilon tamlığı 1.0 ve bunu izleyen en büyük decimal (onlu sayılar) arasındaki farktır.

## **MATLAB da Diziler**

Dizi en genel matematiksel tanımı ile nümerik ve metinsel değerler topluluğudur. **MATLAB** da her şey dizi olarak işleme konur ve en temel veri elemanıdır. Bir dizinin eleman sayısı, satır ile sütun sayısının çarpımıdır.

- ❖ Reel sayıları ve karmaşık sayıları ifade eden çiftkat veya nümerik diziler.  
Metin ifade eden diziler. Hücre diziler.

Bir nümerik dizi, skaler, vektör veya matris olabilir ve tüm nümerik diziler double array formatındadır.  $1 \times 1$  dizisi, bir **skaler (scaler)** gösterir. ( $a=3, b=-6.5$ ).  $m \times 1$  veya  $1 \times n$  dizisi, bir **vektör (vector)** gösterir.  $m \times n$  veya  $n \times m$  dizisi, bir **matris (matrix)** gösterir. Bu çerçevede  $1 \times 1$  dizisi sabit matris veya tek elemanlı matris,  $n \times 1$  dizisi sütun matrisi ve  $1 \times n$  dizisi ise satır matrisi olarak da düşünülebilir.

**Karakter Dizisi(string):** Elemanları rakamlar yerine karakterler olan satır vektörüne karakter dizisi, katar veya string adı verilir. Karakter dizileri ya açılıp kapanan tek tırnaklar ("") içerisinde ya da dizinin her bir karakterinin açılıp kapanan tek tırnaklar içerisinde yer aldığı bir satır vektörü formunda tanımlanırlar. Bu her iki metodun da kullanımına birer örnek aşağıdaki **MATLAB** komut penceresinde verilmiştir.

```
>>isim='deniz'
isim=
deniz
>>isim=['d' 'e' 'n' 'i' 'z']
isim=
deniz
```

## MATLAB da Vektörler

**Kısaca Matris Tanımı:** Bir matrisi **MATLAB** programlama ortamına aktarabilmek için açılıp kapanan **köşeli parantezlerden([ ])** faydaladığımızı daha önceki konularda bahsetmiştik. Matrisin her bir satırındaki elemanlar birbirinden bir boşluk veya virgül karakteri ile ayrılr. Matrisin bir alt satırına inebilmek için ise **noktalı virgül(;)** karakteri kullanılır. Yani iki boyutlu bir matris **MATLAB** komut penceresine şu şekilde taşınır.

```
>> Matris=[1 2 3;4 5 6]
Matris =
1 2 3
4 5 6
```

İki boyutundan herhangi bir tanesi **1** değerine sahip olan matrlslere **vektör** adı verilir. Yani vektör özel bir matristir.(Her vektör bir matristir ama her matris bir vektör değildir.) Satır sayısı yani ilk boyutu 1 olan vektörler **satır vektörleri**, sütun sayısı yani ikinci boyutu 1 olan vektörler ise **sütun vektörleri** olarak adlandırılırlar. **n** elemanlı bir satır vektörünün boyutu  **$1 \times n$**  ve **n** elemanlı bir sütun vektörünün boyutu da  **$n \times 1$** 'dir. Yapılan bu tanımlardan hareketle matrislerin her bir satırının aslında bir satır vektörü olduğunu ve matris satırlarının **MATLAB** programlama ortamına aktarılması için kullanılan kuralın satır vektörleri içinde geçerli olduğunu söyleyebiliriz. Dolayısıyla satır vektörlerini ve sütun vektörlerini, vektör

elemanlarını açılıp kapanan köşeli parantezler içerisinde birbirinden boşluk veya virgül ile ayırarak tanımlarız ve **MATLAB** değişkenine aşağıdaki şekilde atarız.

### **MATLAB' da Vektör Tanımlama:**

***satir\_vektor = [1 2 3] veya [1, 2, 3] : sütun\_vektor = [1; 2; 3]***

Tanımladığımız vektörler MATLAB'ın çalışma alanında isim olarak görünecektir.

**MATLAB' da Bir Vektörün Transpozu:** **MATLAB'** da bir vektörün transpozunu alma işlemi satır ve sütunların yer değiştirmesi durumudur. Yani satır vektör olarak tanımlanmış bir x vektörü **x'** ifadesi ile sütun vektöre çevrilmiş olur. Aşağıdaki y vektörü x vektörünün transpozudur.

```
>> x=[1 3 5];
>>x(3)          %x vektörünün 3. Elemanına ulaşılır.
ans=
5
>>y=x'
y =
1
3
5
```

**MATLAB'** da vektör tanımlamanın başka bir yolu da aşağıda verilmiştir. Eğer vektörü oluşturacak elemanlar arasında bir kural varsa (belli bir artım miktarı ile artarak devam ediyorsa), bu durumda bir vektörü aşağıdaki gibi tanımlayabiliriz.

***tek\_sayılar = 1 : 2 : 11  
1 = başlangıç değeri , 11 = bitiş değeri , 2 = adım büyüklüğü***

```
>> tek_sayılar=1:2:11
tek_sayılar =
1 3 5 7 9 11
```

**MATLAB' da linspace komutunun kullanımı:** MATLAB' da vektör oluşturmanın diğer bir yolu da **linspace komutunun** kullanılmasıdır. Hatırlayacağımız üzere **tek\_sayılar=1:2:11** şeklinde başlangıç değeri, artım miktarı ve bitiş değerlerini belirterek bir vektör oluşturabiliyoruz. Benzer bir şekilde, **linspace komutunu** kullanarak; başlangıç,bitiş değeri ve **ELEMAN SAYISI' nı** belirterek de bir vektör oluşturabiliriz. Başlangıç değeri x1, bitiş değeri x2 olan n elemanlı eşit aralıklı bir dizi oluşturur. Örnek uygulama aşağıda gösterilmektedir.

```
>> sayılar=linspace(1,15,7)      %linspace(başlangıç,bitiş,eleman sayısı)
sayılar =
1.0000 3.3333 5.6667 8.0000 10.3333 12.6667 15.0000
```

Vektörlerin tamamını veya bir kısmını, matematik işlemlerine tabii tutabiliyoruz. Yapacağımız işlem vektörlerin tüm öğelerine etki eder. İsterseniz, 1 ile 2 arasına 5 tane sayı koyup daha sonra bu sayılar üzerinde farklı birkaç işlem yapıp ne demek istediğimi daha açık şekilde görelim.

```
>> vektor = linspace(1,2,5) %linspace ile 5 elemanlı vektör tanımı  
vektor =  
    1.0000  1.2500  1.5000  1.7500  2.0000  
>> vektor/2                      %vektör elemanlarının 2 ye bölünmesi  
ans =  
    0.5000  0.6250  0.7500  0.8750  1.0000  
>> sin(vektor)                  %vektör elemanlarının sinüsü  
ans =  
    0.8415  0.9490  0.9975  0.9840  0.9093  
>> exp(vektor)  
ans =  
    2.7183  3.4903  4.4817  5.7546  7.3891  
>> vektor                         % bu işlemlerden asıl vektör etkilenmez.  
vektor =  
    1.0000  1.2500  1.5000  1.7500  2.0000
```

**Vektör indisleri:** Bir vektörün elemanlarına atanılan değer değişkenler editörü veya eleman adresi vasıtasıyla değiştirilebilir. Vektör indisleri 1 den başlamaktadır. Örnek olarak, 5 elemanlı bir A vektörü oluşturalım ve A vektörünün 3. elemanını 27 ile değiştirelim. Benzer şekilde A vektörünün 2. elemanını silelim. Vektörün elemanına [ ] değeri atandığında eleman silinir.

```
>> A=[1 2 3 4 5];                 %5 elemanlı bir A vektörü tanımlanı.  
>> A(3)=27                        %3. Elemana 27 değeri atandı.  
A =  
    1   2   27   4   5  
>> A(2)=[]                          %2. Eleman silindi.  
A =1   27   4   5
```

Değer atanacak elemanın adresi eleman sayısından fazla ise aradaki elemanlara otomatik olarak 0 değeri atanır.

Örnek olarak 5 elemanlı A vektörünün 9. elemanına 12 değeri atanırsa, aradaki elemanlara 0 değeri atanır ve son elemanın değeri 12 olur.

```

>> A=[1 2 3 4 5];
>> A(9)=12
A =
    1   2   3   4   5   0   0   0   12
>> A(end)
ans =
    12

```

**Vektör İşlemleri:** Skalerler ile yapılan 4 işlem vektörün her elemanına uygulanır. Vektörler arasında yapılacak işlemler lineer cebir kurallarını sağlamak durumundadır.

Örnek olarak, iki vektörün birbirileyle çarpılabilmesi için ilk vektörün sütun sayısı ile ikinci vektörün satır sayısı birbirine eşit olmalıdır.

Vektörlerde eleman elemana işlemleri (.) operatörü ile gerçekleştirilir.

```

A = 1   2   3   4   5   0   0   0   12
>> A=A+3
A =
    4   5   6   7   8   3   3   3   15
>> A=A*2
A =
    8   10  12  14  16  6   6   6   30

```

Yeni bir vektörün oluşturulmasında hafızadaki vektörlerden istifade edilebilir. Benzer şekilde hafızadaki bir vektörün parçalarından da yeni vektörler oluşturulabilir.

```

>> A=[2 4 5 7];
>> AAA=[A A A]
AAA =
    2   4   5   7   2   4   5   7   2   4   5   7
>> B=AAA(1:4)
B =
    2   4   5   7

```

## Özel Vektör Yapıları

**zeros(100)**

**zeros(1,n) :** Tüm elemanları sıfır olan n elemanlı satır vektör.

**zeros(n,1) :** Tüm elemanları sıfır olan n elemanlı sütun vektör.

**ones(1,n) :** Tüm elemanları bir olan n elemanlı satır vektör.

**ones(n,1)** : Tüm elemanları bir olan n elemanlı sütun vektör.

**rand(1,n)** : Elemanları 0 ile 1 arasından rastgele seçilmiş n elemanlı satır vektör.

**rand(n,1)** : Elemanları 0 ile 1 arasından rastgele seçilmiş n elemanlı sütun vektör.

**randn(1,n)** : Ortalaması 0 ve standart sapması 1 olan normal dağılımlı elemanlardan oluşan n elemanlı sütun vektör.

**randn(n,1)** : Ortalaması 0 ve standart sapması 1 olan normal dağılımlı elemanlardan oluşan n elemanlı sütun vektör.

**r = randi([-10 10],100,1);** -10 ile 10 arasında 100x1 boyutunda bir matris üretir. R = randi([IMIN,IMAX],...) returns an array containing integer values drawn from the discrete uniform distribution on IMIN:IMAX.

### **MATLAB da Skalerler**

MATLAB, 1x1 boyutundaki özel matrisi skaler olarak adlandırır. MATLAB değişkenlerine atadığımız veya MATLAB deyimleri içerisinde basit aritmetik manipülasyonlarda kullandığımız tam sayılar ya da ondalıklı sayılar aslında skalerlerdir. Skalerler MATLAB komut penceresinde özel bir matris türü olmaları nedeniyle [ ] arasında da tanımlanabilirler ya da köşeli parantezler bu amaçla hiç kullanılmazlar. Skaler adlı bir değişkene komut penceresinden yapılan skaler atamalarına 2 örnek aşağıda gösterilmiştir.

```
>> skaler=[49]
skaler =
    49
>> skaler=49
skaler =
    49
```

**İPUCU:** Bir değişken atamasının en genel halini `değişkenAdı=değer` şeklinde indirgelyebiliriz. Burada `değer` denilen şey herhangi bir matematiksel ifade, bir skaler, bir vektör, bir karakter dizisi, bir matris veya bunların birden fazlasının MATLAB operatörleri ile birleştirilmiş bir kombinasyonu olabilir.

Matrisler ve vektörler üzerinde gerçekleştirilebilen işlemler skalerler üzerinde de gerçekleştirilebilir.

### **MATLAB da Genel Matris İşlemleri**

#### **Matris Tanımlama:**

A matrisi 3\*3 lük bir kare matris olsun, bu matrisi oluştururken komut satırına aşağıda ki komut girilir.

```
>> A=[15 2 4;3 7 11;1 2 9]
A =
15   2   4
3   7   11
1   2   9
```

Yukarıdaki kod parçasından da anlaşılacağı üzere yeni satır geçmek için ";" işaretini kullanılır. ; işaretini kullanmadan da matris oluşturmak mümkündür. Bunun için ";" yerine **enter** kullanılarak da matris oluşturulabilir.

**MATLAB** matris elemanları karmaşık sayılardan da oluşabilir.

```
>> B=[3+6i 8i;6+10i 14]
B =
3.0000 + 6.0000i    0 + 8.0000i
6.0000 +10.0000i   14.0000
```

Yukarıdaki komutu komut satırına girdiğimiz zaman yukarıdaki gibi bir matris görüntüsü elde edilir.

### **MATLAB' da bir matrisin elemanlarına ulaşma**

**MATLAB'** da tanımladığımız matrisler **MATLAB workspace'** inde saklanır ve orada görünür. Çalışma anında **MATLAB** komut ekranından matrisin tamamını çağrılabileceğimiz gibi matrisin bir parçاسını ya da bir elemanını da çağrılabilmekteyiz. Bunun için **A(satır\_indeksi ; sutun\_indeksi)** şeklinde bir format kullanırız. Aşağıda önce bir matris tanımlayıp daha sonra da o matrisin bir elemanın değerini başka bir değişkene atamaktayız.

```
>> A= [15 2 4 ; 3 7 11 ; 1 2 9];
>> a23 =A(2,3)
a23 =
11
```

Yukarıdaki komutlar sayesinde A matrisinin 2. Satır 3. elemanına ulaşmaktadır.

### **MATLAB Matris de Kolon Operatör**

İlk olarak kolon operatöryle başlayalım; kolon operatörü iki nokta üst üste ( : ) ile gösterilir. Örneğin:

```
>> 2:8  
ans =  
2 3 4 5 6 7 8
```

komutıyla 2 ile 8 arasındaki sayılarla ulaşmış olduk. Bir başka örnek;

```
>> 0:6  
ans =  
0 1 2 3 4 5 6
```

yne aynı işlemle 0 ile 6 arasındaki sayılarla ulaşmış olduk. Fakat dikkat edilmesi gereken bir nokta var: bu işlemlerde artış miktarımız 1 dir. Artış miktarı 1 den farklı ise istediğimiz artış miktarını, başlangıç ve bitiş sayılarının arasına yine iki nokta üst üste ile ayırarak yazarız. Artış miktarı 1 olduğu zamanlarda ise 1'i yazmaya gerek yoktur.

```
>> 4:3:20  
ans =  
4 7 10 13 16 19
```

örneğinde de görüldüğü gibi, 4 ten başlayarak üçer artışla 20 ye en yakın sayıya ulaştık. Bir diğer nokta da şu: başlangıç sayısı girdiğimiz sayı olmalıdır fakat örnekte de görüldüğü gibi bitiş sayısı girdiğimiz sayı olmayabilir.

### Bir matrisin alt matrislerine ve alt vektörlerine ulaşma

**MATLAB** da çok karşılaşılan uygulamalardan birisi ise matrisin içinde alt matrislere ulaşma veya sadece bir satır / sütuna ulaşma durumudur. Bu durumda, bütün durumlar için aşağıdaki yapıyı kullanacağız:

**A (satır\_bilgisi ; sutun\_bilgisi)**

```
>> alt_matris=A(1:2,2:3)  
alt_matris =  
2 4  
7 11
```

Gösterimi ile ulaşılacak alt matris, A' nın 1-2. aralığındaki satırları ile 2-3. aralığındaki sütunlarının kesişim bölgesidir.

```
>> alt_matris=A(1:2, : )  
alt_matris =  
15 2 4  
3 7 11
```

Gösterimi ile ulaşılacak alt matris, A'nın 1.-2. Aralığındaki satırları ile tüm sütunlarının kesişimidir. Yani 1. ve 2. Satırlardan oluşan matristir.

**A(3 , 2:3)** → gösterimi ile A matrisinin 3.satırındaki 2-3 aralığındaki sütunundaki elemanları veren bir vektördür.

**A(:, end)** → gösterimi bütün satırlarla, son sütünün kesişimini verecektir. Yani son sütunu verecektir.

### MATLAB' da Matrislerin Boyutunun Değiştirilmesi

**MATLAB** da **A(m x n)** boyutunda bir matris var ise, bu matris **m\*n = p\*q** olmak şartıyla **B(p x q)** boyutunda bir matrise dönüştürülebilir.

Aşağıdaki A matrisi ( $2 \times 3$ ) boyutundadır. A matrisinin ( $3 \times 2$ ) boyutuna getirilmesi aşağıda gösterilmiştir.

```
A = [10 8 6 ; 1 3 5];  
B = reshape(A,3,2);  
B =  
10 3  
1 6  
8 5
```

repmat :

### MATLAB da Matrislerin Genişletilmesi

**MATLAB** da matrisin bir elemanına atama yapılmışsa, bu matrisi uygun boyutta alması için gereken ama herhangi bir değer ataması yapılmamış elemanlarına otomatik olarak sıfır atanır.

```
>> Z(3, 1 : 2) = [5 6]
```

Z =

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 5 & 6 \end{bmatrix}$$

### MATLAB da Bir Matrise Satır veya Sütun Eklenmesi

C = [B x] → satır eklenmesi

C = [B ; y] → sütun eklenmesi

$$B = \begin{bmatrix} 13 & 6 & 7 \\ 14 & 5 & 9 \\ 12 & 4 & 11 \end{bmatrix} \quad x = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \quad y = [15 \ 2 \ 3]$$

B matrisine x vektörünün eklenmiş hali aşağıda gösterilmiştir.

```
>> B = [13 6 7 ; 14 5 9 ; 12 4 11];
```

```
>> x = [4 ; 5 ; 6];
```

```
>> C=[B x]
```

C =

$$\begin{bmatrix} 13 & 6 & 7 & 4 \\ 14 & 5 & 9 & 5 \\ 12 & 4 & 11 & 6 \end{bmatrix}$$

### MATLAB da Bir Matrisin Satırının/Sütununun Silinmesi

Aşağıda X matrisinin ilk satırının silinmesi gösterilmektedir.

```
>> X = [5 11 10 ; 4 8 7 ; 2 3 9];
```

```
>> X(1, :)=[ ]
```

X =

$$\begin{bmatrix} 4 & 8 & 7 \\ 2 & 3 & 9 \end{bmatrix}$$

Bir matrisin içinden tek bir eleman silinemez çünkü matrisin tek bir elemanın silinmesi boyut uyumsuzluğuna sebep olacaktır.

### MATLAB da Matrislerin Birleştirilmesi

Aşağıda x ve y matrislerinin birleştirilmesi gösterilmektedir.

```

>> X = [2 10 ; 5 7];
Y = [11 13 ; 4 9];
>>
>> X = [2 10 ; 5 7];
>> Y = [11 13 ; 4 9];
>> W =[X Y] % X ve Y matrislerinin satır olarak birleştirilmesi.
W =
    2   10   11   13
    5     7   4     9
>> W =[X;Y] % X ve Y matrislerinin sütun olarak birleştirilmesi.
W =
    2   10
    5     7
    11   13
    4     9

```

## MATLAB' Matrislerin Aritmetik İşlemleri

Bu bölümde sırasıyla göreceğimiz aritmetik işlemler:

- 1- **MATLAB**'da matrislerin toplanması işlemi,
- 2- **MATLAB**'da matrislerin birbirinden çıkarılması işlemi,
- 3- **MATLAB**'da matrislerin çarpılması işlemi,
- 4- **MATLAB DA ELEMAN ELEMANA HESAPLAMA İŞLEMLERİ**,

### MatLab'da Matrislerin Toplanması:

Matrislerde toplama işlemi yapılabilmesi için matrislerin aynı türden olması yani **eşit satır** ve **eşit sütun** sayılarına sahip olması gereklidir. Toplama işlemi yapılırken aynı satır ve sütün değerlerine sahip elemanlar toplanır.

Aşağıdaki kod kısmında aynı türden X ve Y matrislerinin toplanması gösterilmektedir.

```

> X = [11 4 5 ; 12 9 7];
Y = [6 4 3 ; 8 1 2];
Z = X + Y
Z =
    17   8   8
    20   10   9

```

### MATLAB' da Matrislerin Birbirinden Çıkarılması:

Matrislerde çıkarma işlemi yapılabilmesi için matrislerin aynı türden olması yani **eşit satır** ve **eşit sütun** sayılarına sahip olması gereklidir. Çıkarma işlemi yapılırken aynı satır ve sütün değerlerine sahip elemanlar çıkartılır.

Aşağıdaki kod kısmında aynı türden X ve Y matrislerinin birbirinden çıkartılması işlemi gösterilmektedir.

```
X = [11 4 5 ; 12 9 7];  
Y = [6 4 3 ; 8 1 2];  
W = X - Y  
W =  
5   0   2  
4   8   5
```

### MATLAB' da Matrislerin Çarpılması:

Matrislerde çarpma işlemi yapılabilmesi için **ilk** matrisin **sütun** sayısı, **ikinci** matrisin **satır** sayısına **eşit** olmalıdır.

### A-B matrisleri ile çarpım yapılırken:

1. A'nın 1. satır elemanları ile B'nin 1. sütun elemanları karşılıklı çarpılıp toplanır. Bu AB çarpım matrisinin **birinci(a11)** elemanıdır.
2. A'nın 1. satır elemanları ile B'nin 2. sütun elemanları karşılıklı çarpılıp toplanır. Bu AB çarpım matrisinin **a12** elemanıdır.
3. Bu çarpım A matrisinin bütün satırları B matrisinin bütün sütunları ile çarpılıp yeni matris elde edilinceye kadar devam eder.

Aşağıda X ve T matrislerinin çarpımı gösterilmektedir.

```
X = [11 4 5 ; 12 9 7];  
T = [6 8 ; 4 1 ; 3 2];  
Q = X*T  
Q =  
97  102  
129  119
```

**Üs alma:** Sadece kare matrislerin üssü alınabilir. Çünkü sadece o durumda birinci matrisin sütun sayısı ikinci matrisin satır sayısına eşit olur.

Aşağıda R matrisinin 3. Dereceden üssünün alınması gösterilmektedir.

```
R=[11 4 ; 12 9];  
U = R^3  
U =  
2819    1396  
4188    2121
```

**NOT:** Matris ile skaler bir değişken arasında yapılacak bir işlemde bu işlem matrisin bütün elemanlarının hepsine uygulanır.

## MATLAB'TA ELEMAN ELEMANA HESAPLAMA İŞLEMLERİ

Eleman elemana hesaplama işlemi eleman eleman icra edilir. Örneğin A ve B'nin 5'er elemanlı birer satır vektörü olduğunu varsayıyalım. Bu değerler ile yeni bir C satır vektörü oluşturmanın bir yolu, aşağıda gösterildiği gibi A ve B deki karşılık gelen değerlerin çarpımlarını almaktır.

$$\begin{aligned}C(1) &= A(1) * B(1); \\C(2) &= A(2) * B(2); \\C(3) &= A(3) * B(3); \\C(4) &= A(4) * B(4); \\C(5) &= A(5) * B(5);\end{aligned}$$

Bu komutlar esasında skaler komutlardır. Çünkü her bir komut tek bir skaler değeri diğer bir tek skalar değer ile çarparak çarpımı üçüncü bir değer olarak saklamaktadır. MATLAB da aynı boyutlu iki matris arasında eleman eleman çarpma işleminin icrası aşağıdaki bildirimde gösterildiği gibi,”\*” işlemcisi ile çok daha kısa yoldan yerine getirilir.

$$C = A.*B$$

Toplama ve çıkarma işleminde eleman elemana hesaplama ve matris işlemleri aynıdır. Dolayısıyla bunlarda farklı işlemci kullanmaya gerek yoktur. Buna karşılık eleman eleman işlemlerinde çarpma, bölme ve üst alma, matris işlemlerindeki çarpma, bölme ve üst alma işlemlerinden farklılık gösterir. Burada en önemli fark çarpma, bölme ve üst alma işlemcileri önünde nokta, “.” işaretini gelmesidir.

Eleman elemana hesaplama işlemleri yalnızca aynı boyutlu iki matris arasında uygulanmayıp aynı zamanda skalar ve skalar olmayan değerler arasında da uygulanır. Bununla beraber bir matrisin bir skalar ile çarpımı ve bölümünde işlemciler noktalı ve noktasız olarak kullanılabilir. Buna göre skalar olmayan bir A matrisi için aşağıda verilen bildirim takımları birbirine denktir.

$$B=3*A; \text{ veya } B=3.*A;$$

$$C=A/5; \text{ veya } C=A./5;$$

Sonuç matrisler B ve C her iki durumda da A ile aynı boyutta bir matris olur.

**Eleman elemana çarpım:** Vektörlerde kullanılan eleman elemana çarpma işlemini göstermek üzere aşağıda verilen iki satır vektörünü ele alalım.

```
>>A=[2 5 6];
>>B=[2 3 5];
>>C=A.*B %bu işlem eleman elemana çarpma sonucunu verecektir.
C=
4 15 10
```

**Eleman elemana bölme:** MATLAB ileri veya sağdan bölme, ./ ve geri veya soldan bölme, .\ olmak üzere iki bölme işlemcisi kullanır. Sağdan eleman elemana bölme komutu **C=A./B** biçiminde olup bu da **C=[1 1.667 1.2]** A'nın her bir elemanın B tarafından bölümünü sonuçlandırır.

Buna karşılık geri ve soldan bölme (ters bölme) işlemi

**C=A.\B**

komutu ile gerçekleşir ki bu B'nin A tarafından bölümünü sonuçlandırır. **C=[1 0.6 0.833]**

**Eleman elemana üst alma:** Bu işlemde eleman elemana üst alınır. Yukarıda tanımlanan A ve B vektörlerini ele alacak olursak;

**C=A.^2;**

**D=A.^B;** komutları yolu ile C ve D vektörleri elde edilir.

**C=[4 25 36];**

**D=[4 125 7776];**

Aynı zamanda bir skaler tabanın vektör üssünü almak mümkündür.

Örneğin

**C=3.^A;** bildirimi **C=[9 243 729];** Sonucunu doğuracaktır. Bu vektör aynı zamanda aşağıdaki bildirimle de hesaplanabilir.

**C=3.^A;**

Burada MATLAB nokta işaretini “.” 3 sabitinin bir parçası sayacak ve ona göre işlem yapacaktır. Bu da matris işlemine karşılık aşağıda geleceğinden kastedilene göre yanlış olacaktır. Buna karşılık aşağıda gösterildiği gibi nokta ile sabit arasında bir boşluk konacak olursa yine doğru sonuç alınır.

**C=3 .^A;**

Bu örnekler eleman elemana hesaplama işlemi yapılrken çok dikkatli olunması gerektiğini göstermektedir. Yazılan ifadenin doğruluğundan tam olarak emin olunmadığı durumlarda basit örneklerle bazı testler yapılması yerinde olacaktır. Bunu aşağıdaki örneklerle gösterebiliriz.

Bundan önceki örneklerde eleman elemana hesaplama işlemlerinde, vektörler kullanılmıştır. Aşağıdaki örneklerde görüldüğü gibi eleman elemana hesaplama işlemlerinde matrisleri de kullanmak mümkündür.

Hesaplama sonuçları aşağıda gösterildiği gibidir.

```

>> d=[1 2 3 4 5;-1 -2 -3 -4 -5];
>> z=[1 1 1 1 1; 1 1 1 1 1];
>> s=d-z;
>> sq=s.^3;
>> d
d =
    1    2    3    4    5
   -1   -2   -3   -4   -5
>> z
z =
    1    1    1    1    1
    1    1    1    1    1
>> s
s =
    0    1    2    3    4
   -2   -3   -4   -5   -6
>> sq
sq =
    0    1    8   27   64
   -8  -27  -64 -125 -216

```

## MATLAB'da Özel Matrisler

Bu bölümde **MATLAB'** da aşağıdaki özel matris işlemlerini sırasıyla öğreneceğiz.

- ❖ **MATLAB Sıfır matrisi:** zeros komutu
- ❖ **MATLAB Birler matris:** ones komutu
- ❖ **MATLAB Birim matris:** eye komutu
- ❖ **MATLAB Random matrisi:** rand komutu
- ❖ **MATLAB Köşegen (diagonal) matris:** diag komutu

**zeros(m, n) komutu:** m x n boyutunda sıfırlardan oluşan bir matris oluşturur.

```

>> zeros(4,3)
ans =
    0    0    0
    0    0    0
    0    0    0
    0    0    0

```

**zeros(m) komutu** m x m boyutunda sıfırlardan oluşan kare bir matris oluşturur.

```
>> zeros(3)
```

```
ans =
```

0	0	0
0	0	0
0	0	0

**ones(m, n) komutu:** m x n boyutunda birlerden oluşan bir matris oluşturur.

```
>> ones(4,3)
```

```
ans =
```

1	1	1
1	1	1
1	1	1
1	1	1

**eye(m, n) komutu:** m x n boyutunda birim matris oluşturur.

```
>> eye(2,3)
```

```
ans =
```

1	0	0
0	1	0

**eye(m) komutu:** m x m boyutunda birim matris oluşturur.

```
>> eye(4)
```

```
ans =
```

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

**rand(n) komutu:** n x n boyutunda elemanları 0-1 aralığından rastgele seçilmiş olarak bir matris oluşturur.

```
>> rand(4)
ans =
0.8147  0.6324  0.9575  0.9572
0.9058  0.0975  0.9649  0.4854
0.1270  0.2785  0.1576  0.8003
0.9134  0.5469  0.9706  0.1419
```

**rands(m, n) komutu:** m x n boyutunda elemanları -1- 1 aralığından rastgele seçilmiş olarak bir matris oluşturur.

```
rands(4,3)
ans =
-0.1565  0.3115  0.3575
0.8315  -0.9286  0.5155
0.5844  0.6983  0.4863
0.9190  0.8680  -0.2155
```

**diag(d) komutu:** köşegeni d vektörünün elemanlarından oluşan diagonal bir matris üretir.

```
>> d=[1 2 3];
>> diag(d)
ans =
1   0   0
0   2   0
0   0   3
```

## MATLAB'da Matrisler ile İlgili Özel Komutlar

Bu bölümde **MATLAB'** da aşağıdaki matris ile ilgili hazır komutları sırasıyla göreceksiniz.

- ❖ **MATLAB'** da bir matrisin determinantının alınması: **det komutu**
- ❖ **MATLAB'** da bir matrisin rankının alınması: **rank komutu**
- ❖ **MATLAB'** da bir matrisin izinin bulunması: **trace komutu**
- ❖ **MATLAB'** da bir matrisin tersinin bulunması: **inv komutu**
- ❖ **MATLAB'** da bir matrisin karakteristik denkleminin bulunması: **poly komutu**
- ❖ **MATLAB'** da bir matrisin özdeğer ve özvektörlerinin bulunması: **eig komutu**
- ❖ **MATLAB'** da bir matrisin ortogonal matrisinin bulunması: **orth komutu**

**det(x) komutu:** Bir kare matrisin determinantını hesaplar.

```
>> d  
>> det(X)  
ans =  
189
```

**rank(X) komutu:** Bir kare matrisin rankını hesaplar.

```
>> X = [9 13 7 ; 11 8 4 ; 2 10 3];  
>> rank(X)  
ans =  
3
```

**trace(X) komutu:** Bir matrisin izini (köşegen elemanlarının toplamını) hesaplar.

```
>> trace(X)  
ans =  
20
```

**inv(X) komutu:** Bir kare matrisin tersini verir.

```
>> inv(X)  
ans =  
-0.0847  0.1640 -0.0212  
-0.1323  0.0688  0.2169  
0.4974 -0.3386 -0.3757
```

**poly(X) komutu:** Tanımlanmış bir matrisin karakteristik denklemini verir.

```
>> poly(X)  
ans =  
1.0000 -20.0000 -74.0000 -189.0000
```

**eig(X) komutu:** Tanımlanmış bir matrisin özdeğerlerini verir.

```
>> eig(X)
ans =
23.4924
-1.7462 + 2.2352i
-1.7462 - 2.2352i
```

[y, z] = eig(X): Matrisin öz vektörlerini ve öz değerlerini verir.

```
>> X = [9 13 7 ; 11 8 4 ; 2 10 3];
>> [y,z]=eig(X)
y =
0.7132      -0.0031 - 0.2981i -0.0031 + 0.2981i
0.6000      -0.3950 + 0.2459i -0.3950 - 0.2459i
0.3624      0.8335          0.8335

z =
23.4924          0          0
0      -1.7462 + 2.2352i      0
0          0      -1.7462 - 2.2352i
```

**orth(X) komutu:** tanımlanmış bir matrisin **ortogonal matrisini** verir.

```
>> orth(X)
ans =
-0.7193  0.1170 -0.6848
-0.5663 -0.6697  0.4804
-0.4025  0.7333  0.5480
```

**Matlab'da logspace komutunun kullanımı:** **logspace** komutu logaritmik vektörler oluşturmamıza yardımcı olur. Logspace yapısı şu şekildedir.

**logspace(a, b, n):** Başlangıç değeri  $10^a$ , bitiş değeri değeri  $10^b$  olan n elemanlı ve elemanları arasındaki katları eşit olan bir dizi oluşturur.

```
x = logspace(1,5,3)
x =
10      1000     100000.
```

**Matlab'da sum komutu:** **sum (x)** = Vektörün bütün elemanlarını birbirleriyle toplar.

```
x = [-8 0 -1 3 4.5];  
toplam = sum(x)  
toplam =  
-1.5000
```

**Matlab'da mean komutu:**  $\text{mean}(x)$  = Vektörün elemanlarının ortalamasını verir.

```
x = [-8 0 -1 3 4.5];  
ortalama = mean(x)  
ortalama =  
-0.3000
```

**Matlab'da length komutu:**  $\text{length}(x)$  = Vektörün uzunluğunu yani satır ve sütunlardan en fazla elemana sahip değeri verir.

```
x = [-8 0 -1 3 4.5];  
elemansayisi = length(x)  
elemansayisi =  
5
```

**Matlab'da max komutu:**  $\text{max}(x)$  = Vektörün en büyük elemanını verir.

**Örnek :**  
*x = [-8 0 -1 3 4.5];  
max(x)  
enbuyukeleman =  
4.5000*

**Matlab'da min komutu**  $\text{min}(x)$  = Vektörün en küçük elemanını verir.

```
x = [-8 0 -1 3 4.5];  
enkucukeleman = min(x)  
enkucukeleman = -8
```

**Matlab'da prod komutu:**  $\text{prod}(x)$  = Vektörün elemanlarını birbiriyile çarpar.

```
x = [-8 0 -1 3 4.5];  
carpim = prod(x)  
carpim =  
0
```

**Matlab'da sign komutu:**  $\text{sign}(x)$  = Vektörün pozitif elemanlarını 1, negatif elemanları için -1, 0 olan elemanları için 0 sonucunu verir.

```
x = [-8 0 -1 3 4.5];  
sign(x)  
ans =  
-1 0 -1 1
```

**Matlab'da find komutu:**  $\text{find}(x)$  = Vektörün sıfır olmayan elemanlarının indeksini verir.

```
x = [-8 0 -1 3 4.5];  
find(x)  
ans =  
1 3 4 5
```

Ayrıca find komutu içinde verilen koşulu sağlayan elemanların indekslerini de verir.

```
x = [-8 0 -1 3 4.5];  
a=find(x>1)  
a =  
4 5
```

**Matlab'da fix komutu:**  $\text{fix}(y)$  = Sıfıra doğru yuvarlama işlemini yapar.

```
y = [-1.1 -3.6 1.6 2.4 0.4];
sifirayuvarla = fix(y)
sifirayuvarla =
-1   -3    1    2    0
```

**Matlab'da floor komutu:** floor(y) = – sonsuza ‘a doğru yuvarlama işlemi yapar.

```
y = [-1.1 -3.6 1.6 2.4 0.4];
eksisonsuzayuvarla = floor(y)
eksisonsuzayuvarla =
-2   -4    1    2    0
```

**Matlab'da ceil komutu:** ceil(y) = + sonsuz ‘a doğru yuvarlama işlemi yapar.

```
y = [-1.1 -3.6 1.6 2.4 0.4];
>> artisonsuzayuvarla = ceil(y)
artisonsuzayuvarla =
-1   -3    2    3    1
```

**Matlab'da round komutu:** round(y) = Kendisine en yakın tam sayıya yuvarlama işlemi yapar.

```
y = [-1.1 -3.6 1.6 2.4 0.4];
enyakinayuvarla = round(y)
enyakinayuvarla =
-1   -4    2    2    0
```

**Matlab'da sort komutu:** sort(y) = Vektörün elemanlarını aksi belirtilmemişçe küçükten büyüğe sıralar.

```
sort(vektör, 'modu')
y = [-1.1 -3.6 1.6 2.4 0.4];
sirala = sort(y)
sirala =
-3.6000  -1.1000   0.4000   1.6000   2.40
```

## **Yerleşik ans Değişkeni ve Matematik Sabitleri**

**MATLAB** bünyesinde yerleşik(built in) bazı matematik sabitleri ve ans adında özel bir değişken vardır. Aşağıda bütün bu matematik sabitleri ve ans değişkeni tanımlanıp uygulamalar ile açıklanacaktır.

**Ans Değişkeni:** İngilizce answer kelimesinin kısaltılmıştır. **MATLAB** da herhangi bir değişkene atanmayan deyimlerin(icerisinde atama operatörü(=) kullanılmayan deyimler) sonuçları yorumlayıcı tarafından değerlendirildikten sonra özel bir **MATLAB** değişkeni olan ans içerisinde saklanır. Dolayısıyla bu tür deyimlerin içerisinde gizli bir ans= olduğunu hayal edebilirsiniz. Ans normal bir değişken gibi aritmetik işlemler içerisinde değerlendirilebilir. Aşağıdaki komut penceresi çıktısı bu durumu özetlemek için kullanılmıştır. Bu örnekte 2 sayısı ile 3 sayısının çarpımının sonucu herhangi bir değişkene atanmadığı için, işlemin sonucu **MATLAB** in ans değişkeni içerisinde saklanmıştır. Bir sonraki aşamada ise ansın mevcut değerinin karesi alınmış ve yine atama operatörü kullanılmadığı için yeni sonuç tekrar ans'ye aktarılmıştır.

```
>>2*3  
ans=  
6  
>>ans^2  
ans=  
36
```

### **Pi( $\pi$ ) Sabiti**

$\Pi$  değeri pi adı altında otomatik olarak saklanmıştır. Programlar içinde kullanılan pi kelimesi doğrudan  $\pi$  değerine karşılık gelir. Aşağıdaki komut penceresi çıktısı **MATLAB** da yerleşik pi sabitinin değerini öğrenebilmek için kullanılmıştır.

```
>> pi  
ans =  
3.1416  
>> ans/pi  
ans =  
1
```

**i, j Sabiti:** Karmaşık sayıların sanal kısımlarını MATLAB ortamında ifade edebilmek için kullanılırlar. **i** ile **j** harfleri doğrudan  $\sqrt{-1}$ değerine ayarlanmıştır. Aşağıdaki komut penceresi çıktısı **MATLAB** da yerleşik **i ve j sabitlerinin** değerini öğrenebilmek için kullanılmıştır. Sonraki bölümlerimizde döngü değişkeni olarak da kullanacağımız **i** ile **i sabiti** karıştırılmamalıdır ve bu duruma özellikle karmaşık sayı uygulamalarında dikkat edilmelidir.

```

>> i
ans =
0 + 1.0000i
>> sqrt(-1)
ans =
0 + 1.0000i

```

**intmin ve intmax Sabitleri:** Günümüz bilgisayarlarının tam sayıları temsil etmek üzere kullanıldığı ikiye tümleyen sayı sisteminde n bit uzunluğundaki işaretli tam sayılar  $-2^{(n-1)}$  ile  $(2^{(n-1)}-1)$  aralığındadır. Dolayısıyla bu sayı sisteminde 32 bitle temsil edilebilecek en küçük işaretli tam sayı  $-2^{(32-1)} = -2^{31} = -214783648$  dir. Bu değeri ezberlemeye gerek yoktur çünkü **MATLAB** da yerleşik **intmin** sabiti ihtiyaç duyduğumuz her an kullanabilelim diye bizler için bu değeri bünyesinde saklar. Benzer şekilde ikiye tümleyen sayı sisteminde 32 bitle temsil edilebilecek işaretli en büyük tamsayı **214783647**' dir. Aşağıda verilen komut penceresi çıktısının ilk 2 komutu **MATLAB** da yerleşik **intmin/intmax** sabitlerinin değerlerini; son iki komut ise fonksiyon olarak kullanılabilen **intmax** ile 64 bitlik işaretsiz bir tam sayının alabileceği maksimum değeri ve **intmin** ile 8 bitlik işaretli bir tamsayının alabileceği minimum değeri öğrenebilmek için kullanılmıştır.

```

>> intmin
ans =
-2147483648
>> intmax
ans =
2147483647
>> intmax('uint64')
ans =
18446744073709551615
>> intmin('int8')
ans =
-128

```

**realmin ve realmax Sabitleri:** IEEE 754 standarı ile belirlenmiş çift duyarlı kayan noktalı sayıların(64 bit) normalize edilmiş en büyük ve en küçük değerlerini bünyelerinde barındıran **MATLAB** sabitleridir. Değerleri aşağıda gösterilmiştir.

```

>> realmin
ans = 2.2251e-308
>> realmax
ans = 1.7977e+308

```

## MATLAB da Sayı Görüntüleme Formatları

$\pi$  sayısının (**MATLAB** da yerleşik pi sabitinin) değerinin **MATLAB** komut penceresinde farklı görünüm şekilleri vardır bu şekiller:

- ❖ **3.14** (ayraçtan sonra iki basamak duyarlılıkla),
- ❖ **3.1416** (ayraçtan sonra 4 basamak duyarlılıkla),
- ❖ **3.141592653589793** (ayraçtan sonra 15 basamak duyarlılıkla),
- ❖ **355/113** (rasyonel sayı formunda) veya
- ❖ **400921fb54442d18** (hekzadesimal sayı formunda)

**MATLAB** yukarıdaki görünüm şekillerinin oluşturulması için farklı konumlara sahip bir anahtar şeklinde de hayal edebileceğimiz **format** adında bir komuta sahiptir. **MATLAB** programlama ortamında kayan noktalı sayıları ifade etmek için varsayılan olarak **IEEE 754** standardının çift duyarlılı düzey kullanılmaktadır. Aşağıda **format** anahtarının 5 farklı konumu ve bu konumların işlevleri gösterilecektir.

### **Format Short**

Bu format varsayılan çıktı formatıdır. Bu formatın komut penceresi çıktıları ondalıklı sayı formuna ve ayraçtan sonra 4 basamak duyarlılığı sahiptir.

### **Format Bank**

Bu formatın komut penceresi çıktıları ondalıklı sayı formuna ve ayraçtan sonra 2 basamak duyarlılığı sahiptir. Aşağıdaki komut penceresi çıktısı format anahtarının bank komutu ile birlikte kullanılmasını özetlemektedir. **MATLAB** in format anahtarı varsayılan olarak short konumundadır ve bu nedenledir ki pi sayısı başlangıçta ayraç sonrası 4 duyarlılıkla ekrana yansımıştır. Anahtar bank konumuna getirildiğinde ise duyarlılık 2 ye düşmüştür. Ekran satırının son satırı ise format anahtarını yeniden varsayılan konum olan short pozisyonuna getirmek için kullanılmıştır.

```
>> pi
ans =
3.1416
>> format bank
>> pi
ans =
3.14
>> format short
>> pi
ans =
3.1416
```

### **Format Long**

Bu formatın komut penceresi çıktıları ondalıklı sayı formuna ve ayraçtan sonra 15 basamak duyarlılığı sahiptir. Aşağıdaki komut penceresi format anahtarının long konumu ile birlikte kullanılışını özetlemektedir.

```
>> format long  
>> pi  
ans =  
    3.141592653589793  
>> format short  
>> pi  
ans =  
    3.1416
```

## Format Rat

Bu formatın komut penceresi çıktıları rasyonel sayı (pay/payda) formuna sahiptirler. Aşağıdaki komut penceresi çıktısı format anahtarının rat konumu ile birlikte kullanılışını özetlemektedir. Rat konumunun rasyonel sayılar ile yapılan işlemleri nasıl kolaylaştırdığını dikkat ediniz. MATLAB sayesinde en küçük ortak kat bularak paydaları eşitemek ve en büyük ortak bölen bularak sonucu sadeleştirmek zahmetinden kurtulmuş oluruz.

```
>> format rat  
>> pi  
ans =  
    355/113  
>> a=1/3  
a =  
    1/3  
>> b=0.5  
b =  
    1/2  
>> c=0.25  
c =  
    1/4  
>> d=a+b+c  
d =  
    13/12  
>> format short
```

## Format Hex

Bu formatın komut penceresi çıktıları hekzadesimal sayı formuna sahiptirler. Aşağıdaki komut penceresi çıktısı format anahtarının hex konumu ile birlikte kullanılışını

özetlemektedir. Bu çıktıyı anlayabilmek için ikili, onaltılı sayı sistemleri ve kayan noktalı sayıların günümüz bilgisayarlarında temsilinin standarı olan **IEEE 754** hakkında bilgi sahibi olmak gereklidir.

**IEEE Kayan Nokta Aritmetiği Standardı** kayan noktalı sayıların gösteriminde en çok kullanılan standarttır. İkilik sistemdeki sayılar bilimsel gösterim ile gösterildikten sonra işaret, üst ve anlamlı kısımdan oluşan üç parça şeklinde ifade edilebilirler. Bu gösterime sonsuz, sayı değil ve sıfırın gösterimi dâhildir. **IEEE 754** standardına göre sayılar **tek duyarlı (32 bit)** ve **çift duyarlı (64 bit)** şekilde gösterilebilirler.

**Tek duyarlı** gösterimde sayı 32 bitle ifade edilir. Bu bitlerden biri işaret, 8'i üst 23 tanesi ise anlamlı kısmın gösterimi için kullanılır. Tek duyarlı gösterimde üst için kaydırma değeri  $2^{8-1}-1 = 127$  olarak hesaplanır.

- ❖ Tek duyarlı gösterimde 6,375 sayısını göstermek istersek;  $6,375 = (110,011)_2$
- ❖ Sayıyı olağan duruma getirirsek:  $110,011 = 1,10011 \times 2^2$
- ❖ Sayı sıfırdan büyük olduğu için işaret biti: 0
- ❖ Sayının üst değerinin saptırılmış hali:  $2+127 = 129 \rightarrow 129_{10} = 10000001_2$
- ❖ Anlamlı kısım:  $10011000000000000000000000000000$
- ❖ Sayı son olarak; 0 10000001 10011000000000000000000000000000 şeklinde ifade edilir.

**Çift duyarlı** gösterimde sayı 64 bitle ifade edilir. Bu bitlerden biri işaret, 11'i üst ve 52 tanesi de anlamlı kismı ifade etmek için kullanılır. Bu gösterimde üst için sapma değeri  $2^{11-1}-1 = 1023$  olarak hesaplanır. MATLAB değişkenleri varsayılan olarak IEEE 754 çift duyarlıklı temsili kullanırlar.

```
>> format hex  
>> pi  
ans =  
    400921fb54442d18  
>> 1  
ans =  
    3ff0000000000000  
>> intmax('uint8')  
ans =  
    ff  
>> 6.375           % MATLAB değişkenleri çift duyarlıklı temsili kullanırlar.  
ans =  
    4019800000000000  
>> format short
```

## MATLAB da Atama Operatörü

MATLAB, kullandığı operatörleri **atama operatörü**, **aritmetik operatörler**, **ilişkisel operatörler** ve **mantıksal operatörler** olarak 4 gruba ayırrı.

Herhangi bir değişkenin içeresine bir değeri veya ifadenin/işlemin sonucunu aktarma operasyonuna **atama** denir. **Atama operatörü** olarak matematik derslerinden bildiğimiz eşittir karakteri ( $=$ ) kullanılır. “**değişken=ifade**” formatında gerçekleştirilen bu atama operasyonu sonucunda atama operatörü olarak adlandırdığımız “ $=$ ” in sağındaki matematikse/mantıksal işlemin sonucu veya sayısal bir değer solundaki değişkenin değeri olarak kaydedilir. Örneğin **MATLAB** komut penceresine yazılacak  $a=3$  şeklindeki bir deyim a adındaki değişkenin içerisinde 3 değerini saklamak içindir. Eğer a adında bir değişken daha önce tanımlanmamış ise **MATLAB** tarafından bu değişken için bellekte yeni bir alan rezerve edilir ve içerisinde 3 değeri yazılır. Bu ada sahip bir değişkenin daha önce tanımlanmış olması durumunda ise içerdeği eski değerin silinmesi ve yerine 3 değerinin atanması işlemi gerçekleştirilir.

**İPUCU:** Bir algoritma geliştirilirken onun hangi amaçlarla kaç farklı değişkene ihtiyaç duyacağıının belirlenmesi büyük önem arz eder.

### **MATLAB da Aritmetik Operatörler**

**MATLAB**ın aritmetik işlemlere yönelik bizlerin kullanımına sunduğu operatörlere aritmetik operatörler denir. “**delta=B<sup>2</sup> – 4AC**” şeklinde tanımlanan matematiksel bir ifadenin herhangi bir programlama dilinde karşılığı yoktur. Bu yüzden matematiksel ifadelerde yer alan ve aritmetik işlemleri tanımlayan operatörlerin kullanılan programlama dilindeki karşılıklarının bilinmesi gereklidir. Aritmetik operatörler aşağı yukarı birçok programlama dilinde aynıdır. **MATLAB** da tanımlı aritmetik operatörler ve kullanımları aşağıdaki tabloda listelenmiştir.

<b>ARİTMETİKSEL İŞLEM</b>	<b>MATLAB OPERATÖRÜ</b>	<b>KULLANIMA ÖRNEK</b>
<b>TOPLAMA</b>	<b>+</b>	<b>4+9</b>
<b>ÇIKARMA</b>	<b>-</b>	<b>12-9</b>
<b>CARPMA</b>	<b>*</b>	<b>25*4</b>
<b>BÖLME</b>	<b>/</b>	<b>32/9</b>
<b>ÜS ALMA</b>	<b>^</b>	<b>2^6</b>

Tablo kullanılarak **delta=B<sup>2</sup>-4AC** şeklindeki matematiksel ifade **MATLAB**ın yorumlayabileceği **delta=B<sup>2</sup>-4\*A\*C** formuna kolaylıkla dönüştürülebilir.

**İPUCU:** *sayı1, sayı2 ve sayı3* adlı 3 değişkenin ortalamasını hesaplayacak aritmetik ifade *sayı1+sayı2+sayı3/3* değildir. Bu ifade ile 3'te biri alınmış *sayı3* değişkeninin değeri *sayı1* ve *sayı2* ile toplanır. Öncelikli olarak 3 değişkeni toplamak ve sonrasında bu değeri 3 e bölperek ortalama hesaplamak için parantezlerden ( ) faydalananmak gereklidir. Yani bizden istenen işlemin sonucunu ancak *(sayı1+sayı2+sayı3)/3* ile elde ederiz.

### **Aritmetik Operatörlerin Öncelik Sıraları**

**MATLAB** yorumlayıcısı birden fazla aritmetik işlem içeren bir deyimi hangi sıra ile işleyeceğini yani **MATLAB**ın aritmetik işlemlerin her birine verdiği öncelik haklarını bilmek bizim bekłentilerimizin dışında sürpriz sonuçlarla karşılaşmamızı engeller. Örneğin  $2^2^3$

işleminin sonucu **64** müdür yoksa **256** müdür? Aşağıdaki tabloda **MATLAB** in aritmetik işlemlere yönelik öncelik sıraları listelenmiştir.

ÖNCELİK SIRASI	İŞLEM	KENDİ İÇERİSİNDE ÖNCELİK SIRASI
1	Parantez	İçten Dışa
2	Üs Alma	Soldan Sağa
3	Çarpma ve Bölme	Soldan Sağa
4	Toplama ve Çıkarma	Soldan Sağa

Yukarıdaki tablodaki işlem önceliklerini göz önüne alduğumuz zaman  $2^2 \cdot 3$  işleminin **64** olduğunu rahatlıkla söyleyebiliriz.

## **MATLAB da İlişkisel ve Mantıksal Operatörler**

### **İLİŞKİSEL OPERATÖRLER**

İlişkisel operatörler, işlenenleri arasındaki eşitlik, eşitsizlik, büyülük veya küçüklük türünden ilişkileri belirleyen operatörlerdir. **MATLAB** da kullanılan ilişkisel operatörler ve kullanım amaçları aşağıda belirtilmiştir.

**Eşit mi?(==):** “Operatörün solundaki işlenen sağındaki işlenene eşit mi?” testi için kullanılır.

**Eşit Değil mi?(~=):** “Operatörün solundaki işlenen sağındaki işlenene eşit değil mi?” testi için kullanılır.

**Büyük mü?(>):** “Operatörün solundaki işlenen sağındaki işlenenden büyük mü?” testi için kullanılır.

**Büyük veya Eşit mi?(>=):** “Operatörün solundaki işlenen sağındaki işlenenden büyük veya eşit mi?” testi için kullanılır.

**Küçük mü?(<):** “Operatörün solundaki işlenen sağındaki işlenenden küçük mü?” testi için kullanılır.

**Küçük veya Eşit mi?(<=):** “Operatörün solundaki işlenen sağındaki işlenenden küçük veya eşit mi?” testi için kullanılır.

İlişkisel operatörler kullanım amaçları doğrultusunda sonuç olarak 1 (true) veya 0 (false) değerlerini üretirler. Bu durum komut penceresinde aşağıdaki tablodaki testlere benzer birkaç test ile daha iyi anlaşılabilir.

KOMUT PENCERESİ					
>> <b>5==6</b> <b>ans = 0</b>	>> <b>5~=6</b> <b>ans = 1</b>	>> <b>5&gt;6</b> <b>ans = 0</b>	>> <b>5&gt;=6</b> <b>ans = 0</b>	>> <b>5&lt;6</b> <b>ans = 1</b>	>> <b>5&lt;=6</b> <b>ans = 1</b>

**İPUCU:** Eşitlik testinde iki adet “==” işaretini kullanılır. Oysa değişken atamalarında kullandığımız eşittir “=” bir tanedir. MATLAB komut penceresinde  $3=5$  deyimini çalıştırduğumda bir hata mesajı ile karşılaşırız. Oysa  $3==5$  çalıştırıldığında bu “3, 5’e eşit midir?” anlamına gelir, karşılaştırma yanlıştır ve MATLAB yorumlayıcısı bu durum için 0 sonucunu üretir.

```
>> 3=5  
???
```

**Error: The expression to the left of the equals sign is not a valid target for an assignment.**

```
>> 3==5  
ans =  
0
```

## İlişkisel Operatörlerin Öncelik Sıraları

İlişkisel operatörler aritmetik operatörlerden daha düşük bir işlem önceliğine sahiptirler. Örneğin komut penceresinde  $3+4<7-5$  şeklinde çalıştırılacak bir **MATLAB** deyimi arka planda önce  $7<2$  formuna dönüştürülür ve ilişkisel test daha sonra gerçekleştirilir. Benzer şekilde  $i+j==10$  gibi bir eşitlik testi, öncelikle hesaplanan  $i+j$  aritmetik işleminin sonucu ile  $10$  sayısı arasında gerçekleştirilir. Karışıklığa sabep olmak istemiyorsak  $(3+7)<(7-5)$  ve  $(i+j)==10$  şeklinde kullanımları tercih edebiliriz.

## MANTIKSAL OPERATÖRLER

İlişkisel operatörler yardımıyla aynı anda sadece bir durumun test edilebileceğini öğrendik. Bilgisayar programları yazarken bir ilişkisel testten çok daha kompleks testler sonrasında bir karara varmamız beklenir. Bu tür durumlarda birden fazla ilişkisel durumu birbirine mantıksal olarak bağlayacak operatörlere ihtiyaç duyuyoruz. **MATLAB**’ın bize sağladığı üç adet mantıksal operatör işlenenlerini mantıksal olarak karşılaştırarak ilişkisel operatörlerde olduğu gibi 1 veya 0 değerlerini üretir. **MATLAB**’da kullanılan mantıksal operatörler ve açıklamaları aşağıda verilmiştir.

```
% true 1 ( 0 dışındaki tüm değerler true kabul edilir )  
% false 0
```

**a** Operatörün solundaki işlenen ile sağndaki işleneni **Mantıksal VE** işlemine tabi tutar. **Mantıksal VE** operatörü ile birleştirilmiş bir ifadenin 1 sonucu üretebilmesi, işlenenlerin her ikisinin de aynı anda **1(true)** olması ile mümkündür. Diğer tüm kombinasyonlarda sonuç **0(false)** değerini alır.

`and(A, B)`

**Note** The symbols & and && perform different operations in the MATLAB® software. The element-wise AND operator described here is &. The short-circuit AND operator is &&.

**Mantıksal VEYA(||):** Operatörün solundaki işlenen ile sağındaki işleneni **Mantıksal VEYA** işlemine tabi tutar. **Mantıksal VEYA** operatörü ile birleştirilmiş bir ifadenin **1(true)** sonucu üretmesi için işlenenlerinden herhangi birsinin **1(true)** olası yeterlidir. Çıkışın **0(false)** olduğu tek durum her iki işlenenin de aynı anda **0(false)** olduğu durumdur.

**Mantıksal DEĞİL(~):** Operatörün sağındaki işleneni **Mantıksal DEĞİL** işlemine tabi tutar. **Mantıksal DEĞİL** operatörü sağındaki işlenen 0 ise 1 değerini, benzer şekilde sağındaki işlenen 1 ise 0 değerini üretmektedir. Yani işlenenin **DEĞİL** ini almaktadır.

Aşağıdaki komut penceresi çıktısı sayesinde Mantıksal operatörlerin kullanımı ile ilgili bir fikir verilmesi amaçlanmıştır.

```
>> a=5;
>> b=7;
>> (a>6) && (b<8)           %0 ve 1 sonuç=0 dir.
ans =
0
>> (a>6) || (b<8)          %0 veya 1 sonuc=1 dir.
ans =
1
>> ~0                        %0 in değil 1 dir.
ans =
1
```

**İPUCU:**  $3 \leq x < 9$  şeklinde verilen matematiksel bir ifadeyi ele alalım. Bu ifadenin **MATLAB** da doğrudan bir karşılığı yoktur. Bu durumda söz konusu ifadeyi **MATLAB** in yorumlayabileceği bir hale dönüştürmeliyiz. Uygum ilişkisel ve mantıksal operatörler yardımıyla bu matematiksel ifadeyi aşağıdaki gibi bir forma dönüştürmemiz gereklidir.

**(3<=x) && (x<9)**

**İPUCU:** İlişkisel ve Mantıksal operatörler her zaman **bool tipinde (1(true),0(false))** bir sonuç geriye döndürülürler. Dolayısıyla **if** ve **while** deyimleri ile bitlikte karar verme amacıyla kullanılabilirler.

**All**

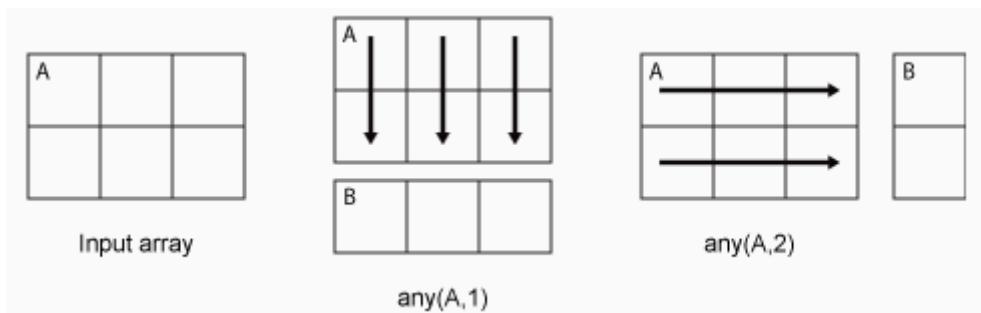
**Any**

**dim** — Dimension to operate along positive integer scalar

Dimension to operate along, specified as a positive integer scalar. If no value is specified, then the default is the first array dimension whose size does not equal 1.

Consider a two-dimensional input array, A:

- `any(A, 1)` works on successive elements in the columns of A and returns a row vector of logical values.
- `any(A, 2)` works on successive elements in the rows of A and returns a column vector of logical values.



*Operatör ile birleştirilebilir. Örneğin sütunlardaki tüm değerlerin pozitif tam sayı olması şartı aranabilir.*

## MATLAB da Fonksiyonlar

Fonksiyonlar farklı programlama dilleri altında değişik adlarla (**prosedür, alt program, rutin, metod**) ifade ediliyor olsalar da temelde belirli işlevleri yerine getirmek üzere tasarlanmış küçük program parçacıkları olarak bilinirler. Aldıkları giriş parametrelerini işleyerek bir çıktıya dönüştürürler. **MATLAB** da tanımlı `sqrt` fonksiyonu ise kendisine parametre olarak verilen bir sayının karekökünü almak üzere tasarlanmıştır. Yani işlev bellidir ve her ihtiyaç duyulduğunda kullanıma hazırlıdır. **MATLAB** da fonksiyonlar kullanıcı-tanımlı fonksiyonlar ile hazır **MATLAB** fonksiyonları olarak ikiye ayrırlar. Kullanıcı tanımlı fonksiyonlardan daha sonra bahsedilecektir. **MATLAB** ile birlikte kullanıma hazır halde gelen ve program yazarken hayatımıza kolaylaşutan bazı matematiksel fonksiyonlar ve kullanım şekilleri aşağıda gösterilmiştir.

MATEMATİKSEL FONKSİYON	MATLAB FONKSİYONU	KULLANIMA ÖRNEK
<b>Sinüs(Açı Değeri Derece Cinsinden)</b>	<code>sind</code>	<code>sind(30)</code>
<b>Sinüs(Açı Değeri Radyan Cinsinden)</b>	<code>sin</code>	<code>sin(pi)</code>
<b>Kosinüs(Açı Değeri Derece Cinsinden)</b>	<code>cosd</code>	<code>cosd(pi)</code>
<b>Eksponansiyel(<math>e^x</math>)</b>	<code>exp</code>	<code>exp(1)</code>
<b>Doğal Logaritma(<math>\ln x</math>)</b>	<code>log</code>	<code>log(exp(1))</code>
<b>2 Tabanında Logaritma(<math>\log_2 x</math>)</b>	<code>log2</code>	<code>log2(16)</code>
<b>10 Tabanında Logaritma(<math>\log_{10} x</math>)</b>	<code>log10</code>	<code>log10(10)</code>

Karekök	<b>sqrt</b>	<b>sqrt(25)</b>
n. Dereceden Kök	<b>nthroot</b>	<b>nthroot(8,3)</b>
Mutlak Değer	<b>abs</b>	<b>abs(-1)</b>

## **MATLAB da Klavyeden Veri Girişi**

**İnput KOMUTU:** Klavyesiz ve monitörsüz bir bilgisayar düşünülemeyeceği gibi(uzaktan erişim imkanı bulunan Linux terminalleri veya benzer diğer bilgisayarlar hariç), işlemek üzere kullanıcıdan klavye yoluyla(veya bir dosyayı okuyarak) veri almaya ve işlediği veriyi kullanıcıya ekran üzerinden (veya bir dosyaya kaydederek) sunmayan bir bilgisayar programı düşünülemez. **MATLAB** programlarına klavye aracılığıyla veri aktarabilmek için **input komutu** kullanılır. İnput komutuyla kullanıcıdan hem bir sayısal veri hem de metinsel bir veri (karakter dizisi, string) temin edilebilir.

### **İnput Komutu ile Klavyeden Sayısal Veri Temini**

Kullanıcıdan sayısal bir veriyi klavye aracılığıyla alabilmek için aşağıdaki gibi bir işlem gerçekleştirilir.

```
>> yas=input('lütfen yaşınızı giriniz:')
lütfen yaşınızı giriniz:21
yas =
21
```

Komut penceresinde çalıştırılan yukarıdaki **MATLAB** deyimi, içerisinde kullanılan input deyimi sayesinde kullanıcıdan aldığı yaş bilgisini yas adında bir değişken içerisinde saklar. Deyim sonuna (;) konulmadığı içinde yapılan bu atama işlemi ekranda görüntülenmiş olur. Açılıp kapanan tek tırnaklar (‘ ’) içerisindeki bir uyarı metni input komutuna parametre olarak verilmelidir. Bu uyarı metni sayesinde kullanıcı, kendisinin girmesi gereken veri hakkında bilgilendirilmiş olur. İnput komutu, işletildikten sonra ekrana yansıtılan uyarı metninin bittiği nokta da imlecin yanıp sönmesini sağlar. Kullanıcı ise tam bu anda klavye yoluyla veriyi girer ve **ENTER** tuşuna basar. Böylece girilen bu veri input komutunun solunda yer alan değişkene aktarılmış olur. Söz konusu bu değişkene yeni bir değer atanmadığı sürece kullanıcının **MATLAB** ortamına aktardığı bu bilgiden istenilen her an faydalанılır.

### **İnput Komutu ile Klavyeden Metinsel Veri Temini**

Kullanıcıdan metinsel bir verinin klavye aracılığıyla programlama ortamına alınmak istenmesi durumunda ise input komutu şu şekilde işletilir.

```
>> isim=input('lütfen isminizi giriniz:')  
lütfen isminizi giriniz:Adem %'s' eklemediğimiz için hata verdi.  
??? Error using ==> input  
Undefined function or variable 'Adem'.  
lütfen isminizi giriniz:27  
isim =  
27
```

```
>> isim=input('lütfen isminizi giriniz:','s')  
lütfen isminizi giriniz:Adem  
isim =  
Adem
```

Bu kullanımda, bir önceki örnektenden farklı şekilde, `input` komutuna birinci parametre olarak aktarılan uyarı metninin yanına ikinci bir parametre olarak da ‘s’ getirilmiştir. Burada s, string kelimesinin ilk harfini temsil eder.

**İPUCU:** Bütün **MATLAB** fonksiyonları parametrelerini açılıp kapanan parantezler () içerisine hapseder. Fonksiyona birden fazla parametre girişi olması durumunda ise parametreler birbirinden virgülerle (,) ayrılır.

## **MATLAB da Ekrana Bilgi Yazma**

Yazdığımız **MATLAB** programlarının üretikleri sonuçları kullanıcıya monitör üzerinden aktarabilmek için kullanabileceğimiz iki **MATLAB** komutu vardır. Bunlar **disp** ve **fprintf** komutlarıdır. Şimdi bu komutları sırasıyla inceleyelim.

### **disp Komutu ile Ekrana Bilgi Yazdırma:disp** komutu ile

- ❖ Metinler,
- ❖ Sayısal değerler veya
- ❖ Metinler ile sayısal değerlerin karışımı ekrana yazdırılabilir.

### **disp Komutu ile Ekrana Metin Yazdırma:**

Yazdığımız programlarda kullanıcıya yönelik uyarı mesajlarını ya da sırf metin çıktılarını genellikle **disp**'in bir kullanımı ile ekrana yansıtırız. **disp** komutunun ekrana metin yazdırma amacıyla kullanımına komut penceresi üzerinden birkaç örnek aşağıda verilmiştir.

```
>> disp('Üzgünüm! Sıfıra Bölüm Hatası Var.');
Üzgünüm! Sıfıra Bölüm Hatası Var.
>> disp('Üzgünüm! Hatalı Giriş Yaptınız.');
Üzgünüm! Hatalı Giriş Yaptınız.
>> disp('Dikkat Payda Sıfır Olamaz!');
Dikkat Payda Sıfır Olamaz!
```

**Disp** komutunun ekrana sadece metin yazdırma amacıyla kullanımı görüldüğü gibi oldukça basittir. Ekrana yazdırılmak istenen metin açılıp kapanan yek tırnak içerisinde alınır ve **disp** komutuna parametre olarak aktarılır. **Disp** komutu ise kendisine parametre olarak verilen metni ekrana yazdırır ve imleci bir alt satırın başına götürür.

### **disp** Komutu ile Ekrana Sayısal Değer Yazdırma

**disp** komutunun ekrana sayısal değer yazdırma amacıyla kullanımına komut penceresi üzerinden verilen birkaç örneği aşağıda inceleyelim.

```
>> skaler=16;
>> satirVektoru=[12 -4 36 25 47];
>> matris=[1 2;3 4];
>> skaler
skaler =
16
>> disp(skaler);
16
>> disp(satirVektoru);
12 -4 36 25 47
>> disp(matris);
1 2
3 4
```

Bu örneğimizde komut penceresi üzerinde özellikle bir skaleri bir satır vektörü ve bir matris ataması gerçekleştirilmiştir. Bu atama deyimlerinin sonuna ise noktalı virgüler koyularak yapılan işlemlerin ekrana çıktı olarak yansımısması engellenmiştir. Daha sonra skaler adlı değişkenin değerinin ekrana yansıtılması için iki farklı yöntem izlenmiştir. İlk yöntem, değişken adının sonuna noktalı virgül koymadan komut penceresinde doğrudan çalıştırırken, ikinci yöntem değeri ekrana yazdırılacak skaler değişkenini **disp** komutuna parametre olarak aktarma yolunu seçmiştir. Örneğimiz son olarak içerisinde bir satır vektörü ve bir matris saklayan değişkenlerin de **disp** ile ekrana kolaylıkla yazdırılabilğini göstermiştir.

**İPUCU:** Bu aşamadan itibaren yazdığımız bütün **MATLAB** deyimlerinin sonuna noktalı virgül koymayı ve ekrana bilgi yazdırmak amacıyla sadece **disp** ve **fprintf** komutlarını kullanmayı alışkanlık haline getirmeliyiz. Satır sonuna eklenmeyen noktalı virgül sayesinde ekrana bilgi yazdırılması işlemini unutalım.

## **disp Komutu ile Ekrana Sayısal Değerleri ve Metinleri Birlikte Yazdırma**

Bir metin ve sayısal bir değer, **disp** komutu ile ekrana yazdırılmak üzere aşağıdaki örnekteki gibi birleştirilebilir.

```
>> tahmin=input('Lütfen Bir Sayı Giriniz: ');
Lütfen Bir Sayı Giriniz: 12
>> disp(['Girdığınız Sayı: ' num2str(tahmin) ' dir. ']);
Girdığınız Sayı: 12 dir.
```

**MATLAB**ın hazır fonksiyonlarından olan **num2str**, kendisine parametre olarak verilen sayısal bir değeri karakter dizisine çevirir. Örneğin 12 sayısını ‘12’ yapar. Metinleri ve sayısal değerleri bir arada kullanmak, iki farklı grubu birleştirmek gibi düşünülebilir. Mevcut durumda birbirlerine dönüştürmek gereklidir. Metinler karakter dizileridir. Sayısal değerleri de **num2str** ile karakter dizilerine dönüştürerek metinlerle bir arada kullanabiliriz. **Disp’**e bu işlem için açılıp kapanan kare parantezler içerisinde tanımlı satır vektörü formunda bir parametre verildiğine dikkat ediniz.

## **fprintf Komutu ile Ekrana Bilgi Yazdırma**

**fprintf komutu/fonksiyonu** ekrana bilgi yazdırmak amacıyla genel olarak aşağıdaki şekliyle kullanılır:

**fprintf(format,değişkenler);**

**fprintf** komutunun format adlı birinci bileşeni/parametresi tek tırnaklar arasında tanımlanan ve ekrana yazdırılacak olan bilgiyi temsil eden karakter dizisidir. Değişkenler isimli ikinci bileşen ise format içerisinde belirlenen kurallara göre değeri ekrana yazdırılacak olan ve birbirinden virgülerle ayrılan değişkenleri ifade eder. Duruma göre **fprintf** sadece format bileşeninden de oluşabilir; tek bir değişkene de sahip olabilir.

Öte yandan format içerisinde kullanılan ve **fprintf** için özel anlam ifade eden 2 karakter vardır. Bunlardan birincisi **% (yüzde)** ve ikincisi de **\(ters bölü)** karakteridir. Söz konusu her iki karakter de kendilerini takip eden karakterle birlikte değerlendirilirler.

Format içerisinde rastlanılan \ karakteri ile onu takip eden karakterin birleşiminin icra edeceği işlev aşağıda listelenmiştir.

**\n:** İmleci bir alt satırın başına götürür.(**n, newline**)

**\t:** İmleci bir **tab** kadar sağa kaydırır.

Diğer taraftan format içerisinde rastlanılan % karakteri ile onu takip eden karakterin birleşimi aşağıdaki tabloda tanımlanan işlevlerden birini yerine getirir.

<b>%c</b>	<b>İlgili değişkenin tek bir karakter olduğunu gösterir.</b>
<b>%s</b>	<b>İlgili değişkenin bir karakter dizisi(string) olduğunu gösterir.</b>
<b>%d</b>	<b>İlgili değişkenin bir tam sayı olduğunu gösterir.</b>

<b>%f</b>	<b>İlgili değişkenin bir ondalıklı sayı olduğunu gösterir.</b>
<b>%g</b>	<b>İlgili değişkeni mümkün olan en kompakt formda gösterir.</b>

**Fprintf** komutunun ise çalışma prensibi ise basitçe şöyledir. **Fprintf**, kendisine açılıp kapanan tek tırnaklar (") içerisinde parametre olarak verilen metni soldan sağa doğru karakter karakter ekrana basar. Bu ekrana basma yolculuğu esnasında % karakteri ile ilk karşılaşlığında durur ve ekranın imlecin bulunduğu noktaya, metni kapatın tek tırnak sonrasında ilk virgülü takip eden değişkenin değerini basar. Yazdırılacak değişkenin sakladığı verini tipinin % karakterini takip eden karakterle tanımlanan cinsten olmasına dikkat edilmelidir. Örneğin ekrana yazdırılacak metin **%d** içeriyorsa, virgülden sonra yer alan ve değeri imlecin bulunduğu pozisyona bastırılacak olan değişkenin içerisinde bir tam sayı olması beklenir. **Fprintf** komutu benzer şekilde karşılaştığı ikinci % karakteri yerine, ekrana, kapanan tek tırnaktan sonraki virgülü takip eden değişkenin değerini basar ve yoluna devam eder. **Fprintf**, ekrana karakter basma yolculuğu sırasında \ karakterine rastlarsa bir sonraki karakterin durumuna göre bir işlem gerçekleştirir. Eğer **n** görürse (\n) bir alt satıra gitme karakterini ekrana basar yani imleci bir alt satırın başına götürür. Eğer **t** görürse (\t) imleci **tab** kadar sağa kaydırır.

Bir örnek üzerinde **fprintf** komutunun kullanımını daha detaylı olarak inceleyelim.

```
>> karakter='d';
>> isim='deniz';
>> tamsayi=25;
>> ondalikliSayi=3.1416;
>> fprintf('Tanimlanan Karakter =%c',karakter);
Tanimlanan Karakter =d>>
>> fprintf('Tanimlanan String =%s\n',isim);
Tanimlanan String =deniz
>> fprintf('Tanimlanan Tamsayı =%d\n',tamsayi);
Tanimlanan Tamsayı =25
>> fprintf('Tanimlanan Ondalıklısı =%f\n',ondalıkliSayi);
Tanimlanan ondalıklısı =3.141600
>> fprintf('Tanimlanan Ondalıklısı =%g\n',ondalıkliSayi);
Tanimlanan ondalıklısı =3.1416
>> fprintf('Tam= %d ve Ondalıklı= %g\n',tamsayi,ondalıkliSayi);
Tam= 25 ve Ondalıklı= 3.1416
```

Yukarıdaki örnekte öncelikli olarak dört farklı değişken ataması gerçekleştirilmiştir. Birinci **fprintf** komutu, ekrana yazdırılacak metnin sonunda \n kullanılmadığı için imleci metnin bittiği yerde bırakmıştır ve >> karakteri satır başında gözükmemiştir. (ikinci **fprintf** komutundan önce **enter** tuşuna basılarak >> karakteri satır başına manuel olarak taşınmıştır.) Son **fprintf** komutu ise metin içerisinde hem **%d** ve hem de **%g** kullanmıştır. Bu sayede metin içerisindeki **%d** karakterinin olduğu pozisyona **tamsayı** adlı değişkenin değerinin ve

**%g** karakterinin olduğu pozisyonda **ondalıkliSayı** isimli değişkenin değerinin bastırılması hedeflenmiştir.

**İPUCU:** **fprintf** içerisinde **%f** nin kullanıldığı durumlarda tam sayı değerler **ondaklı** olarak temsil edilebilsinler diye **MATLAB** tarafından tam sayıların sağına gereksiz sıfırlar eklenir. Çoğu zaman estetik olmayan sonuçlar doğuran bu tür durumların önüne geçebilmek için **%g** kullanılabilir. Örneğin yukarıdaki örnekte dördüncü **fprintf** komutu içerisinde **%f** kullanılmıştır ve ondalıklı sayı ekrana 3.141600 şeklinde yazdırılmıştır. Öte yandan, beşinci **fprintf** komutu **%f** yerine **%g** formatını kullanmıştır ve bu sayede, ondalıklı sayı ekrana sonundaki sıfırlar atıldıktan sonra en kompakt haliyle, yani sadece 3.1416 şeklinde yazdırılmıştır. Özetlemek gerekirse, herhangi bir matematiksel işlemin sonucunun tam sayı mı ondalıklı sayı mı olacağının önceden kestirilemeyeceği durumlarda sonucu saklayan değişkenin değerini **fprintf** komutu ile birlikte **%g** formatını kullanarak ekrana yazdırmak estetik açıdan daha uygundur.

**İPUCU:** **fprintf** komutu ile içerisinde **%** karakteri barındırmayan dolayısıyla değişken bileşeni olmayan karakter dizileri de ekrana yazdırılabilir. Örneğin **disp('Şampiyon FENERBAHÇE');** deyiminin yerine **fprintf('Şampiyon FENERBAHÇE');** deyimi de kullanılabilir. Ve bu sayede ekranda **Şampiyon FENERBAHÇE** yazısı görüntülenebilir.

**İPUCU:** **disp** komutu ekrana çıktı verdikten sonra imleci bir alt satırın başına otomatik olarak götürür. İmleci, **fprintf** komutu çıktısı sonrası bir alt satırın başına götürmek ise kullanıcı sorumluluğundadır ve bu işlem ancak **\n** kullanımı ile mümkündür. Ayrıca **disp** komutu satır veya sütun vektörleri ile matrisleri ekrana kolayca yazdırırken, aynı işlemi **fprintf** ile yapabilmek daha çok işlem gerektirmektedir.

### **fprintf ile Alan, Hızalama ve Duyarlılık Kontrolü**

**fprintf** komutu ile **alan, hızalama ve duyarlılık** kontrolü de yapılabilir. Şöyledir ki:

- ❖ **fprintf** ile ondalıklı sayıların ayraçtan itibaren kaç basamak duyarlılıkla ekrana yazdırılacağı belirlenebilir.
- ❖ **fprintf** ile ekrana yazdırılacak verinin imlecin bulunduğu noktadan itibaren kaç karakterlik bir alanı kullanacağı belirlenebilir.
- ❖ **fprintf** ile bir verinin kendisine rezerve edilen alanın sağunda veya solunda hızalanabileceği belirlenebilir.

Bu türden uygulamalarda **%** karakteri ile onu takip eden ve bir önceki bölümde tanımlanan özel karakterlerin arasına yeni karakterler eklenir. Açıklamalarımızda kullanmak üzere örneğin **%f** yi ele alalım ve **%± 6.2f** nin hangi yeni çıktı formatlarını beraberinde getirdiğini inceleyelim.

**%± 6.2f** içerisindeki:

<b>f</b>	İlgili değişkenin bir ondalıklı sayı olduğunu gösterir.
<b>2</b>	İlgili değişkenin değerinin ayraçtan sonra 2 basamak duyarlılıkla ekrana yazdırılması gerektiğini belirtir.

.	Ayraç karakteridir.
6	İlgili değişkenin değerinin imlecin bulunduğu noktadan itibaren kendisine rezerve edilecek olan 6 karakterlik bir alana yazdırılacağını belirtir.
+	İlgili değişkenin kendisine rezerve edilen alanın sağına hizalanmış olarak ekrana yazdırılacağını belirtir.
-	İlgili değişkenin kendisine rezerve edilen alanın soluna hizalanmış olarak ekrana yazdırılacağını belirtir.

Aşağıdaki komut penceresi çıktısı üzerinden örnekler bu konuyu pekiştirmek amacıyla kullanılmıştır.

```
>> fprintf('pi Sayısının Değeri: %f dir.\n',pi);
pi Sayısının Değeri: 3.141593 dir.
>> fprintf('pi Sayısının Değeri: %.3f dir.\n',pi);
pi Sayısının Değeri: 3.142 dir.
>> fprintf('pi Sayısının Değeri: %8.3f dir.\n',pi);
pi Sayısının Değeri: 3.142 dir.
>> fprintf('pi Sayısının Değeri: %-8.3f dir.\n',pi);
pi Sayısının Değeri: 3.142 dir.
>> fprintf('Merhaba %10s\n','Dünya');
Merhaba    Dünya
>> fprintf('%9dx8+%d=%d\n',123,3,987);
123x8+3=987
>> fprintf('%-9dx9+%2d=%10d\n',12345,6,111111);
12345  x9+ 6=  111111
```

## MATLAB da Veri Tipleri

Veri tipi (data type) program içinde kullanılacak değişken, sabit, fonksiyon isimleri gibi tanımlayıcıların tipini, yani bellekte ayrılmak bölgemin büyüklüğünü, belirlemek için kullanılır. Bir programcı, bir programlama diliinde ilk olarak öğrenmesi gereken, o dile ait veri tipleridir. Çünkü bu, programcının kullanacağı değişkenlerin ve sabitlerin sınırlarını belirler.

**MATLAB** da toplam 14 adet temel veri tipi bulunmaktadır. Daha öncede ifade edildiği üzere varsayılan olarak girilen her değişken double tipinde olduğu kabul edilir. Özel bir tanımlama yapmak gerektiğinde yalnızca o değişken için seçilen tipi belirtmek gereklidir. Data tipleri C++ da tanımlandığı gibi aynı isimlerle burada da kullanılabilir. Daha geniş bilgi için help komutundan faydalananarak dat tipleri konusunda bilgi alınabilir. C++ da olduğu gibi **MATLAB** ayrıca işaretli (signed) ve işaretsiz (unsigned) veri tiplerini de tanımlayabilmektedir.

MATLAB da veriler mantıksal, karakter, sayısal, hücresel ve yapısal olmak üzere beşen ayrırlar. Buradaki en önemli bölüm sayılardır. Sayılarda işaretli ve işaretsiz tamsayılar, single ve double olmak üzere üç gruba ayrılır. Bu grupların tablo gösterimi ve açıklamaları aşağıda yer almaktadır.

**MATLAB** da yerleşik fonksiyonların hemen hemen tamamı double ve char veri türleriyle işleme konur.

Tür	Tanım	Uzunluk(Byte)	Sınır Aralığı
<b>int8</b>	<b>8 bit işaretli tam sayı</b>	<b>1</b>	<b>[-128,+127]</b>
<b>uint8</b>	<b>8 bit işaretsiz tam sayı</b>	<b>1</b>	<b>[0,+255]</b>
<b>int16</b>	<b>16 bit işaretli tamsayı</b>	<b>2</b>	<b>[-32768,+32767]</b>
<b>uint16</b>	<b>16 bit işaretsiz tamsayı</b>	<b>2</b>	<b>[0,+65535]</b>
<b>int32</b>	<b>32 bit işaretli tamsayı</b>	<b>4</b>	<b>[-2147483648,+2147483647]</b>
<b>uint32</b>	<b>32 bit işaretsiz tamsayı</b>	<b>4</b>	<b>[0,+4294967295]</b>
<b>int64</b>	<b>64 bit işaretli tamsayı</b>	<b>8</b>	<b>[-92234e14,+92234e14]</b>
<b>uint64</b>	<b>64 bit işaretsiz tamsayı</b>	<b>8</b>	<b>[0,+18447e15]</b>
<b>single</b>	<b>Tek hassasiyetli sayı</b>	<b>4</b>	<b>[-3e38,+3e38]</b>
<b>double</b>	<b>Çift hassasiyetli sayı</b>	<b>8</b>	<b>[-1e308,+1e308]</b>
<b>logical</b>	<b>Lojik sayı</b>	<b>1</b>	<b>0 , 1</b>
<b>char</b>	<b>Karakter</b>	<b>2</b>	<b>[0 ,+65355]</b>

**İşaretli Tamsayılar(int8, int16, int32, int64):** Ondalıklı (küsuratlı) olmayan sayılardır. Pozitif veya negatif ve 8, 16, 32 veya 64 bit uzunlukta olabilirler. 8 bit bir byte olduğuna göre sırasıyla bellekte 1, 2, 4 ve 8 byte yer kaplarlar. Bu türlerde tanımlanan değişkenlerde tutulabilecek sayıların aralığı değişkenin bit uzunluğu ile tayin edilebilir. Yukarıdaki tabloda işaretli tamsayıların sınır aralıkları yer almaktadır.

**İşaretsiz Tamsayılar(uint8, uint16, uint32, uint64):** Ondalıklı (küsuratlı) olmayan sayılardır. İşaretli tamsayıların aksine sadece pozitif değerler içerebilirler. 8, 16, 32 ve 64 bit uzunluğunda olabilirler. Bu türlerde tanımlanan değişkenlerde tutulabilecek sayıların aralığı değişkenin bit uzunluğu ile tayin edilebilir. Yukarıdaki tabloda işaretsiz tamsayıların sınır aralıkları yer almaktadır.

**Tek Hassasiyette Sayılar(single):** Ondalıklı sayılardır. Bellekte 4 byte yer kaplarlar. Bu türde tanımlanan değişkenlerde double veri türüne göre daha küçük sayılar tutulabilir. Sınır aralığı yukarıdaki tabloda görülebilir.

**Çift Hassasiyette Sayılar(double):** Ondalıklı sayılardır. Bellekte 8 byte yer kaplarlar. Bu türün MATLAB için çok büyük bir önemi vardır. MATLAB da tanımlı bütün matematiksel işlemler çift hassasiyette sayılarla gerçekleştirilirler. Dolayısı ile tamsayı ve tek hassasiyette sayılarla gerçekleştirilecek işlemlerde, bütün değişkenler işlemden önce double türüne dönüştürülmelidir. Ancak tamsayı ve tek hassasiyette sayılar çift hassasiyette sayılarla göre hafızayı daha verimli kullanırlar.

**Mantıksal(Lojik) Sayılar:** Lojik bir değişken 1 ya da 0 sayılarından oluşarak doğru ya da yanlış değerlerini temsil ederler. MATLAB, ilişkisel ve lojik işlemlerden geriye lojik değerler döndürür.

**Karakter(char):** Karakterleri tutar. Dizideki her katar aynı uzunlukta olduğu sürece mxn boyunda katar dizisini temsил etmek için char kullanılır. Eşit uzunlukta olmayan katarı temsил etmek için cell array (hücre dizi) kullanılır.

**Karakter Katarı (String):** Karakterler katarı veya yalnızca katar/sözcük (string), iki tek tırnak arasındaki ifade edilen, gerçekte ASCII kod tablosunda sayısal kodlarla belirtilen ilk 127 karakterden oluşan karakter dizileridir (char array). Katar uzunluğu, katardaki karakter sayısıdır. Her bir karakter bellekte 1 byte (8 bit) yer kaplar. Stringlerle ilgili işlemlerin yer aldığı konu ilerleyen derslerde daha ayrıntılı olarak işlenecektir.

**Hücre dizi (cell array):** Farklı tür ve boydaki diziler, bir hücre dizi içindeki hücrelerde saklanabilir. Hücre Dizileri (Cell arrays) { } ile tanımlanır. Büylesi bir dizi, farklı matrisleri aynı isim altında tutmak ve işlemek için kullanılmaktadır. Örneğin,

`C{1}=[1 2;3 5],C{2}=[4 4 4 4];C{3}=([('yıldız teknik'),(' elektronik')]);` girildiğinde, C bir hücre dizisi olur. Bu hücre geri çağrıldığında,

```
>> C{1}=[1 2;3 5],C{2}=[4 4 4 4];C{3}=([('yıldız teknik'),(' elektronik')]);
C =
[2x2 double]
>> C
C =
[2x2 double] [1x4 double] [1x24 char] %her bir hücre ayrı ayrı işlenir.
```

**C=cell(n):** nxn hücreden oluşan boş bir hücreyi C'ye atar. Örneğin n=2 için aşağıdaki hücre oluşturulur.

```
>> C=cell(2)  
C =  
  []  []  
  []  []
```

Bir hücrenin içine istenilen sayıda yeni hücreler eklemek mümkündür. Örneğin, C{1}{1}=[2,3] ile C aşağıdaki biçimde değişir;

```
>> C{1}{1}=[2,3]  
C =  
  {1x1 cell}  []  
  []  []
```

**Yapı (structure):** Benzer olmayan veri türlerini saklayan hücre diziye benzer. Ancak bu durumda veriler, hücrelerde değil de adlandırılmış alanlarda (named field) saklanır. Yapı dizileri (Structure arrays), veri tabanları için oldukça kullanışlı bir dizi türüdür.

**Kısaca Yapı Dizilerinin Oluşturulması:** Yapı dizisini oluşturacak birimler için ortak bir yapı dizisi değişken ismi belirlenir. İlk yapı dizisinin ismi olmak üzere, yapı dizisini oluşturacak birimlere isimler verilir ve aralarına nokta(.) işaret konulur. Yapı dizileri için genel kullanım, yapı adı. birim adı şeklindeki bir yapı dizisi örneği verilmiştir.

```
>> A.isim= 'Adem';  
>> A.soyisim= 'Ulu';  
>> A.universite= 'KOÜ';  
>> A.sehir= 'KOCAELİ';  
>> A.yil= '2015';  
>> A  
A =  
  isim: 'Adem'  
  soyisim: 'Ulu'  
  universite: 'KOÜ'  
  sehir: 'KOCAELİ'  
  yil: '2015'
```

Hücre ve yapı dizileri, mat uzantılı dosyalar olarak, daha önce açıklanan save komutuyla kaydedilip, load komutuyla geri çağrılabilir.

## MATLAB da Şartlı Deyimler

### IF-ELSEIF-ELSE YAPISI

**İf Komutu:** if komutu bir şartın (condition) gerçekleşmesi / gerçekleşmemesi durumunda yapılacak işlemleri (statement) belirler. Buna örnek olarak bir sayının belli bir değerden büyük / küçük / eşit olup olmadığı şartları olabilir. if komutunun MATLAB' da farklı kullanımları vardır.

- ❖ if yapısı
- ❖ if-else yapısı
- ❖ if-elseif-else yapısı

if deyiminin kullanım şekilleri için temel yazım şekilleri aşağıdaki gibidir.



**ÖRNEK:** MATLAB ekranından öğrencinin notu sorulsun. Kullanıcı ekrandan öğrencinin notunu gırsın. Eğer öğrenci 60 ve 60'dan yukarı not almış ise o zaman 'Öğrenci sınavı geçti.' mesajı versin.

```
%METİN DÜZENLEYİCİSİ%
sinav_notu = input('ogrencinin notunu giriniz: ');
if sinav_notu >= 60
    disp('Ogrenci sinavi gecti')
end
```

```
%KOMUT PENCERESİ%
>>ogrenci_notu
ogrencinin notunu giriniz: 60
Ogrenci sinavi gecti
>>ogrenci_notu
ogrencinin notunu giriniz: 30
```

If komutunun yanındaki şart (condition) doğru olduğunda bu 1 sinyali üretecektir. Yanlış olduğunda ise 0 sinyali üretecektir. if komutu, yanındaki ifade doğru olduğunda (1 sinyalı ürettiğinde), if ile end arasındaki ifadeleri gerçekleştirecektir. Yukarıdaki örnekte, kullanıcı tarafından girilen öğrenci notu 60 ve 60'dan büyük ise 'Ogrenci sinavi gecti' mesajı verecektir. Eğer değil ise hiç bir işlem gerçekleştirmeyecektir.

**ÖRNEK:** Şimdi ise kullanıcı ekrandan öğrencinin notunu girsin. Eğer öğrenci 60 ve 60'dan yukarı not almış ise o zaman ‘Öğrenci sınavı geçti.’ , eğer değil ise ‘Öğrenci sınavdan kaldı’ mesajı versin.

```
%METİN DÜZENLEYİCİSİ%
sinav_notu = input('Ogrencinin notunu giriniz' );
if sinav_notu >= 60
    disp('Ogrenci sınavı geçti')
else
    disp('Ogrenci sınavdan kaldı')
end
```

```
%KOMUT PENCERESİ%
>> ogrenci_notu
Ogrencinin notunu giriniz50
Ogrenci sınavdan kaldı
>> ogrenci_notu
Ogrencinin notunu giriniz80
Ogrenci sınavı geçti
```

If komutununlarındaki şart (condition) doğru olduğunda (1 sinyali ürettiğinde) altındaki (if ile end arasındaki ) ifadeleri gerçekleştirdiğini bir önceki örnekte göstermişik. Ancak bir önceki örnekte şart gerçekleşmez ise (0 sinyali ürettiğinde) program hiç bir işlem gerçekleştirmemişti. Şimdi bu MATLAB koduna else ifadesi ekleyerek, mevcut şartların hiç biri gerçekleşmez ise programa ne yapacağını belirtiyoruz. Bundan dolayı else ifadesinin yanında hiçbir zaman bir şart ifadesi yazmıyoruz. Çünkü else hiçbir koşulun gerçekleşmediği durumda yapılacak ifadeyi belirtir. Kendisi yeni bir şart getirmez. Yeni bir şart işlemini ‘elseif ‘ ifadesi ile getiririz. Onun için ‘elseif’ ile ‘else’ ifadeleri karıştırılmamalıdır.

Yukarıdaki örnekte kullanıcı tarafından girilen sınav notunun 60 ve 60'dan büyük olmaması durumda ne yapılacağı (‘Öğrenci sınavdan kaldı’ mesajının verilmesi) else ile belirtilmiştir.

Şimdiki örneğimizde ise kullanıcının girdiği sınav notuna göre bunu “zayıf – orta – iyi – pekiyi” şeklinde sınıflandıralım.

- ❖ 0-49 Zayıf
- ❖ 49 – 69 Orta
- ❖ 70- 84 İyi
- ❖ 85-100 Pekiyi

### %METİN DÜZENLEYİCİSİ%

```
sinav_notu = input('Ogrencinin notunu giriniz: ');
if sinav_notu > 84
    disp('Pekiyi')
elseif sinav_notu > 69
    disp('İyi')
elseif sinav_notu > 49
    disp('Orta')
else
    disp('Zayıf')
```

### %KOMUT PENCERESİ%

```
>> ogrenci_notu
Ogrencinin notunu giriniz: 70
İyi
>> ogrenci_notu
Ogrencinin notunu giriniz: 10
Zayıf
>> ogrenci_notu
Ogrencinin notunu giriniz: 100
Pekiyi
>> ogrenci_notu
Ogrencinin notunu giriniz: 60
Orta
```

If – elseif -else yapılarında program sırasıyla şartları sınar, sırasıyla sınadığı şartlardan herhangi birisi doğru olduğunda (1 sinyali ürettiğinde ) geri kalan şartları sınamaz, onların doğru olup olmadığını kontrol etmez. Misal verecek olursak, kullanıcı tarafından sınav notu 90 olsun. Bu hem 84'den büyük olma, hem 69'dan büyük olma, hem de 49'dan büyük olma şartlarını sağlar. Ancak program çalışırken 84'den büyük olma koşulunu sağlar ve ‘Pekiyi’ mesajını yazdırır. Bu koşulu sağladı ve gerekli ifadeyi gerçekleştirdiği için diğer ifadeleri sınamaz. Yani ‘iyi’ ve ‘orta’ mesajlarını VERMEZ.

Bir önceki örnekte de vurguladığımız gibi ‘else’ ifadesinin yanında bir şart ifadesi bulunmaz. Else geriye kalan tüm olasılıklar demektir. Bu da bu program için 0 ila 49 arasında bulunma durumudur.

**ÖRNEK:** 0 ile 10 arasında rastgele üretilen bir sayıyı bulmak için kullanıcından sayı isteyen ve girilen sayının üretilen sayıdan büyük mü, küçük mü yada eşit mi olduğunu bulan MATLAB programını yazınız.

```
%METİN DÜZENLEYİCİSİ%
uretilen = round ( 10 * rand (1));
girilen = input(' tahminizi giriniz: ');
if ( uretilen < girilen )
    fprintf(' daha küçük bir sayı giriniz\n');
elseif ( uretilen > girilen )
    fprintf(' daha büyük bir sayı giriniz\n');
else
    fprintf(' tebrikler bulundunuz\n');
end
```

```
%KOMUT PENCERESİ%
>> sayı_bulma
tahminizi giriniz: 5
daha küçük bir sayı giriniz
>> sayı_bulma
tahminizi giriniz: 5
daha büyük bir sayı giriniz
>> sayı_bulma
tahminizi giriniz: 5
tebrikler bulundunuz
```

## SWITCH – CASE YAPISI

İf yapısının bir benzeri olan bu yapıda anahtar değere karşılık gelen birçok seçenek içinden uygun olanı seçmeliśiniz. Bu yapı ile yapılan her algoritma if ile de yapılır. Ancak bu işlemleri if ile yapmak istersek çok fazla elseif yapısı kullanmamız gereklidir. Kullanımı;

**Switch (durum)**

```
case (durum1)
    işlemler
case (durum2)
    işlemler
.
.
.
otherwise
    ► isteğe bağlı
```

**end**

**ÖRNEK:** Verilen ay değerine karşılık, o ayın kaç günden olduğunu hesaplayan MATLAB programını yazınız.

```
%METİN DÜZENLEYİCİSİ%
ay = input('ay giriniz(1,2,...,11,12): ');
switch ay
    case {1,3,5,7,8,10,12}
        fprintf(' 31 gün\n');
    case 2
        yil = input('yıl giriniz: ');
        if(mod(yil,4)==0)
            fprintf(' 29 gün\n');
        else
            fprintf(' 28 gün\n');
        end
    case {4,6,9,11}
        fprintf(' 30 gün\n');
    otherwise
        fprintf(' hatalı ay girişi\n');
end
```

```
%KOMUT PENCERESİ%
>> ay_gun
ay giriniz(1,2,...,11,12): 1
31 gün
>> ay_gun
ay giriniz(1,2,...,11,12): 2
yıl giriniz: 2015
28 gün
>> ay_gun
ay giriniz(1,2,...,11,12): 4
30 gün
>> ay_gun
ay giriniz(1,2,...,11,12): 30
hatalı ay girişi
```

**ÖRNEK:** Kullanıcının gireceği bir düzgün çokgen için, iç açılar toplamını hesaplayan bir MATLAB programı yazınız.

**%METİN DÜZENLEYİCİSİ%**

```
cokgen = input('bir çokgen giriniz : ','s');
switch cokgen
    case 'ucgen'
        kenar_sayisi=3;
    case { 'dortgen', 'dörtgen', '4gen' }
        kenar_sayisi=4;
    case 'beşgen'
        kenar_sayisi=5;
    otherwise
        fprintf(' geçersiz giriş \n ');
        kenar_sayisi=0;
end
sonuc = (kenar_sayisi - 2)*180;
if kenar_sayisi ~=0
    fprintf('sonuc : %d \n ',sonuc );
end
```

**%KOMUT PENCERESİ%**

```
>> cokgen_aci
bir çokgen giriniz : ucgen
sonuc : 180
>> cokgen_aci
bir çokgen giriniz : 4gen
sonuc : 360
>> cokgen_aci
bir çokgen giriniz : beşgen
sonuc : 540
>> cokgen_aci
bir çokgen giriniz : ongen
geçersiz giriş
```

**MATLAB** in ondalıklı sayıları yuvarlama amacıyla sıkılıkla kullanılan 4 adet hazır fonksiyonu vardır. Bu fonksiyonlar **fix**, **round**, **ceil** ve **floor** fonksiyonlarıdır. Aşağıda bu fonksiyonlar sırayla açıklanacaktır.

**Fix Fonksiyonu:** **fix(x)** fonksiyonu kendisine parametre olarak aldığı x ondalıklı sayısını 0 yönünde karşılaştığı ilk tamsayıya yuvarlar. Aşağıdaki tablo fix fonksiyonunun nasıl çalıştığını özetlemek amacıyla kullanılmıştır.

>>fix(3.2) ans= 3	>>fix(3.5) ans= 3	>>fix(3.99) ans= 3	>>fix(-3.865) ans= -3
----------------------	----------------------	-----------------------	--------------------------

**Round Fonksiyonu:** **round(x)** fonksiyonu kendisine parametre olarak aldığı x ondalıklı sayısını kendisine en yakın tamsayıya yuvarlar. (*Ondalıklı kısmı 0.5 e eşit veya ondan büyük pozitif sayılar kendisinden büyük en yakın tam sayıya, negatif sayılar ise kendisinden küçük en yakın tam sayıya yuvarlanır.*) Aşağıdaki tablo round fonksiyonunun nasıl çalıştığını özetlemek amacıyla kullanılmıştır.

>>round(3.2) ans= 3	>>round(3.5) ans= 4	>>round(3.99) ans= 4	>>round(-3.865) ans= -4
------------------------	------------------------	-------------------------	----------------------------

**Ceil Fonkdiyonu:** **ceil(x)** fonksiyonu kendisine parametre olarak aldığı x ondalıklı sayısını pozitif sonsuz yönünde karşılaştığı ilk tamsayıya yuvarlar. Ceil fonksiyonunun negatif ondalıklı sayılar için fix fonksiyonu ile aynı işlemi görüdüğünne dikkat ediniz. Aşağıdaki tablo ceil fonksiyonunun nasıl çalıştığını özetlemek amacıyla kullanılmıştır.

>>ceil(3.2) ans= 4	>>ceil(3.5) ans= 4	>>ceil(3.99) ans= 4	>>ceil(-3.865) ans= -3
-----------------------	-----------------------	------------------------	---------------------------

**Floor Fonksiyonu:** **floor(x)** fonksiyonu kendisine parametre olarak aldığı x ondalıklı sayısını negatif sonsuz yönünde karşılaştığı ilk tamsayıya yuvarlar. floor fonksiyonunun pozitif ondalıklı sayılar için fix fonksiyonu ile aynı işlemi görüdüğünne dikkat ediniz. Aşağıdaki tablo floor fonksiyonunun nasıl çalıştığını özetlemek amacıyla kullanılmıştır.

>>floor(3.2) ans= 3	>>floor(3.5) ans= 3	>>floor(3.99) ans= 3	>>floor(-3.865) ans= -4
------------------------	------------------------	-------------------------	----------------------------

### Bir Sayının Tam Sayı veya Ondalıklı Sayı Olup Olmadığının Araştırılması

Bir sayının tam sayı mı yoksa ondalıklı sayı mı olduğu herhangi bir yuvarlama fonksiyonu kullanılarak anlaşılabılır. Şöyled ki; bir sayı tam sayı ise sayının kendisi sayının yuvarlanması eşittir. Benzer şekilde bir sayı yuvarlanması eşit değilse bu sayının bir ondalıklı sayı olduğu sunucu çıkarılabilir. **MATLAB** da eşitlik testi için kullanılan (==) veya

eşitsizlik testi için kullanılan ( $\sim=$ ) ilişkisel operatörleri sayesinde, bu türden işlemler aşağıdaki örneklerinde gösterdiği üzere kolaylıkla gerçekleştirilebilir.

>>3==round(3) ans= 1	>>3.8==round(3.8) ans= 0	>>4.793 ~=round(4.793) ans= 1
-------------------------	-----------------------------	----------------------------------

Yukarıdaki testler sonuç olarak **1(true)** veya **0(false)** üretiklerine göre if deyiminin yanında koşul olarak kullanılabilirler.

## **Mod, Rand, Zeros ve Ones Fonksiyonları**

### **Mod Fonksiyonu**

**Mod Fonksiyonu ile Kalan Bulma:** Kendisine iki parametre alan ve en genel formu  $\text{mod}(x,y)$  şeklinde tanımlanabilen MATLAB fonksiyonu  $x$  sayısının  $y$  sayısına bölümünden kalanı verir. Aşağıdaki komut penceresi çıktıları mod fonksiyonunun nasıl çalıştığını özetlemek amacıyla kullanılmıştır.

>>mod(25,3) ans= 1	>>mod(246,17) ans= 8	>>mod(1563,379) ans= 47
-----------------------	-------------------------	----------------------------

**Mod Fonksiyonu ile Bir Sayının Çift veya Tek Sayı Olup Olmadığının Araştırılması:** Bir sayının çift sayı mı yoksa tek sayı mı olduğu mod fonksiyonu kullanılarak anlaşılır. Şöyle ki; bir sayı ikiye tam olarak bölünebiliyorsa yani sayının ikiye bölümünden kalan sıfırsa bu sayı çift sayıdır. Benzer şekilde sayının ikiye bölümünden kalan 1 ise bu sayının bir tek sayı olduğu sonucu çıkarılabilir. Bir sayının çiftliği veya tekliği aşağıdaki örneklerin de kullanıldığı testler sayesinde kolaylıkla anlaşılır.

>>mod(8,2)==0 ans= 1	>>mod(9,2)==0 ans= 0	>>mod(9,2)==1 ans= 1
-------------------------	-------------------------	-------------------------

Yukarıdaki testler sonuç olarak **1(true)** veya **0(false)** üretiklerine göre if deyiminin yanında koşul olarak kullanılabilirler.

**İPUCU:** Bir  $x$  sayısının bir  $i$  sayısına kalansız olarak bölünüp bölünmediğini test edebilmek için aşağıdaki 2 MATLAB deyiminden herhangi birisi kullanılabilir.

- i.      **if**  $\text{mod}(x,i) == 0$
- ii.     **if**( $x/i == \text{fix}(x/i)$ )

**İPUCU:** Bir  $x$  sayısı bir  $y$  sayısına bölündüğünde bölüm  $\text{fix}(x/y)$  ve kalan da  $\text{mod}(x,y)$  sayesinde hesaplanabilir. Şöyle ki;

$$X = Y * \text{Bölüm} + \text{Kalan} \quad \text{veya} \quad X = Y * \text{fix}(X/Y) + \text{mod}(X/Y)$$

## Rand Fonksiyonu

**Rand Fonksiyonu ile Rastgele Sayı Üretilmesi:** Rastgele sayı üretilmesine yardımcı olan hazır fonksiyonlar bütün programlama dillerinde mevcuttur ve programcılar bu fonksiyonlar sayesinde ilginç uygulamalar geliştirebilirler. **MATLAB** da tanımlı **rand(n,m)** fonksiyonu nxm boyutunda (n satırlı ve m sütunlu ) ve her bir elemanı 0 ile 1 arasında rastgele bir ondalıklı sayı içeren bir matris üretmek amacıyla kullanılır. Oluşturulan bu rastgele sayılar düzgün dağılımlıdır. Üretilmek istenen matris bir kare matris ise ( $n=m$ ) **rand(n,n)** yerine **rand(n)** kullanılabilir. Dolayısıyla **0 ile 1** arasında rastgele ondalıklı bir skaler sayı üretmek için **rand(1)** fonksiyonunun kullanımı yeterlidir.

0 ile 1 aralığında rastgele ondalıklı sayılarından oluşan bir matrisin tüm elemanlarını **k** gibi bir skaler ile çarparak her bir matris elemanın değerini **0 ile k** aralığına çekebiliriz. Ondalıklı sayılarından oluşmuş bir matrisin tüm elemanlarını yuvarlayıp tam sayı yapmak için ise herhangi bir yuvarlama fonksiyonundan, örneğin **round** fonksiyonundan yararlanabiliriz.

Aşağıdaki örnekler **MATLAB** da **rand** fonksiyonu kullanılarak rastgele sayı üretmenin nasıl yapıldığını özetlemektedir.

```
>> a=rand(2,3)
a =
0.8147  0.1270  0.6324
0.9058  0.9134  0.0975
```

Yukarıdaki **MATLAB** deyimi komut penceresinde her çalıştırıldığında 2 satırlı ve 3 sütunlu bir a matrisi oluşturur ve her bir matris elemanına 0 ile 1 aralığında rastgele bir ondalıklı sayı atar.

```
>> a=40*rand(2,3)
a =
11.1399  38.3003  6.3045
21.8753  38.5955  38.8237
```

Yukarıdaki **MATLAB** deyimi komut penceresinde her çalıştırıldığında 2 satırlı ve 3 sütunlu bir a matrisi oluşturur ve her bir matris elemanına **0 ile 40** aralığında rastgele bir ondalıklı sayı atar.

```
>> a=round(40*rand(2,3))
a =
38  32  17
19  6  37
```

Yukarıdaki **MATLAB** deyimi komut penceresinde her çalıştırıldığında 2 satırlı ve 3 sütunlu bir **a matrisi** oluşturur ve her bir matris elemanına **0 ile 40** aralığında rastgele bir **tamsayı** atar.

**İPUCU:** Yukarıdaki üçüncü örnekte içten dışa doğru bir yol takip edilerek sonuca ulaşıldığına dikkat ediniz. Öncelikle  $rand(2,3)$  kullanılarak  $2 \times 3$  boyutunda ve her bir elemanı  $0$  ile  $1$  aralığında rastgele bir ondalıklı sayı içeren bir matris oluşturulur. Daha sonra bu matrisin her bir elemanı  $40$  sayısı ile çarpılarak matris elemanlarının  $0$  ile  $40$  arasında rastgele ondalıklı sayılarından oluşması sağlanır. En sonunda  $40 * rand(2,3)$  deyimi  $round$  fonksiyonu içerisine hapsedilerek. Herbir matris elemanın kendisine en yakın tamsayıya yuvarlanması sağlanır. Böylelikle her bir elemani  $0$  ile  $40$  aralığında rastgele bir tamsayı içeren ve  $a$  adlı değişken içerisinde saklanan  $2$  satırlı ve  $3$  sütunlu bir matris oluşturulmuş olur. Bu deyim MATLAB komut penceresinde her çalıştırıldığında farklı rastgele sayılar içeren bir  $a$  matrisinin oluşturulacağını unutmuyınız.

```
>> a=round(10+40*rand(2,3)) %10 ile 50 aralığında rastgele tamsayılar atar.  
a =  
43 32 17  
19 16 37
```

```
>> a=round(rand(2,3)) %her bir elamana rastgele 0 veya 1 atar.  
a =  
1 1 0  
0 0 1
```

```
>> a=40*round(rand(2,3)) %her bir elamana rastgele 0 veya 40 atar.  
a =  
40 0 40  
40 40 0
```

```
>> a=10+40*round(rand(2,3)) %her bir elamana rastgele 10 veya 50 atar.  
a =  
10 10 50  
50 50 10
```

```
>> a=round(1+999*rand(1)) %1 ile 1000 aralığında rastgele tamsayı skaler üretir.  
a =  
602
```

**Rand** fonksiyonu ile oluşturulan  $0$  ile  $1$  aralığındaki rastgele bir ondalıklı sayı  $round$  kullanıldığında ya  $0$  a ya da  $1$  e yuvarlanır.

**Rand** fonksiyonu ile oluşturulan  $0$  ile  $1$  aralığındaki rastgele bir ondalıklı sayı  $fix$  kullanıldığında her zaman  $0$  a yuvarlanır.

```
>> a=fix(rand(2,3))  
a =  
    0     0     0  
    0     0     0
```

### Zeros ve Fonksiyonu

**Zeros Fonksiyonu ile Sadece 0 İçeren Bir Matrisin Oluşturulması:** MATLAB da tanımlı **zeros(n,m)** fonksiyonu nxm boyutunda n satırlı ve m sütunlu ve her bir elemanı 0 olan bir matris üretmek amacıyla kullanılır. Üretilmek istenen bir kare matris ise sadece **zeros(n)** kullanılabilir. Aşağıdaki komut penceresi çıktıları zeros fonksiyonunun nasıl çalıştığını özetlemek amacıyla kullanılmıştır.

```
>> a=zeros(3,4)  
a =  
    0     0     0     0  
    0     0     0     0  
    0     0     0     0  
>> a=zeros(1,5)  
a =  
    0     0     0     0     0  
>> a=zeros(2)  
a =  
    0     0  
    0     0  
>> a=fix(rand(2,3))  
a =  
    0     0     0  
    0     0     0
```

**İPUCU:** Yukarıdaki komut penceresi çıkışının da özetlediği üzere **zeros(n,m)** fonksiyonu kullanılmadan da **fix(rand(n,m))** sayesinde sadece 0 içeren n xm boyutunda bir matris oluşturulabilir.

### Ones Fonksiyonu

**Ones Fonksiyonu ile Sadece 1 İçeren Bir Matrisin Oluşturulması:** **MATLAB** da tanımlı ones(n,m) fonksiyonu nxm boyutunda n satırlı ve m sütunlu ve her bir elemanı 1 olan bir matris üretmek amacıyla kullanılır. Üretilmek istenen bir kare matris ise sadece ones(n) kullanılabilir. Aşağıdaki komut penceresi çıktıları ones fonksiyonunun nasıl çalıştığını özetlemek amacıyla kullanılmıştır.

```
>> a=ones(3,5)
a =
    1   1   1   1   1
    1   1   1   1   1
    1   1   1   1   1
>> a=ones(4,1)
a =
    1
    1
    1
    1
>> a=ones(3)
a =
    1   1   1
    1   1   1
    1   1   1
>> a=ones(1,3)
a =
    1   1   1
>> a=ceil(rand(2,4))
a =
    1   1   1   1
    1   1   1   1
```

**İPUCU:** Yukarıdaki komut penceresi çıktısının da özetlediği üzere **ones(n,m)** fonksiyonu kullanılmadan da **ceil(rand(n,m))** sayesinde sadece 1 içeren nxm boyutunda bir matris oluşturulabilir.

## **MATLAB da Döngüler**

## FOR DÖNGÜSÜ

Başlangıç, bitiş ve adım değerleri arasında dönen bir döngüdür. Eğer adımlarınız birer birer artıyor ise adım değerini yazmasanız da olur.

```
for parametre = başlangıç değeri : artım değeri : bitiş değeri
    olay1
    olay2
    ....
end
```

ÖRNEK: 1 den 10 a kadar olan sayıları ekranaya yazdırın programı yazınız.

```
%KOMUT PENCERESİ%
clc;
for i=1:1:10
    fprintf(' %d - ',i);
end
1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - >>
```

ÖRNEK: Kullanıcıdan istenen bir sayının faktöriyelini hesaplayan MATLAB programını yazınız.

```
%METİN DÜZENLEYİCİSİ%
sayi =input(' bir sayı giriniz : ');
if(sayı>0)
    faktoriyel =1; % çarpım olduğundan ilk değeri 1 aldık
    for i=1:1:sayı
        faktoriyel =faktoriyel * i;
    end
    fprintf(' %d\n',faktoriyel);
else
    fprintf(' sizin 1 den küçük olamaz \n');
end
```

```
%KOMUT PENCERESİ%
```

```
>> faktor
```

```
bir sayı giriniz : 5
```

```
120
```

```
>> faktor
```

```
bir sayı giriniz : -5
```

```
sayınız 1 den küçük olamaz
```

**ÖRNEK:** Bilinen bir alt değişkenden üst değişkene kadar olan sayıların ardışık toplamını ekrana yazdırın MATLAB programı yazınız.

```
%METİN DÜZENLEYİCİSİ%
```

```
toplam=0;
```

```
bas=input('bir başlangıç değeri giriniz : ');
```

```
son=input('bir bitiş değeri giriniz : ');
```

```
if bas>son
```

```
    fprintf('başlangıç değeri bitiş değerinden büyük olmamalı!\n');
```

```
else
```

```
    for i=bas:1:son
```

```
        toplam=toplam+i;
```

```
    end
```

```
    fprintf('toplam : %d\n',toplam);
```

```
end
```

```
%KOMUT PENCERESİ%
```

```
>> toplam_deger
```

```
bir başlangıç değeri giriniz : 20
```

```
bir bitiş değeri giriniz : 30
```

```
toplam : 275
```

```
>> toplam_deger
```

```
bir başlangıç değeri giriniz : 20
```

```
bir bitiş değeri giriniz : 10
```

```
başlangıç değeri bitiş değerinden büyük olmamalı!
```

## İÇ İÇE DÖNGÜLER

Algoritma geliştirirken bir döngü içerisinde bir veya daha fazla döngü kullanmak gerekebilir. Bu gibi durumlarda dışarıdaki döngünün bir kez dönmesi sonucu içerisindeki döngü tam döngüsünü tamamlayacaktır. Yani ilk olarak içerisindeki döngü tamamlanacaktır.

ÖRNEK: Resimdeki gibi  $10 \times 10$  çarpım tablosunu ekrana yazdırın MATLAB programını yazınız.

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>1</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>2</b>	<b>2</b>	<b>4</b>	<b>6</b>	<b>8</b>	<b>10</b>	<b>12</b>	<b>14</b>	<b>16</b>	<b>18</b>	<b>20</b>
<b>3</b>	<b>3</b>	<b>6</b>	<b>9</b>	<b>12</b>	<b>15</b>	<b>18</b>	<b>21</b>	<b>24</b>	<b>27</b>	<b>30</b>
<b>4</b>	<b>4</b>	<b>8</b>	<b>12</b>	<b>16</b>	<b>20</b>	<b>24</b>	<b>28</b>	<b>32</b>	<b>36</b>	<b>40</b>
<b>5</b>	<b>5</b>	<b>10</b>	<b>15</b>	<b>20</b>	<b>25</b>	<b>30</b>	<b>35</b>	<b>40</b>	<b>45</b>	<b>50</b>
<b>6</b>	<b>6</b>	<b>12</b>	<b>18</b>	<b>24</b>	<b>30</b>	<b>36</b>	<b>42</b>	<b>48</b>	<b>54</b>	<b>60</b>
<b>7</b>	<b>7</b>	<b>14</b>	<b>21</b>	<b>28</b>	<b>35</b>	<b>42</b>	<b>49</b>	<b>56</b>	<b>63</b>	<b>70</b>
<b>8</b>	<b>8</b>	<b>16</b>	<b>24</b>	<b>32</b>	<b>40</b>	<b>48</b>	<b>56</b>	<b>64</b>	<b>72</b>	<b>80</b>
<b>9</b>	<b>9</b>	<b>18</b>	<b>27</b>	<b>36</b>	<b>45</b>	<b>54</b>	<b>63</b>	<b>72</b>	<b>81</b>	<b>90</b>
<b>10</b>	<b>10</b>	<b>20</b>	<b>30</b>	<b>40</b>	<b>50</b>	<b>60</b>	<b>70</b>	<b>80</b>	<b>90</b>	<b>100</b>

```
%METİN DÜZENLEYİCİSİ%
fprintf('\t%3d %3d %3d %3d %3d %3d %3d %3d %3d %3d \n',1,2,3,4,5,6,7,8,9,10);
fprintf('-----\n');
for i=1:10
    fprintf('%2d |',i);
    for j=1:10
        fprintf('%3d ',i*j);
    end
    fprintf('\n');
end
```

%KOMUT PENCERESİ%

```
>> carpim_tablosu
      1 2 3 4 5 6 7 8 9 10
-----
```

<b>1</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>2</b>	<b>2</b>	<b>4</b>	<b>6</b>	<b>8</b>	<b>10</b>	<b>12</b>	<b>14</b>	<b>16</b>	<b>18</b>	<b>20</b>
<b>3</b>	<b>3</b>	<b>6</b>	<b>9</b>	<b>12</b>	<b>15</b>	<b>18</b>	<b>21</b>	<b>24</b>	<b>27</b>	<b>30</b>
<b>4</b>	<b>4</b>	<b>8</b>	<b>12</b>	<b>16</b>	<b>20</b>	<b>24</b>	<b>28</b>	<b>32</b>	<b>36</b>	<b>40</b>
<b>5</b>	<b>5</b>	<b>10</b>	<b>15</b>	<b>20</b>	<b>25</b>	<b>30</b>	<b>35</b>	<b>40</b>	<b>45</b>	<b>50</b>
<b>6</b>	<b>6</b>	<b>12</b>	<b>18</b>	<b>24</b>	<b>30</b>	<b>36</b>	<b>42</b>	<b>48</b>	<b>54</b>	<b>60</b>
<b>7</b>	<b>7</b>	<b>14</b>	<b>21</b>	<b>28</b>	<b>35</b>	<b>42</b>	<b>49</b>	<b>56</b>	<b>63</b>	<b>70</b>
<b>8</b>	<b>8</b>	<b>16</b>	<b>24</b>	<b>32</b>	<b>40</b>	<b>48</b>	<b>56</b>	<b>64</b>	<b>72</b>	<b>80</b>
<b>9</b>	<b>9</b>	<b>18</b>	<b>27</b>	<b>36</b>	<b>45</b>	<b>54</b>	<b>63</b>	<b>72</b>	<b>81</b>	<b>90</b>
<b>10</b>	<b>10</b>	<b>20</b>	<b>30</b>	<b>40</b>	<b>50</b>	<b>60</b>	<b>70</b>	<b>80</b>	<b>90</b>	<b>100</b>

WHİLE DÖNGÜSÜ

Belirlenen şart sağlandıkça dönen döngü tipidir. Bu döngüde dikkat edilmesi gereken nokta şart sağlandıkça döngünün sürekli döneceğidir. Eğer önlem alınmaz ise sonsuz döngüye sebep olur. While ve for döngülerini birbirinin yerine kullanılabılır, doğru olanı seçmek bizim görevimizdir.

## while şart

şart sağlandıkça olaylar  
end

ÖRNEK: Küpü 4000 den büyük ve 3 ile bölünebilen en küçük doğal sayıyı bulan programı yazınız.

```
%METİN DÜZENLEYİCİSİ%
sayi=1;
x=1;
while x==1;
    if ( (sayi^3)>=4000 && mod(sayi,3)==0 )
        fprintf('aranan rakam : %d\n',sayi);
        x=0;
    else
        fprintf('olmadı : %d\n',sayi);
        sayi=sayı+1;
    end
end
```

```
%KOMUT PENCERESİ%
>> while_dongu
olmadı : 1
olmadı : 2
olmadı : 3
olmadı : 4
...
olmadı : 15
olmadı : 16
olmadı : 17
aranan rakam : 18
```

ÖRNEK: x=10 ve y=7 olmak üzere, bu sayıların toplamı 52757 den büyük oluncaya kadar, x sayısını 2 kez katlayarak toplama işlemeye devam ediliyor. Buna göre x sayısı kaç kez katlanmıştır.

```
%METİN DÜZENLEYİCİSİ%
x=10;
y=7;
toplam = x+y;
katlanma =0;
while(toplam<52757)
    x = 2 * x;
    toplam = x+y;
    katlanma =katlanma +1;
end
fprintf(' ay : %d \n', katlanma);
```

```
%KOMUT PENCERESİ%
```

```
>> kac_kat
```

```
ay : 13
```

## BREAK DEYİMİ

Bir döngüyü sona erdirmek için kullanılan komuttur. MATLAB bu komutu gördüğünde içinde bulunduğu döngüyü sonlandırır ve bu komutun altındaki komutları değerlendirmez.

**ÖRNEK:** Kullanıcı negatif sayı girinceye kadar yeni sayı isteyen ve girilen sayıları toplayan bir MATLAB programı yazınız.

```
%METİN DÜZENLEYİCİSİ%
toplam=0;
while(1)
    sayi = input('sayi giriniz = ');
    if( sayi < 0)
        break;
    end
    toplam=toplam + sayi ;
end
fprintf('toplam = %g\n ', toplam);
```

```
%KOMUT PENCERESİ%
```

```
>> negatif  
sayı giriniz = 5  
sayı giriniz = 9  
sayı giriniz = 25  
sayı giriniz = 32  
sayı giriniz = 7  
sayı giriniz = -9  
toplam = 78
```

## CONTINUE DEYİMİ

Döngüyü sonraki çevirmeye geçirir. Bu komutun altındaki komutlar o anki çevirme için değerlendirilmez.

**ÖRNEK:** Continue kullanarak 1 ile 100 arasındaki tek sayıları ekrana alt alta yazdırın MATLAB programını yazınız.

```
%METİN DÜZENLEYİCİSİ%
```

```
for i=1:100  
    if mod(i,2)==0  
        continue;  
    end  
    fprintf('%d\n ',i);  
end
```

```
%KOMUT PENCERESİ%
```

```
>> tek_sayı  
1  
3  
5  
7  
9  
...  
87  
89  
91  
93  
95  
97  
99
```

## Try – Catch Yapısı

Mantıksal hatayı kontrol eder ( yazı hatalarını kontrol etmez ). İf yapısına benzer şekilde dallanma sağlanır. İki durumda hangisi mantıksal olarak doğru ise onu çalıştırır. İkisi de doğru ise try kısmı çalışır.

```
try  
    durum1  
catch  
    durum2  
end
```

```
%METİN DÜZENLEYİCİSİ%  
a=10;%başlangıç değeri 10  
try a=a+20; % a nin yeni değeri 30 oldu  
    a=a+g; % g diye bir sayı olmaz ( mantık hatası ) bu satrı atlar  
catch  
    a=50; % yeni değeri 50  
end  
display(a); % sonucu 30 yerine 50 yazar , çünkü 2. satrı atlar
```

```
%KOMUT PENCERESİ%  
>> deneme  
a =  
50
```

**RETURN KOMUTU:** Yazdığımız MATLAB programlarının herhangi bir anda ( programın normalde sona erdiği noktadan daha önce ) sonlandırılması için kullanılır. Bir diğer deyiş ile programı durdurup kapatma komutudur.

**ERROR:** Return' a benzer bir komut olan **error** komutunda, program bir hata ile birlikte sonlandırır. Return komutunda ise herhangi bir hata durumu yoktur.

**PAUSE KOMUTU:** **pause** komutu programınızı geçici olarak dondurmanızı sağlar. Devam etmek için klavyeden herhangi bir tuşa basabilirsiniz ya da belirtilen süre sonunda otomatik olarak kendiliğinden devam eder. **pause(n)** komutu n saniye kadar geçici durdurma yapar.

## Trigonometrik Fonksiyonlar

Trigonometrik hesaplamalarda açıların radyan olarak verilmesi gerekmektedir. Bu ayrıntı unutulmamalıdır. Aynı şekilde ters trigonometrik hesaplamalar sonucunda da sonuçlar

radyan cinsinden çıkacaktır. Aşağıdaki işlemlerde açılar önce derece olarak tanımlanmış daha sonra radyana çevrilmiştir. Aşağıda kullanılan trigonometrik fonksiyonların isimleri ve bu fonksiyonlar ile ilgili örnekler sırayla verilecektir.

Sin	Sinüs fonksiyonu
Cos	Kosinüs fonksiyonu
Tan	Tanjant fonksiyonu
Asin	Arksinüs fonksiyonu
Acos	Arkkosinüs fonksiyonu
Atan	Arktanjant fonksiyonu
Atan2	Arktanjant fonksiyonu(açının birim çemberdeki yeri)
Sinh	Hiperbolik sinüs fonksiyonu
Cosh	Hiperbolik kosinüs fonksiyonu
Tanh	Hiperbolik tanjant fonksiyonu
Asinh	Hiperbolik arksinüs fonksiyonu
Acosh	Hiperbolik arkkosinüs fonksiyonu
Atanh	Hiperbolik arktanjant fonksiyonu

```
>> d=[0 30 60 90 120 150 180];
>> r=pi/180*d;
>> sin(r)
ans =
    0   0.5000   0.8660   1.0000   0.8660   0.5000   0.0000
>> cos(r)
ans =
    1.0000   0.8660   0.5000   0.0000  -0.5000  -0.8660  -1.0000
>> tan([0 30 60 120 150 180]*pi/180)
ans =
    0   0.5774   1.7321  -1.7321  -0.5774  -0.0000
>> s=[0 1 -1 0.5 3^0.5/2];
>> 180/pi*asin(s)
ans =
    0   90.0000  -90.0000   30.0000   60.0000
>> 180/pi*acos(s)
ans =
    90.0000      0  180.0000   60.0000   30.0000
>> 180/pi*atan(s)
ans =
    0   45.0000  -45.0000  26.5651  40.8934
```

**NOT:** atan2 fonksiyonunun genel yapısı atan2(y,x) eklindedir. Birim çemberi düşünelim. y ve x, işaretleriyle göz önüne alınarak açının pozitif x ekseninden ölçülen değeri elde edilir.  
**Örnek:**

```

>> aci=180/pi*atan2(1,1) %önce y eksenindeki değer verilir atan2(y,x)!!!!!
aci =
    45
>> aci=180/pi*atan2(-1,1)
aci =
   -45
>> aci=180/pi*atan2(1,-1)
aci =
   135
>> aci=180/pi*atan2(-1,-1)
aci =
  -135

```

```

>> sinh([1 -1 0 2])
ans =
    1.1752  -1.1752      0   3.6269
>> cosh([1 -1 0 2])
ans =
    1.5431  1.5431  1.0000  3.7622
>> tanh([1 -1 0 2])
ans =
    0.7616  -0.7616      0   0.9640
>> asinh([1 -1 0 2])
ans =
    0.8814  -0.8814      0   1.4436
>> acosh([1 -1 0 2])
ans =
    0           0 + 3.1416i    0 + 1.5708i  1.3170
>> atanh([1 -1 0 2])
ans =
    Inf        -Inf          0       0.5493 + 1.5708i

```

## LOGARİTMİK VE ÜSTEL FONKSİYONLAR

On tabanında logaritma	$\log_{10}(x)$	$\log10(x)$
2 tabanında logaritma	$\log_2(x)$	$\log2(x)$
Doğal logaritma( $\ln x$ )	$\log_e(x)$	$\log(x)$
Üstel fonksiyon	$e^x$	$\exp(x)$
Karekök fonksiyonu		$\sqrt{x}$

**MATLAB** da matematikten bildiğimiz doğal logaritma gösterimi  $\ln$  olarak değil doğrudan  $\log$  olarak gösterilmektedir. Yine bildiğimiz gibi  $\ln x = \log e^x$  demektir. Genel yazım formatı bir  $x$  değeri için  $\ln x$ , **MATLAB** da  $\log(x)$  şeklindedir.

```
>> log(1) %logaritma e tabanında 2: ln2  
ans =  
0  
>> log(10) %logaritma e tabanında 10: ln10  
ans =  
2.3026
```

MATLAB da e sabit sayısı yani  $e=2.71828$  veya kısaca  $e=2.71$  sayısı e olarak tanımlanmamıştır. Bunun yerine bir sonraki konuda göreceğimiz  $\exp(1)$  fonksiyonu kullanılabilir. Biliyoruz ki  $\ln e = 1$  dir ve bunu MATLAB da sağlayalım:

```
>> exp(1)  
ans =  
2.7183
```

**MATLAB** da matematikten bildiğimiz normal logaritma 10 tabanındadır ve bir x değeri için genel yazım formatı  $\log_{10}(x)$  şeklindedir. Ayrıca **MATLAB**, 2 tabanındaki logaritma içinde hazır bir fonksiyon sağlar. Bir x değeri için genel yazım formatı  $\log_2(x)$  şeklindedir. Doğal logaritma da olduğu gibi negatif sayıların logaritmaları reel sayı değildir ve sıfır için değeri sonsuzdur. 0 ile 1 arasındaki (0 ve 1 dahil değil) değerleri negatiftir. Şimdi sırasıyla  $\log 1$ ,  $\log 10$ ,  $\log 100$  ve  $\log 1000$ , sonra  $\log 2$ ,  $\log 5$ ,  $\log 3/5$  ve sonra da  $\log 0$  ve  $\log(-4)$  değerlerini bulalım.

```
>> log10(1)  
ans =  
0  
>> log10(10)  
ans =  
1  
>> log10(1000)  
ans =  
3
```

Bu konu başlığı altında çok kullanılan bir diğer fonksiyonumuz karekök alma işlemini gerçekleştiren **sqrt fonksiyonudur**. Genel yazım formatı bir x değeri için  $\sqrt{x}$  şeklindedir.  $\sqrt{x}$  fonksiyonunu kullanarak sırasıyla  $\sqrt{2}$ ,  $\sqrt{23}$ ,  $\sqrt{144}$ , işlemlerini yapalım.

```
>> sqrt(2)
ans =
1.4142
>> sqrt(23)
ans =
4.7958
>> sqrt(144)
ans =
12
>> sqrt(3+sqrt(2))
ans =
2.1010
```

## KARMAŞIK (KOMPLEKS) SAYI İŞLEMLERİ

Bilindiği gibi kompleks sayıların tipik genel formatı  $a+bi$ ,  $a+bj$  veya  $a+ib$ ,  $a+jb$  şeklindedir. MATLAB dilinde bu notasyon  $a+bi$ ,  $a+bj$  veya  $a+i*b$ ,  $a+j*b$  şeklinde ifade edilir. Bu gösterim şekli aynı zamanda kartezyen gösterim olarak da adlandırılır. Sayılarda  $i$  veya  $j$  kullanımı arasında fark yoktur her ikisi de aynı şeyi ifade ederler. Örnek olarak  $2-3j$  karmaşık sayısını ele alalım.

```
>> 2-3j
ans =
2.0000 - 3.0000i
>> 2-j*3
ans =
2.0000 - 3.0000i
>> 2-j3
??? Undefined function or variable 'j3'.
```

### Temel kompleks sayı işlemleri:

- ❖ **Abs:** Mutlak değer (Absolute value)
- ❖ **Angle:** Faz açısı (Phase angle)
- ❖ **Conj:** Kompleks eşlenik (Complex conjugate)
- ❖ **Imag:** Kompleks imajiner kısım (Complex imaginary part)
- ❖ **Real:** Kompleks reel kısım (Complex real part)

$x=3+4j$  sayısı için özetlersek;

```

>> x=3+4j;
>> real(x)           % karmaşık sayının reel kısmı
ans =
    3
>> imag(x)           % karmaşık sayının sanal kısmı
ans =
    4
>> abs(x)            % karmaşık sayının mutlak değeri
ans =
    5
>> angle(x)          % karmaşık sayının faz açısı
ans =
    0.9273
>> conj(x)           % karmaşık sayının eşleniği
ans =
    3.0000 - 4.0000i

```

**abs (absolute) komutu** sadece karmaşık sayı işlemlerinde değil diğer tüm mutlak değer alma  $|x|$  işlemlerinde kullanılabilir. Fonksiyon adı abs olup genel formatı bir x değeri için abs(x) şeklindedir. Ör: » abs(sqrt(3)-1) ans = 0.7321

### MATLAB da Betimsel İstatistik Komutları

**max(x), min(x), sort(x), prod(x), sum(x), mean(x), median(x), conv(x) ve std(x)** komutları MATLAB da kullandığımız betimsel istatistik komutlarıdır. Bu komutlar yapmak istediğimiz işlemleri büyük ölçüde kolaylaştırmaktadır. Bu komutları aşağıda örnekler yardımıyla inceleyelim.

**Max Komutu:** Verilen bir sayı dizisi içerisindeki en büyük değeri bulmak için kullanılan komuttur. Komutun kullanımına göre en büyük sayı değerinin dizi içerisindeki sırası da elde edilebilir.

```

>>x=[3,-5,4,10,5,20,4,15,12,-8];
>>max(x)
Komut sonucu : 20
>>[y,k]=max(x)
Komut sonucu : 20 (En büyük sayı)
6 (En büyük sayının dizideki yeri)

```

**Min Komutu:** Verilen bir sayı dizisi içerisindeki en küçük değeri bulmak için kullanılan komuttur. Komutun kullanımına göre en küçük sayı değerinin dizi içerisindeki sırası da elde edilebilir.

```
>>x=[3,-5,4,10,5,20,4,15,12,-8];
>>min(x)
Komut sonucu : -8
>>[y,k]=min(x)
Komut sonucu : -8
10
```

Eğer bir matrisin en büyük değeri veya en küçük değeri bulunmak isteniyorsa;

```
>> A=rand(4,4)
A =
0.9501 0.8913 0.8214 0.9218
0.2311 0.7621 0.4447 0.7382
0.6068 0.4565 0.6154 0.1763
0.4860 0.0185 0.7919 0.4057
>> max(max(A))
ans =
0.9501
>> min(min(A))
ans =
0.0185
```

**Sort Komutu:** Verilen bir sayı dizisi içerisindeki sayıları küçükten büyüğe doğru sıralamak için kullanılan komuttur.

```
>> sort(x)
ans =
1 2 4 5 7 8 9
```

**Prod Komutu:** Dizideki tüm elemanların çarpımını veren komut.

```
>> x=[1 2 3 4]
x =
1 2 3 4
>> prod(x)
ans =
24
```

**Sum:** Verilen bir sayı dizisi içerisindeki elemanların toplamını bulmak için kullanılan komuttur.

```
>>x=[3,-5,4,10,5,20,4,15,12,-8];
>>sum(x)
Komut sonucu : 60
```

**Mean Komutu:** Verilen bir sayı dizisi içerisindeki sayıların ortalamasını bulmak için kullanılan komuttur.

**Median Komutu:** Dizinin median değerini veren komut.

**Std Komudu:** Verilen bir sayı dizisi içerisindeki sayıların standart sapmasını bulmak için kullanılan komuttur

```
>> x=[1 1.3 1.5];
>> mean(x)
ans =
1.2667
>> median(x)
ans =
1.3000
>> std(x)
ans =
0.2517
```

## **MATLAB da Polinom Uygulamaları**

Bu bölümde polinomlar ile ilgili **MATLAB** komutlarını göreceğiz. Sırasıyla aşağıdaki işlemleri gerçekleştireceğiz.

- ❖ **MATLAB da** Bir Polinomun Tanıtılması
- ❖ **MATLAB da** Polinomun köklerinin bulunması: **MATLAB** roots komutu
- ❖ **MATLAB da** Polinomların Toplanması
- ❖ **MATLAB da** Polinomların Çarpılması: **MATLAB** conv komutu
- ❖ **MATLAB da** Kökleri Bilinen Bir Polinomu Elde Etme: **MATLAB** poly komutu
- ❖ **MATLAB da** Bir Matrisin Karakteristik Denkleminin Bulunması: **MATLAB** poly komutu
- ❖ **MATLAB da** Polinomda Bilinmeyenin yerine değer atanması: **MATLAB** polyval komutu
- ❖ **MATLAB da** Bir Polinomun Türevinin Alınması: **MATLAB** polyder komutu
- ❖ **MATLAB da** Bir Polinomun İntegralinin Alınması: **MATLAB** polyint komutu
- ❖ **MATLAB da** Polinomial Eğri Uydurulması **MATLAB** polyfit komutu

### **Bir Polinomun MATLAB a Tanıtılması**

$P(s) = a s^4 + b s^3 + c s^2 + d s + e$  şeklindeki bir polinomu tanıtmak için, polinomun katsayılarını kullanacağız. ‘**katsayı**’ adında bir satır vektör tanımlarsak;

```
>> katsayı = [a b c d e];
```

Böylelikle katsayılarını satır vektör olarak girdiğim polinom üzerinde MATLAB ‘ın hazır komutlarını kullanarak, polinomların kökünün bulunması, çarpılması gibi işlemleri kolaylıkla yapabileceğiz.

$P(x) = 5x^5 + 23x^2 + 8$  şeklindeki bir polinomun katsayıları

```
>> katsayı = [5 0 0 23 0 8];
```

$P(s) = s^4 + 2s^3 + s^2 + 3s + 6$  şeklindeki bir polinomu:

```
>> katsayı = [1 2 1 3 6];
```

ekinde komut ekranından girebiliriz.

### MATLAB da Polinomun Köklerinin Bulunması

Polinomun köklerini bulmak için ‘roots’ komutunu kullanacağız. Daha önce katsayılarını bir satır vektöre atadığımız Polinomun köklerini aşağıdaki şekillerde MATLAB a buldurabiliriz.

**Matlab roots komutu:**

**roots(katsayılar)** veya **kokler = roots(katsayılar)**

$P(s) = 1s^2 + 2s + 1$  polinomunun köklerinin **MATLAB** yardımıyla bulunması;

```
>> katsayılar = [1 2 1];
>> roots(katsayılar)
ans =
-1
-1
```

$P(s) = s^4 - 6s^3 + 11s^2 - 6s$  polinomunun köklerinin **MATLAB** da bulunması:

```
>> katsayilar = [1 -6 11 -6 0];
>> kokler = roots(katsayilar)
kokler =
0
3.0000
2.0000
1.0000
```

Bu örnekte görüldüğü üzere polinomun kökleri istenirse, bir değişkenine atanabilir. Burada ‘**kokler**’ isimli değişkene atanmıştır. Polinomun değerleri 0, 3, 2 ve 1’dir.

### MATLAB da Polinomların Toplanması:

$P_1(s) = 3 s^3 + 2 s + 1$  ve  $P_2(s) = 2 s^2 + 10 s + 1$  gibi iki polinomu MATLAB da toplamak isteyelim.

```
>> katsayilar1 = [3 0 2 1];
>> katsayilar2 = [0 2 10 1];
>> katsayilar3 = katsayilar1 + katsayilar2
katsayilar3 =
3 2 12 2
```

İki polinomun toplamı bize  $P_3(s) = 3 s^3 + 2 s^2 + 12 s + 2$  polinomunu verecektir. Burada dikkat edilecek husus vektör toplamında boyut eşitliği arandığından 2. polinomun mertebe olarak bulunmayan mertebelerinin katsayılarına sıfır konmuştur.

### MATLAB da Polinomların Çarpılması

**MATLAB conv komutu:** MATLAB da polinom çarpımı için ‘**conv**’ hazır fonksiyonu vardır. Çarpmak istediğimiz polinomların katsayılarını bu fonksiyona girdiğimizde, cevap olarak çarpım sonucu elde edilen polinomun katsayılarını alırız.

$P_1(s) = 3 s^3 + 2 s + 1$  ve  $P_2(s) = 2 s^2 + 10 s + 1$  gibi iki polinomu çarpmak isteyelim.

```
>> kokler1 = [3 0 2 1];
>> kokler2 = [2 10 1];
>> kokler3 = conv(kokler1, kokler2)
kokler3 =
6 30 7 22 12 1
```

İki polinomun çarpımı bize  $P_3(s) = 6 s^5 + 30 s^4 + 7 s^3 + 22 s^2 + 12 s + 1$  polinomunu verecektir. ‘**conv**’ hazır fonksiyonu katsayılar dizilerinin eşit boyutta (eşit eleman sayısında) olmasını gerektirmez.

### MATLAB da Polinomların Bölünmesi

**Matlab deconv komutu:** Matlab da polinom bölümü için ‘**deconv**’ hazır fonksiyonu vardır. Bölmek istediğimiz polinomların katsayılarını sırasıyla pay ve payda olacak şekilde bu fonksiyona girdiğimizde, cevap olarak bölüm sonucu elde edilen polinomun katsayılarını alırız. ‘**deconv**’ hazır fonksiyonun kullanım şekli aşağıdaki gibidir.

[bolum, kalan]= **deconv(bolunen, bolen )** şeklindedir.

**Matlab** yardımıyla  $P_1(s) = 3 s^3 + 2 s + 1$  polinomunu  $P_2(s) = 2 s^2 + 10 s + 1$  polinomuna bölmek isteyelim.

```
>> kokler1 = [3 0 2 1];
>> kokler2 = [ 2 10 1];
>> [bolum, kalan] = deconv(kokler1, kokler2)
bolum =
    1.5000 -7.5000
kalan =
    0     0   75.5000  8.5000
```

Yani  $P_3(s) = P_1/P_2 = 1.5 s - 7.5 + (75.5 s + 8.5)/(2 s^2 + 10 s + 1)$  şeklinde olacaktır.

### MATLAB da Kökleri Bilinen Bir Polinomu Elde Etme

**Matlab poly komutu:** Matlab da kökleri bilinen bir polinomun katsayıları ‘**poly**’ hazır fonksiyonu ile bulunur. Katsayılarını bulmak istediğimiz polinomun köklerini bir satır vektörde tanımlarız. Bu satır vektörü ‘**poly**’ fonksiyonuna girdiğimizde, cevap olarak polinomun katsayılarını alırız. ‘**poly**’ fonksiyonu bir nevi ‘**roots**’ fonksiyonun tersi yönde işlem görür. ‘**roots**’ fonksiyonu katsayılarından köklere, ‘**poly**’ fonksiyonu ise köklerden polinom katsayılarına ulaşmamızı sağlar.

Kökleri **-2 , -3, ve -4** olan polinomu tanımlayınız. Bu polinom  $P(s) = (s + 2)(s + 3)(s + 4)$  şeklinde tanımlanır.

```
>> kokler = [-2 -3 -4];
>> poly(kokler)
ans =
    1    9   26   2
```

Yani bu üç ifadenin çarpımıyla elde edilecek polinom  $P(s) = s^3 + 9 s^2 + 26 s + 2$

Kökleri **1-3i ve 1+3i** olan polinomu tanımlayınız.

```
>> kokler = [1-3i 1+3i];
>> poly(kokler)
ans =
    1   -2   10
```

Yani  $P(s) = s^2 - 2 s + 10$

## MATLAB da Bir Matrisin Karakteristik Denkleminin Bulunması

**Matlab poly komutu:** A bir matris olmak üzere  $\det(sI - A) = 0$  denklemi bize matrisin karakteristik denklemini verir. Matlab da bir matrisin karakteristik denklemini ‘poly’ hazır fonksiyonu ile buluruz. Genel kullanım formu:

**Karakteristik\_polinom = poly(matris)** şeklindedir.

$A = [0 \ 1; \ 3 \ 5]$  matrisinin karakteristik denklemini bulalım.

```
>> A = [0 1 ; 3 5];
>> poly(A)
ans =
1.0000 -5.0000 -3.0000
```

## Polinomda Bilinmeyenin Yerine Değer Atanması

**Matlab polyval komutu:** Matlab da polinomda bilinmeyenin yerine bir değerin atanması için ‘polyval’ fonksiyonunu kullanılır. Bu fonksiyonum genel kullanım şekli şu şekildedir: **polinomun\_degeri = polyval(katsayılar, atanacak\_deger)**

$P(s) = s^5 + 4 s^3 + 2 s^2$  ise  $z = P(10)$  değeri nedir?

```
>> katsayılar = [1 0 4 2 0 0];
>> z = polyval(katsayılar, 10)
z =
104200
```

## Matlab da Bir Polinomun Türevinin Alınması

**Matlab polyder komutu:** Matlab da bir polinomun türevini almak için ‘polyder’ hazır fonksiyonu vardır. Türevini almak istediğimiz polinomun katsayılarını bir satır vektöre atarız. Bu satır vektörü , ‘polyder’ fonksiyonuna girdiğimizde cevap olarak, polinomun türevi alınmasıyla elde edilen polinomun katsayılarını verecektir.

$P(s) = s^5 + 4 s^3 + 2 s^2$  fonksiyonun türevini **Matlab** yardımıyla bulunuz.

```
>> katsayılar = [1 0 4 2 0 0];
>> turev = polyder(katsayılar)
turev =
5 0 12 4 0
```

Yani  $P(s) = s^5 + 4 s^3 + 2 s^2$  polinomunun türevi  $P(s) = 5 s^4 + 12 s^2 + 4 s$  dir. Genel olarak bir polinomun türevini almak kolaydır. Belki bunun için **Matlaba** çok ihtiyaç duymaya biliriz. Ancak iki polinomun çarpımının türevini elle almak daha zahmetlidir. Neyse ki bu işlemi de **Matlab** ile kolaylıkla yaptırabiliyoruz.

$P_1(s) = 3 s^3 + 2 s + 1$  ve  $P_2(s) = 2 s^2 + 10 s + 1$  gibi iki polinomun çarpımının türevini **Matlab** yardımıyla bulalım.

```
>> katsayi1 = [3 0 2 1];
>> katsayi2 = [ 2 10 1];
>> turev = polyder(katsayi1, katsayi2)
turev =
30 120 21 44 12
```

**Alternatif Çözüm:**

```
>> katsayi1 = [3 0 2 1];
>> katsayi2 = [ 2 10 1];
>> katsayi = conv(katsayi1, katsayi2);
>> turev = polyder(katsayi)
turev =
30 120 21 44 12
```

$P(s) = P_1(s) P_2(s)$  polinomunun türevi:  $dP(s) / ds = 30 s^4 + 120 s^3 + 21 s^2 + 44 s + 12$

### Matlab'da Bir Polinomun İntegralinin Alınması

**Matlab polyint komutu:** Matlab'da bir polinomun integralini almak için ‘**polyint**’ hazır fonksiyonu vardır. İntegralini almak istediğimiz polinomun katsayılarını bir satır vektöre atarız. Bu satır vektörü, ‘**polyint**’ fonksiyonuna girdiğimizde cevap olarak, polinomun integrali alınmasıyla elde edilen polinomun katsayılarını verecektir.

$P_1(s) = 3 s^2 + 2 s + 1$  polinomunun integralini **Matlab** yardımıyla bulalım.

```
>> kokler1 = [3 2 1];
>> integral = polyint(kokler1)
integral =
1 1 1 0      %s^3 + s^2 + s+0
```

Denklemden de görüldüğü üzere integral sabiti default olarak 0 alınmıştır. Eğer sıfırdan başka bir integral sabiti denkleme girilmek isteniyorsa, bu fonksiyonda ikinci bir parametre olarak girilebilir.

```
>> integral = polyint(kokler1,3)
integral =
1 1 1 3
```

Görüldüğü üzere ‘**polyint**’ hazır fonksiyonunda girilen ikinci parametre ile integral sabiti 3 olarak girilmiş oldu.

## Matlab'da Polinomial Eğri Uydurulması

**Matlab polyfit komutu:** Bir dinamik sistemimiz varsa ve bu sisteme çeşitli girişler uygulayıp çıktıları ölçuyorsak, bu veriler ile giriş ve çıkış arasındaki ilişkiyi polinom olarak ifade edebiliriz. Genel kullanım yapısı aşağıdaki gibidir.

**polyfit(giris\_verileri, cikis\_verileri, polinomun\_mertebesi)**

Giriş 0 1 2 3

Çıkış 3 6 40 100

Giriş verilerini x satır vektörüne, çıkış verilerini ise y satır vektörüne atayalım.

```
>> x = [0 1 2 3];
>> y = [3 6 40 100];
>> polyfit(x,y,3)
ans =
-0.8333 18.0000 -14.1667 3.0000
```

Sonuç olarak;  $y(x) = -0.8333x^3 + 18x^2 - 14.1667x + 3$  elde ederiz.

## MATLAB'DA PROGRAMLAMA

MATLAB da programlama genel olarak iki yolla yapılır:

- ❖ Komut penceresinde(inline) programlama
- ❖ M- dosyaları ile (m-files) programlama
  - ❖ Düzyazı(script) m-dosyaları ile programlama
  - ❖ Fonksiyon (function) m-dosyaları ile programlama

**MATLAB**, herhangi bir problemin çözümüne yönelik geliştirdiğimiz algoritmamızı doğrudan komut penceresi üzerinde bir bilgisayar programına dönüştürmemize imkan sağlar. Bu durumda programın kullanacağı veriler ve programın kendisi bellek üzerinde saklanmaktadır. MATLAB kapatıldığında ya da programın yazıldığı bilgisayar enerjisiz kaldığında bütün herşey buhar olup uçup gideceği için komut penceresi üzerinde program geliştirilmesi yolu tercih edilmemelidir. Bu yöntem daha çok küçük çaplı programlama pratikleri yaparken kullanılmalıdır.

Komut penceresi üzerinde programlamanın yukarıda bahsedilen dezavantajından dolayı **MATLAB** bizlere, programlarımızı uzantısı .m olan dosyaların içerisinde yazacağımız ve m-dosyaları ile programlama adı verilen alternatif bir yol sunmaktadır. M-dosyaları ile programlama ise kendi içerisinde ikiye ayrılmaktadır. Düzyazı m-dosyaları ile programlama ve fonksiyon m-dosyaları ile programlama.

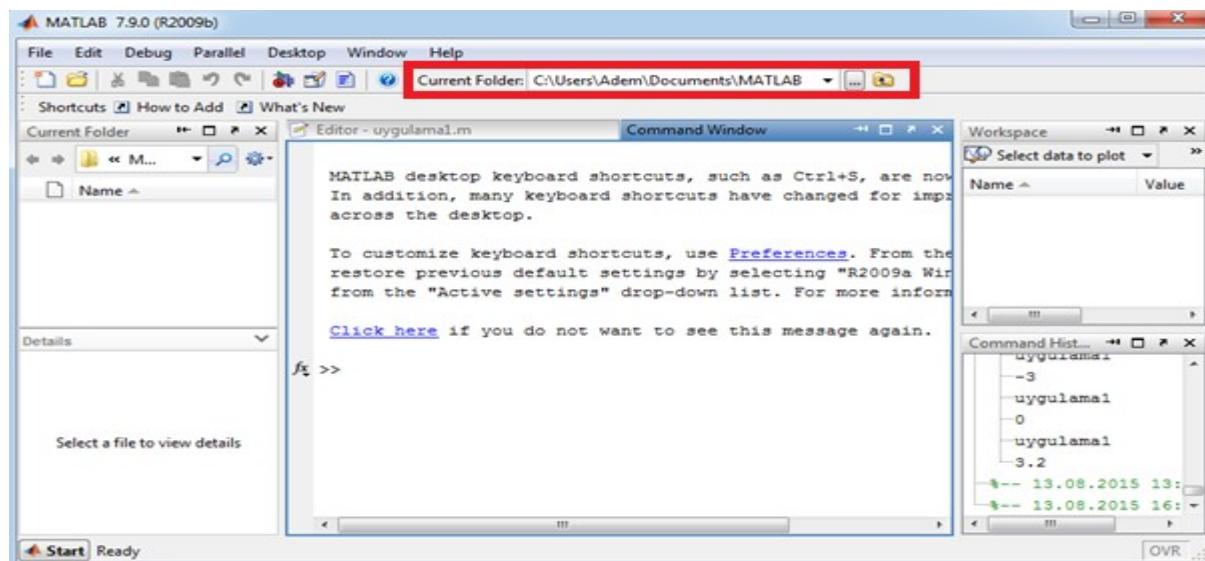
### Düzyazı M-Dosyaları ile Programlama

Bir düzyazı m-dosyası ya da senaryo dosyası, içerisinde ard arda sıralanmış **MATLAB** deyimlerinin saklandığı basit bir metin dosyasından başka bir şey değildir. MATLAB in bizlere programlama ortamı içerisinde sunduğu kendi **metin düzenleyicisi(editor)** oldukça fonksiyoneldir.

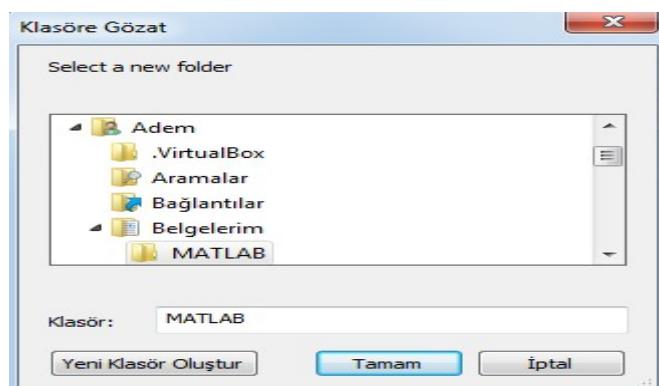
## Düzyazı M-Dosyası Oluşturulması ve Çalıştırılması

Yazdığımız m-dosyalarını ancak ve ancak çalışma dizinimiz altında saklarsak komut penceresinden çalıştırabiliriz. O yüzden çalışma dizinimiz olarak özel bir dizin belirlemeli ve bütün m-dosyalarını bu dizinde saklamalıyız.

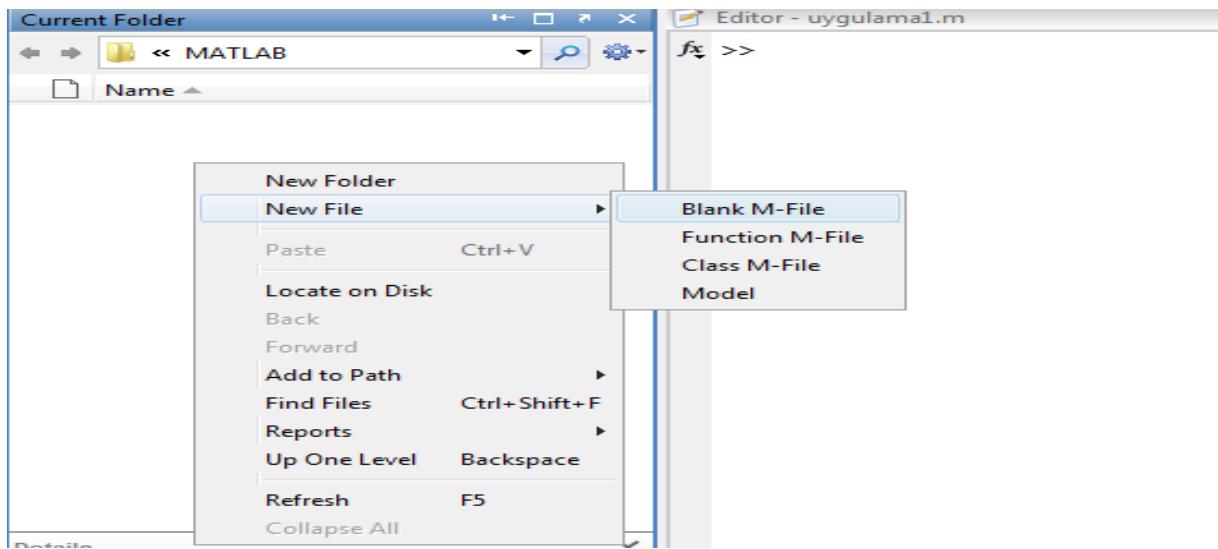
Uygulamayı çalıştırduğumuzda karşımıza aşağıdaki ekran gelmektedir. M- dosyası oluşturmadan önce yapacağımız ilk iş **D** dizini içerisinde **MATLAB\_UYGULAMALARI** adında bir klasör oluşturuyoruz bu klasör sayesinde yaptığımız bütün uygulamaları burada saklayabiliriz. Bu klasörü çalışma dizinimiz yapabilmek için kırmızı ile gösterdiğimiz **Current Folder** bölümünün sonundaki 3 noktayı tıklıyoruz.



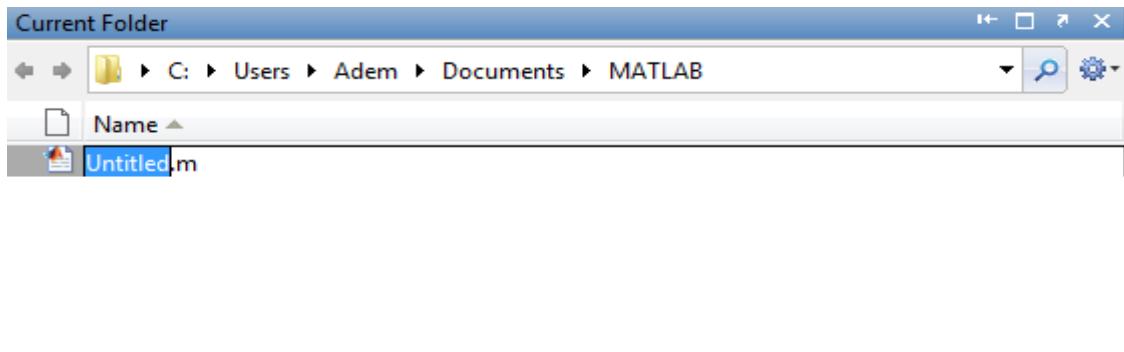
Açılan **Klasöre Gözat** penceresi üzerinde **D** dizininin altında oluşturduğumuz **MATLAB\_UYGULAMALARI** adlı alt dizinine kadar iniyoruz ve bu dizini seçili hale getirdikten sonra **OK** düğmesini tıklıyoruz.



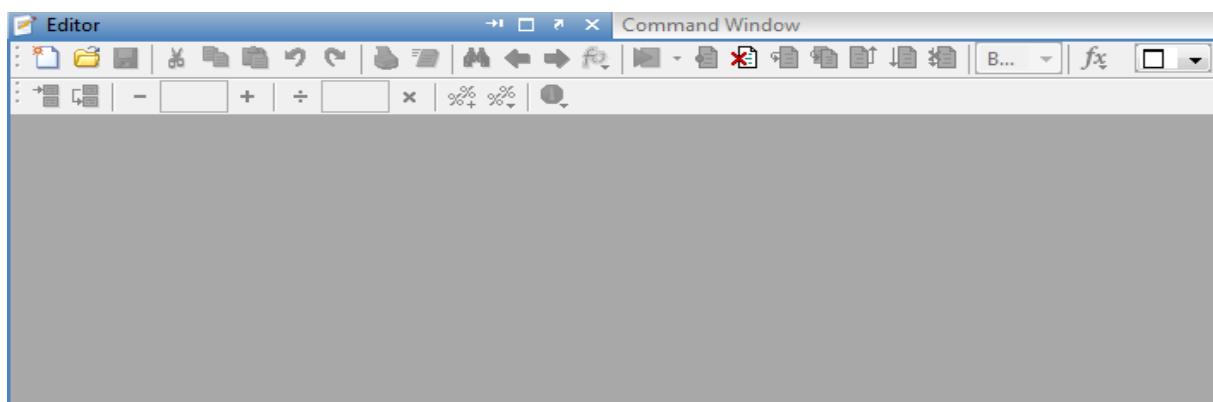
**Çalışma Dizini** üzerinde boş bir alanda sağ tıkladıkten sonra **New File–Blank M-File** yolunu takip ederek boş bir dosya oluşturabiliriz.



**Çalışma Dizini** penceresi üzerinde **Untitled.m** adında bir dosya oluşacaktır. **Untitled** kelimesi seçili olacaktır. Bu isim yerine istediğiniz ismi verebilirsiniz. Dosya uzantısını yani **.m** kısmını kesinlikle değiştirmeyiniz. MATLAB dosyalarını adlandırırken Paskal Notasyonu kullandığımızı unutmayalım.



Yapacağımız uygulamaları **Editör** üzerine yazdıktan sonra **kaydet** düğmesi ile kaydediyoruz. Programı çalıştırmak için **komut penceresine** gidiniz ve **.m** uzantısını kullanmadan sadece dosyanın adını yazarak **ENTER** tuşuna basınız. **Komut Penceresi** üzerinde programın çıktı verdiğiğini göreceksiniz.



## **Fonksiyon M-Dosyaları ile Programlama**

Düzyazı m-dosyaları işleyecekleri bilgileri kullanıcıdan input komutu ile klavye yoluyla almaktadırlar. Fonksiyon m-dosyaları ise programlama ortamına bilgi aktarma işlemini daha pratik ve kolay bir yol olan parametreler sayesinde yine klavye yoluyla gerçekleştirirler. Örneğin **MATLAB** da tanımlı **sqrt** fonksiyonu “**lütfen karekökü alınacak sayıyı giriniz:**” şeklinde bir uyarı metnini ekrana sunarak kullanıcıdan bilgi almaz. Karekökü alınmak istenen sayı yani işlenecek bilgi açılıp kapanan parantezler içerisinde ( ) fonksiyona doğrudan parametre olarak verilir.**(sqrt(16))**

### **Fonksiyonlar ve Genel Özellikleri**

Fonksiyonlar da bilgisayar programlarıdır. Fonksiyonlar sayesinde, bilgisayarlar yardımıyla çözülmeye çalışılan problemler yönetilmesi ve test edilmesi kolay, küçük parçacıklara bölünürler. Bu metoda “**Böl ve Yönet**” adı verilir.

Fonksiyonlar ihtiyaç duyulduğunda kullanılmak üzere belirli işlevleri yerine getirmek amacıyla tasarlanırlar. Programlarımızı tek bir büyük parça halinde yazmayacağız ve onların hangi küçük bileşenlerin (alt fonksiyonların veya yardımcı fonksiyonların) bileşimi şeklinde yazılabileceği konusunda zihin jimnastikleri yapacağız. Ayrıca bizden çözümü istenen bir problemin çözümüne yönelik sıfırdan bir program yazmaya çalışmayaacağız. Daha önceden yazdığımız programları mümkün mertebe tekrar kullanacağız ya da gelecekteki muhtemel kullanımlarını da dikkate alarak tekrar kullanabilecek fonksiyonlar yazmaya özen göstereceğiz. Bütün bu bilgiler ışığında var olanı kullan ve tekerlegi yeniden icat etme prensiplerini program yazarken hep aklimızda bulunduracağız.

Fonksiyonların genel özellikleri şu şekilde listelenebilir:

- ❖ Her fonksiyon kendine özgü bir isme sahiptir.
- ❖ Her fonksiyon genellikle kendine, üzerinde işlem yapacağı bir parametre ya da parametreler alır.
- ❖ Fonksiyonların parametreleri açılıp kapanan parantezler ( ) arasına konulur.
- ❖ Birden fazla parametrelî fonksiyonların parametreleri birbirinden virgül (,) karakteri yardımıyla ayrılır.
- ❖ Fonksiyonlar genellikle geriye bir değer döndürürler. (bir skaler, bir vektör veya bir matris.)

### **Fonksiyon M-Dosyalarının Bileşenleri**

Bir fonksiyon m-dosyasının iki bileşeni vardır:

- 1- **Fonksiyonun İmzası veya Prototipi:** Fonksiyonun dış dünyayı ilgilendiren ara yüzüdür; fonksiyonun işlemi ile ilgili hiçbir bilgi içermez ve ilgili m-dosyasının ilk satırında tanımlanır.

- 2- **Fonksiyonun Tanımı:** Fonksiyonun dış dünyayı ilgilendiren iç yapısını yani fonksiyonun işlevini ifade eder. İlgili m-dosyasının ikinci satırından başlar ve dosya sonuna kadar devam eder.

### **Fonksiyonun İmzası veya Prototipi**

Bir fonksiyon m-dosyasının ilk satırında tanımlanan, **function** anahtar kelimesiyle başlayan ve o fonksiyona ait isim, giriş parametresi ve çıkış parametresi bilgilerini içeren bileşendir.

İşleyişi yani iç yapısı bilinmeyen bir kıyma makinasının dış dünyadaki kullanım amacı herkes tarafından bilinmektedir. Bu kıyma makinası bir fonksiyon m-dosyasının ilk satırında bir **MATLAB** imzasına aşağıdaki şekilde dönüştürülür.

#### ***function Kıyma\_Et=Kıyma Makinası(Parça\_Et)***

Bu imza da **function** anahtar kelimedir. Atama operatörünün “=” sağında fonksiyonun adı ve solunda fonksiyonun işledikten sonra geri döndüreceği bilgiyi saklayan değişken yer alır. Açılıp kapanan parantezler içerisinde ise fonksiyonun değerlendirmesi istenen bilgiyi içeren değişken(parametre) adı yazılır. Bir fonksiyon m-dosyasının ilk satırı muhakkak yukarıda tanımladığımız ve fonksiyon imzası olarak adlandırdığımız bir satırda olmalıdır.

Aşağıda karşılaştırılması muhtemel birkaç fonksiyon imzası türü verilmiştir.

**İMZA 1: *function cikis parametresi= FonksiyonAdı (giris parametreleri 1,2,..,n):*** İmzası ile tanımlanan fonksiyon kendisine birden fazla giriş parametresi almakta ve geriye sadece bir çıkış değeri göndermektedir.

**İMZA 2: *function [cikis parametreleri 1,2,...,n]= FonksiyonAdı (giris parametreleri 1,2,..,n):*** İmzası ile tanımlanan fonksiyon kendisine birden fazla giriş parametresi almakta ve geriye birden fazla çıkış değeri göndermektedir.

**İMZA 3: *function cikis parametresi = FonksiyonAdı ()***

**İMZA 4: *function cikis parametresi = FonksiyonAdı***

Kendisine değerlendirmek üzere bir giriş parametresi almayan ama geriye bir değer döndüren bir fonksiyona ait İMZA 3 veya İMZA 4 ile tanımlanabilir.

**İMZA 5: *function FonksiyonAdı (giriş parametreleri 1,2,...,n ):*** İmzası ile tanımlanan fonksiyon kendisine birden fazla giriş parametresi almakta ve geriye hiçbir çıkış değeri göndermemektedir. Bu fonksiyon imzasında, geriye değer döndürülmediği için Atama Operatörü kullanılmamaktadır. Geriye bir değer döndürmeyen fonksiyonlar genellikle fpintf veya disp komutu ile ekrana çıktı verirler.

### **Fonksiyon M-Dosyaları Oluşturulurken Dikkat Edilmesi Gereken Hususlar**

- 1- Her **MATLAB** fonksiyonunun imzası **function** anahtar kelimesi ile başlamalıdır.
- 2- İmza satırında kullanılan **FonksiyonAdı** ile fonksiyon m-dosyasına verilen isim aynı olmalıdır.

- 3- Bir **MATLAB** fonksiyonu komut penceresinden fonksiyon adı ve eğer varsa parantez içerisinde birbirinden virgülerle ayrılmış parametrelerle çağrılmalıdır.
- 4- Bir fonksiyonu çağrıran herhangi bir program o fonksiyona, fonksiyonun imzasında tanımlanan sayıda parametre geçmeye dikkat etmelidir.

### **Fonksiyon M-Dosyalarının Oluşturulması ve Çalıştırılması**

Fonksiyon m-dosyaları da çalışma dizini penceresi üzerinde boş bir alan sağ tıklanarak ve düz yazıl m-dosyaları oluşturulurken izlenen adımlar takip edilerek oluşturulabilirler. Yani aslında **script file** olarak oluşturduğumuz yeni bir düz yazıl m dosyasını ilk satırında **function** anahtar kelimesi ile tanımladığımız bir imza sayesinde fonksiyon m-dosyası gibi kullanabiliriz.

Aşağıdaki verilen örneklerden birincisi iki boyutlu uzayda **(x1,y1)** koordinatı ile tanımlanan bir nokta ile **(x2,y2)** koordinatı ile tanımlanan bir başka nokta arasındaki mesafeyi düz yazıl m-dosyası mantığıyla, ikincisi ise aynı problemi fonksiyon m-dosyası yaklaşımı ile çözmektedir.

```
%METİN DÜZENLEYİCİSİ-DÜZYAZI M-FİLE%
x1=input('lütfen x1 koordinatını giriniz: ');
y1=input('lütfen y1 koordinatını giriniz: ');
x2=input('lütfen x2 koordinatını giriniz: ');
y2=input('lütfen y2 koordinatını giriniz: ');
uzaklik =sqrt((x2-x1)^2+(y2-y1)^2);
fprintf('girilen iki nokta arasındaki mesafe: %g\n',uzaklik);
```

```
%KOMUT PENCERESİ%
>> UzakligiHesapla
lütfen x1 koordinatını giriniz: 2
lütfen y1 koordinatını giriniz: 3
lütfen x2 koordinatını giriniz: 7
lütfen y2 koordinatını giriniz: 9
girilen iki nokta arasındaki mesafe: 7.81025
```

```
%METİN DÜZENLEYİCİSİ-FONKSİYON M-FİLE%
function uzaklik2= UzakligiHesapla2(x1,y1,x2,y2)
uzaklik2 =sqrt((x2-x1)^2+(y2-y1)^2);
end
```

```
%KOMUT PENCERESİ%
>> UzakligiHesapla2(2,3,7,9)
ans =
7.8102
```

Yukarıdaki m-dosyaları aynı işlemleri yapmaktadır ikisi de noktalar arasındaki uzaklığını hesaplamaktadır. Şekillerden de anlaşılacağı gibi fonksiyon m-dosyası ile programlama da işlemler daha kısa sürmektedir.

**MATLAB** da daha önce öğrendiğimiz değişken tanımlama kuralları, bir fonksiyonun imzası içinde giriş-çıkış parametrelerini temsil etmek üzere kullanılan değişkenler için de aynen geçerlidir. Bu yüzden parametre değişkenler kullanım maksatlarına uygun olarak isimlendirilmelidirler. Örneğin **UzakligiHesapla** fonksiyonu aslında jenerik parametre değişkenler kullanılarak da oluşturulabilir.

Fonksiyonun ara yüzünü tasarlarken yani m-dosyamızın imza satırını oluştururken istediğimiz isme sahip parametre değişkenler kullanabiliriz.

## Nargin ve Nargout Komutları

**nargin(number of input arguments)**: Fonksiyonda kaç tane input kullanıldığını geri döner, çeşitli kontroller de yapılabilir.

```
function x=fon(a,b,c);
if nargin==3
x=a+b+c;
else
disp('yazılan argumenler yetersiz');
end
```

**nargout(number of output arguments)**: Kaç tane output kullanıldığı bilgisini geri döner, kullanılacak olan input ve output sayısını varargout, varargin kullanarak tam olarak belirtmesek de olur.

```
function varargout=fon(varargin) %Şeklinde tanımlama yapılır ve istege
göre kontroller yapılabilir
```

## Ana Fonksiyon/Alt Fonksiyon Yaklaşımı ile Programlama

Bir **A** fonksiyonu işlevini yerine getirirken bir **B** fonksiyonundan yardım alabilir. Böyle bir durumda **A** fonksiyonu **ana** ve **B** fonksiyonu da **alt** veya yardımcı fonksiyon olarak adlandırılır. Ana fonksiyonların genellikle birden fazla alt fonksiyon barındırdıkları unutulmamalıdır. Aşağıdaki ekran çıktıları bu durumu özetlemek amacıyla kullanılmışlardır.

```

function sonuc = FonksiyonA( x )
%sonuc=x^2+x+1;
arasonuc=FonksiyonB(x);%alt fonksiyonu çağır.
sonuc=arasonuc+x+1;
%2. ve 3. satırların yerine tek başına aşağıdaki satırda kullanılabilir.
%sonuc=FonksiyonB(x)+x+1;
end

```

```

function sonuc = FonksiyonB( y )
sonuc = y^2;
end

```

```

>> FonksiyonA(3)
ans =
13

```

Yukarıdaki birinci ekran çıktısının anlamı şudur. Kullanıcı-tanımlı **FonksiyonA** adındaki fonksiyon kendisine bir  $x$  sayısını parametre olarak alır, bu sayıyı kullanarak  $x^2+x+1$  işleminin sonucunu doğrudan değil de **FonksiyonB** adlı bir alt fonksiyon sayesinde dolaylı olarak hesaplar ve geriye döndürür. Öte yandan ikinci ekran çıktısının da özetlediği üzere **FonksiyonB** nin işlevi oldukça basittir ve kendisine argüman olarak aktarılan herhangi bir  $y$  sayısının karesini almaktan ibarettir. Komut penceresinde **3** parametresi ile çalıştırılan **FonksiyonA** ise sonuç olarak **13** değerini arka planda sırasıyla aşağıdaki adımları takip ederek üretir:

- ❖ **FonksiyonA**' ya giriş parametresi olarak aktarılan 3 sayısı  $x$  değişkeninin değeri olarak kayıt altına alınır.
- ❖ **FonksiyonA** bu değer ile **FonksiyonB** yi çağırır.
- ❖ **FonksiyonB** 3 sayısının karesini alarak ürettiği 9 sayısını değerlendirilmek üzere kendisini çağıran **FonksiyonA** ya geri gönderir.
- ❖ **FonksiyonA** **FonksiyonB** den gelen 9 cevabını **arasonuc** adlı değişken içerisinde saklar.
- ❖ En son adımda ise **FonksiyonA** **arasonuc** değişkeninin değerine  $x+1=3+1=4$  ekler ve 13 değerini komut penceresinde kullanıcıya sunar.

Bütün bu bilgiler ışığında **FonksiyonA** nın da gerektiğinde bir başka **MATLAB** fonksiyonuna yardımcı olabileceği sonucu kolaylıkla çıkarılabilir. Fonksiyon m-dosyaları ile programlama yaparken dikkat edilmesi gereken en önemli hususlardan birisi de hem ana fonksiyonun ve hem de alt fonksiyonların mevcut dizinin altında bulunması zorunluluğudur.

## **MATLAB da Özel Fonksiyonlar**

**Cross(a,b) Fonksiyonu:** a ile b vektörlerinin vektörel çarpımını hesaplayan MATLAB özel fonksiyonudur. Örnek kullanım:

```
A = [4 -2 1];
B = [1 -1 3];
C = cross(A,B)
C =
-5 -11 -2
```

**Dot(a,b) Fonksiyonu:** a ile b vektörlerinin skaler çarpımını hesaplayan MATLAB özel fonksiyonudur. Örnek kullanım:

```
>> a=[1 2 3 4 5];
>> b=[6 7 8 9 10];
>> dot(a,b)
ans =
130
```

**Factor(n) Fonksiyonu:** n sayısının çarpanlarını bulan MATLAB özel fonksiyonudur. Örnek kullanım:

```
>> factor(150)
ans =
2 3 5 5
```

**isprime(n) Fonksiyonu:** n sayısının asal olup olmadığını denetleyen, asal ise 1 değilse 0 değerini döndüren MATLAB özel fonksiyonudur. Örnek kullanım:

```
>> isprime(9)
ans =
0
>> isprime(11)
ans =
1
```

**Primes(n) Fonksiyonu:** n sayısına kadar olan asal sayıları listeleyen MATLAB özel fonksiyonudur. Örnek kullanım:

```
>> primes(30)
ans =
2 3 5 7 11 13 17 19 23 29
```

**Gcd(a,b) Fonksiyonu:** a ile b sayılarının OBEB ini hesaplayan MATLAB özel fonksiyonudur. Örnek kullanım:

```
>> gcd(30,45)  
ans =  
15
```

**Lcm(a,b) Fonksiyonu:** a ile b sayılarının OKEK ini hesaplayan MATLAB özel fonksiyonudur. Örnek kullanım:

```
>> lcm(30,45)  
ans =  
90
```

**Rats(a) Fonksiyonu:** a sayısını rasyonel sayıya çeviren MATLAB özel fonksiyonudur. Örnek kullanım:

```
>> rats(4.5)  
ans =  
9/2  
>> rats(pi)  
ans =  
355/113
```

**Perms(a) Fonksiyonu:** a stringinin permütasyonlarını bulan MATLAB özel fonksiyonudur. Örnek kullanım:

```
>> perms('ada')  
ans =  
ada  
...  
daa  
aad
```

**Factorial(n) Fonksiyonu:** Girilen n sayısının faktöriyelini hesaplayan MATLAB özel fonksiyonudur. Örnek kullanım:

```
>> factorial(5)  
ans =  
120
```

**Nchoosek(n,r) Fonksiyonu:** n'nin r'li kombinasyonlarının sayısı hesaplayan MATLAB özel fonksiyonudur. Örnek kullanım:

```
>> nchoosek(5,2)  
ans =  
10
```

## MATLAB da Fonksiyon Uygulamaları

**Örnek:** Verilen n sayılarından küçük olan tüm asal sayıları listeleyen matlab fonksiyonunu alt fonksiyon kullanarak yazınız.

```
function liste = asal_sayıları_bul(n)
liste=[];
for i=2:n
    if(asal_mi(i)==1)
        liste=[liste i];
    end
end
end
function sonuc = asal_mi(sayı)
boolenler=[];
for i=1:sayı
    if(mod(sayı,i)==0)
        boolenler=[boolenler i];
    end
end
if(length(boolenler)==2)
    sonuc=1;
else
    sonuc=0;
end
end
```

```
>> asal_sayıları_bul(10)
ans =
2 3 5 7
```

**Örnek:** Kendine giriş parametresi olarak aldığı x ve y değişkenlerinin değerlerini değiştiren MATLAB fonksiyonunu yazınız.

```
function [x,y]=yerlerini_degistir(x,y)
gecici=x;
x=y;
y=gecici;
end
```

```
sayi1=53;
sayi2=1989;
[sayi1,sayi2]=yerlerini_degistir(sayi1,sayi2);
sayi1=1989
sayi2=53
```

Cıkış parametresinin birden çok olduğu durumlarda çıkan sonucu değişkene almak için, aşağıdaki yöntem uygulanır.

[x,y]=degistir(2,3); Aksi halde editör sadece ilk değeri, ans değişenine atar ve ikincisini atamaz.

**Palindrom Kelime:** Sağdan ve soldan okunuşu aynı olan kelimeye denir. Örnek kelimeler: ece, sos, elle, kazak, küllük, nacican...

Ey edip adanada pide ye

**Örnek:** Verilen bir kelimenin palindrom olup olmadığını test eden bir MATLAB fonksiyonu yazınız.

```
function sonuc = palindrom_mu(kelime)
boyut = length(kelime);
yarisi = fix(boyut/2);
sonuc ='bu kelime palindromdur';
for i=0:yarisi
    if kelime(i+1)== kelime(boyut-i)
        continue
    else
        sonuc ='bu kelime palindrom değildir';
    break end end
```

**Fibonacci dizisi:** Her sayının kendinden öncekiyle toplanması sonucu oluşan bir sayı dizisidir. 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, şeklinde devam etmektedir.

**Örnek:** 1 ile 1000 arasındaki fibonacci dizi elemanlarını yazdırın MATLAB programını yazınız.

```
liste=[];
liste(1)=1;
liste(2)=1;
for i=3:1000
eleman =liste(i-1) + liste(i-2);
liste(i)=eleman;
end
for i=1:1000
fprintf('%d ',liste(i));
end
```

```
fibonacci
ans=
```

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
```

**Örnek:** Girilen n sayısına karşılık, n satır ve n sütundan oluşan, \* lar kullanarak dik üçgen çizdireن MATLAB fonksiyonunu yazınız.

```
function ucgen_ciz(n)
for i=1:n
for j=1:i
fprintf('*');
end
fprintf('\n');
end
```

```
ucgen_ciz(5)
ans=
*
**
***
****
*****
```

**Armstrong Sayılar:** 3 basamaklı sayılardan, basamak değerlerinin küpleri toplamı sayıya eşit ise bu sayıya Armstrong sayı denir. **Örneğin:** 153 bir Armstrong sayısıdır, çünkü:  $1^3 + 5^3 + 3^3 = 153$  tür.

**Örnek:** Tüm Armstrong sayıları bulan bir MATLAB fonksiyonu yazınız.

```
function liste = armstrong_bul()
liste=[];
for i=100:999
s=num2str(i);
s1=str2num(s(1));
s2=str2num(s(2));
s3=str2num(s(3));
toplam=s1^3 + s2 ^3 + s3^3;
if i==toplam
liste=[liste i];
end end end
```

```
armstrong_bul()
ans=
153    370    371    407
```

**Örnek:** Aşağıdaki gibi bir ekran çıktısı verecek ekranaRakamBas.m adında bir fonksiyon M dosyası yazınız. (Fonksiyonunuz kendisine hiçbir argüman almayacak ve geriye yine hiçbir değer çevirmeyecektir.)

```
>>ekranaRakamBas()
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
function ekranaRakamBas()
clc;
for i=1:9 %satır sayısı
for j=1:(10-i) %sutun sayısı
fprintf('%d ',j);
end
fprintf('\n');
end
```

**Örnek:** Girilen bir sayı vektörünün ortalama değerini ve standart sapmasını bulan bir fonksiyon yazalım.

```
function [ort, stds] = ort_stdsapma (x)
% ort_stdsapma fonksiyonu girilen matristeki her bir sütunun ortalama değerini
% ve standart sapmasını bulur.
[m,n] = size(x);
if m == 1
m = n;
end
ort = sum(x)/m;
stds=sqrt(sum(x.^2)/m-ort.^2);
```

```
%MATLAB komut satırında aşağıdaki satırlar sırasıyla yazılırsa
>> a=[1 2 3 4 5];
>> [x,y]=ort_stdsapma(a)
x =
3
y =
1.4142
```

## MATLAB da Hata Bulma ve Düzeltme

**MATLAB** m-dosyaları ile programlama yapmamıza imkan sağlayan **MATLAB** metin düzenleyicisinin birçok gelişmiş özelliğe sahip olduğundan daha önce söz etmiştik. İşte bu metin düzenleyicisi öte yandan, daha bizler programlarımızı yazma aşamasındayken ve henüz daha çalıştırılmamışken kendi penceresinin sağında hatalı bölümü içeren satır hizasında turuncu çizgiler şeklinde dinamik olarak uyarılar ve hata mesajları oluşturur. Bunlar genellikle yazın hataları gibi, açılıp kapatılmayan parantezler veya tek tırnaklar, benzer şekilde end ile kapatılmayan if deyimleri gibi problemlerdir. Metin düzenleyicisi ayrıca program içerisindeki hatalı kelimelerin veya satırların ya da sorunlu bölgelerin altlarını yine turuncu bir çizgi ile işaretli hale getirir. Diğer yandan metin düzenleyicisi penceresi imlecimizin pozisyonunu işaret etmekte satır ve sütunlara ayrılmıştır. Yazdığınıız programınızı kaydettikten sonra ve komut penceresi üzerinde çalıştığımız an itibariyle de hata mesajlarıyla karşılaşabiliriz. Bu durumda **MATLAB** bizlere hatalı satır ve sütun bilgileriyle birlikte hatanın tipine yönelik ipuçlarını da komut penceresi üzerinden bize yansıtır. Eğer işaret edilen satırda bir hataya rastlamazsa, bu satırın bir üstündeki veya bir altındaki satırı incelemek gerekebilir. Bu başlıkta anlatılanları özetleyen bir örnek ekran çıktısı aşağıda verilmiştir.

The screenshot shows the MATLAB Editor window and the Command Window. The Editor window displays a script named 'deneme.m' with the following code:

```
1 ayNo=input('Lutfen (1-12) Araliginda Bir Ay Numarasini giriniz.');
2 switch ayNo
3 case 2
4     disp('Bu ay 28 gunden olusur.');
5 case {1, 3, 5, 7, 8, 10, 12}
6     disp('Bu ay 31 gunden olusur.');
7 case {4, 6, 9, 11}
8     disp('Bu ay 30 gunden olusur.');
9 otherwise
10    disp('Dikkatli olunuz. Yanlis bir ay numarası girdiniz.');
11 end
```

A tooltip box is overlaid on the Editor window, pointing to the first line of the string input. It contains two error messages:

- Line 1: A quoted string is unterminated.
- Line 1: There might be a parenthesis imbalance.

In the Command Window below, the user has run the script with the command `>> deneme`. The output shows the error message:

```
?? Error: File: deneme.m Line: 1 Column: 12
A MATLAB string constant is not terminated properly.
```

Yukarıdaki resimde uyarı metninde açılan ama kapanmayan tek tırnağın oluşturduğu hata görülmektedir. Hata içeren satırın sonundaki kırmızı çizgi tıklandığında hataya ilgili ipucu görülmektedir. Alt kısımdaki komut penceresinde 1. Satır 12. Sütunda bir hata olduğu belirtilmektedir.

## Hata Ayıklama

Yazdığımız bilgisayar programları bir hata nedeniyle ya hiç çalışmazlar ya da çalışırlar ama beklediğimiz çıktıları üretmezler. Bir hata vererek çalışmayan **MATLAB** programındaki sorunların giderilmesine yönelik yapılabilecekleri bir önceki bölümde irdeledik. Çalıştığı halde içeriği mantıksal hatalar (*geliştirilen veya bilgisayar programına dönüştürülmüş algoritmadan, döngü değişkeninin sınır değerlerinin yanlış belirlenmesinden, if veya while deyimlerinde kullanılan koşulların yanlışlığından kaynaklanabilecek vs.*) yüzünden beklenilen sonuçları üretmeyen programlarda sorunun kaynağını bulmanın standart ve kolay bir yolu maalesef yoktur. Bu tür durumlarda hata ayıklama adı verilen işleme başvurulur. **MATLAB** in da metin düzenleyicisi penceresi üzerinden bizlerin kullanımına sunduğu bir hata ayıklama metodu mevcuttur. Bu mod, bizlerin programımızın herhangi bir satırına bir kesme koyabilmesine imkan sağlar. Programın kesme noktası konulan satırı öncesindeki bölümü bilgisayar tarafından işletilir ve kesme noktasına ulaşıldığından kontrol bilgisayar tarafından kullanıcıya devredilir. Kontrolü devralan kullanıcı ise programı satır satır işletir, işletilen herhangi bir satır sonrası program akışının nereye kayacağını ve hangi sonuçlar üretileceğini not alır. Nihayetinde o satırı sanki kendisi **MATLAB** yorumlayıcısı gibi işletir ve gerçek durumla beklenilerini karşılaştırarak sorunu bulmaya çalışır. Kullanılan bu hata ayıklama yöntemi, aynı zamanda nasıl çalıştığını veya kodlandığını tam olarak anlayamadığımız programlarla ilgili bizlere ipuçları verme noktasında da oldukça faydalıdır. Şimdi **MATLAB** ile hata ayıklamanın nasıl yapılabileceğini adım adım öğrenelim.

**ADIM 1:** İçerisindeki hataları ayıklayacağımız programı çalışma dizini penceresi üzerinde çift tıklayarak açalım.

**ADIM 2:** Programımızın metin düzenleyicisi penceresi üzerinde açıldığından emin olalım.

**ADIM 3: Çalışma Alanı** penceresinin açık olduğundan emin olalım.

**ADIM 4:** Komut penceresinde **clear** komutunu çalıştırarak **Çalışma Alanı** penceresini temizleyelim, yani tanımlı bütün değişkenleri bellekten silelim.

**ADIM 5:** Komut penceresinde **clc** komutunu çalıştırarak komut penceresini temizleyelim.

**ADIM 6:** Metin Düzenleyicisi penceresinin sol tarafında programın satır numaraları yer alır. Kontrolü devralmak istediğimiz satırın satır numarasının sağındaki **tire karakterini (-)** tıklayalım ve onun içi dolu kırmızı bir yuvarlağa dönüşmesini sağlayalım.

**ADIM 7:** Komut penceresine gidip programı çalıştıralım. Komut penceresindeki **>>** sembolünün **K>>** formuna dönüştüğünü göreceğiz. Bu bizim başarıyla hata ayıklama moduna geçtiğimizin işaretidir.

**ADIM 8:** Kesme noktasında bilgisayar tarafından bize devredilen kontrolü ele alalım ve **F10** tuşu sayesinde adım adım ilerleyelim. (*Alt programlar içерisine dallanmak için F11'i kullanalım.*)

**ADIM 9:** Kesme noktasının yanında yeşil bir okun varlığına ayrıca dikkat edelim. Biz **F10** tuşuna her bastığımızda bu yeşil okun program akışına göre yeni pozisyonuna ilerlediğini göreceğiz.

**ADIM 10:** **F10** tuşuna basmadan önce bu okun nerede olduğuna dikkat edelim ve **F10** tuşuna bastığımızda hangi satırı gideceğini kestirmeye çalışalım.

**ADIM 11: Çalışma Alanı** penceresi üzerinde var olmayan bir değişkenin sırası geldiğinde nasıl sıfırdan oluşturulduğuna ve var olanın değerinin ise nasıl güncellendiğine dikkat edelim. Faremizi programımızda bir değişkenin üzerine taşıdığımızda onun mevcut değerinin ekranда görüneceğine dikkat edelim.

**ADIM 12:** İşimiz bittiği zaman programımızdaki kesme noktasını (*ichi dolu kırmızı yuvarlak*) bir kez tıklayalım ve onu yeniden **tire (-)** karakterine dönüştürelim.

**ADIM 13:** Komut penceresi üzerinde **return** komutunu çalıştıralım ve hata ayıklama modundan çıkışalım. Böylece **K>>** sembolünün yeniden **>>** haline Dönüşüğünü görelim.

Bütün bu anlatılan işlemleri özetleyen örnek bir ekran çıktısı aşağıda verilmiştir.

The screenshot shows the MATLAB IDE interface. The Editor window displays a script named 'deneme.m' with the following code:

```
1 - AyNo = input('Lutfen (1-12) Araliginda Bir Ay Numarası Giriniz:');
2 - swi AyNo: 1x1 double =
3 -      5
4 -      gunden olusur.';
5 - case {1, 3, 5, 7, 8, 10, 12}
6 -     disp('Bu ay 31 gunden olusur.');
7 - case {4, 6, 9, 11}
8 -     disp('Bu ay 31 gunden olusur.');
9 - otherwise
10 -    disp('Dikkatli olunuz. Yanlis bir ay numarasi girdiniz.');
11 - end
```

The Command Window shows the following interaction:

```
>> deneme
Lutfen (1-12) Araliginda Bir Ay Numarası Giriniz:5
3      case 2
5      case {1, 3, 5, 7, 8, 10, 12}
K>> return
```

The Workspace browser on the right shows a variable 'AyNo' with the value 5.

Yukarıdaki resim üzerinde programın 3 numaralı satırına konulmuş bir kesme noktası, fare değişken adı üzerine kaydırıldığında açılan değişken değeri penceresi, işletilen satırlar sonrasında tanımlı tek değişken ve değeri, hata ayıklama simgesi ve hata ayıklama modundan çıkmak için kullanılan return komutu görülmektedir.

## Karakter Katarı (String) İşlemleri

Karakterler katarı veya yalnızca katar/sözcük (string), iki tek tırnak arasındaki ifade edilen, gerçekte ASCII kod tablosunda sayısal kodlarla belirtilen ilk 127 karakterden oluşan karakter dizileridir (char array). Katar uzunluğu, katardaki karakter sayısıdır. Her bir karakter bellekte 1 byte (8 bit) yer kaplar.

```
>> ders='algoritmalar'  
ders =  
algoritmalar
```

Burada verilen ders değişkeninde her bir harf ASCII kod tablosundaki sayısal bir koda sahiptir.

```
>> kod=double(ders)  
kod =  
97 108 103 111 114 105 116 109 97 108 97 114
```

Tam tersi olarak ASCII karşılığı verilen bir katarın karakter karşılığını bulmak için char fonksiyonu kullanılır.

```
>> char(kod)  
ans =  
algoritmalar
```

## Çok Boyutlu Katar Gösterimleri

```
>> x='10014';  
>> a=[ders, 'ders kodu:', x]  
a =  
algoritmalar ders kodu:10014
```

## Katarların Karşılaştırılması

```
>> katar1='adem';  
>> katar2='adam';  
>> katar3='adem';
```

**strcmp:** İki katarın aynı olup olmadığını belirler.

```
>> strcmp(katar1,katar2)
ans =
  0
>> strcmp(katar1,katar3)
ans =
  1
```

**strcmp:** iki katarın harf durumu dikkate alınmadan aynı olup olmadığını belirler.

```
>> strncmp(katar1,katar3)
ans =
  1
>> strncmp(katar1,katar2)
ans =
  0
```

**strncmp:** iki katarın ilk n karakterinin aynı olup olmadığını belirler.

```
>> strncmp(katar1,katar2,2)
ans =
  1
>> strncmp(katar1,katar2,3)
ans =
  0
```

**strncmp:** ilk katarın harf durumu dikkate alınmadan ilk n karakterinin aynı olup olmadığını belirler.

```
>> strncMPI(katar1,katar2,3)
ans =
  0
>> strncMPI(katar1,katar2,2)
ans =
  1
```

### Katarların Tek Tek Karşılaştırılması

Katar dizileri aynı boyutlu veya en az biri skaler olmak koşuluyla karakter eşitlik karşılaştırılması için ilişkisel operatörler ( $>$ ,  $\geq$ ,  $<$ ,  $\leq$ ,  $=$ ,  $\sim=$ ) kullanılabilir.

```
>> x='matlab';
>> y='matema';
>> x==y
ans =
1 1 1 0 0 0
```

### Büyük-Küçük Harf Dönüşümleri

**upper:** Katardaki tüm harfleri büyük harfe dönüştürür.

```
>> katar1='matlab';
>> upper(katar1)
ans =
MATLAB
```

**lower:** Katardaki tüm harfleri küçük harfe dönüştürür.

```
>> katar1='FeNeRbAhÇe';
>> lower(katar1)
ans =
fenerbahçe
```

### Sayı-Katar Dönüşümü

**num2str:** Kayan noktalı sayıyı (tamsayı ya da ondalıklı) katara dönüştürür.

```
>> a=236;
>> akatar=num2str(a)
akatar =
236
>> double(akatar)
ans =
50 51 54
```

**int2str:** Yalnızca bir tamsayıyı katara dönüştürür.

```
>> sayi=25;          %bir sayı tanımlandı.
>> int2str(sayi)    %sayı katara dönüştürüldü.
ans =
25
```

### Katar-Sayı Dönüşümleri

**str2num:** ASCII katarı sayıya dönüştürür.

```
>> z='123';
>> x=str2num(z)
x =
123
>> 123*2
ans =
246
```

**eval:** Sayısal biçimden sayı içeren katara dönüştürür.

```
>> d='234';
>> eval(d)
ans =
234
>> katar='k=1:5'
katar =
k=1:5
>> eval(katar)
k =
1 2 3 4 5
```

**Katarlar ile ilgili örnek:**

Bir metin içinde kaç tane “a” harfi olduğunu bulan programın yazılması.

```
a='There is no function or block with this name. You may want to Check that you
have spelled the name correctly.';
s=0;
for i=1:1:length(a)
    if a(i)=='a'
        s=s+1;
    end
end
s
```

**Çalışma Sorusu:** Rakamlarla yazılan bir sayının söylenildiği gibi ekrana basılmasını sağlayan programın yazılması. (1234: Bin iki yüz otuz dört)

## Lineer Cebir Uygulamaları

A matrisi için aşağıdaki işlemleri komut penceresinde gerçekleştiriniz.

$$\begin{matrix} 1 & 3 & 5 \\ 7 & 8 & 11 \\ A = 100 & 1 & 4 \end{matrix}$$

- ❖ A matrisini giriniz.

```
>> A=[1 3 5; 7 8 11; 100 1 4]
```

A =

$$\begin{matrix} 1 & 3 & 5 \\ 7 & 8 & 11 \\ 100 & 1 & 4 \end{matrix}$$

- ❖ A matrisinin determinantını hesaplayınız.

```
>> det(A)
```

ans =

$$-728$$

- ❖ A matrisinin tersini bulunuz. Çıkan sonucu bir B matrisine atayınız.

```
>> inv(A)
```

ans =

$$\begin{matrix} -0.0288 & 0.0096 & 0.0096 \\ -1.4725 & 0.6813 & -0.0330 \\ 1.0893 & -0.4107 & 0.0179 \end{matrix}$$

```
>> B=ans;
```

- ❖ A\*B işlemini yapınız. Elde edilen sonucu irdeleyiniz.

```
>> A*B
```

ans =

$$\begin{matrix} 1.0000 & 0 & 0.0000 \\ 0 & 1.0000 & 0.0000 \\ 0 & 0.0000 & 1.0000 \end{matrix} \quad \%Birim\ matris!!!$$

- ❖ A matrisinin 1. sütununu a1, 3. sütununu a3 vektörlerine atayınız.

```
>> a1=A(:,1);
```

```
>>a3=A(:,3);
```

- ❖ Köşegenleri A matrisinin köşegenlerinden oluşan bir C köşegen matrisi oluşturunuz.

```
>> C=diag(diag(A))
C =
  5
  6
  7
```

- ❖ a1'in devriği ile a3 vektörünü çarpınız.

```
>> a1'*a3
ans =
  482
```

- ❖ a1 ile a3 vektör elemanlarını karşılıklı çarpınız.

```
>> a1.*a3
ans =
  5
  77
  400
```

- ❖ A'nın 3. satırını, diğer satır elemanlarını girmeden, [5 6 7] olarak değiştiriniz.

```
>> A(3,:)=[5 6 7]
A =
  1   3   5
  7   8  11
  5   6   7
```

- ❖ A'nın 1 ve 2. satırlarını siliniz.

```
>> A([1 2],:)=[]
A =
  5   6   7
```

- ❖ A matrisinin transpozunu(devriğini) bulunuz.

```
>> A'  
ans =  
1 7 100  
3 8 1  
5 11 4
```

- ❖ A matrisinin özvektörlerini bulunuz.

```
>> A=[1 3 5; 7 8 11; 100 1 4];  
>> [x,y]=eig(A)  
x =  
0.2040 0.1933 -0.0046  
0.5072 0.3444 -0.8577  
0.8373 -0.9187 0.5142  
y =  
28.9750 0 0  
0 -17.4175 0  
0 0 1.4425
```

B matrisi için aşağıdaki işlemleri komut penceresinde gerçekleştiriniz.

$$\begin{matrix} 10 & 5 & 5 \\ 70 & 8 & 7 \\ B=10 & 1 & 3 \end{matrix}$$

- ❖ B matrisini giriniz.

```
>> B=[10 5 5;70 8 7;10 1 3]  
B =  
10 5 5  
70 8 7  
10 1 3
```

- ❖ B matrisini mevcut çalışma klasörünüze katsayılar ismiyle kaydediniz.

```
>>save katsayilar B
```

- ❖ Dosyanın kaydedilip kaydedilmediğini kontrol ediniz. (Open Files penceresinden)
- ❖ MATLAB oturumundaki tüm değişkenleri siliniz (clear)
- ❖ Çalışma penceresine yazılmış tüm ifadeleri temizleyiniz. (clc)

```
>> clear  
>> clc
```

- ❖  $B^2$  işlemini yapınız.

```
>> B*2  
??? Undefined function or variable 'B'.
```

- ❖ B matrisini geri çağrıınız.

```
>> load katsayilar
```

- ❖ B matrisinin üst ve alt üçgen matrislerini oluşturunuz.

```
>> triu (B)  
ans =  
10 5 5  
0 8 7  
0 0 3  
>> tril (B)  
ans =  
10 0 0  
70 8 0  
10 1 3
```

- ❖  $C=[B \text{ zeros}(3,2)]$  işlemini yapınız.

```
>> C=[B zeros(3,2)]  
C =  
10 5 5 0 0  
70 8 7 0 0  
10 1 3 0 0
```

## Sembolik Matematik ve Uygulamaları

**MATLAB** da bir denklemin çözümünü bulmak için örneğin  $x^2-2x-15=0$  denkleminin çözümünü bir m dosyasına gerekli kodları yazarak yapabiliriz. Bu programın m dosyası aşağıdaki gibi olabilir.

```

a=input('a = ');
b=input('b = ');
c=input('c = ');
delta=b*b-4*a*c;
if delta>0
x1=(-b-delta^0.5)/(2*a);
x2=(-b+delta^0.5)/(2*a);
fprintf('İki reel kök: x1 = %f x2 = %f ',x1,x2);
elseif delta==0
fprintf('Tek kök var: x1 = x2= %f ',-b/(2*a));
else
fprintf('Kökler sanal ');
end;

```

Program çalıştırıldığında **a** **b** **c** katsayılarına sırasıyla, **1**, **-2** ve **-15** değerlerini girerek denklemin köklerini -3 ve 5 olarak buluruz. İkinci derece bir denklemin çözümünü veren formüllerini bildiğimizden bunun programını (çok kolay olmasa da) yazabildik. Daha karmaşık ve zor işlemleri yaptırabilmek için **MATLAB** daki **Sembolik Mantık (Sembolik Nesne)** kavramını kullanmalıyız. Bu konu ile ilgili açıklama ve yardım almak için komut satırına **help symbolic** yazmak yeterlidir.

### Sembolik Matematikte Bazı Komutlar ve Anlamları

**sym ve syms komutları:** Bir değişkeni sembolik nesne yapmaya yarar. Örneğin **x** değişkenini sembolik nesne yapmak için; **x=sym('x');** komutu kullanılabilir. Aynı işlemi **syms x;** komutu ile de yapabiliriz. **x, y** ve **z** değişkenlerini sembolik nesne yapmak için; **syms x y z** komutu kullanılabilir. Bir ondalık sayının kesir olarak karşılığını bulabilmek için **sym** komutundan faydalananabiliriz. Örneğin 3.98 ondalık sayısının rasyonel sayı karşılığını bulmak için **sym(3.98,'r')** veya **sym(3.98)** komutu kullanabiliriz. Benzer şekilde 22/7 kesrini ondalık sayıya çevirmek için de **sym(22/7,'d')** komutu kullanılır.

**pretty komutu:** Sembolik nesnenin görüntüsünü ekranada net olarak anlaşılır biçimde görünmesini sağlayan komuttur.

### Sembolik İntegral İşlemi

Sembolik integral alma fonksiyonu **int** olup genel formatları aşağıdaki gibidir. Aşağıda MATLAB da gerçekleştirilen integral alma işlemleri gösterilmektedir.

- ❖ **int(S):** **S**'in belirsiz integralini alır.

```
>> syms x
>> S=-2*x^5-4*x+20;
>> int(S)
ans =
-1/3*x^6-2*x^2+20*x
```

❖ **int(S,v):** S'in v'ye göre belirsiz integralini alır.

```
>> syms x
>> syms y
>> S=x^2+y^2+5;
>> int(S,y)
ans =
y^3/3 + (x^2 + 5)*y
```

❖ **int(S,a,b):** S'in varsayılan sembolik değişkene göre a'dan b'ye kadar belirli integralini alır.

```
>> S=3*x^2+2*x+5;
>> int(S)
ans =
x*(x^2 + x + 5)
>> int(S,0,3)
ans =
51
```

❖ **int(S,v,a,b):** S'in v'ye göre a'dan b'ye kadar belirli integralini alır.

```
>> syms x
>> syms y
>> S=x^2+y^2+5;
>> int(S,y)
ans =
y^3/3 + (x^2 + 5)*y
>> int(S,y,1,2)
ans =
x^2 + 22/3
```

## Sembolik Türev İşlemi

Türev alma işleminde kullanılan fonksiyon adı **diff** olup sembolik işlem mantığı çerçevesinde genel formatları aşağıdadır.

❖ **diff(S):** S'in türevini alır.

```
>> syms x  
>> S=x^3+x^2+x+10;  
>> diff(S)  
ans =  
3*x^2 + 2*x + 1
```

❖ **diff(S,v):** S'in v'ye göre türevini alır.

```
>> S=x^3+y^3+x^2*y^2+10*x*y;  
>> diff(S)  
ans =  
3*x^2 + 2*x*y^2 + 10*y  
>> diff(S,y)  
ans =  
2*x^2*y + 10*x + 3*y^2
```

❖ **diff(S,n):** n pozitif bir tamsayı olmak üzere n. dereceden türevini alır.

```
>> S=x^3+x^2+x+10;  
>> diff(S,2)  
ans =  
6*x + 2
```

❖ **diff(S,'v',n) veya diff(S,n,'v'):** S'in v'ye göre n. dereceden türevini alır.

```
>> S=x^3+y^3+x^2*y^2+10*x*y;  
>> diff(S,y,2)  
ans =  
2*x^2 + 6*y  
>> diff(S,2,y)  
ans =  
2*x^2 + 6*y
```

Türevin  $x=p$ 'deki değerini bulmak için;

```
>> subs(turev,x,p)
```

```

>> S=x^3+x^2+x+10;
>> turev=diff(S)
turev =
3*x^2 + 2*x + 1
>> subs(turev,x,1)
ans =
6

```

**ÖRNEK:**  $f(x)=5x^3+ax^2+bx -14$  (a ve b sabit değerdir) fonksiyonunun türevini bulunuz.

```

>> syms a b c x
>> f=5*x^3+a*x^2+b*x-14;
>> diff(f)
ans =
15*x^2+2*a*x+b

```

### Denklem Sistemlerinin Çözümü

**solve fonksiyonu:** Cebirsel denklemlerin sembolik çözümünü verir. Genel formatı:

`solve('denk1','denk2',...,'denkN')` şeklindedir.

**Örnek:**  $f(x)=x^2-x-6$  fonksiyonun çözüm kümesini bulunuz.(Ç.K:{-2,3})

```

>> syms x
>> S=x^2-x-6;
>> solve(S)
ans =
-2
3

```

**Önnek:**

$$x^2+x*y+y=3$$

$$x^2-4x+3=0$$

denklem sisteminin çözüm kümesini bulunuz.

```

>> syms x
>> syms y
>> S1=x^2+x*y+y-3;
>> S2=x^2-4*x+3;
>> [x,y]=solve(S1,S2)
x =
1
3
y =
1
-3/2

```

**Diferansiyel Denklem Çözümü:** **dsolve** fonksiyonu ile gerçekleştirilir.

$dy/dt=1+y^2$  diferansiyel denklemini çözünüz.

```

>> dsolve('Dy=1+y^2')
ans =
i
-i
tan(C3 + t)

```

$y(0)=0$ ,  $y'(0)=0$  şartları altında  $y''+6y'+13y=10\sin 5t$  ikinci dereceden diferansiyel denklemin çözümünü bulunuz.

```

>>Q=dsolve('D2y+6*Dy+13*y=10*sin(5*t)','y(0)=0','Dy(0)=0','t')
>>pretty(simplify(Q))
25 10 25 125
- -- cos(5t) - -- sin(5t) + -- exp(-3t)cos(2 t) + --- exp(-3t)sin(2t)
87 87 87 174

```

$\cos x$  fonksiyonunu 9. terime kadar Taylor serisine açınız.

```

>> f=cos(x)
>>T=taylor(f,9)
T =
1-1/2*x^2+1/24*x^4-1/720*x^6+1/40320*x^8

```

## MATLAB da Grafik İşlemleri

**MATLAB** akademik dünyada en yaygın kullanılan hesap kitap araçlarındanadır. Bu kadar yaygın olmasının sebeplerinden biride ihtiyaç duyduğumuz birçok özelliğe sahip olmasıdır. Bunlardan biride grafik bileşenidir.

Matlab'da grafik çizdirmek oldukça kolaydır. 2 Boyutlu, 3 Boyutlu çizimler Matlab'da rahatlıkla yapılır.

Matlab' da birçok işlemde olduğu gibi çizim yaptmak için vektörlere (dizilere) ihtiyacımız olacaktır.

**Örnek:**  $y = x^2 + 1$  eşitliğinin grafiğini çizdirelim.

Komutlarımızı **Matlab** komut ekranından yazabiliriz. İlk olarak grafiğin yatay ekseni oluşturacak x değişkeni için x dizisini oluşturalım.

Bu ifadeyle 0'dan başlayan 0.5 adım büyüğünü ile artan ve 20'de biten 41 elemanlı bir dizi şeklinde oluşturalım. Oluşan x dizisini Matlab penceresinin **Workspace** kısmında görebiliriz.

Grafiğin dikey ekseni oluşturacak y değişkeni için de y dizisini oluşturmamız gerekiyor. y dizisinin elemanlarının değerlerin x'e bağlıdır. **Workspace** de x ve y dizisini görülebilir. Dizilerin eleman sayılarını ve minimum-maksimum değerlerini de **Workspace** den görebiliriz.

**Matlab'da Grafik** çizimi yapabilmemiz için eşit boyutlu iki diziye ihtiyacımız vardır. Dizinin elemanlarının her biri bir x-y çifti oluşturacaktır.

Bu x-y kartezyen koordinat sisteminde bir noktayı belirtecektir. Bu noktaların birleştirilmesi ile de grafik çizilmiş olacaktır.

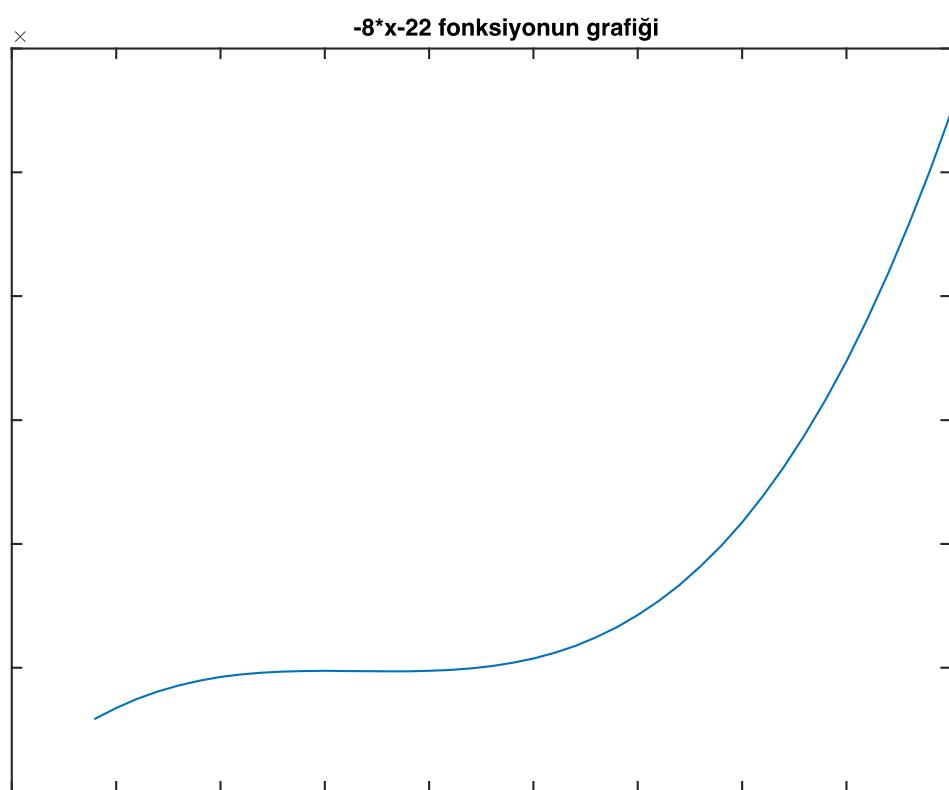
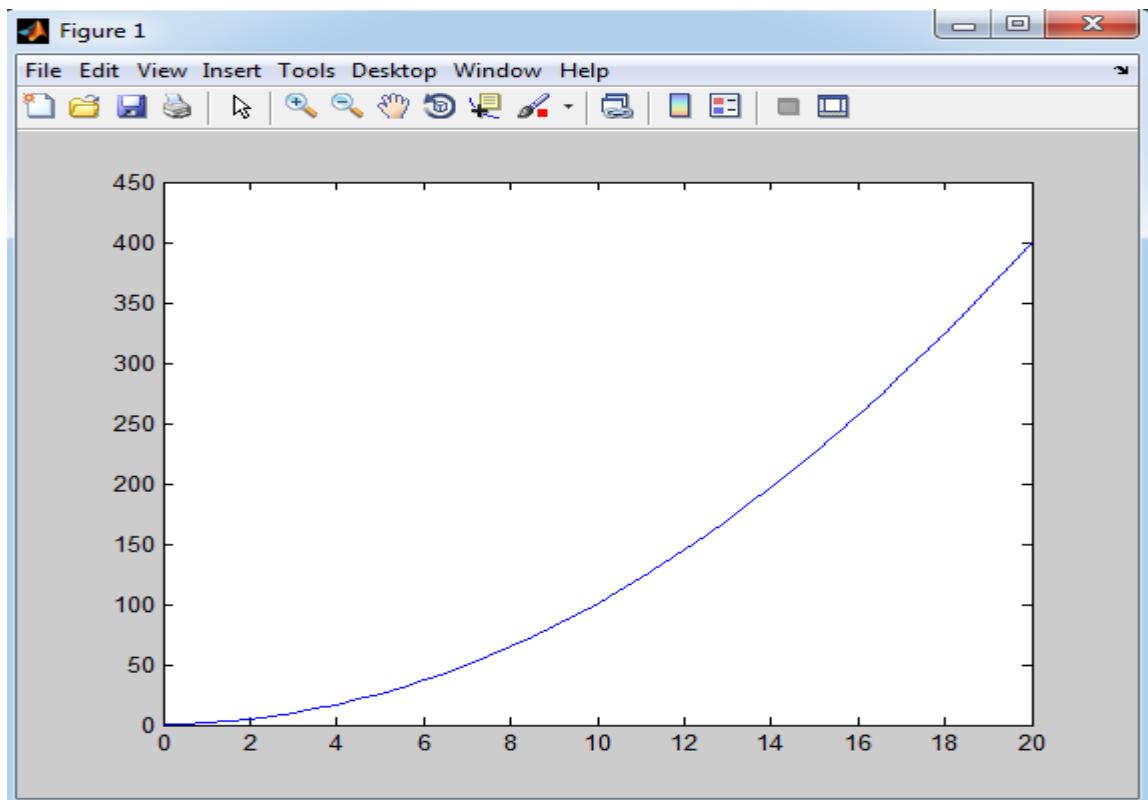
Grafiği çizdirmek için 'plot' komutunu kullanacağız.

**Plot:** Plot fonksiyonu sayesinden grafiklerimizi en hızlı şekilde çizdirebiliriz.,

**plot(x,y)** şeklinde kullanılan fonksiyon x ve y dizilerindeki karşılık gelen her elemana ait noktayı belirler ve bu noktaları birleştirerek grafiği oluşturur.

Bizim çizimimizde x eksenini x dizisi, y eksenini y dizisi oluşturacaktır. Aşağıdaki komutları yazıp enter tuşuna bastıktan sonra aşağıdaki grafik penceresi açılır.

```
>> x = 0:0.5:20;  
>> y = 1 + x.^2;  
>> plot(x,y)
```



**Matlab'da Sinüs Grafiğinin Çizdirilmesi**

Grafiğin yatay eksenini açı değerleri, düşey eksenini ise bu açı değerlerinin sinus değerleri oluşturacaktır. Onun 2 adet diziye ihtiyacımız var. Birinci dizimiz açılar olacaktır.

```
>> angles = 0: pi/ 360 : 2*pi;
```

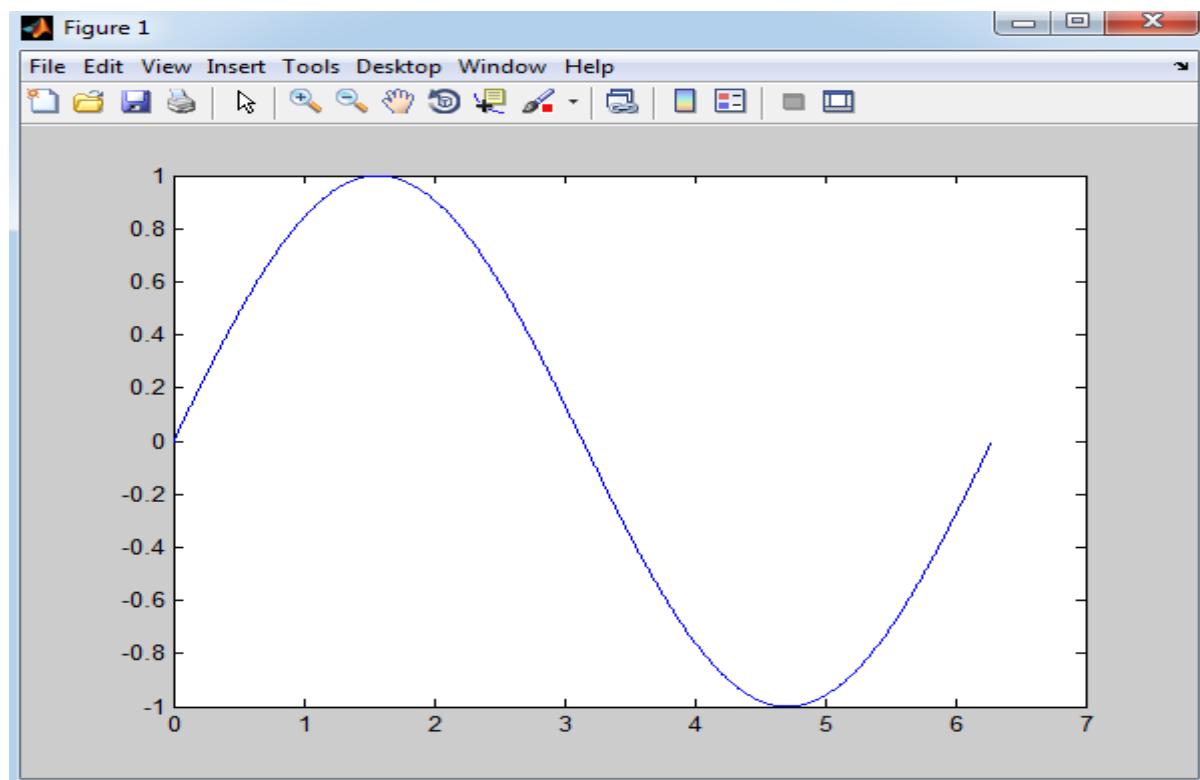
Bu komutun anlamı 0'dan 2\*pi'ye pi/360 (çok küçük bir adım değeri, farklı bir değer seçilebilirdi, 0.1 gibi) adımla dizi oluştur demektir. Sıra düşey ekseni için dizinin oluşturulmasında;

```
>> y = sin(angles);
```

Workspace'e baktığımızda elimizde aynı boyutta (aynı eleman sayısına sahip) angles ve x adında 2 dizimizin olduğunu görürüz. O zaman bu 2 diziyi kullanarak bir çizim yapabiliriz.

Bizim çizdirmek istediğimiz grafikte x ekseninde açı değerleri, y ekseninde ise sinüs değerleri olsun istiyoruz.

```
>> angles = 0: pi/360 : 2*pi;  
>> y = sin(angles);  
>> plot(angles, y)
```

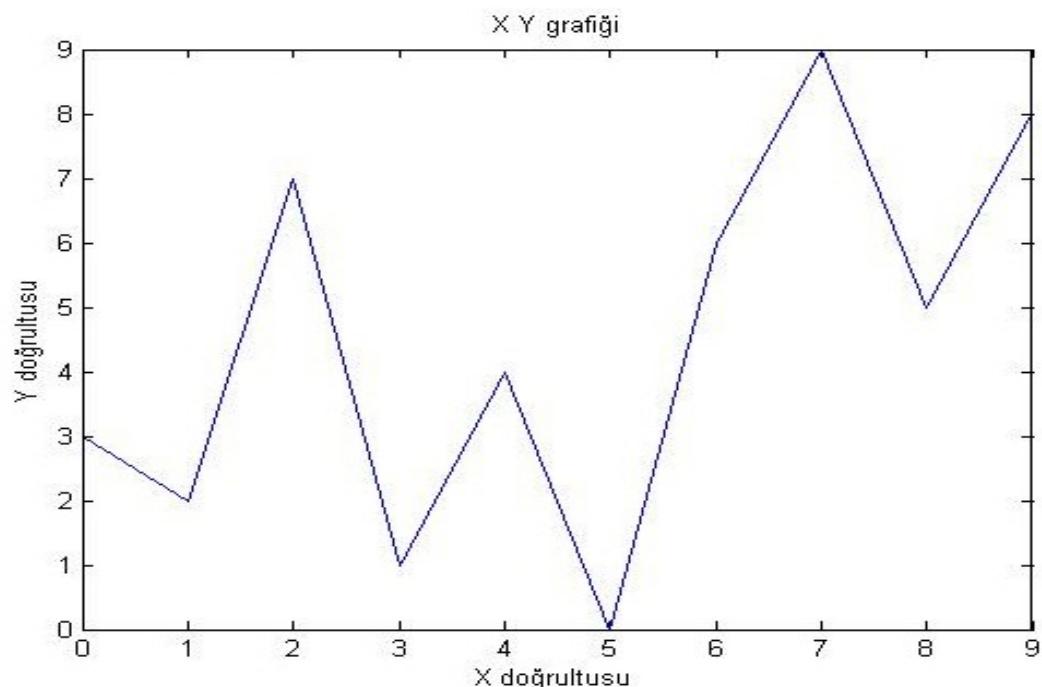


### Matlab label ve title komutu

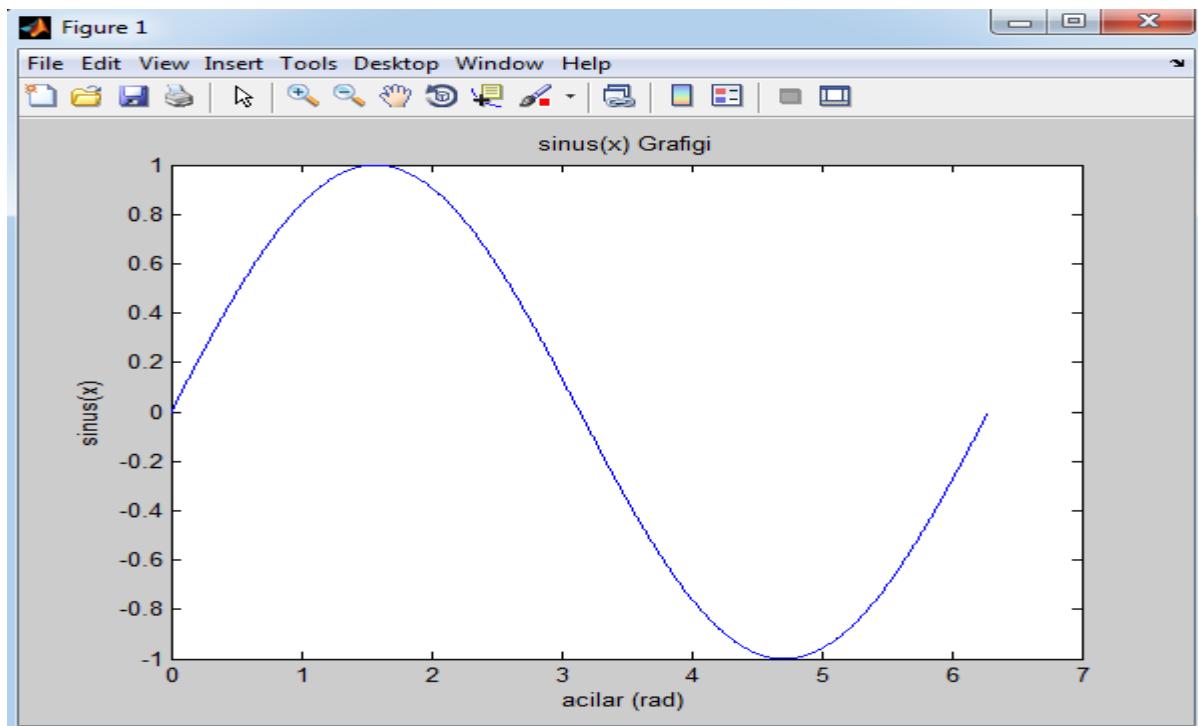
Bir önceki örnekte  $\sin(x)$  grafiğini elde ettik. Ancak bu ifadeyi daha güzel hale getirebiliriz: Eksenleri etiketleyebiliriz (label), grafiğe başlık verebiliriz (title) ve ızgara ekleyebiliriz (grid). label ve title komutunun genel kullanım formatı:

- ❖ **xlabel('x eksenine verilecek etiket'):** Yataydaki doğrultuyu isimlendirirken kullanılan fonksiyon,
- ❖ **ylabel('y eksenine verilecek etiket'):** Dikey doğrultuyu isimlendirirken kullanılan fonksiyon,
- ❖ **title('grafiğin başlığı'):** Grafiği isimlendirirken kullanılan fonksiyondur.

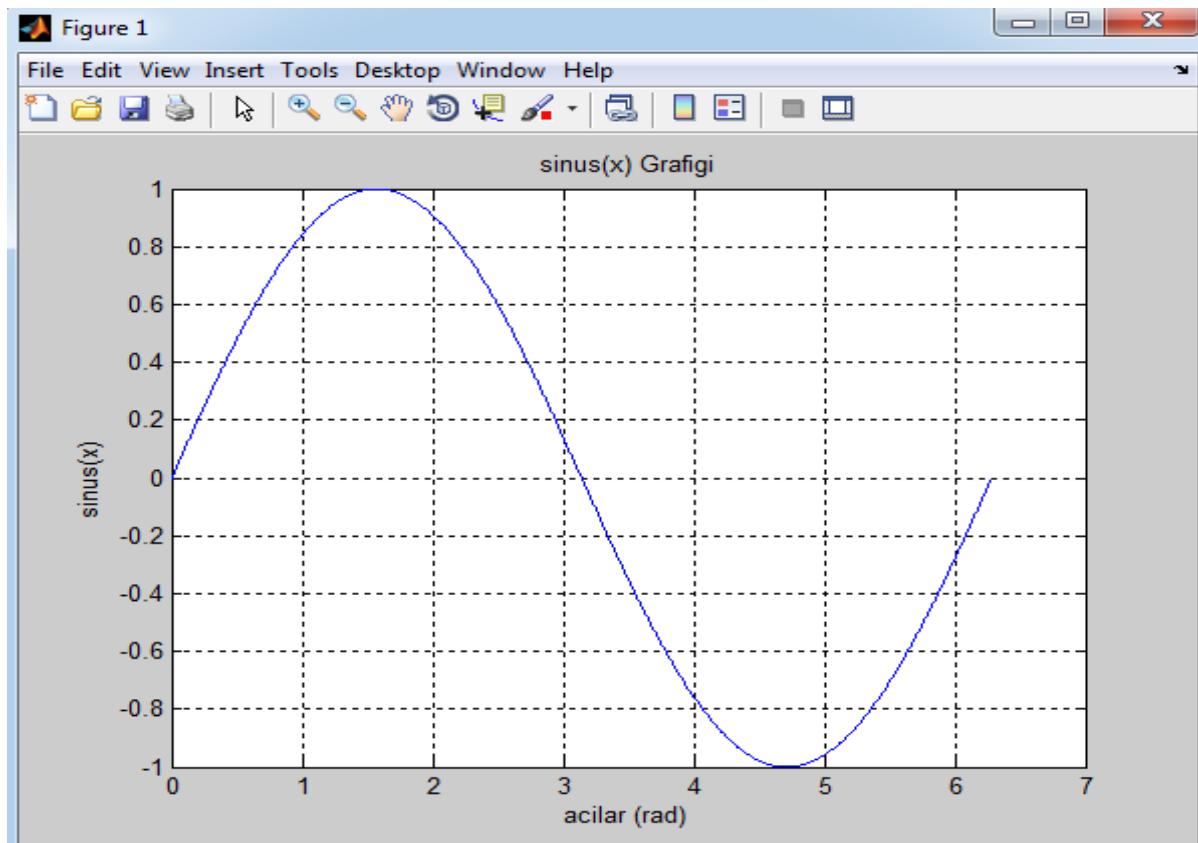
```
>>x=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
y= [3, 2, 7, 1, 4, 0, 6, 9, 5, 8];
plot(x,y);
xlabel('X doğrultusu');
ylabel('Y doğrultusu');
title('X Y grafiği');
```



```
>> angles = 0: pi/ 360 : 2*pi;
>> y = sin(angles);
>> plot(angles, y)
>> xlabel('açilar (rad)');
>> ylabel('sinus(x)');
>> title('sinus(x) Grafiği')
```



**Matlab grid komutu:** grafiğimize ızgara eklemeye yarar. Bu bize değerleri grafikten daha rahat okumamıza olanak sağlar. Yazdığımız komutlara grid komutunu da eklersek: `>>grid;`



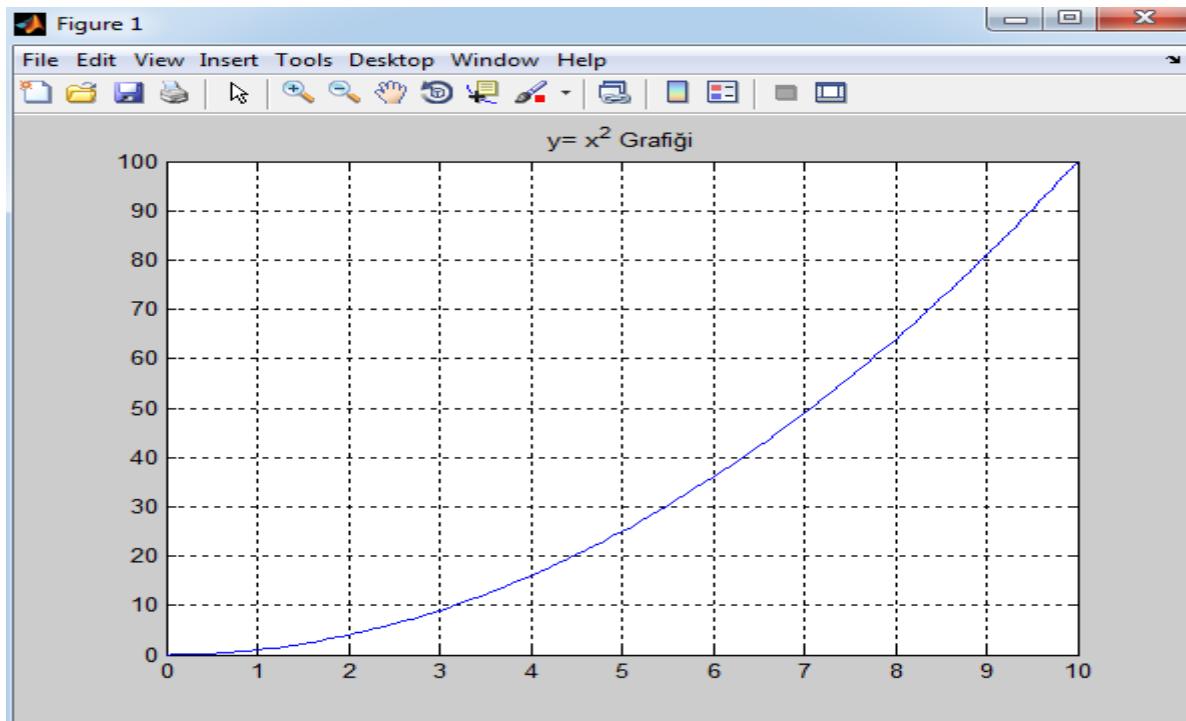
### Matlab'da Grafiğe Yazı Eklenmesi (text, gtext)

Çızdirdiğimiz bir grafiğe yazı eklemek isteyebiliriz. Matlab'da grafiğe yazı eklemek için 2 farklı komut vardır:

- ❖ text komutu
- ❖ gtext komutu

Bu bölümde text ve gtext komutunun kullanımını öğreneceğiz. İlk önce örnek bir grafik çizelim.  $y = x^2$ 'nin grafiğini çizelim.

```
>> x = 0:0.1:10;  
y = x.^2;  
plot(x,y);  
title('y= x^2 Grafiği');  
grid;
```



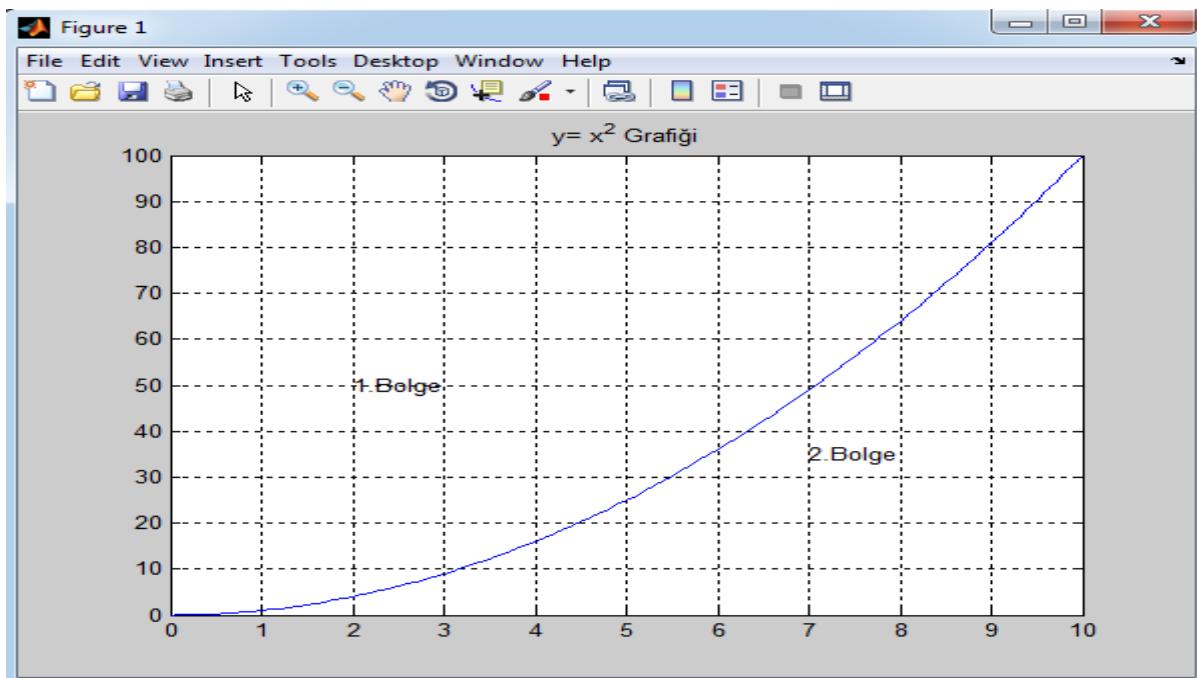
t

**Matlab text komutu:** text komutunun genel kullanım formatı:

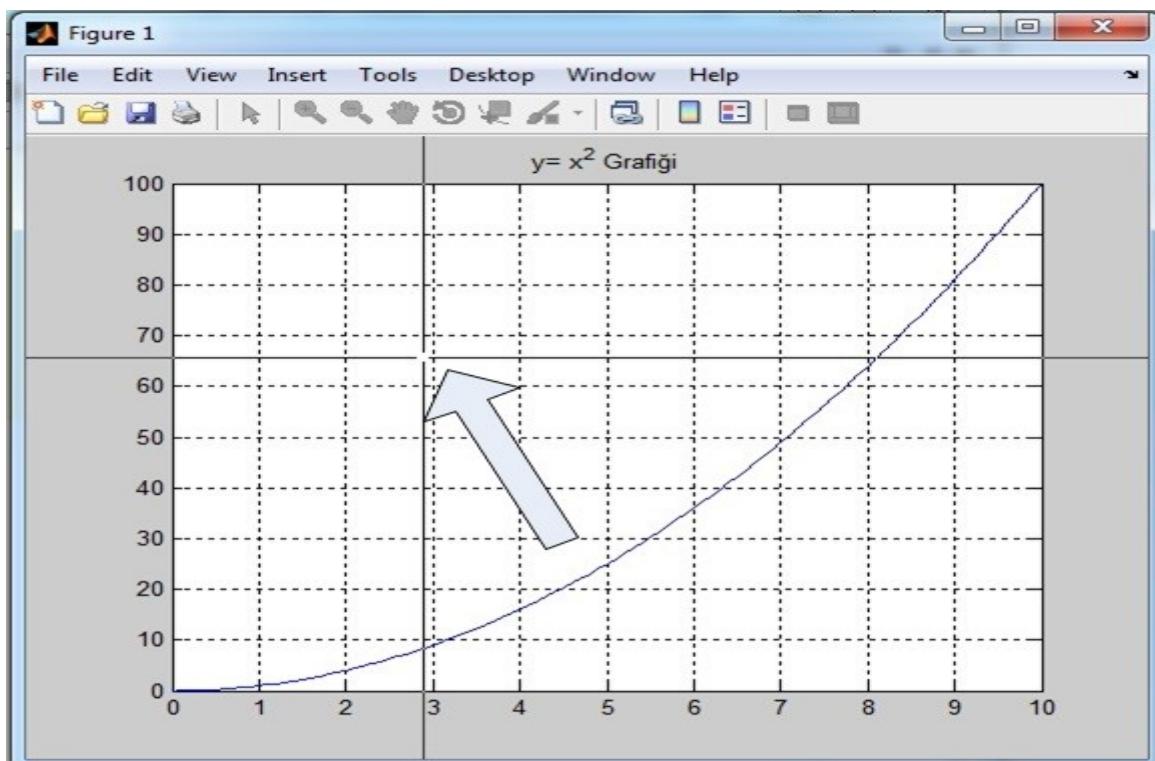
**text(x kordinati, y kordinati, 'Yazı')**

Yukarıdaki örneğe aşağıdaki komut satırlarını ekleyelim:

- ❖ >> text(2,50,'1.Bölge')
- ❖ >> text(7,35,'2.Bölge')

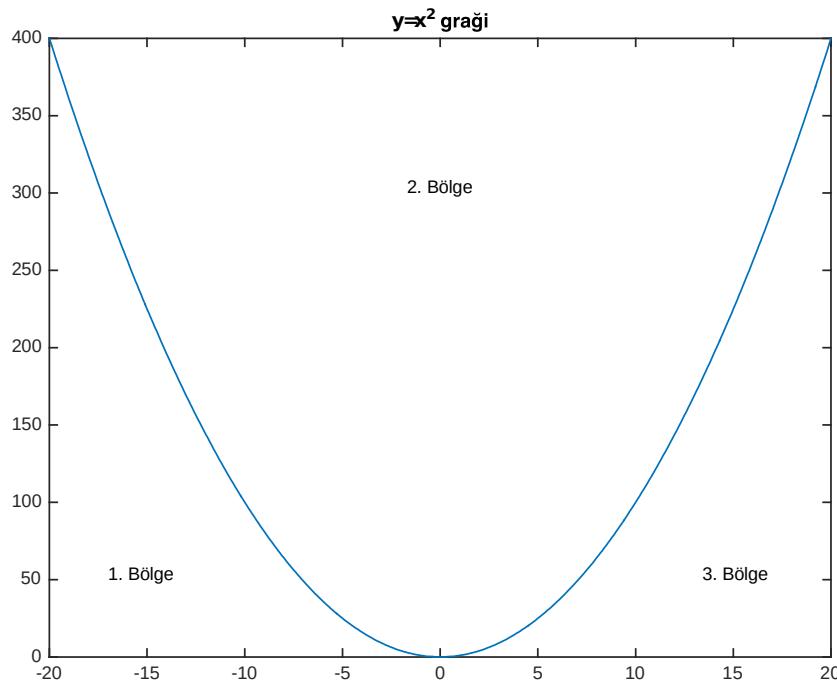


Grafikten görüleceği üzere ‘1. Bölge’ yazısını (2,50) koordinatlarına, ‘2. Bölge’ yazısını da (7,35) koordinatlarına koydu. **Matlab’da** grafiğe yazı eklemenin diğer bir yolu da **gtext komutudur**. Matlab’da gtext komutunu kullanarak grafiğe yazı eklemek için **text** komutundan farklı olarak yazının koordinatlarını girmek gerekmek.



Açılan pencerede **Matlab** fare ile bir noktayı tıklamanızı beklemektedir.

İlk tıkladığınız yere ‘1. Bölge’ ve ikinci tıkladığınız yere de ‘2. Bölge’ yazacaktır.



### Matlab da Çoklu Grafik Çizimi

Aynı pencerede farklı iki grafiği çizdirmek isteyebiliriz. Bu bölümde Matlab'da bir pencerede birden fazla grafiği nasıl çizdireceğimizi göreceğiz. Aynı pencerede hem  $\sin(x)$  hem de  $\cos(x)$  grafiğini çizdirelim. Aşağıdaki kodu yazarız:

```
>> x = 0: pi/360 : 2*pi; % 0'dan 360'a açı dizisi oluşturduk.
y1 = sin(x);
y2 = cos(x);
plot(x,y1,x,y2);
grid;
xlabel('açı (rad)')
title('sin(x) ve cos(x) Grafiği');
```



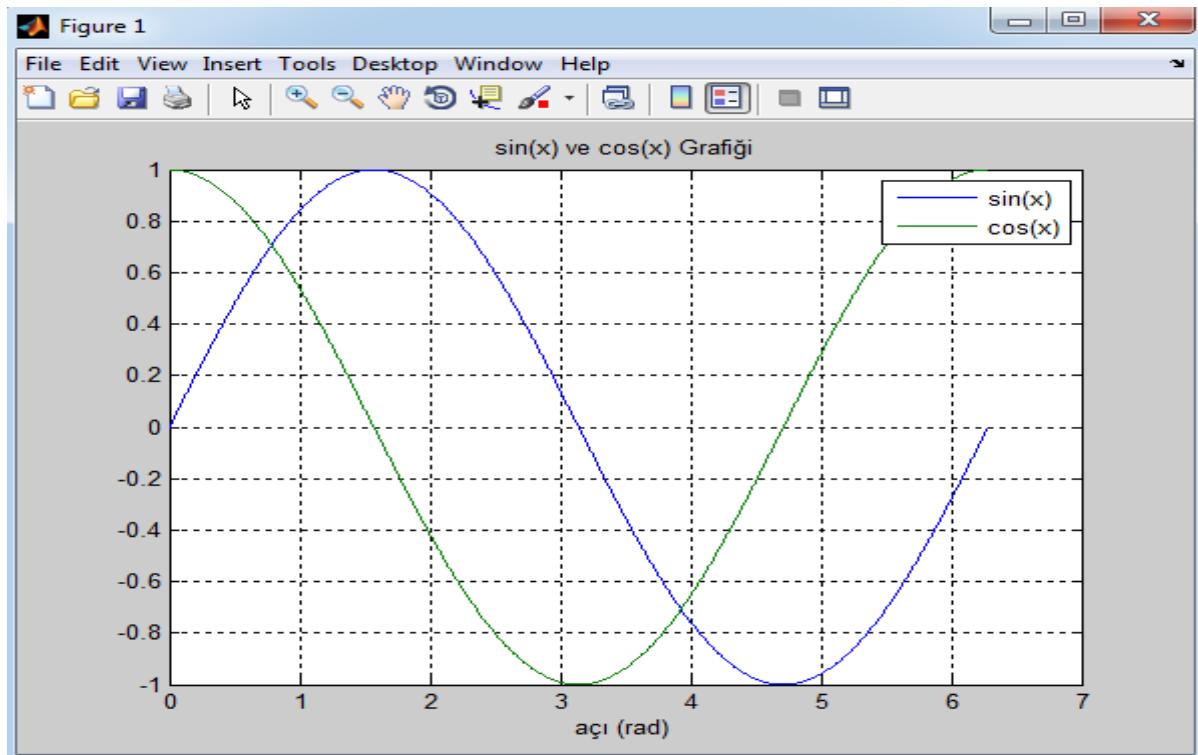
Görüleceği üzere aynı pencere içinde hem  $\sin(x)$  hem de  $\cos(x)$  grafiğini çizdirdik. Bu grafiklerin hangisinin  $\sin(x)$  hangisinin  $\cos(x)$  olacağını nasıl bileyeciz. Tabi ki 0 derecede 1 değerinde olan  $\cos(x)$  diğer de  $\sin(x)$ 'dir. Ancak bu ayrimı yapmak her grafikte bu kadar kolay olmayabilir. Bu durumda legend komutu işimize yarayacaktır.

**Matlab legend komutu:** Önceki problemdeki kodu aynen yazacağız. Sonuna sadece legend'lı satırı ekleyeceğiz.

```
legend('sin(x)', 'cos(x)');
```

Aşağıdaki grafikte legend yardımıyla hangi grafiğin  $\sin(x)$  hangi grafiğin  $\cos(x)$  olduğu seçilir.

```
>> x = 0: pi/360 : 2*pi; % 0'dan 360'a açı dizisi oluşturduk.
y1 = sin(x);
y2 = cos(x);
plot(x,y1,x,y2);
legend('sin(x)', 'cos(x)')
grid
xlabel('açı (rad)')
title('sin(x) ve cos(x) Grafiği');
```



Yukarıdaki örnekte çizimi tek bir plot komutu kullanarak yaptık: **plot(x,y1,x,y2)**

Ancak çizdirilecek çok sayıda grafiğimiz olabilir. Her bir grafiği teker teker ancak yine aynı pencerede çizdirmek isteyebiliriz. Bu durumda **hold komutunu** kullanacağız.

**Matlab hold komutu:** Aynı grafiği tek bir plot komutu değil de 2 adet plot komutu kullanarak çizdireceğiz. **hold komutu** mevcut pencerenin kapanmasını engelleyerek yeni gelen grafiğin de aynı pencerede çizilmesini sağlar.

```
>>plot(x,y1,'r');
>>hold on;
>>plot(x,y2,'g');
```

Sonuç olarak aynı grafiği elde ederiz.

**Hold komutu** koda eklenmeseydi ilk önce **ilk plot** komutu gerçekleştirilecekti. Yani **sinus(x)** grafiği çizilecekti. Ancak **ikinci plot** komutuna sıra geldiğinde bu pencere kapatılacak yeni **cos(x)** grafiği çizilecekti. Dolayısıyla program tamamlandığında mevcut pencerede sadece son plot komutu (**cos(x) grafiği**) kalacaktı.

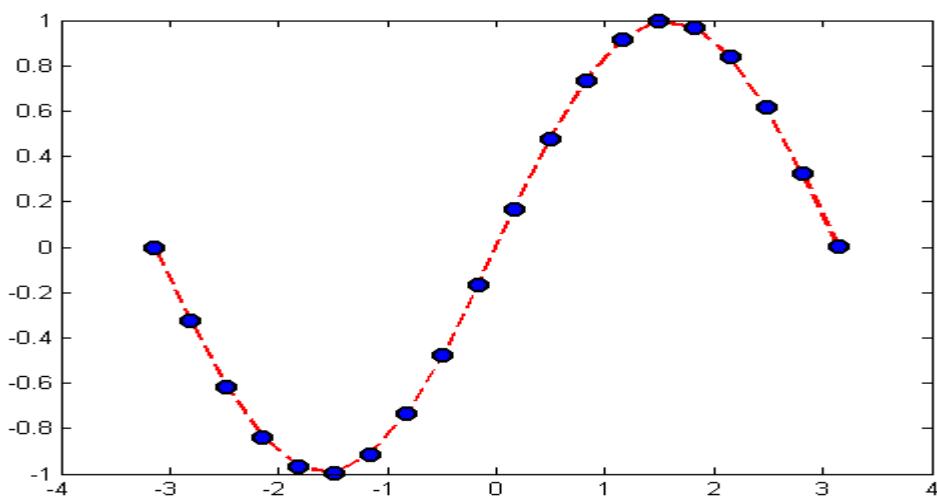
Eğer grafikleri ayrı ayrı pencerelerde çizdirmek istiyorsak, her **plot** komutundan önce **figure(1)**, **figure(2)**,.... diye pencereleri isimlendirmek gereklidir.

Grafiklerimize daha anlaşılır olmalarını sağlamak için şekil ve renk verebiliriz.

Renk	İşaretleme Tipi	Çizgi Tipi
<b>y: yellow</b> (sarı)	.	: nokta
<b>m: magenta</b> (mor)	<b>O</b>	: yuvarlak
<b>b: blue</b> (mavi)	<b>X</b>	: x işaretleri
<b>r: red</b> (kırmızı)	<b>+</b>	: artı işaretleri
<b>g: green</b> (yeşil)	*	: yıldız işaretleri
<b>w: white</b> (beyaz)	<b>S</b>	: square (kare)
<b>k: black</b> (siyah)	<b>d</b>	: diamond (elmas)
	<b>v</b>	: aşağı üçgen
	<b>^</b>	: yukarı üçgen
	<	: sola üçgen
	>	: sağa üçgen
	<b>p</b>	: pentagram (beşgen)

MATLAB dokümantasyonundan bakarsak bu iş için tonla çizim fonksiyonu var. Ben en çok kullandıklarımı yazıyorum.

```
x = linspace(-pi,pi,20);
y = sin(x);
plot(x,y,'--ro','LineWidth',2,...
      'MarkerEdgeColor','k',...
      'MarkerFaceColor','b',...
      'MarkerSize',8)
```



**LineWidth:** Doğrunun kalınlığı. Örnekte 2 verilmiş.

**MarkerEdgeColor:** İşaretçinin çerçeve rengi. Örnekte siyah verilmiş.

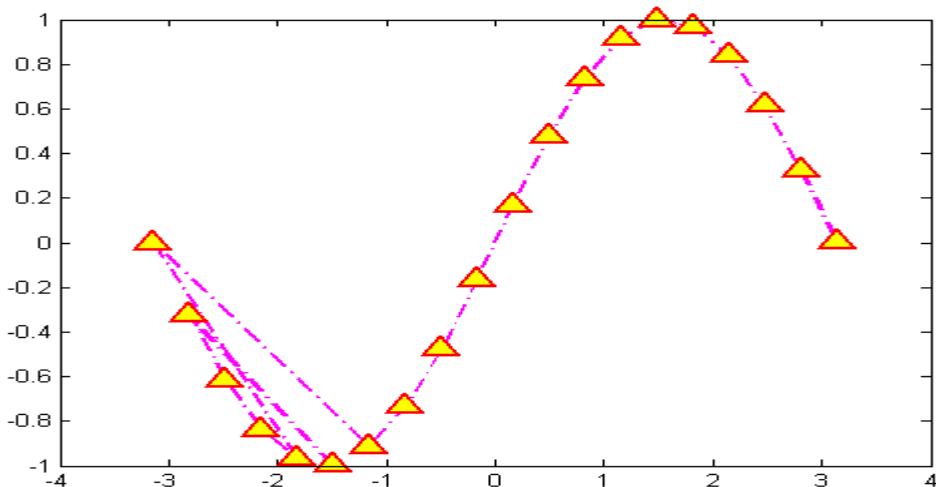
**MarkerFaceColor:** İşaretçinin iç rengi. Örnekte mavi verilmiş.

**MarkerSize:** İşaretçinin boyutu. Örnekte 8 verilmiş.

Tüm parametreleri vermek gerekli değil, üstteki kodu canınız nasıl isterse değiştirip sonuçlarına bakabilirsiniz.

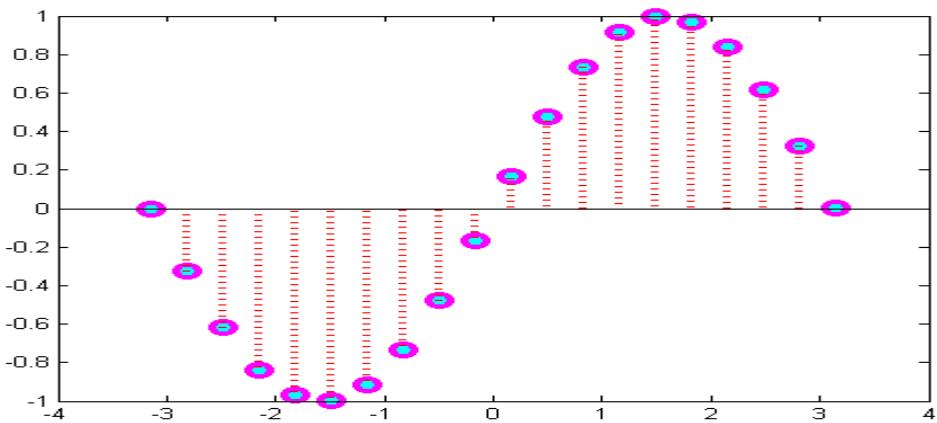
**plot** fonksiyonu iki nokta arasına doğru parçaları koyar, o yüzden sırası önemli. Örneğin ilk elemanların sırası şöyle olsaydı:

```
k = [6 2 3 4 5 1 7:length(x)];  
plot(x(k),y(k),'-.m^','LineWidth',2,...  
'MarkerEdgeColor','r',...  
'MarkerFaceColor','y',...  
'MarkerSize',12)
```

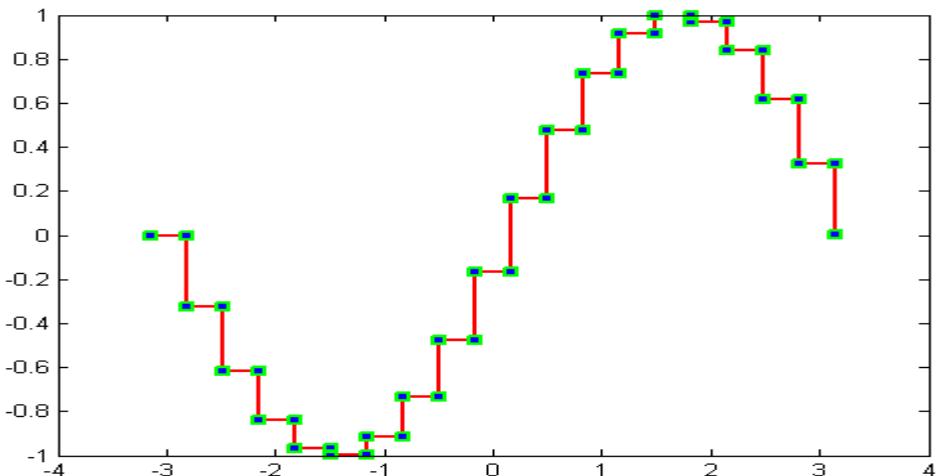


Görüldüğü üzere çizim güzel olmayacağı. Tabii çizimde çizgi kullanmayı bilirdik ve sıralı olması farketmezdi:

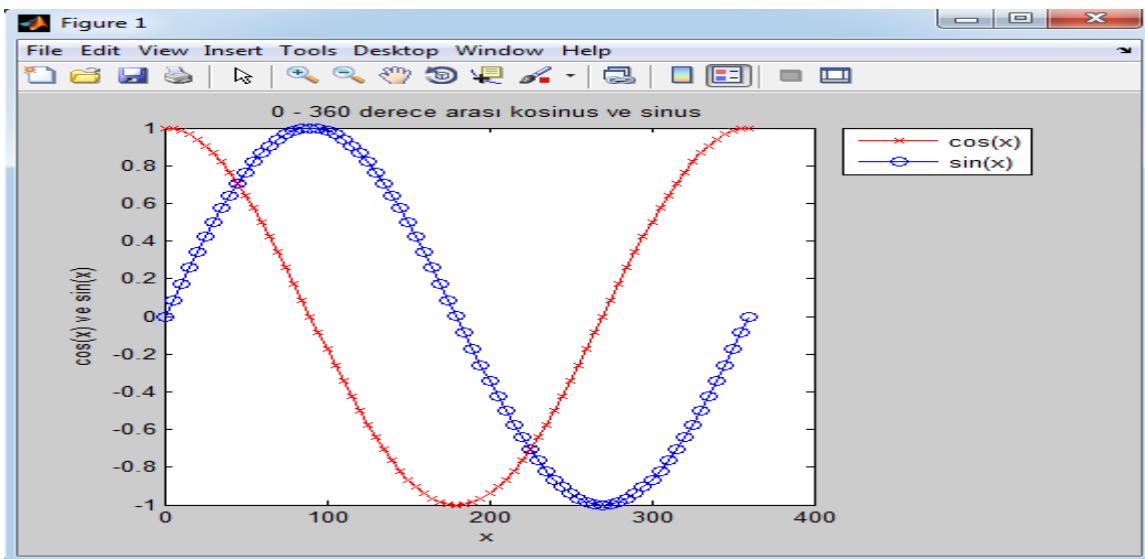
```
stem(x,y,:ro','LineWidth',3,...  
'MarkerEdgeColor','m',...  
'MarkerFaceColor','c',...  
'MarkerSize',10)
```



```
stairs(x,y,'-rs','LineWidth',2,...  
'MarkerEdgeColor','g',...  
'MarkerFaceColor','b',...  
'MarkerSize',5)
```



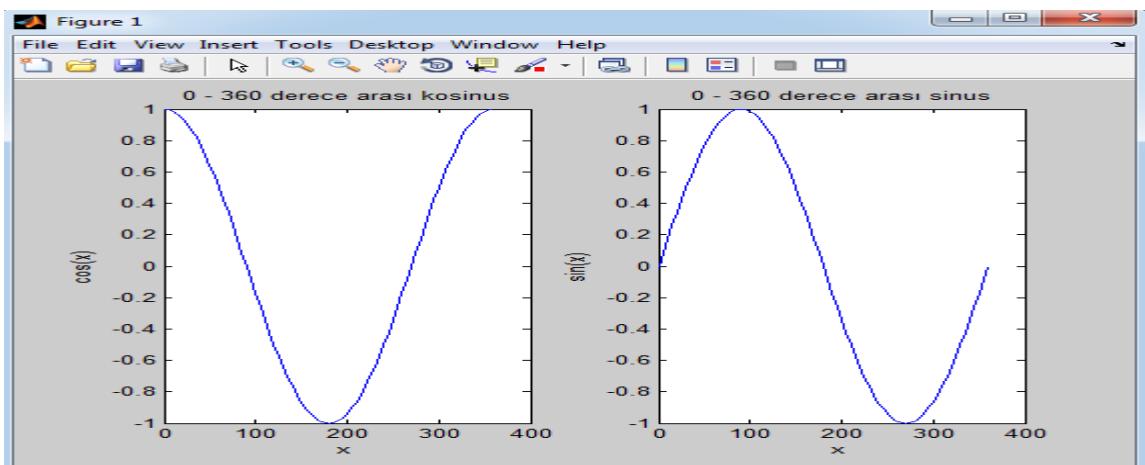
```
x=[0:5:360];  
c=cos(x*pi/180); %Matlab da trigonometrik işlemler radyan cinsinden yapılır  
s=sin(x*pi/180); %Matlab da trigonometrik işlemler radyan cinsinden yapılır  
plot(x,c,'r-x',x,s,'b-o');  
xlabel('x');  
ylabel('cos(x) ve sin(x)');  
title('0 - 360 derece arası kosinus ve sinus');  
legend('cos(x)', 'sin(x)', -1);
```



Yukarıdaki çizimde sinüs ve kosinüs daha anlaşılır bir şekilde görülmektedir. Birden çok grafiği aynı anda çizdirmek istiyorsak bu grafikleri yan yana çizdirmekte mümkündür. Bu yan yana çizdirme işlemi için **subplot** ve **plot** fonksiyonlarını birlikte kullanıyoruz.

**Subplot Fonksiyonu:** Subplot fonksiyonu sayesinde MATLAB da birden fazla grafiği aynı anda ekranaya çizdirebiliriz.

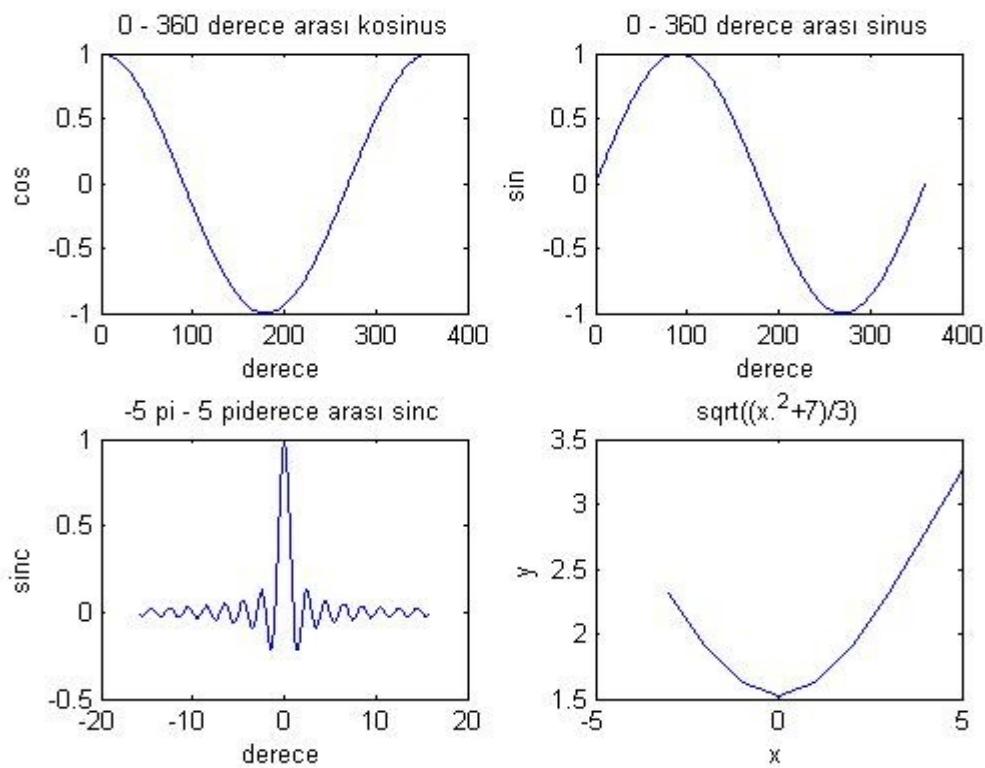
```
x=[0:5:360];
c=cos(x*pi/180); %Matlabda trigonometrik işlemeler radian cinsinden yapılır
s=sin(x*pi/180); %Matlabda trigonometrik işlemeler radian cinsinden yapılır
subplot(1,2,1), plot(x,c);
xlabel('x');
ylabel('cos(x)');
title('0 - 360 derece arası kosinus');
subplot(1,2,2), plot(x,s);
xlabel('x');
ylabel('sin(x)');
title('0 - 360 derece arası sinus');
```



```

derece=[0:1:360];
c=cos(derece*pi/180); %Trigonometrik işlemler radyan cinsinden yapılır
s=sin(derece*pi/180); % Trigonometrik işlemler radyan cinsinden yapılır
derece2=-5*pi:.1:5*pi;
snc=sinc(derece2);
x=[-3:1:5];
y=sqrt((x.^2+7)/3);
subplot(2,2,1), plot(derece,c);
xlabel('derece');
ylabel('cos');
title('0 - 360 derece arası kosinus');
subplot(2,2,2), plot(derece,s);
xlabel('derece');
ylabel('sin');
title('0 - 360 derece arası sinus');
subplot(2,2,3), plot(derece2,snc);
xlabel('derece');
ylabel('sinc');
title('-5 pi - 5 piderece arası sinc');
subplot(2,2,4), plot(x,y);
xlabel('x');
ylabel('y');
title('sqrt((x.^2+7)/3)');

```

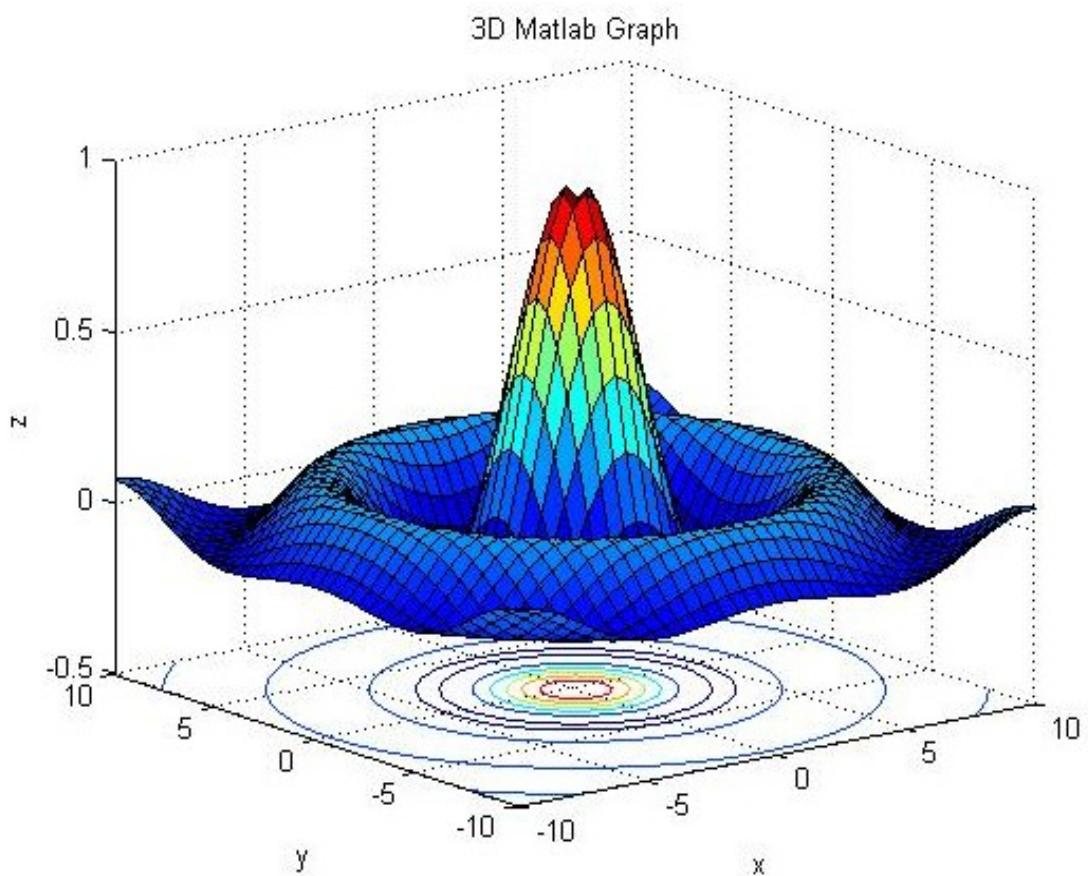


Yukarıda çizdiğimiz grafiklerin tümü 2 boyutlu grafiklerdi. Bunların yanı sıra MATLAB da 3 boyutlu grafikler ile veri grafikleri de çizilebilmektedir.

### 3D Grafikler

MATLAB' da çizilebilecek 3 boyutlu grafiklere aşağıdaki gibi örnekler verebiliriz.

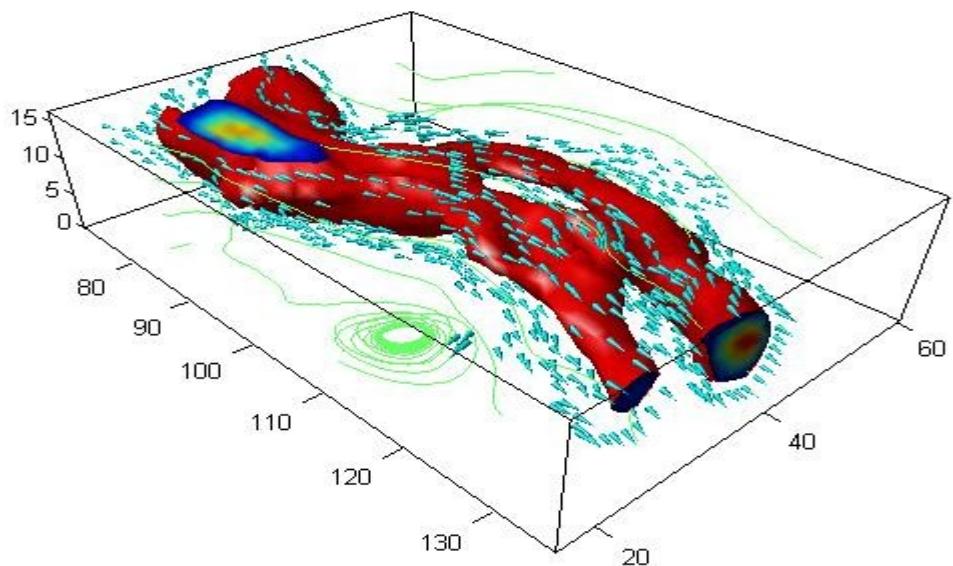
```
>>y = -10:0.5:10;
x = -10:0.5:10;
[X, Y] = meshgrid(x, y);
Z = sin(sqrt(X.^2+Y.^2)) ./ sqrt(X.^2+Y.^2);
surf(X, Y, Z);
view(-38, 18);
title('3D Matlab Graph');
xlabel('x');
ylabel('y');
zlabel('z');
```



```

load wind x y z u v w;
spd = sqrt(u.*u + v.*v + w.*w);
figure;
p = patch(isosurface(x, y, z, spd, 40));
isonormals(x, y, z, spd, p)
set(p, 'FaceColor', 'red', 'EdgeColor', 'none');
p2 = patch(isocaps(x, y, z, spd, 40));
set(p2, 'FaceColor', 'interp', 'EdgeColor', 'none')
daspect([1 1 1]);
[f, verts] = reducepatch(isosurface(x, y, z, spd, 30), .2);
h = coneplot(x, y, z, u, v, w, verts(:, 1), verts(:, 2), verts(:, 3), 2);
set(h, 'FaceColor', 'cyan', 'EdgeColor', 'none');
[sx, sy, sz] = meshgrid(80, 20:10:50, 0:5:15);
h2 = streamline(x, y, z, u, v, w, sx, sy, sz);
set(h2, 'Color', [.4 1 .4]);
colormap(jet);
box on;
axis tight;
camproj perspective;
camva(34);
campos([165 -20 65]);
camtarget([100 40 -5]);
camlight left;
lighting gouraud;

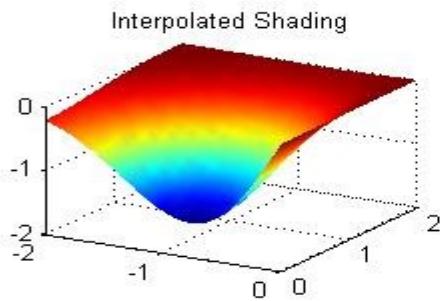
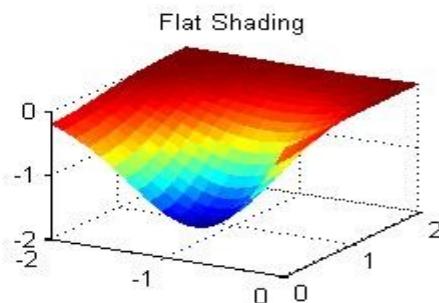
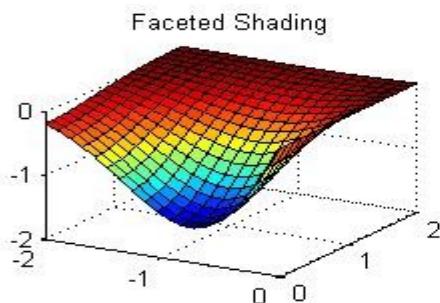
```



```

points = linspace(-2, 0, 20);
[X, Y] = meshgrid(points, -points);
Z = 2./exp((X-.5).^2+Y.^2)-2./exp((X+.5).^2+Y.^2);
subplot(2, 2, 1);
surf(X, Y, Z); view(30, 30);
shading faceted;
title('Faceted Shading');
subplot(2, 2, 2);
surf(X, Y, Z); view(30, 30);
shading flat;
title('Flat Shading');
subplot(2, 2, 3);
surf(X, Y, Z); view(30, 30);
shading interp;
title('Interpolated Shading');
subplot(2, 2, 4, 'Visible', 'off');
text(0, .5, sprintf('%s\n%s\n%s', ...
    'shading faceted', 'shading flat', 'shading interp'), ...
    'VerticalAlignment', 'middle', ...
    'FontName', 'Courier New', ...
    'FontWeight', 'bold', ...
    'FontSize', 12);

```

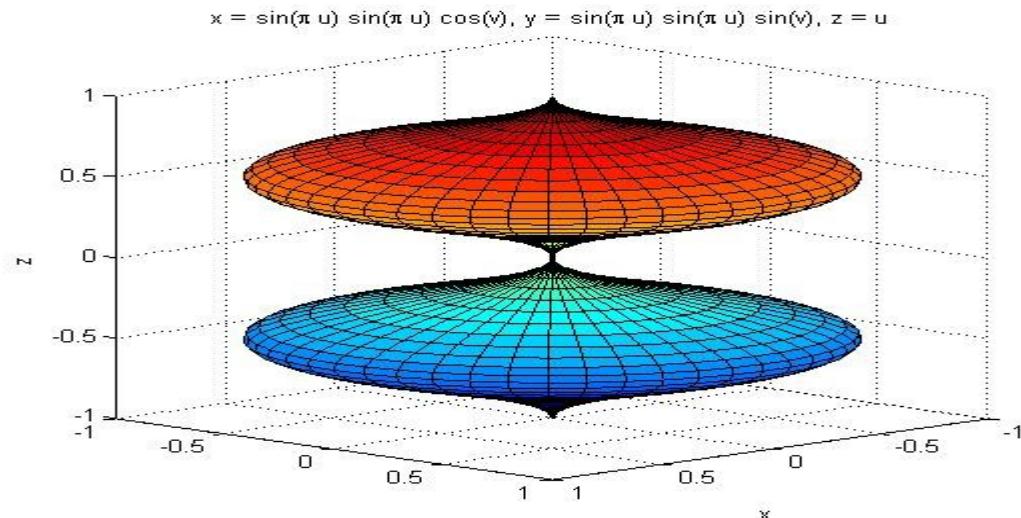


**shading faceted**  
**shading flat**  
**shading interp**

```

figure;
ezsurf('sin(pi*u)*sin(pi*u)*cos(v)', ...
        'sin(pi*u)*sin(pi*u)*sin(v)', ...
        'u', [-1 1 0 2*pi]);
view(135,15);

```



Yukarıdaki şekilde de görüldüğü gibi **MATLAB** 3D grafik çizmede oldukça başarılıdır. **MATLAB** da 3D grafik işlemleri için MathWorks referans olarak kullanılmıştır.

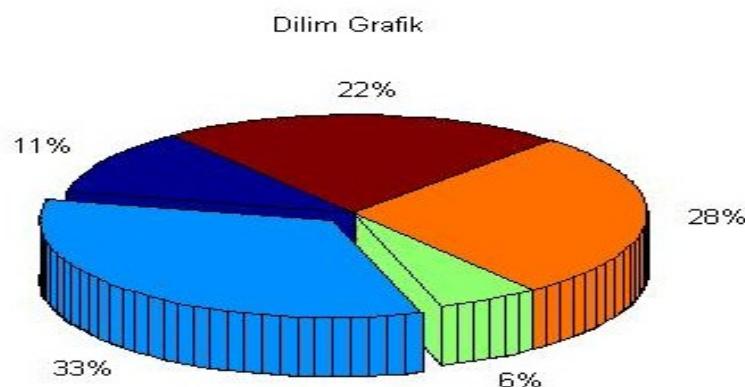
## Veri Grafikleri

**MATLAB** ile eldeki verileri kullanarak istatistiksel grafikler çizmekte mümkün.

```

>>x = [1 3 0.5 2.5 2];
figure;
explode = [0 1 0 0 0];      %1 olan kısmın işaret ettiği dilim ayrı olur.
pie3(x, explode);          %dilim grafik çizmeyi sağlayan fonksiyon.
title('Dilim Grafik');

```

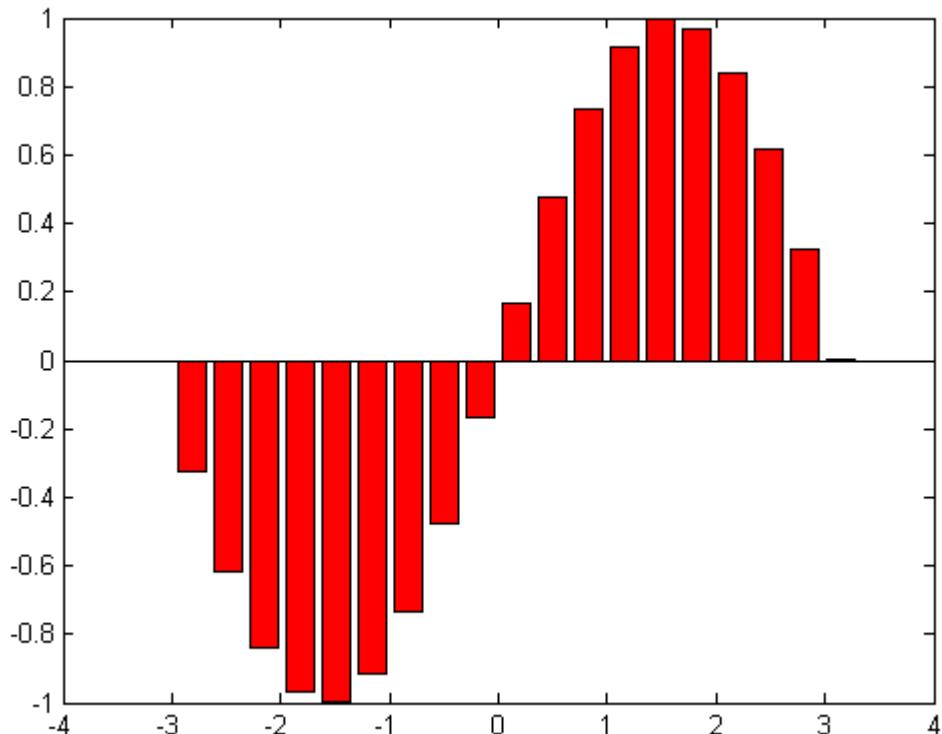


Bar diye de güzel bir fonksiyon var:

```

x = linspace(-pi,pi,20);
y = sin(x);
bar(x,y,'r')

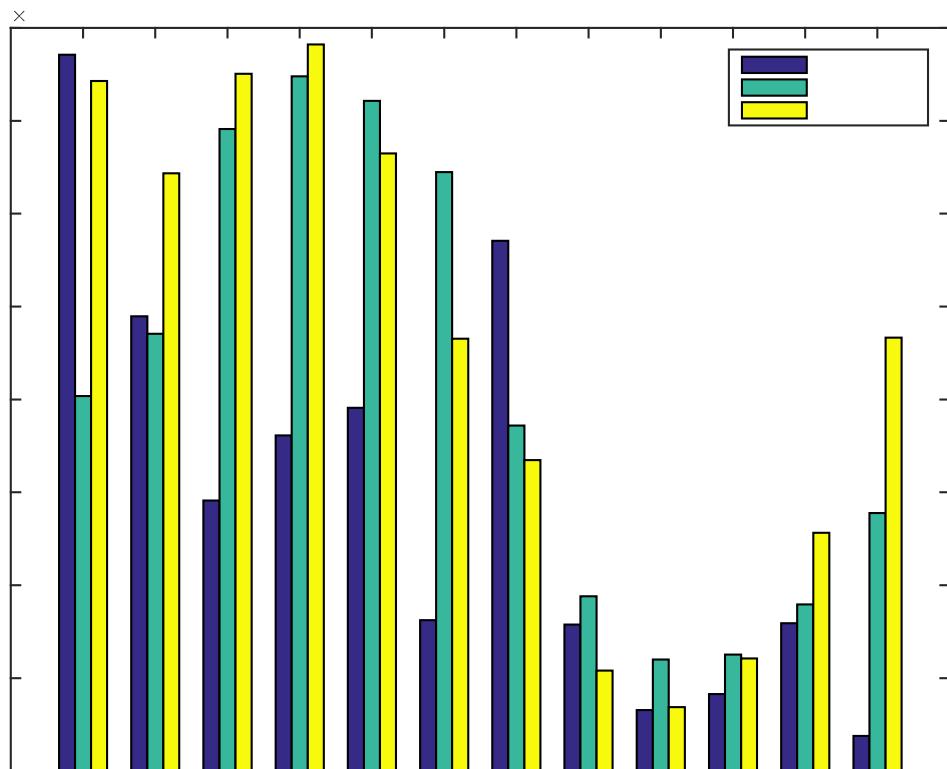
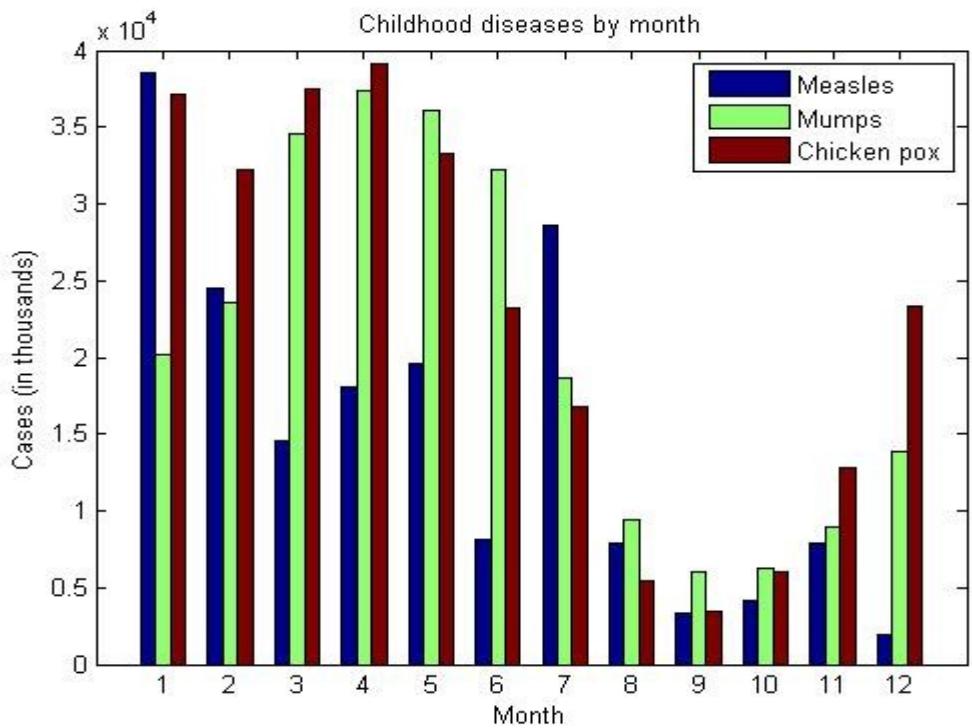
```



```

measles = [38556 24472 14556 18060 19549 8122 28541 7880 3283 4135 7953
1884];
mumps = [20178 23536 34561 37395 36072 32237 18597 9408 6005 6268 8963
13882];
chickenPox = [37140 32169 37533 39103 33244 23269 16737 5411 3435 6052
12825 23332];
figure;
bar(1:12, [measles' mumps' chickenPox'], 1);
axis([0 13 0 40000]);
title('Childhood diseases by month');
xlabel('Month');
ylabel('Cases (in thousands)');
legend('Measles', 'Mumps', 'Chicken pox');

```



Bu çizimleri ilk yazındaki gibi **subplot** ile aynı figür içine yerlestirebilir, değişik başlıklar, vb. ekleyebilirdik.

Sıra geldi grafiği dokümanlara eklemek için kaydetmeye. Bunu ara yüzü kullanarak **File -> Save** seçeneği ile yapmak mümkün ama otomatik de yapılabılırken niye uğraşalım!

```
>>saveas(gcf,'ornek_resim.jpg')
```

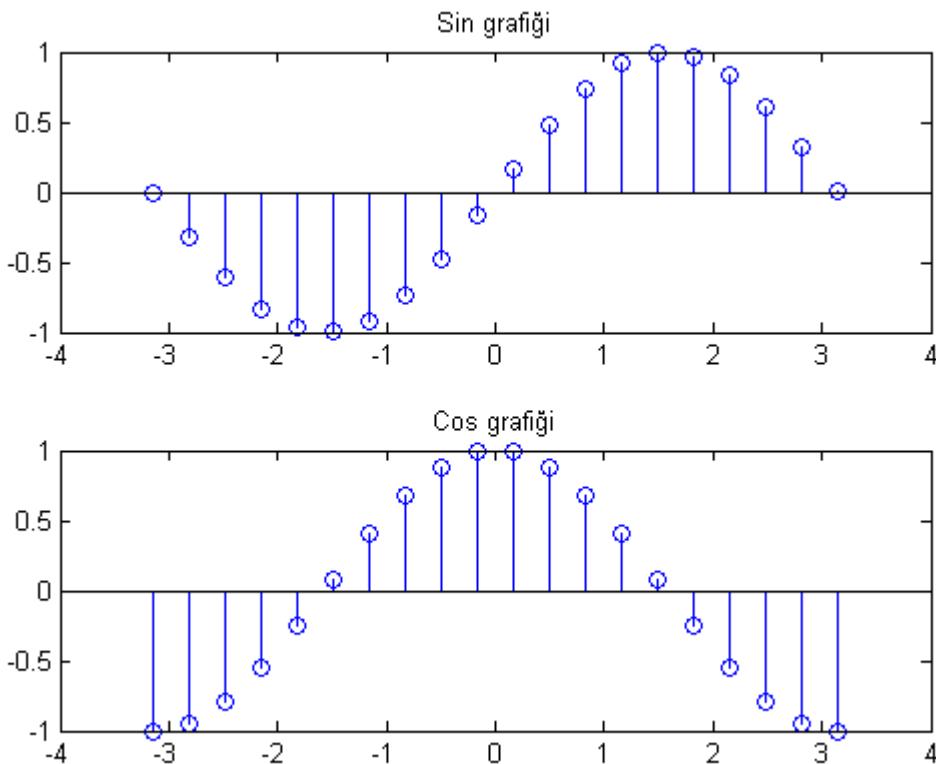
Üstte görüldüğü üzere **gcf** ile yakalanan figürü, yani şu anki gözüken figürü (**'get current figure'**) bir **jpg** olarak kaydettik.

Birçok format destekleniyor: **bmp**, **eps**, **pdf**, **png**, **ppm**, vb. Ayrıca **fig** de destekleniyor. O şekilde kaydederseniz daha sonra MATLAB a geri yükleyip figürün üstünde elle değişiklikler yapıp tekrar kaydedebilirsiniz.

Bitirmeden küçük ama önemli bir özelliği de ekleyelim. **Subplot** fonksiyonunu öğrenmiştık. Fakat bir figür içinde birden çok grafiğiniz varsa bu grafikleri birbirine bağlamak isteyebilirsiniz.

Bağlamaktan kastedilen birinde yaklaşma/uzaklaşma/kaydırma/vb. yapılınca diğerinin de görünümünün benzer şekilde güncellenmesi.

```
x = linspace(-pi,pi,20);
s = sin(x);
c = cos(x);
ax(1) = subplot(2,1,1);
stem(x,s)
title('Sin grafiği')
ax(2) = subplot(2,1,2);
stem(x,c)
title('Cos grafiği')
linkaxes(ax,'xy');
```



Üstteki örnekte xy parametresi ile hem x hem de y eksenlerini bağladık. Bu dokümdanda gözükmez ama oluşan grafikte bir pencereye yaklaşma uygularsanız, diğerinin de aynı şekilde hareket edeceğini göreceksiniz.

Parametre olarak yalnızca x veya y vererek sadece o eksenler de bağlanılabilir.

### **Matlab da Bazı Grafik Nesneleri**

- ❖ **Axis:** manuel eksen ölçüği
- ❖ **hold(on-off):** ekrandaki grafiği tut
- ❖ **ginput:** mouse konumlu girdi
- ❖ **figure:** yeni grafik penceresi aç

**axis komutu:** grafik eksenlerinin ölçeklerini ayarlamak için kullanılır. Genel yapısı:

**axis([xmin xmax ymin ymax])** şeklindedir.

Bu komut şeşlin ekranда fare kullanılmaksızın aktif hale getirilmesini sağlar. Ekranı ikiye bölüp yarısını **MATLAB** komut sayfası, yarısını grafik ekran olarak kullanarak da **grid** ve **axis** komutlarının etkilerini anında gözlemek mümkündür.

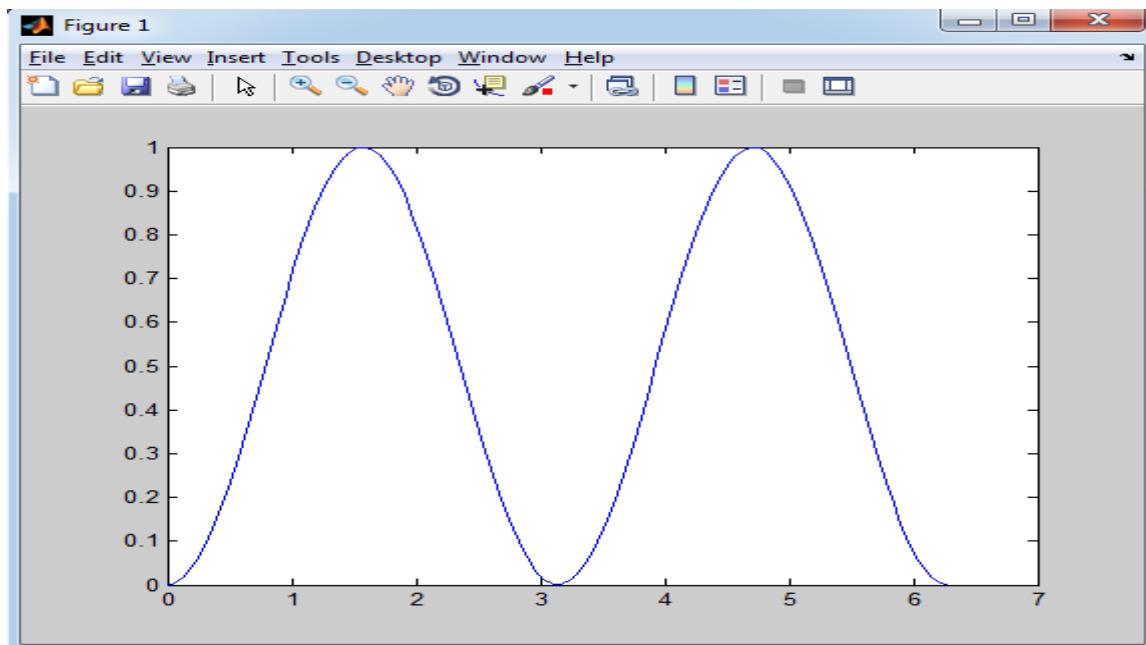
```
t = 0:pi/50:2*pi;  
[x, y, z] = cylinder(2+sin(t));  
figure;  
surf(x, y, z, 'LineStyle', 'none', 'FaceColor', 'interp');  
colormap('summer');  
axis('square', 'off');  
grid('off');
```



**Figüre Komutu:** Bir grafik çizdikten sonra onu kaybetmeden başka bir grafik elde etmek için figure komutunu kullanmamız gereklidir.

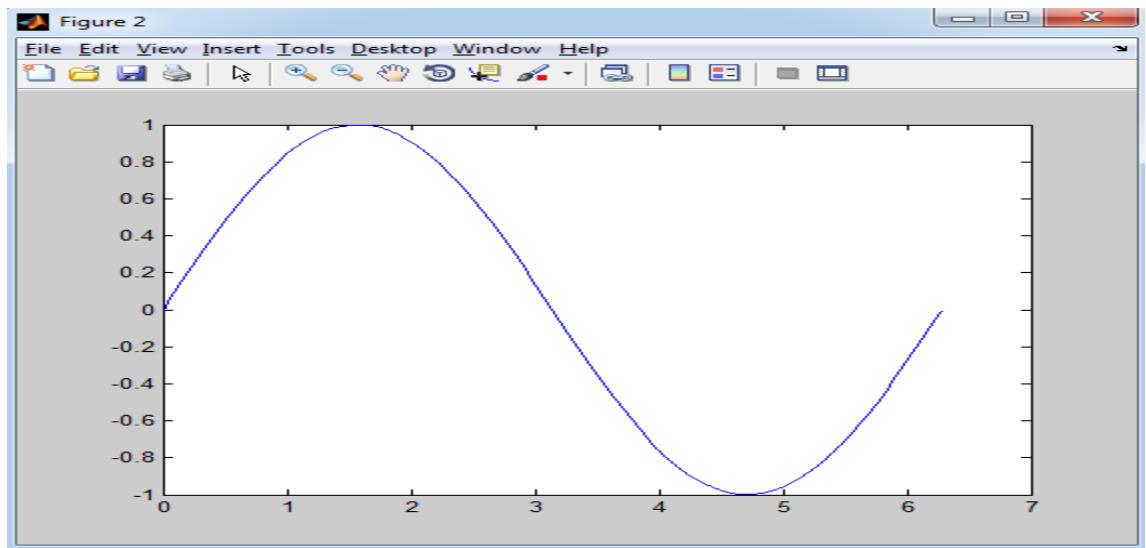
**Örnek:** İlk komut bütün grafik pencerelerini kapatmaya yarar.

```
>> close all  
>> x=pi/180*linspace(0,360);  
>> y1=sin(x).^2;  
>> plot(x,y1)
```



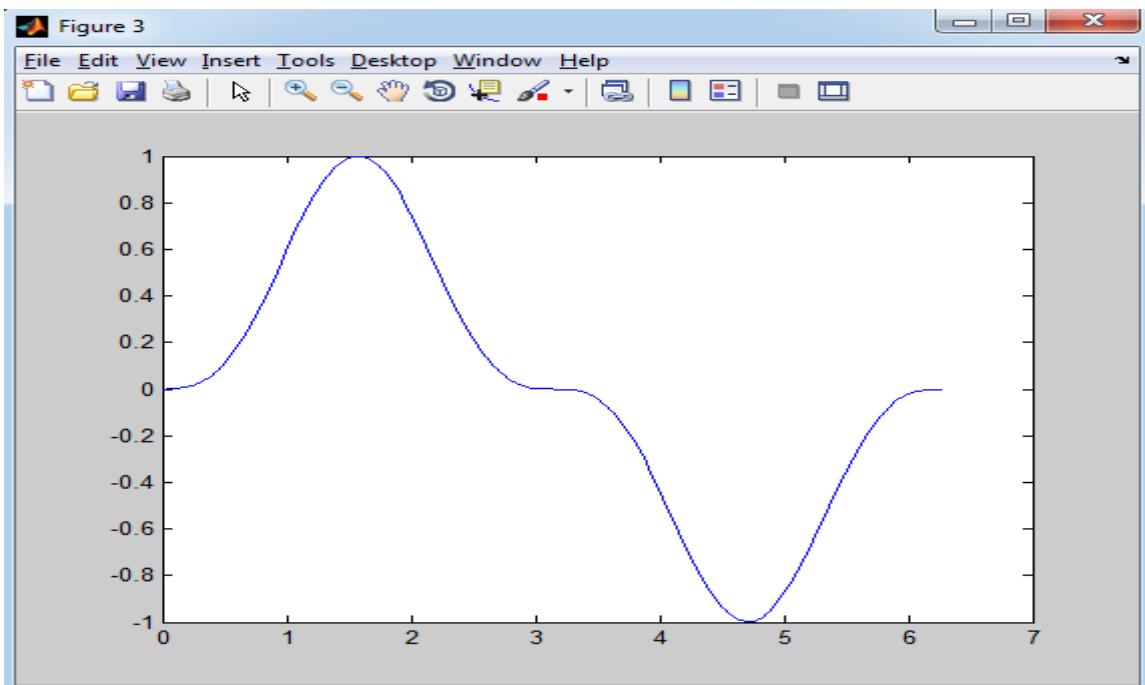
Şimdi eskisi kalmak üzere yeni bir grafik elde edelim.

```
>> figure  
>> y2=sin(x);  
>> plot(x,y2)
```



Bir grafik daha oluşturup öncekileri de muhafaza etmek için

```
>>h3=figure;  
>> plot(x,y1.*y2)
```



Daha önce iki grafik bulunduğu için  $h3$ 'ün değeri 3 dür. Şimdi hangi grafiği aktif hale getirmek istiyorsak ya fareyi onun üzerine sürükleyip tıklamalıyız ya da aşağıdaki komutları kullanmalıyız.

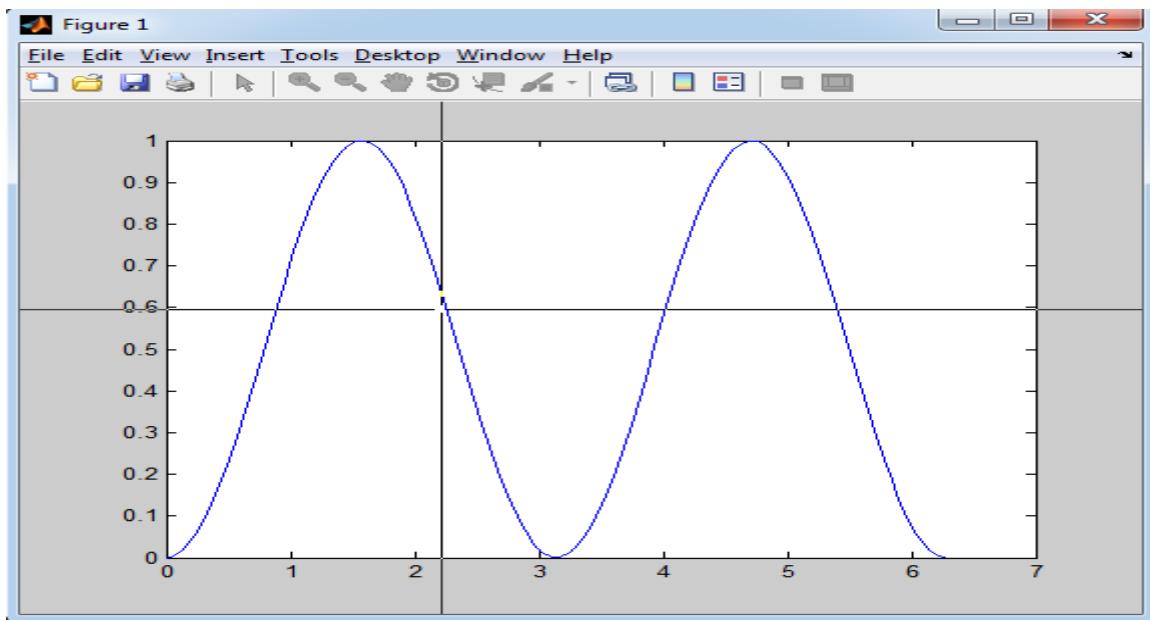
```
>> figure(1)
>>figure(2)
>>figure(h3)
```

Bu komutlar grafikleri aktif hale getirmek için kullanılır. Pencerelerden diyalog ikincisini kapatmak istiyoruz. Bunun için **close(2)** komutunu kullanmak gereklidir. Kapatmaksızın bir grafik penceresinin içeriğini silmek için **clf** komutu kullanılır.

**Ginput Komutu:** Grafik üzerindeki noktaların koordinatını elde etmek için kullanılan bir komuttur.

**>>[x,y]=ginput(n):** Komutuyla grafik üzerinde bulunan n tane noktanın koordinatlarını fareyi kullanarak elde etmek mümkündür. **n** noktayı tamamlamadan işlemi bitirmek için **enter** tuşunu kullanmak gereklidir. Bu komut **n** parametresini vermeden kullanılırsa **enter** tuşuna basana kadar x ve y için veri toplamaya devam eder.

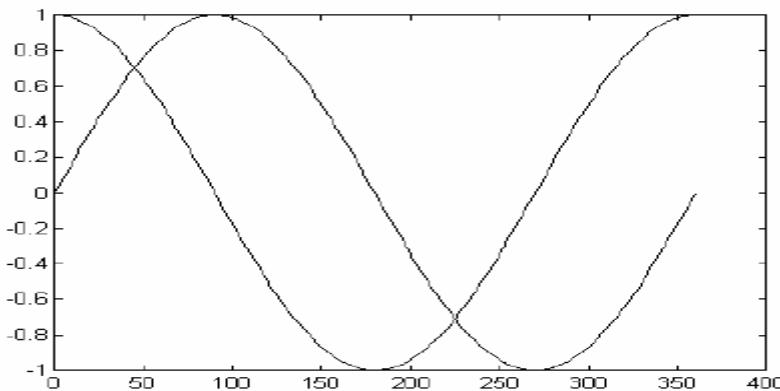
```
>> [x1,y1]=ginput(1)
x1 =
 2.2661
y1 =
 0.5863
```



**Hold Komutu:** **hold** fonksiyonu mevcut bir grafiği koruyarak bir sonraki çizimin aynı grafik üzerinde gösterilmesini sağlar. Bu moddan çıkmak için **hold off** komutunu kullanmak gereklidir.

Örnek: sinüs ve kosinüs fonksiyonlarını aynı grafik üzerinde gösterelim:

```
>>xd=0:360;
>>x=xd*pi/180;
>>y=sin(x);
>>z=cos(x);
>>plot(xd,y,xd,z)
```



Bu örnek aynı anda birden çok grafiğin çizilebileceğini göstermektedir. Bunun başka bir yolu **hold** fonksiyonunu kullanmaktır.

```
>> plot(xd,y)
>>hold
Current plot held
>> plot(xd,z)
```

Sonuç yukarıdaki grafiğin aynısı olacaktır. Daha fazla detay için **help hold** komutunu kullanabiliriz.

## Matlab da Resim İşlemleri

Bilindiği üzere resimler farklı renklere sahip küçük küçük blokların bir araya gelmesiyle oluşur. Yani resimlerin yapıtaşları piksel adını verdigimiz bu bloklardır. Eğer ekran çözünürlüğünü biraz azaltıp ekrana yaklaşırsanız pikselleri rahatlıkla gözlemlileyebilirsiniz. Bahsedilen pikselleri değerler olarak düşünürsek resmi de bu değerlerden oluşan matris olarak düşünmeliyiz. Bu değerleri kaç sefersek deri rengine benzer olur, en yüksek kaç seçebiliriz, skaler mi olmalı yoksa vektör olabilir mi gibi soruları cevaplamalıyız. Ancak bu sorulara çözüm getirdiğimiz takdirde (yani değerlerin ifade edilişi hakkında bir uzlaşıya vardığımızda) ekrandan istediğimiz renkleri göstermesini bekleyebiliriz.

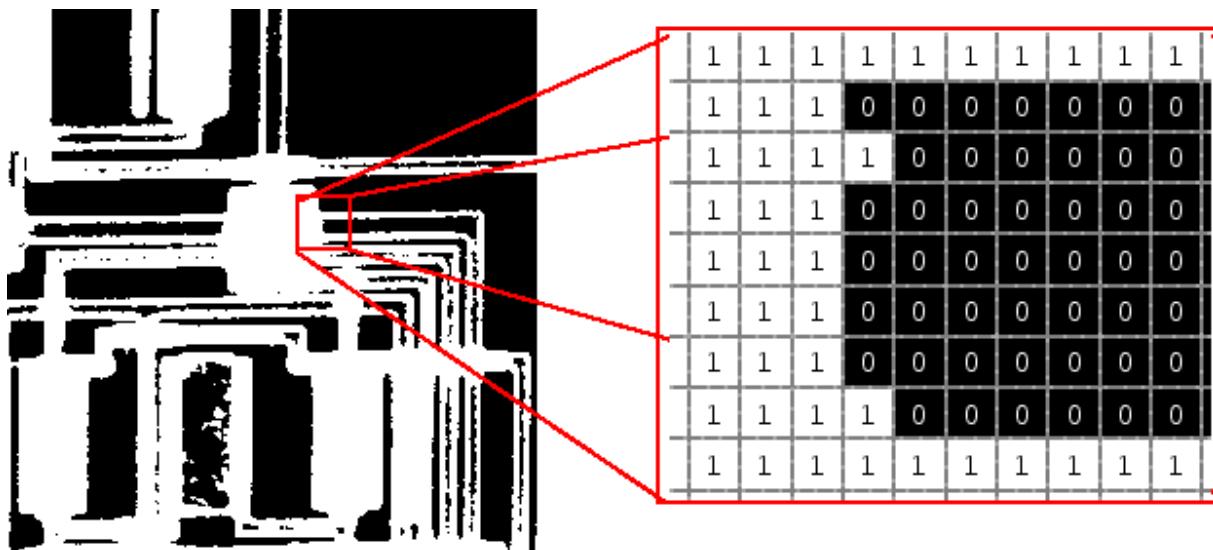
### MATLAB de Resim Türleri

Resimleri temel olarak dört tipe ayıralım:

- ❖ İkilik resimler
- ❖ İndekslenmiş resimler
- ❖ Gri seviye resimler
- ❖ Gerçek-renkli resimler

Şimdi teker teker bunları ele alalım.

**İkilik Resimler:** İkilik bir resimde her piksel 0 (siyah) veya 1 (beyaz) değerine sahiptir ve resim logical tipinde saklanır. Yazılarda ikilik resimler için **bw** önekini kullanacağız.

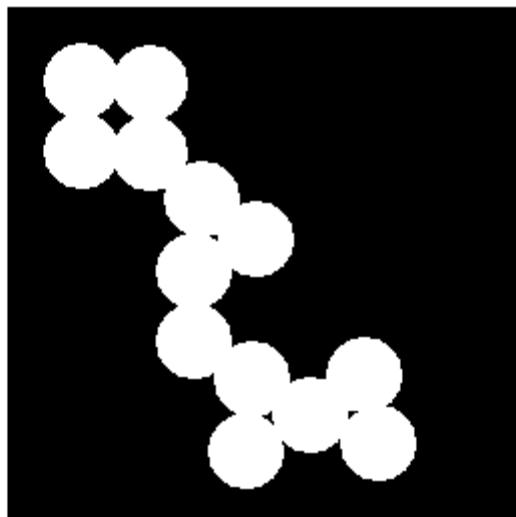


Yukarıda ikilik bir resme ve piksel değerlerine güzel bir örnek gösterim var. Şimdi de biz bir örnek yapalım. MATLAB in yüklenmiş olduğu dizini kök dizin olarak kabul edersek **toolbox/images/imdemos** dizininde örnek resimler var. Bunlardan **circles.png** resmini okuyalım.

**NOT:** Bu yazındaki kodları çalıştırabilmeniz için **İmge İşleme Araçkutusu (Image Processing Toolbox)** yüklü olmalıdır. Bunu, komut satırına ver komutunu girerek öğrenebilirsiniz. Örneğin bendeki MATLAB de **Image Processing Toolbox** sürüm 6.4 yüklü.

```
bwImg = imread('circles.png');
imshow(bwImg)
title('İkilik resim')
```

İkilik resim



Bakalım bwImg'nin boyutları ve tipi neymiş?

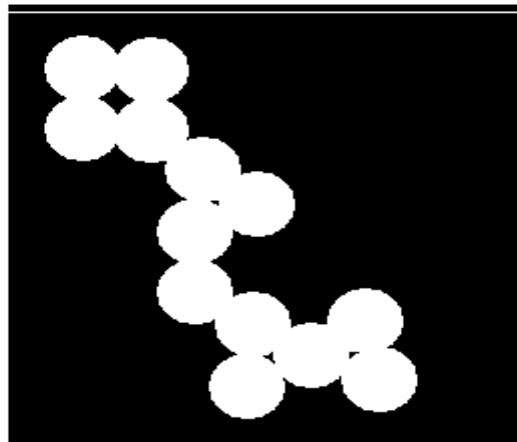
```
sz = size(bwImg)
imgType = class(bwImg)
sz =
256 256
imgType =
logical
```

Peki içerdiği farklı değerler nelermiş?

```
unique(bwImg)
ans =
0
1
```

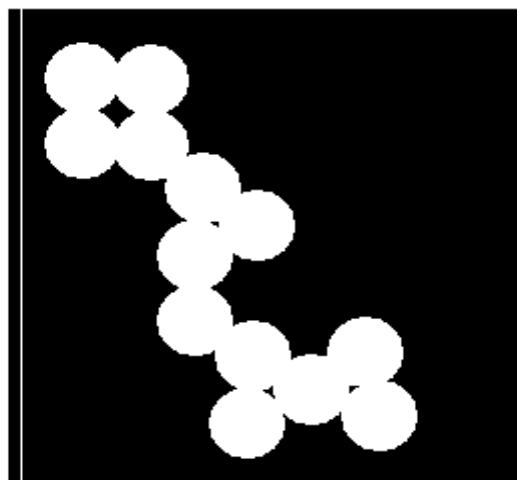
Görüldüğü üzere bu resim 256x256'lık bir matris ve **logical** tipinde. Yani sadece **0** ve **1** değerlerinden oluşuyor. Peki bu matrisin 5. satırındaki elemanlar resimde nereye tekabül ediyor? Beyaza boyayıp görelim.

```
bwImgCopy = bwImg;
bwImgCopy(5,:) = 1;
imshow(bwImgCopy)
```



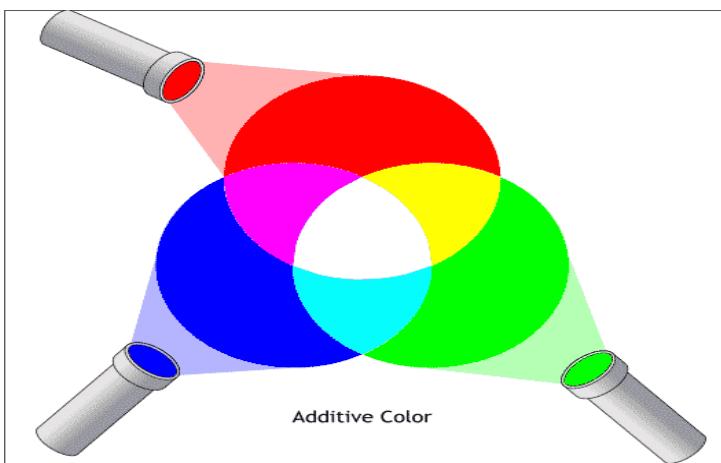
Peki 7.sütun?

```
bwImgCopy = bwImg;
bwImgCopy(:,7) = 1;
imshow(bwImgCopy)
```

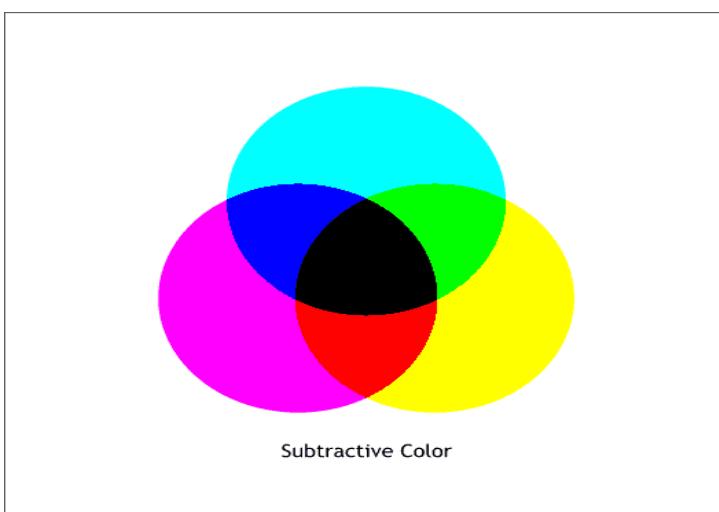


Demek ki resmi ifade eden matrisin satırları resmin yatay elemanlarını, sütunları da resmin dikey elemanlarını oluşturuyor, fakat **(1,1)** koordinatları resmin sol alt değil de **sol üst** kösesine denk geliyor. Görüntü işleme de ikilik resimleri genelde maske olarak kullanırız. Yani işlem yapacağımız bir resimle eş boyutlarda bir maske oluştururuz. Resmi işleme sokarken yalnızca ilgili masked'e beyaz olan yerleri kullanırız. İlerideki yazınlarda buna uygun örnekler mutlaka denk gelir, şimdilik bitirelim.

**İndekslenmiş resimler:** Bu konuya başlamadan bilmemiz gereken bir şey var: Bilgisayar ekranında gördüğümüz tüm renkler üç ana rengin belli oranlarında karışımından oluşur. Işıktaki ana renkler kırmızı, yeşil ve mavidir. Literatürde toplamsal (additive) renkler olarak geçer. RGB (R=kırmızı(red), G=yeşil(green), B=mavi(blue)) kısaltması olarak da karşımıza çıkar.



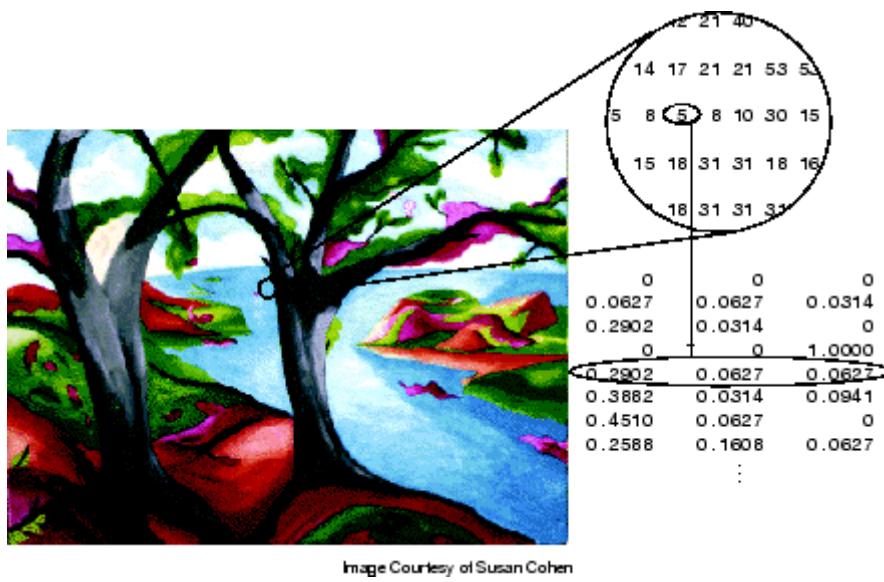
Halbuki bize resim dersinde yeşil yerine sarıyi göstermişlerdi :) Resim dersinde gördüğümüz renkler çıkarımsal (subtractive) renklerdi. Yazıcıda da kullanılan bu renkler cam göbeği (cyan), magenta ve sarıdır.



Artık herhangi bir rengin 3 farklı bileşenin karışımından olduğunu biliyoruz. Şimdi söyle düşünün, siz bana her piksel için ona ait üç değeri değil de sadece bir değer göndermek istiyorsunuz. Benim gelen renkleri çözümleyemem için bir anahtara (eşleştirme haritasına)

sahip olmam lazım ki örneğin bana 3 yolladığınız da turuncu demek istediğinizi bileyim. Yani bana yollayacağınız her farklı değer için bir tane de 3 elemanlı anahtar yollamanız gereklili.

İşte bu anahtara **colormap**, skalar değerlerden oluşan bu matrise de indekslenmiş resim diyoruz. Bu **değerler logical, uint8, uint16, single veya double** tipinden olabilirler. **m** adet farklı değer olduğunu düşünürsek, indekslenmiş resim **single** veya **double** tipinde ise değerler **1 ile m** arasında, diğer tipler içinse değerler **0 ile m-1** arasında olmalı. **colormap** de **mx3'lük**, her satırı bir renge denk düşen **double** bir matris olmalı.



Üstteki örnekte görüldüğü üzere 5 ile ifade edilen renk aslında 5. satırdaki **0.2902xR + 0.0627xG + 0.0627xB**'dır. Şimdi de kendi örneğimize geçelim ve **palyaço (clown)** resmini yükleyelim. X değerini indekslenmiş resim, **map**'i de anahtar olarak kullanalım.

```
s = load('clown')
s =
X: [200x320 double]
map: [81x3 double]
caption: [2x1 char]
```

Dikkat ederseniz resimleri okumak için genelde **imread** kullanıyoruz, fakat indekslenmiş resimlerin renk anahtarına ihtiyacımız olduğu için önceden hem değerleri hem de anahtarları kaydetmişiz ve şimdi onları **load** ile yükliyoruz.

Mesela palyaço resminin (2,5) koordinatında hangi renk olacakmış, bir bakalım. (**Not:** s değişkeni bir nesne ve bu nesneye ait değişkenlere ulaşabilmek için nokta operatörünü kullanıyoruz.)

```
s.X(2,5)
ans =
69
```

Peki 69 ile ifade edilen renk hangisi?

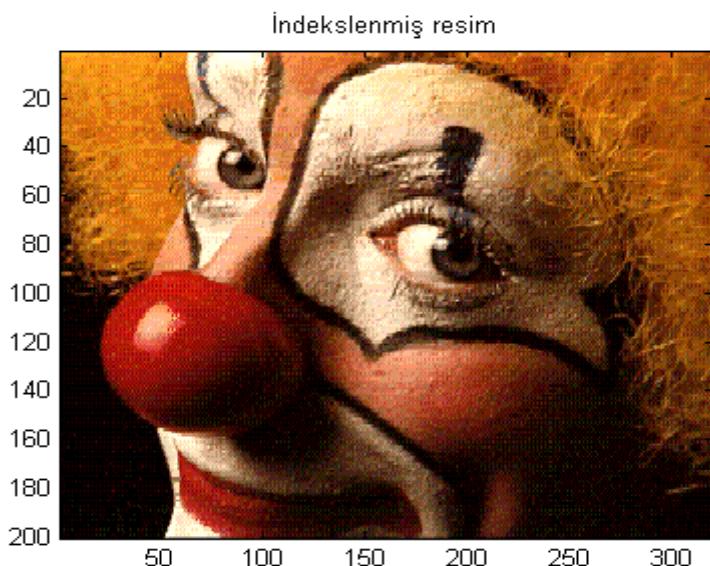
```
s.map(69,:)
ans =
0.9961  0.7031  0.1250
```

O halde bu renk bol miktarda kırmızı ve yeşil, az miktarda da mavi içeriyor. Kaç farklı renk bulunduğuna bakalım:

```
size(s.map,1)
ans =
81
```

İndekslenmiş resimleri göstermek için iki adım kullanacağız.

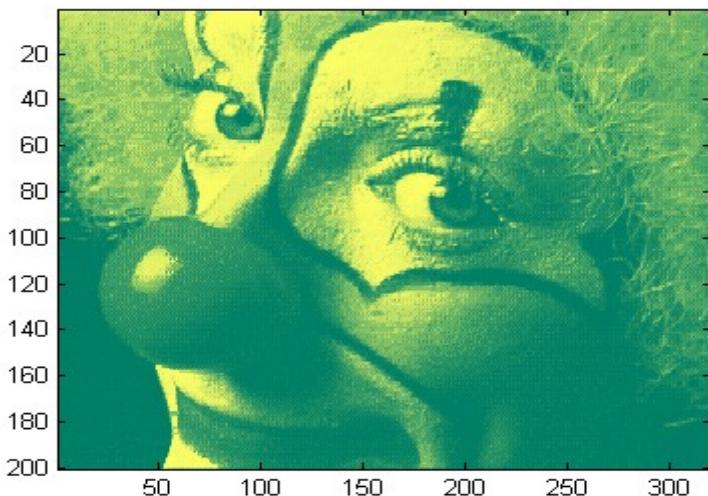
```
image(s.X)
colormap(s.map)
title('İndekslenmiş resim')
```



Başka renk anahtarları kullandık, başka resimler gözleyebilirdik! colormap fonksiyonunun kullanımına bakarsanız önceden tanımdı bazı değerlerle de kullanabilirsiniz. Haydi birisini deneyelim:

```
colormap(summer)
title('"summer" renk anahtarı ile indekslenmiş resim')
```

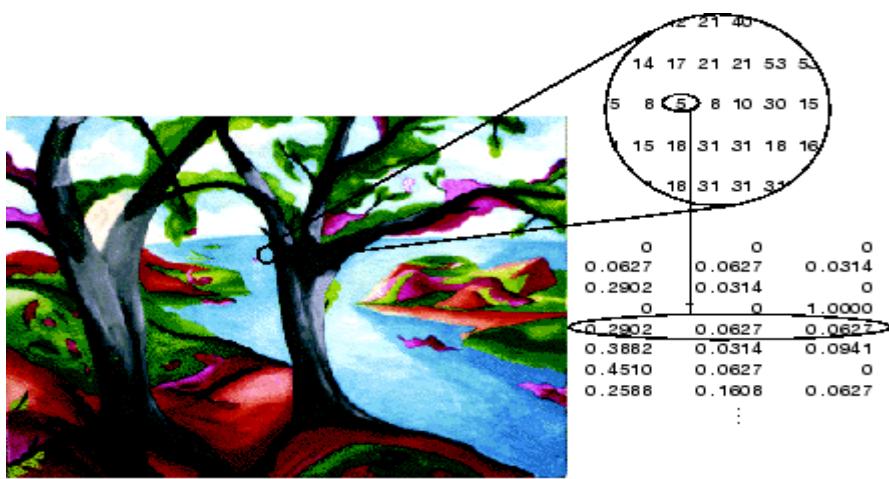
'summer' renk anahtarı ile indekslenmiş resim



Bu alt konuyu bitirmeden önce bu kullanımın faydalarnı düşünelim. Her piksel için 3 elemanlı vektör kullandığımızda çok yer kaplayacaktı. Hele bir de bu değerlerin 8 bitlik olacağını düşünürsek her piksel için 24 bit kullanmalıyız. (**Antiparantez:** *Hani hep duyarız 24 bitlik resim diye: 0-255 arası bileşenler, yani 8 bit kırmızı, 8 bit yeşil, 8 bit mavi. 8 bit de şeffaflık kullandığımız 32 bit olurdu.*) Fakat indekslenmiş resim kullanırsak, örneğin 60 farklı renk varsa  $m=2^n$  ve  $n \geq 6$ ,  $n$  bir tam sayı olmak üzere piksel başına  $m=6$  bit kullanarak işin içinden çıkabiliriz. Tabii ki colormap için kullanacağımız  $mx3$  kadar double hafızasını da gözardı etmemeliyiz.

İşin zor kısmı bitti, şimdi daha aşina olduğunuz gri (aydınlanık) seviyeli resimlere geçelim.

**i.r:** Gri seviye resimler elemanları belli aralıklar arasındaki aydınlanma değerlerini gösteren matrislerdir. **uint8**, **uint16**, **int16**, **single** veya **double** tipi olabilirler. **single** veya **double** tipinde ise değerleri 0 siyahı, 1 beyazı belirtmek üzere 0-1 aralığındaki gerçek sayılardır. Diğer tipler ise **intmin(class(I))** en düşük aydınlanma değerini, yani siyahı, **intmax(class(I))** ise en yüksek aydınlanma değerini, yani beyazı temsil eder. Örneğin **unit8** 8 bitlik gri seviye resmin sınıfıdır ve değerleri 0 ile 255 arasında değişen tamsayılardır.



Yukarıdaki örnekte **double** tipindeki aydınlanma değerlerini gözlemlayabilirsiniz. Şimdi de kendi örneğimizi yapalım. Bunun için de **bozuk paraları (coins.png)** kullanalım.

```
img = imread('coins.png');  
imshow(img)  
title('Gri seviye resim')
```



Resmin tipine bakalım:

```
class(img)  
ans =  
uint8
```

Demek ki, MATLAB gri seviye resimleri uint8 tipinde yükliyor. Ortalardan bir bölüme bakalım:

```
img(51:60,28:33)  
ans =  
58 57 61 113 173 189  
58 57 59 109 171 184  
59 59 57 100 168 180  
59 59 56 89 161 181  
58 59 56 80 157 183  
59 60 57 68 140 184  
58 60 60 59 108 174  
58 60 61 57 84 162  
59 60 60 58 65 139  
58 58 60 61 57 100
```

Görüldüğü üzere değerler tamsayı ve şu anki kısımdan görülmemiği ama bizim bildiğimiz üzere 0 ile 255 arasındalar. (0,255 dahil) Bizim **double** tipinde bir resme ihtiyacımız olsaydı **im2double** ile dönüşümü yapabilirdik. Haydi yapalım:

```
img = im2double(img);
class(img)
ans =
double
img(51:60,28:33)
ans =
0.2275 0.2235 0.2392 0.4431 0.6784 0.7412
0.2275 0.2235 0.2314 0.4275 0.6706 0.7216
0.2314 0.2314 0.2235 0.3922 0.6588 0.7059
0.2314 0.2314 0.2196 0.3490 0.6314 0.7098
0.2275 0.2314 0.2196 0.3137 0.6157 0.7176
0.2314 0.2353 0.2235 0.2667 0.5490 0.7216
0.2275 0.2353 0.2353 0.2314 0.4235 0.6824
0.2275 0.2353 0.2392 0.2235 0.3294 0.6353
0.2314 0.2353 0.2353 0.2275 0.2549 0.5451
0.2275 0.2275 0.2353 0.2392 0.2235 0.3922
```

Değerlerin [0-1] aralığında **double** değerlere dönüştüğü gözüküyor. Aynı şekilde **im2uint8** fonksiyonu ile de ters yönde dönüşüm yapabilirdik. Diğerleri için de yardıma bakabilirsiniz.

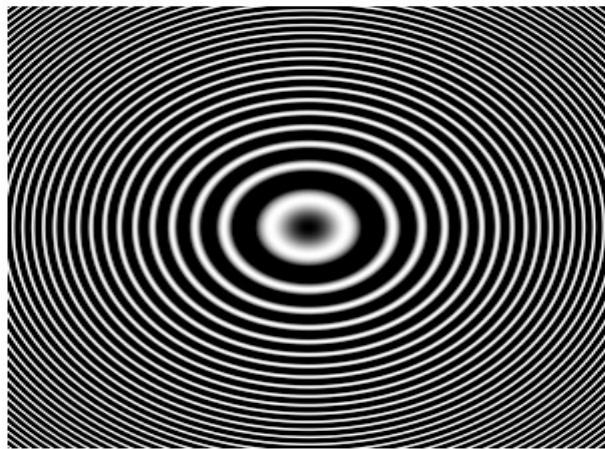
Buradan şöyle bir sonuç çıkarmalıyız. Resimlerde çalışırken hangi tür olduğunun yanısıra elemanlarının sınıfı da (veri tipi) önemlidir. Örneğin bazı fonksiyonlar **uint8** tipi için çalışmaz, o yüzden resimleri okurken hep aynı tipe dönüştürmek iyi bir alışkanlıktır.

Haydi bir tane de örnek resmi biz sentezleyelim. Önce bir ızgara oluşturalım:

```
x = 1:3;
y = 7:10;
[X, Y] = meshgrid(x, y)
X =
1 2 3
1 2 3
1 2 3
1 2 3
Y =
7 7 7
8 8 8
9 9 9
10 10 10
```

Örnekte görüldüğü gibi X ve Y'nin aynı hücredeki elemanlarını yanyana koyarsak x ve y'lerin sıralı kombinasyonları oluşur. Şimdi daha büyük bir ızgara oluşturalım, sonra bir fonksiyon yazalım ve bu ızgarayı tanım kümesi olarak kullanalım.

```
x = linspace(-pi, pi, 300);  
[X, Y] = meshgrid(x);  
A = 10;  
imgSynth = sin(A*(X.^2 + Y.^2));  
imshow(imgSynth)
```



Güzel halkalar çizdik! Çok dağıtmadan veri tipine, en küçük ve en büyük değerlere bakalım ve bitirelim.

```
class(imgSynth)  
ans =  
double  
max(max(imgSynth))  
ans =  
1.0000  
min(min(imgSynth))  
ans =  
-1.0000
```

Artık biliyoruz ki -1 siyah, +1 beyaz, aradaki değerler de gri seviyeler oldular.

#### Gerçek-renkli resimler:

Son olarak gerçek-renkli resimlere değineceğiz. Kırmızı, mavi ve sarıyi oluşturacağımız güzel bir örnekle başlayalım:

```

RPlane = [1 0 1];
GPlane = [0 0 1];
BPlane = [0 1 0];
rgbImg = cat(3, RPlane, GPlane, BPlane);
size(rgbImg)
ans =
1 3 3
imagesc(rgbImg)
axis image
title('Kırmızı, mavi ve sarı renkler')

```



**cat** fonksiyonu ile **1x3**'luk üç vektörü arka arkaya birleştirip **1x3x3**'luk yeni bir değişken oluşturduk. Evet, gerçek-renkli resimler **mxnx3**'luk dizilerden başka birşey değil.

Bunu (R,G ve B'nin oranlarını ifade eden) üç gri seviye resmin arka arkaya konulması olarak düşününebiliriz.

Örnekte olduğu gibi sarı rengi kırmızı ve yeşilin karışımından elde ederiz.

(*Not: Bu arada piksellerin orta noktalarının tam değerlere geldiğine dikkat edin! Ayrıca **imshow**, **image** ve **imagesc** arasındaki farkları yardımdan öğrenin.*)

Gerçek-renkli resimlerle çalışırken renk anahtarları bir işe yaramaz!

```

colormap(winter)
title('Yine kırmızı, yine mavi, yine sarı!')

```



Şimdi de alttaki örneğe bakarak gerçek bir resimde bahsettiğimizin neye tekabül ettiğini görelim:



Artık kendi örneğimize geçmenin vakti geldi. Bu sefer MATLAB dizinindekilerden değil de bize ait bir resmi kullanalım. Bu resim şu an çalıştığımız dizinin içindeki **resimler** dizininde bulunsun ve adı **ornekresim.jpg** olsun.

```
rgbImg = imread('resimler/ornekresim.jpg');
imshow(rgbImg)
```



```
size(rgbImg)
ans =
385 385 3
```

Harika, bize ait bir resmi MATLAB'e aktardık sonunda! Açılan pencerede alttaki gibi '**data cursor**' basılı iken pikselleri seçip koordinatlarına ve renk bileşenlerine bakabilirsiniz.

### Gri Seviye Resimlerle İşlemler

Resimlere ve piksel renklerine ait temelleri öğrendik, dahasını getirelim, biraz da basit işlem yapıp konuyu sonlandıralım. Öncelikle elimizdeki gerçek-renkli resmi griye çevirelim:

```
img = rgb2gray(rgbImg);
imshow(img)
truesize
title('Örnek resim')
```

Örnek resim



```
size(img)
```

```
ans =
```

```
385 385
```

Artık üç kanaldan değil de tek kanaldan oluşuyor resmimiz. Resim ekrana basılırken **truesize** kullandık ki her resim pikseli ekrandaki bir piksele denk gelsin. Mesela ekrandaki **0.5x0.5** piksellik bölgeye denk gelmesi için şöyle de yapabilirdik:

```
imshow(img)
```

```
truesize(0.5*size(img))
```

```
title('Ekrandaki bir piksele dört resim pikseli geldi')
```

Ekrandaki bir piksele dört resim pikseli geldi



Dikkat edin, resmi ölçeklemedik, sadece ekran görüntüsünü küçülttük. Şimdi de resmi ölçekleyelim. Bunu yaparken de '**bilinear**' interpolasyon kullanalım.

```
imgRescaled = imresize(img, 0.75, 'bil');  
imshow(imgRescaled)  
title('0.75 oranda ölçeklenmiş resim')
```

0.75 oranda ölçeklenmiş resim



```
imgRescaled = imresize(img, [100 150], 'bil');  
imshow(imgRescaled)  
title('100x150 piksele ölçeklenmiş resim')
```

100x150 piksele ölçeklenmiş resim



Resmi döndürseydik, sonra da orijinal boyutuna kırpıştık:

```
imgRotated = imrotate(img, 30, 'bil', 'crop');  
imshow(imgRotated)  
title('30 derece döndürüüp orijinal boyutuna kırpılmış resim')
```

30 derece döndürülüp orijinal boyutuna kırpılmış resim



Geometrik dönüşümleri başka zaman daha uzun konuşuruz. Şimdi matematiksel birkaç işlem uygulayalım resme. Ama bundan önce veri tipine bakalım:

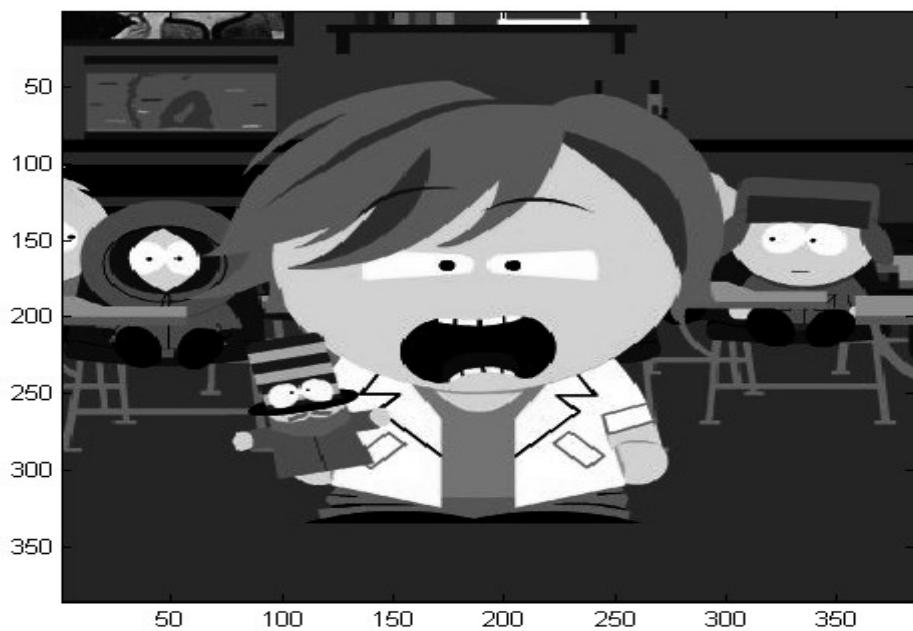
```
class(img)
ans =
uint8
```

Uygulayacağımız işlemler double tipinde veri istiyor, o halde dönüşümü yapalım.

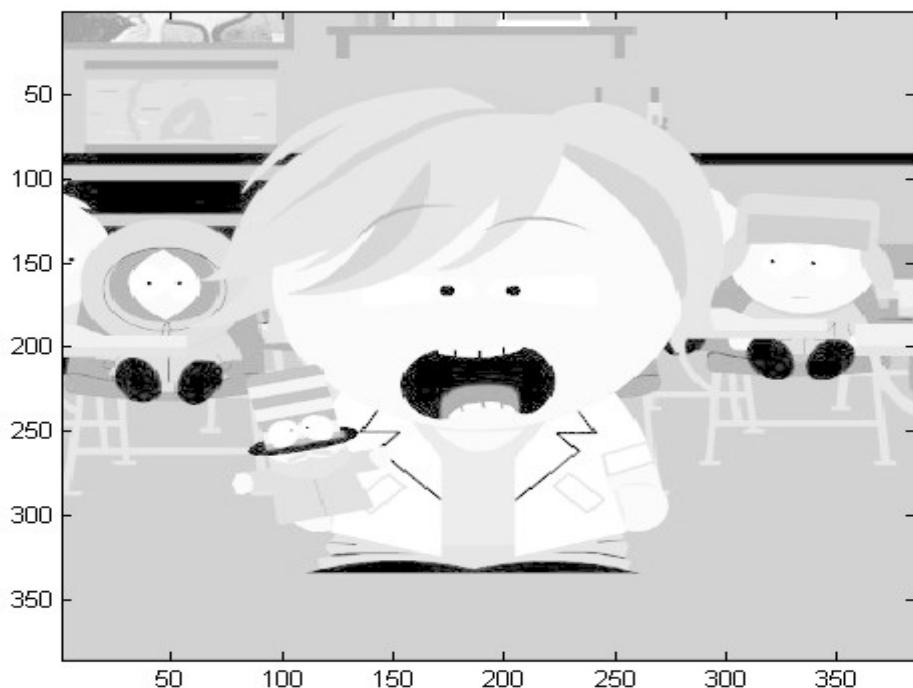
```
img = im2double(img);
class(img)
ans =
double
```

Şimdi bu resim için hiçbir anlam ifade etmeyen birkaç işlemin sonucunu gözlemleyelim:

```
imagesc(img.^2)
```



```
imagesc(log(img))
```



Son olarak anlamlı bir işlem yapalım ve resmi  $5 \times 5$ 'lik bir Gauss çekirdeği ile filtreleyelim, sonra da yine **resimler** dizinine kaydedelim. Önce çekirdeği oluşturalım:

```
gaussKernel = [1 4 7 4 1]' * [1 4 7 4 1] / 273  
gaussKernel =  
0.0037 0.0147 0.0256 0.0147 0.0037  
0.0147 0.0586 0.1026 0.0586 0.0147  
0.0256 0.1026 0.1795 0.1026 0.0256  
0.0147 0.0586 0.1026 0.0586 0.0147  
0.0037 0.0147 0.0256 0.0147 0.0037
```

Şimdi bu çekirdeği kullanarak resmi bulandırılm, yani Gauss ile evrişimini (konvolusyonunu) alalım. Sonuç, orijinalle aynı boyutta olsun:

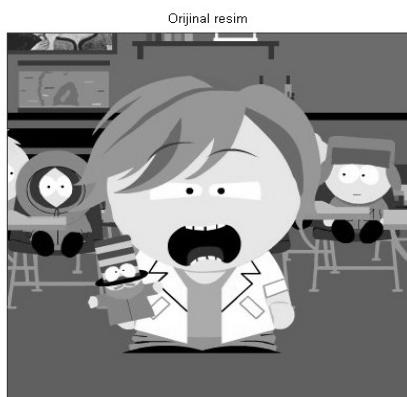
```
imgBlurred = conv2(img, gaussKernel, 'same');
```

Bu resmi png formatında diske yazalım:

```
imwrite(imgBlurred, 'resimler/yeniresim.png')
```

Son olarak orijinal resmi ve filtreleme sonucunu yan yana çizelim ve bu çizimi diske kaydedelim.

```
subplot(1, 2, 1),  
imshow(img), title('Orjinal resim')  
subplot(1, 2, 2),  
imshow(imgBlurred), title('Filtreleme sonucu'), truesize  
saveas(gcf, 'resimler/ikiResimYanyana.jpg')
```



## Matlab da Dosya İşlemleri

Şu ana kadar yazdığımız bütün **MATLAB** programlarında, programların kullanacakları verileri ya input komutu yardımıyla ya da fonksiyonlara parametre olarak kullanıcıya parametre olarak kullanıcının klavye aracılığıyla doğrudan alıyordu. Bu giriş verileriyle programlarımızın ürettiği çıktılar ise bilgisayarın kalıcı olmayan hafızasında bir başka deyişle ana belleğinde tutuluyordu. Yani, bilgisayar kapatıldığında veya **MATLAB** programı sona erdirildiğinde bu verilerde kayboluyordu. Oysa pek çok uygulamada hem giriş verilerinin hem de program çıktılarının daha sonra tekrar tekrar kullanılmak üzere bilgisayarımızın kalıcı hafızasında yani sabit diskinde saklanması gereklidir. Bunun için bütün programlama dilleri söz konusu verileri bir dosyadan okuyabilmeye veya çıktıları bir dosyaya kaydedebilmeye imkan tanıyan yapılar içerir.

### Üç Adımda Dosya Yöntemi:

- ❖ İşleyeceğin dosyayı aç.(**fopen**)
- ❖ Açıığın dosyadan veri oku ya da açtığı dosyaya veri yaz.(**fscanf**, **textscan**, **fprintf**)
- ❖ Açıığın dosyayı muhakkak kapat(**fclose**)

### fopen Komutuyla Dosya Açılması

MATLAB'ın fopen komutu, veri kaydetmek veya veri okumak için, var olan bir veri dosyasını yazmaya/okumaya açar veya var olmayan bir dosyayı sıfırdan oluşturur. Kullanım formatı aşağıdaki gibidir:

**DosyaDeğişkeni=fopen('dosyaAdı', 'mod');**

Burada:

- ❖ **DosyaDeğişkeni** >> Dosya kontrolünde kullanılan MATLAB değişkeni
- ❖ **dosyaAdı** >> Verilerin kaydedileceği(okunacağı) dosyanın adı
- ❖ **mod** >> Dosya işleme modunu: r, w, a, ... vs. ifade eder.

### Bazı Dosya İşleme Modları

Mod	İşlev
'r'	Dosyayı sadece okumaya açar.(read)
'w'	Var olan bir veri dosyasının içeriğini siler, dosya yoksa oluşturur ve dosyayı yazmaya açar.(write)
'a'	Var olan bir veri dosyasını yazmak için açar, dosya yoksa oluşturur ve girilecek bilgileri dosyanın sonuna ekler. (append) Mevcut dosyanın içerisinde var olan bilgiler silinmezler. Yeni bilgiler önceki bilgilerin sona erdiği noktadan itibaren dosyaya kaydedilirler.

### Fclose Komutu ile Dosyaların Kapatılması

MATLAB in **fclose** komutu yazma ve/veya okumaya açılan dosyanın MATLAB ile ilişkisini kesmeye (dosyayı kapatmaya) yarayan komuttur. (*açılan bütün dosyalar muhakkak kapatılmalıdır.*)

**Fclose** komutunun kullanım formatı aşağıdaki gibidir:

**fclose(dosyaDeğişkeni);**

Burada **dosyaDeğişkeni**, bir veri dosyasının işlenmek üzere **fopen** komutu ile açıldığı an ilişkilendirildiği MATLAB değişkenidir.

### **Fprintf Komutu ile Bir Dosyaya Yazma**

Bilgilerin ekrana basılmasında kullanılan **fprintf** fonksiyonu, aynı zamanda **fopen** komutuyla açılmış bir dosyaya veri yazmak için de kullanılır.

**Fprintf** komutunun bir dosyaya bilgi kaydetme amacıyla kullanım formatı aşağıdaki gibidir:

**fprintf(dosyaDeğişkeni, '%format %format',değişken1,değişken2);**

Bu yeni **fprintf** in daha önce öğrendiğimiz ve ekrana bilgi yansıtmak amacıyla sıkılıkla kullandığımız **fprintf** komutunun kullanımından tek farkı **dosyaDeğişkeni** olarak adlandırdığımız **MATLAB** değişkenin fonksiyon içerisinde ilk parametre olarak kullanılmasıdır.

### **Fscanf Komutu ile Bir Dosyadan Bilgi Okuma**

MATLAB in fscanf komutu formatlanmış veriler içeren bir dosyadan bilgi okumaya yardımcı olur. Fscanf komutunun bir dosyadan bilgi okuma amacıyla en genel kullanım formatı aşağıdaki gibidir:

**[değişken sayı]=fscanf(dosyaDeğişkeni,'format',adet);**

Burada **adet** ile ifade edilen şey dosyadan okunması istenilen bilginin miktarıdır. Format ise dosyadan okunacak verinin türüdür. Eğer dosyadaki verilerin tamamının okunması isteniyorsa **adet** yerine **MATLAB** da tanımlı olan özel bir değişken olan **inf(infinity)** kullanılmalıdır. Böyle bir durumda dosyadan okunan bilginin miktarına da **sayı değişkeni** üzerinden ulaşılabilir. Dosyadan okunan bilgiler ise **değişken** içerisinde saklanır ve bu değişkene dosyadan aktarılan bilgiler **sütun vektörü** formundadır.

### **Textscan Komutu ile Bir Dosyadan Bilgi Okuma**

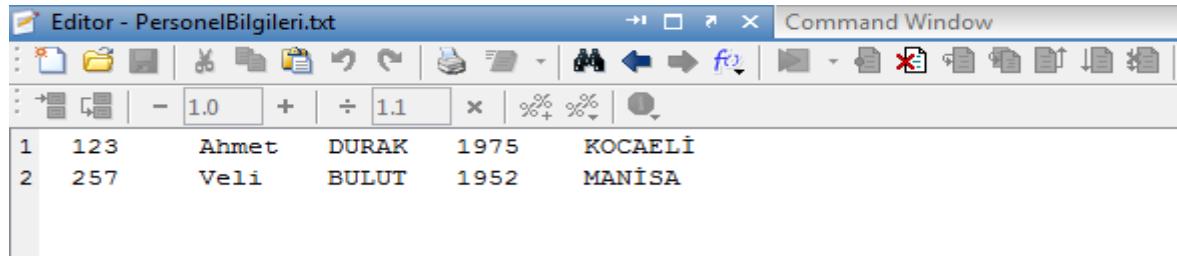
MATLAB in textscan komutu bir dosya içerisinde belirli bir format çerçevesinde (dosyanın herbir satırı birden fazla farklı veri tipinde sütunlardan oluşduğunda) saklanan verileri/satırları/kayıtları bir hücre dizisine atamak için kullanılır.

Textscan komutunun bir dosyadan bilgi okuma amacıyla en genel kullanım formatı aşağıdaki gibidir:

**hcreDizisi=textscan(dosyaDeğişkeni,'format');**

Bu kullanımda MATLAB'ın varsayılan sütun ayrıcısı(delimiter character) boşluk ve yeni satır karakteridir. Eğer dosyanın her bir satırındaki sütunlar başka bir ayraç karakteri ile birbirinden ayrılmışlarsa kullanım formatı aşağıdaki şekilde dönüşür.

```
hucreDizisi=textscan(dosyaDeğişkeni, 'format', 'delimiter', 'ayracKarakteri');
```



**PersonelBilgileri.txt** adlı birUBYAZI dosyasının içerisinde yukarıdaki ekran çıktısı ile verilen bilgilerin saklı olduğunu kabul edelim ve aşağıdaki verilen komutları çalıştırarak textscan komutunun nasıl çalıştığını anlamaya çalışalım.

```
>> dosyaDegiskeni=fopen('PersonelBilgileri.txt','r');
>> hucreDizisi=textscan(dosyaDegiskeni, '%d %s %s %d %s');
>> fclose(dosyaDegiskeni);
>> hucreDizisi{1}
ans =
    123
    257
>> hucreDizisi{1}(2)
ans =
    257
>> hucreDizisi{5}
ans =
    'KOCAELİ'
    'MANİSA'
>> hucreDizisi{5}(1)
ans =
    'KOCAELİ'
```

Dosyanın 1. ve 4. Sütunlarında tam sayı değerler saklı olduğu için textscan komutunda bu sütunlara karşılık gelen formatlar olarak %d ve diğer sütunlarda string değerler saklandığı için bu sütunlara karşılık gelen formatlar olarak %s kullanılmıştır. Textscan ile dosya okuma işlemi gerçekleştirildikten sonra okunan bilgilerin içerisinde saklandığı hücre dizisinin ilk indisinde dosyanın 1. Sütundaki tüm bilgilerin ve benzer şekilde diğer indislerinde de ilgili sütunlara ait bilgilerin saklandığı gözlemlenmiştir. **hucreDizisi{1}** deyimi ile ilk sütun bilgilerini saklayan diziye **hucreDizisi{1}(1)** deyimi ile de ilk sütundaki tam sayı değere ulaşılmıştır.

## Matlab ile Excel Arasındaki İşlemler

Herhangi bir bilgisayar programının gerçek anlamda bir işlev görebilmesi veriler ile mümkündür. Bu yüzden profesyonel uygulamalar veri tabanı adı verilen sunucular ile birlikte çalışır. Excel her bir veri tabanı için bir geçiş noktası olabileceğinden çok önemli bir yer tutar.

### **Excel Dosyası Hakkında Bilgi Alma**

Matlab da **xlsinfo fonksiyonu** bir excel belgesinin durumu hakkında bilgi almak için kullanılır. 3 farklı kullanım şekli vardır.

**[durum] = xlsinfo ('belge\_adi.uzanti')**: Biçimindeki verilen belgenin varsayılan klasörde olup olmadığı kontrol edilir ve durum değişkeni altında bildirilir.

**[durum, sekme] = xlsinfo('belge\_adi.uzanti')**: Biçimindeki verilen belgenin varsayılan klasörde olup olmadığı ve belgenin sekme isimleri kontrol edilir sekme değişkeninde saklanır.

**[durum, sekme, format] = xlsinfo('belge\_adi.uzanti')**: Biçiminde verilen belgenin varsayılan klasörde olup olmadığı, belgenin sekme isimleri ve belgenin formatı kontrol edilir. Belgenin formatı **format** değişkeninde saklanır.

```
[durum,sekme,format] = xlsinfo('deneme.xlsx')
durum =
Microsoft Excel Spreadsheet
sekme =
'Sayfa1'      'Sayfa2'      'Sayfa3'      'alper'
format =
xlOpenXMLWorkbook
```

### **Excel Belgesinden Veri Okuma**

Matlab da bir excel belgesinden veri okumak için **xlsread fonksiyonu** kullanılır. Bu fonksiyonun çok çeşitli kullanımı vardır. Bazıları şu şekildedir.

**num = xlsread('belge\_adi.uzantısı')**: Bu kullanımda verilen belgenin içeriğindeki ilk sekmenin nümerik (sayısal) değerleri **num** matrisine aktarılır.

**[num,str] = xlsread('belge\_adi.uzantısı')**: Bu kullanımda verilen belgenin içeriğindeki ilk sekmenin nümerik (sayısal) değerleri **num** matrisine, **string** değerleri **str** matrisine aktarılır.

**[num,str,tum] = xlsread('belge\_adi.uzantısı')** : Bu kullanımda verilen belgenin içeriğindeki ilk sekmenin nümerik (sayısal) değerleri **num** matrisine, **string** değerleri **str** matrisine ve tüm matris **tum** matrisine aktarılır.

```
[num,str,tum] = xlsread('deneme.xlsx')
num =
    1 2
    3 4
str =
    'a' 'b'
tum =
    'a' 'b'
    [1] [2]
    [3] [4]
```

**num = xlsread('belge\_adi.uzantisi','sekme\_adi')** : Komutu istenilen sekmenin içeriği okunabilir. Burada sekme adları kullanılabileceği gibi sekme sırası da kullanılabilir

```
num = xlsread('deneme.xlsx','alper')
veya
num = xlsread('deneme.xlsx',4)
num =
    1 2 3 4 5 6 7
```

**num = xlsread('belge\_adi.uzantisi','sekme\_adi','aralik')** : Komutu istenilen sekmenin, istenilen hücrelerinin okunması anlamına gelir.

```
num = xlsread('deneme.xlsx','alper','A7:C9')
num =
    1 2 3
    2 2 2
    3 4 4
```

**num = xlsread('belge\_adi.uzantisi',-1)** : Komutu MATLAB in excel belgesini açarak kullanıcının istediği veriyi seçmesini sağlar.

```
num = xlsread('deneme.xlsx',-1)
num =
    4 5 6
```

## Excel Belgesine Veri Yazma

Matlab da bir excel belgesi oluşturmak için **xlswrite fonksiyonu** kullanılır.

Bu fonksiyon istenilen belge var ise üzerine yazar yok ise oluşturur. Farklı kullanımları vardır.

**xlswrite('belge\_adi.uzantisi',A)** : Kullanımında A verisi istenilen belgenin ilk sekmesinde A1 hücreinden itibaren yazılır.

```
A = [11,12,13];
xlswrite('yaz.xls',A)
```

**xlswrite('belge\_adi.uzantisi',A,'sekme')** : Kullanımında A verisi istenilen belgenin, istenilen sekmesine A1 hücreinden itibaren yazılır.

```
A = [11,12,13];
xlswrite('yaz.xls',A,'Sayfa2')
```

**xlswrite('belge\_adi.uzantisi',A,'sekme','aralik')** : Kullanımında A verisi istenilen belgenin, istenilen sekmesine, istenilen aralığa yazılır. Eğer veri aralıkta büyük ise kalan kısmı yazılmaz.

```
A = [11,12,13];
xlswrite('yaz.xls',A,'Sayfa2','F2:H12')
```

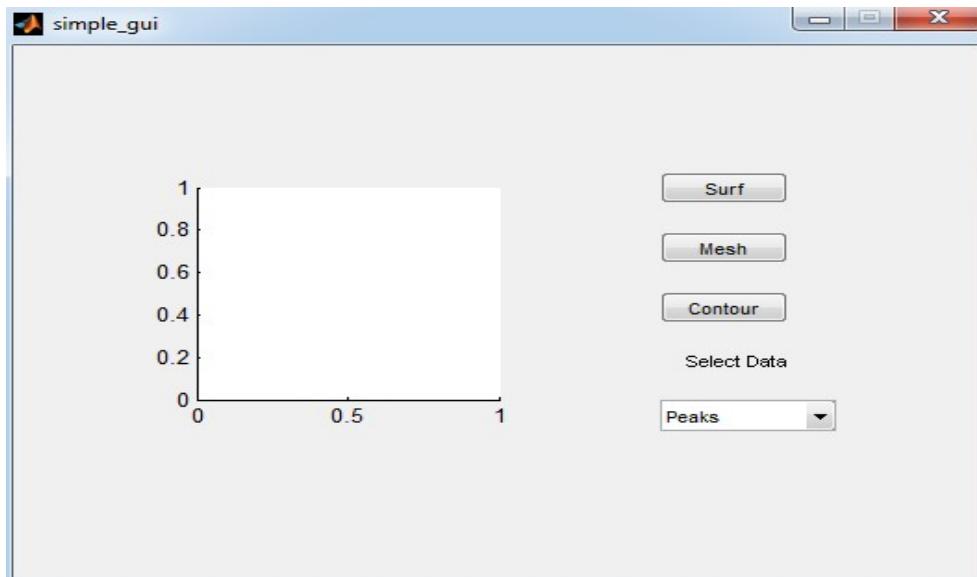
**[drm,msj] =xlswrite('belge\_adi.uzantisi',A)** : Komutu işlem başarılı ise **drm** değişkenine 1 değerini atar **msj** değişkenine bir şey atamaz. İşlem başarılı değil ise **drm** değişkenine 0 atar **msj** değişkenine başarısızlığın nedenini yazar.

```
A = [11,12,13];
[drm,msj] = xlswrite('yaz.xls',A,'Sayfa','F2:H12')
drm = 0
msj =
message: [1x94 char]
identifier: 'MATLAB:xlswrite:LockedFile'
```

İçeriğinde yer alan nesnelerin kullanılması ile kullanıcıya etkileşim sağlayan ve bir işin veya bir programın koşturulmasını sağlayan grafiksel bir program arayüzüdür. Açılmış Graphical User Interface (GUI) dir.

GUI nesneleri menüler, araç çubukları, radio butonlar, liste kutuları veya kaydırıcılar olabilir. Bunların yanında MATLAB GUI ile MATLAB'ın sunduğu hesaplama imkânları kullanılarak da veri alımı ve grafik çizimi gibi pek çok işlem gerçekleştirilebilir.

Aşağıdaki şekilde basitçe bir GUI arayüzü görülmektedir.



### Grafiksel Kullanıcı Arabirimini (GUI) Nasıl Çalışır?

Her bir nesne (veya komponent) GUI için tanımlanan programlama dosyasında **callback** diye adlandırılan ayrı alt rutin programlama parçalarına sahiptir. Bu şekilde her bir nesnede oluşan olaylara (örnek olarak bir buton nesnesinin tıklanması ile **click event** oluşması gibi) GUI o olaya ait **callback** rutinlerini icra ettirir. Yani, GUI hem bir **arayüz** hem de bir program çağrılarını icra ettirme mekanizması olarak çalışır.

Yukarıda bahsedilen programlama olay tabanlı programlama diye adlandırılır. Bu tür programlamada her bir olaylara ait alt program parçaları birbirinden bağımsız olarak **MATLAB GUI** tarafından çalıştırılır.

### MATLAB da GUI Oluşturma Yöntemleri

MATLAB GUI tasarımları üç ayrı yöntem kullanılarak yapılabilir. Bunlar,

- ❖ **M-File programlama yöntemi kullanılarak**
- ❖ **MATLAB GUIDE aracı kullanılarak,**
- ❖ **App Designer**

Özellikle GUI tasarımda hızlı arayüzler dizayn etmek ve bu işe ilk başlayan programcılar için **MATLAB GUIDE** aracının kullanılması büyük bir kolaylık sağlar. Bu aracın kullanılması ile GUI arabirimini kolaylıkla ve yorulmadan sürükle bırak ve açılan pencerelerde

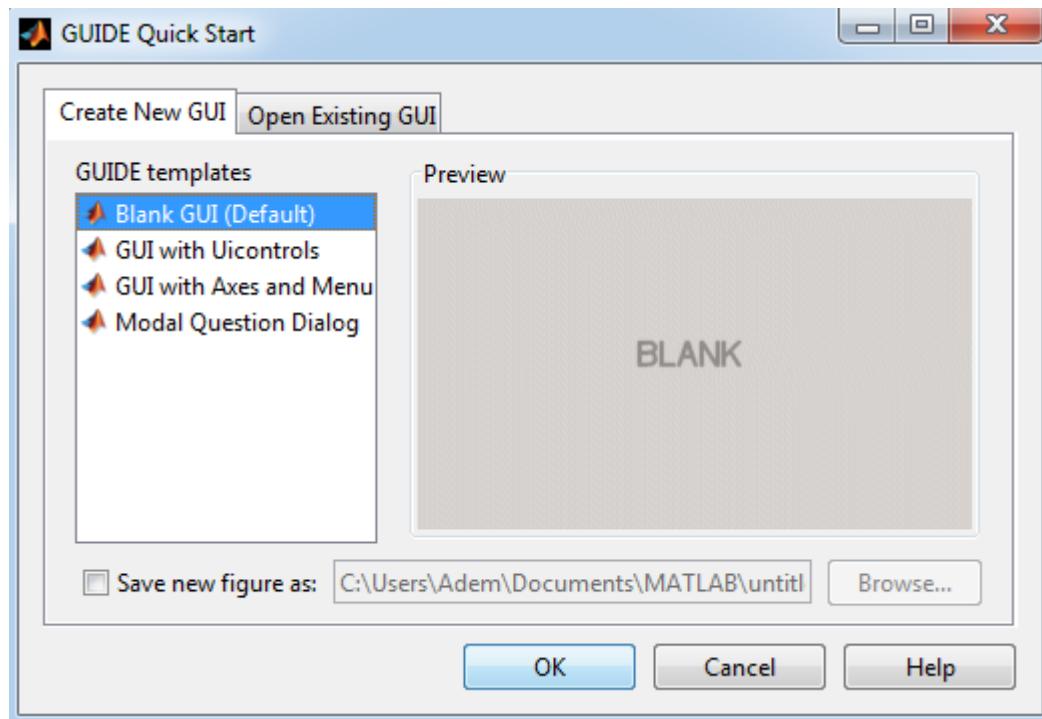
özelliklerin değiştirilmesine dayanan bir yöntem kullanılır. Ayrıca, bu yöntemi kullanmanın ileride var olan bir GUI nin düzenlenmesi ve değişiklik yapılması bakımından da çok yararlıdır.

M-File programlama yönteminde tüm GUI tasarımları ve **callback** program parçalarının yazılması tamami ile programlama kodları kullanılarak yapılır. Burada tasarımcı her şeye hakimdir ve bu teknik uzman bir programlama bilgisi gerektirir. Bu yöntem ile tasarım zamanı uzamasına rağmen programcı her türlü manipülasyonu yapabildiği için programcı açısından çok yararlıdır.

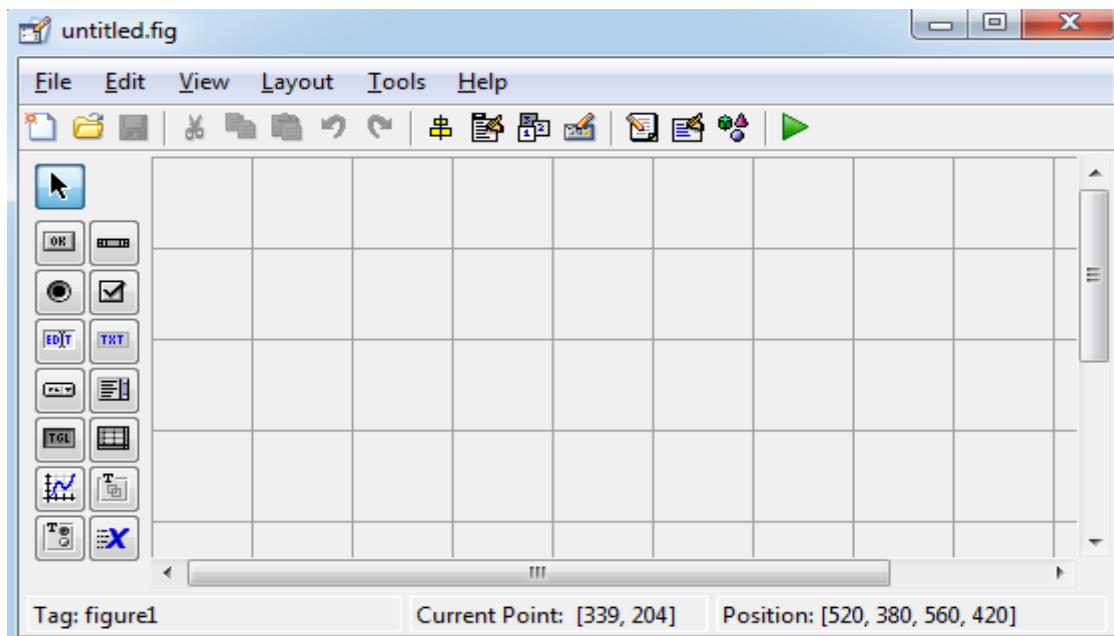
## MATLAB GUIDE ARACI İLE GUI TASARIMI OLUŞTURMA

GUIDE MATLAB in GUI tasarımcılarına sunduğu içerisinde çeşitli araçlar içeren ve kolaylık sağlayan bir grafiksel GUI geliştirme ortamıdır. GUIDE kullanılarak tıkla ve sürükle-bırak tekniği ile GUI arayüzüne nesneler (örneğin butonlar, text kutuları, liste kutuları, grafikler v.s.) kolaylıkla eklenebilir. Ayrıca, eklenen nesnelerin hizalanması, tab sırasının değiştirilmesi, görsel ayarlar üzerinde manipülasyonlar yapılması da bu ortamın tasarımcılara sunduğu imkânlardan bazlıdır.

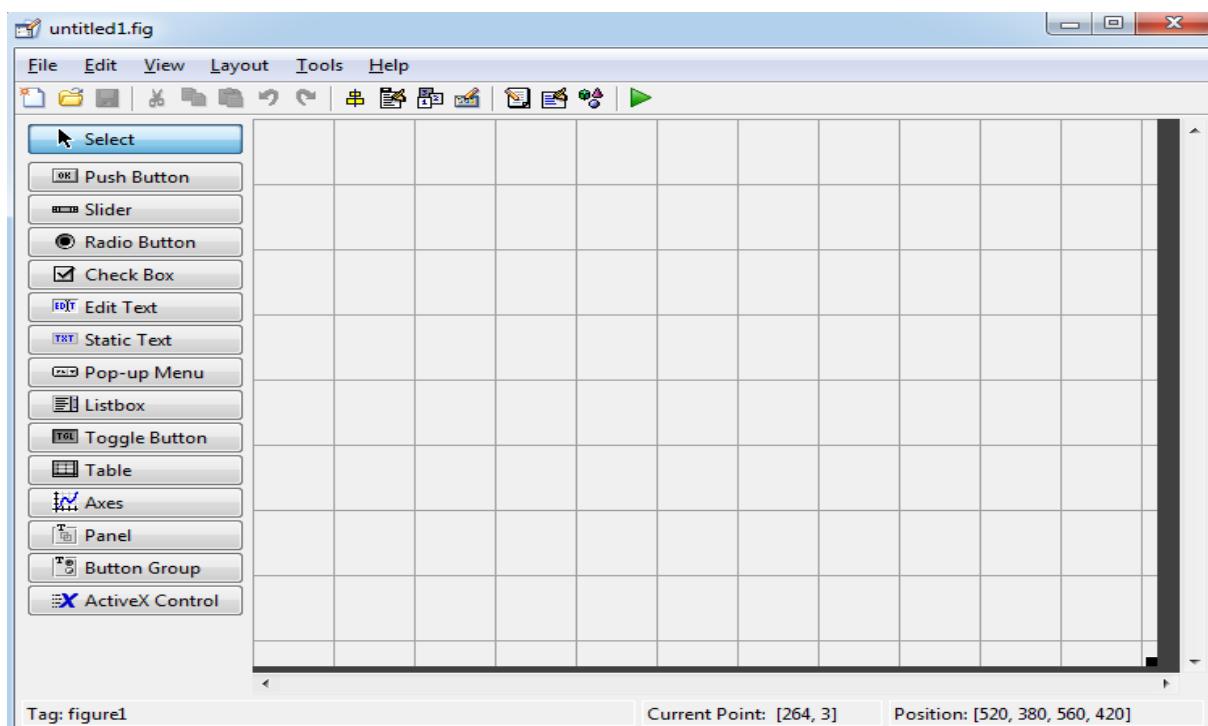
**MATLAB GUIDE** aracını tanıyalım. Bu aracını çalıştmak için ya MATLAB komut satırından **guide komutu** verilir ya da **Start düğmesi** tıklanarak **MATLAB/GUIDE** komutu verilir. Bu adımdan sonra karşımıza aşağıdaki gibi bir pencere gelir.



Bu pencereden eğer yeni bir **GUI tasarımlı** yapacak ise **Blank GUI** seçeneğini seçeriz. Şayet önceden yapılmış bir tasarım açmak istiyor ise **Open Existing GUI** sekmesinden sonra istenilen dosyayı seçeriz. Burada yeni bir tasarım oluşturulacağını kabul edelim. Bundan sonra **OK düğmesi** tıklayarak aşağıdaki gibi bir **GUIDE LAYOUT Editor (GUIDE Çalışma Alanı)** penceresine ulaşırız.

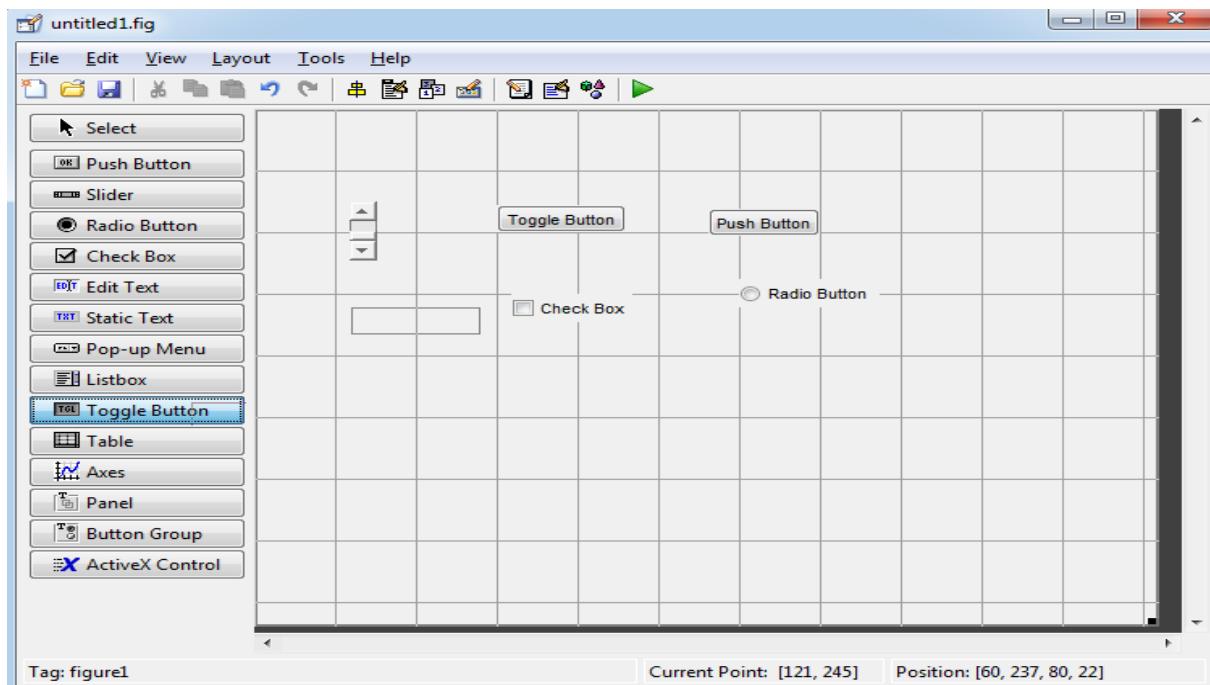


Bu adımdan sonra **File/Prefences/Guide** yolunu kullanılarak gelen pencereden “**Show names in component palette**” seçeneğini tıklayıp **OK** düğmesine basalım. Karşımıza aşağıdaki gibi bir pencere gelecektir.



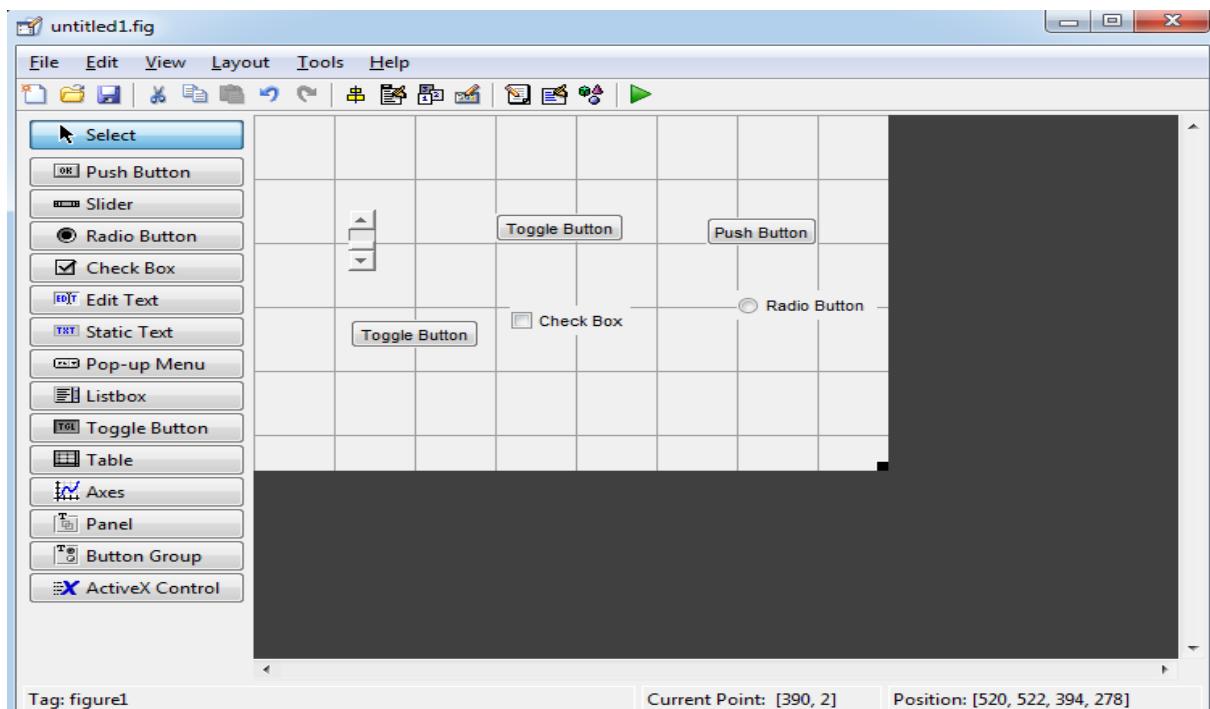
### Komponentleri Çalışma Alanına Ekleme

Bunun için sol tarafta bulunan nesne butonlarından istenilen nesneye ait buton tıklanır ve daha sonra çalışma alanında uygun görülen bir noktaya tıklandığında o noktaya ilgili nesne eklenmiş olacaktır. İstenirse çalışma alanındaki bir nesne farenin sol tuşu ile tıklanıp bırakılmadan çalışma alanının herhangi bir yerine sürüklenebilir. Bu durum aşağıdaki şekilde görülmektedir.



## Çalışma Alanının Boyutlarını Değiştirmek

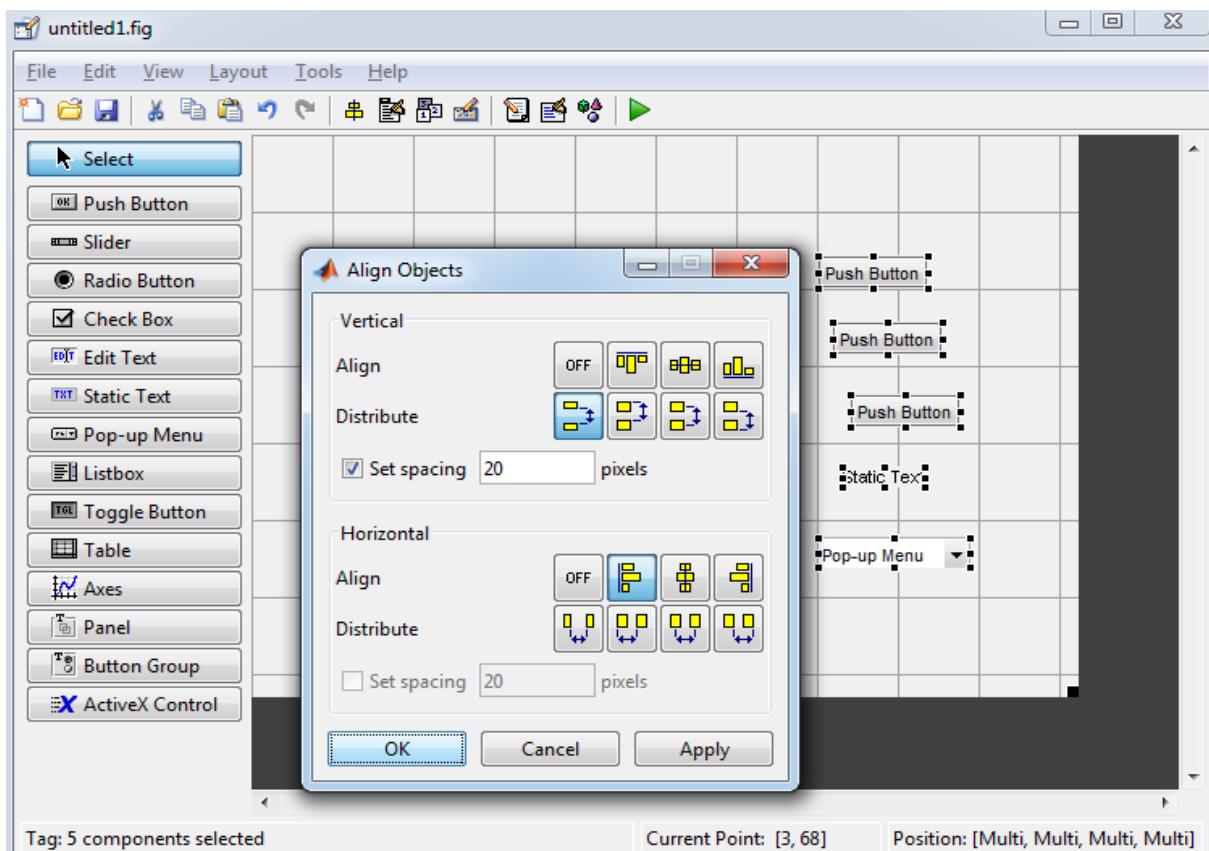
Burada çalışma alanının sağ alt tarafında bulunan siyah karenin üzerine fare işaretçisi getirilir ve fare işaretçisi konum değiştiğinde farenin sol tuşu basılı tutularak çalışma alanı istenilen boyutlarda olacak şekilde düzenleme yapılabilir.



## Nesneleri Hizalamak

Bu işlemi yapmak için öncelikle hizalanacak nesneler seçilir. Topluca seçmek için çalışma alanında fare işaretçisini herhangi bir yere tıklayıp sürükleyerek açılan kesik kenarlı pencerenin içinde nesneler kalacak şekilde hareket ettirip, hizalanacak nesneler bu çerçeve

içinde kalınca farenin sol tuşunu bırakın. Bu şekilde sadece o çerçeveye içinde kalan nesneler seçilmiş olacaktır. Ayrıca, nesneleri **Ctrl tuşunu** basılı tutarak farenin sol tuşu ile teker teker de seçme imkânı bulunmaktadır. Hizalanacak nesneler seçildikten sonra **Tools/Align Objects...** yolunu kullanarak **Alignment Tool (Hizalama Aracı)** penceresini açınız. Aşağıdaki şekildeki gibi bir ekran ile karşılaşırız. Burada yatay ve dikey hizalamaları kendimize göre butonlardan seçip **OK butonuna** bastığımız zaman nesnelerimiz hizalanmış olacaktır. Eğer ki hizalama istenilen gibi olmadı ise **Ctrl + Z kısayolu** ile yapılan işlemler geri alınabilir.

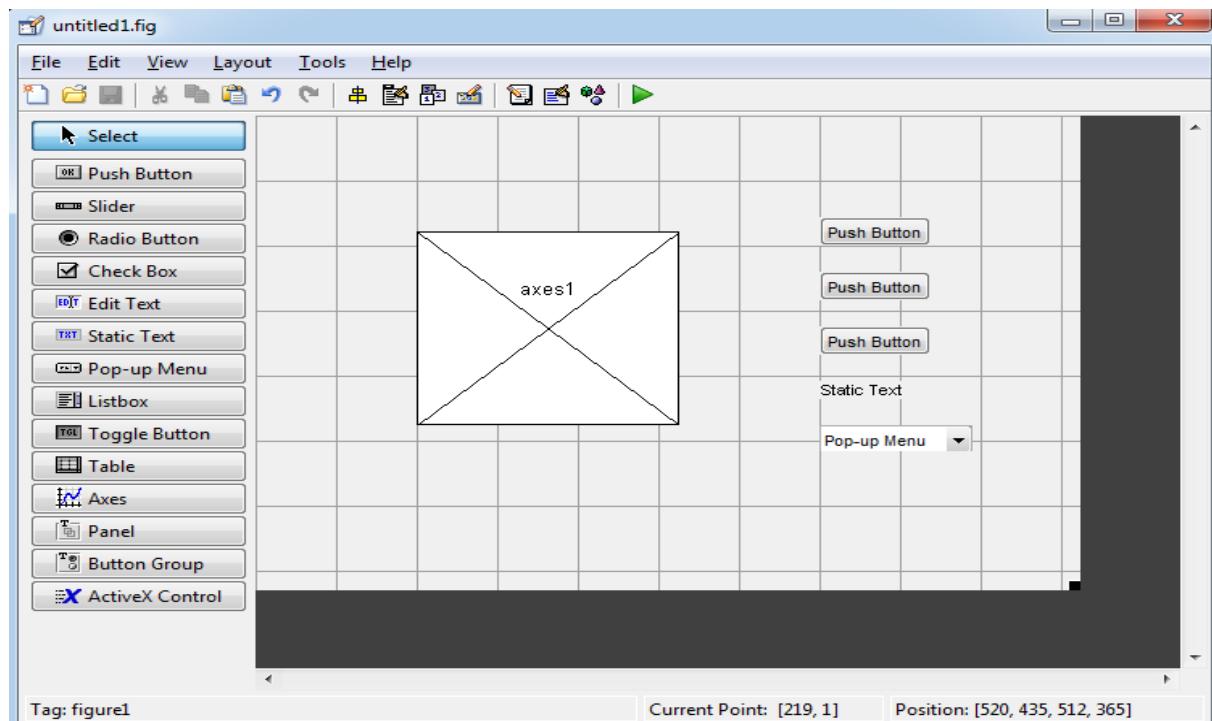


Burada aşağıdaki gibi bir GUI hazırlanmış oluruz.

Burada GUI arayüzünde

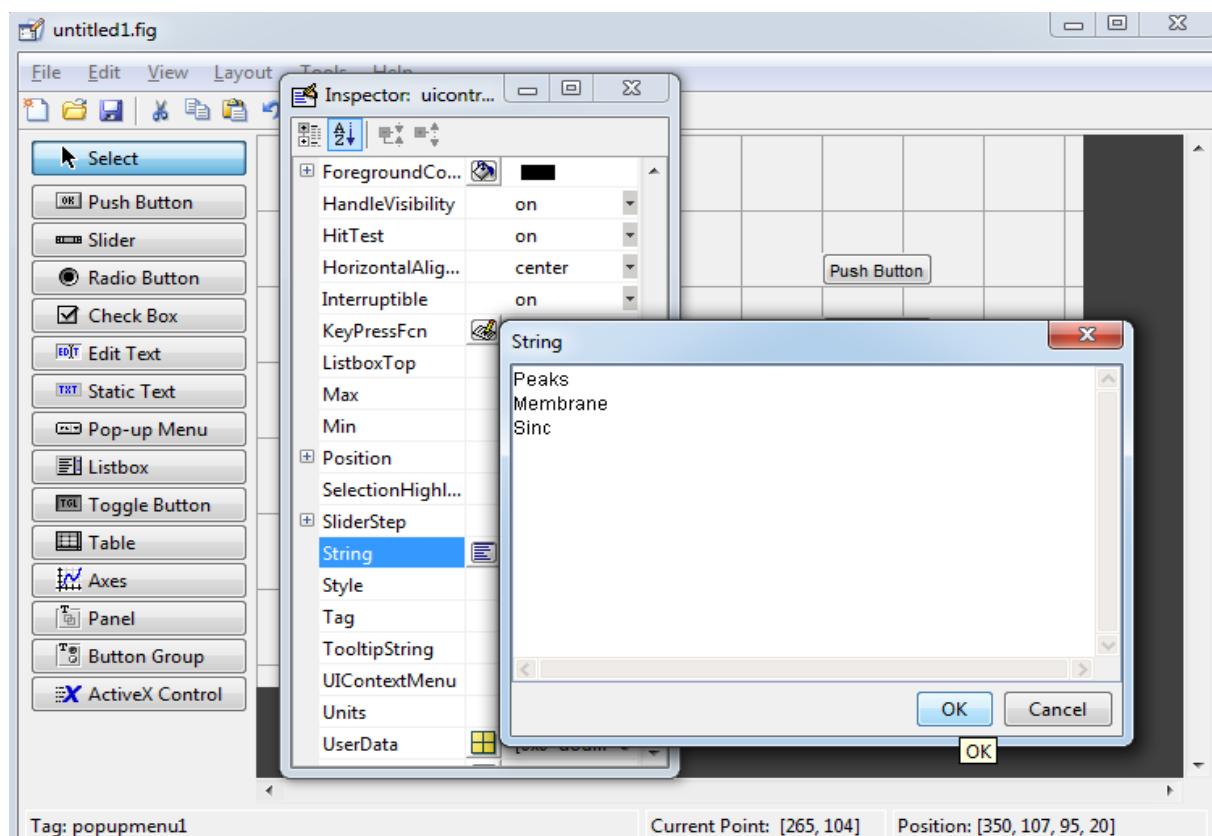
- ❖ Bir adet **grafik çizim (axes) nesnesi**,
- ❖ Bir adet veri setlerini gösteren **popup menü**,
- ❖ Bir adet **popup menü başlığı sunan static text nesnesi**,
- ❖ Üç adet **push button** nesneleri yer almaktadır.

Şekilde de görüleceği gibi hizalama işlemi başarılı bir şekilde gerçekleşmiştir.

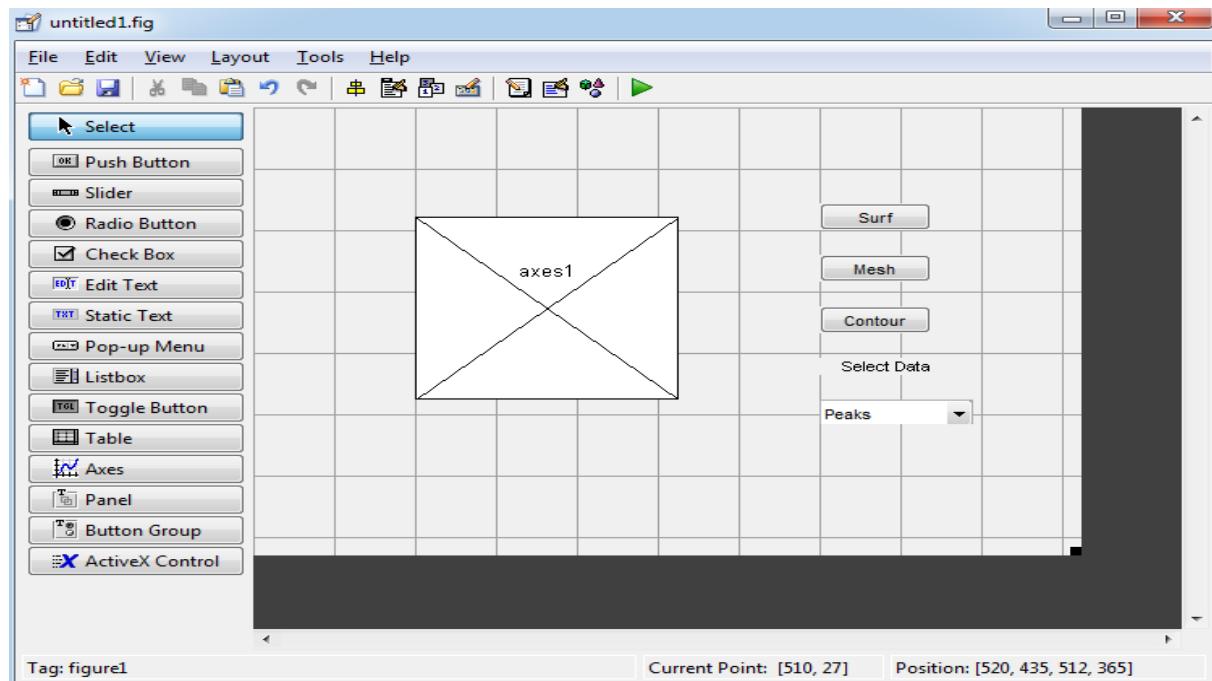


## Nesnelere Yazı Ekleme ve Özelliklerini Değiştirme

Nesnelerin özelliklerini değiştirmek istersek ya ilgili nesne farenin sol tuşu ile çift tıklanır ya da ilgili önce seçilip daha sonra **View/Property Inspector** komutu ile özellikler penceresi açılır. Buradan örneğimizde eklenen **popup menu** içeriğine **Peaks**, **Membrane** ve **Sinc** içeriklerini alt alta **popup menu** nesnesini seçtikten sonra **String** özelliğine ekleyiniz.

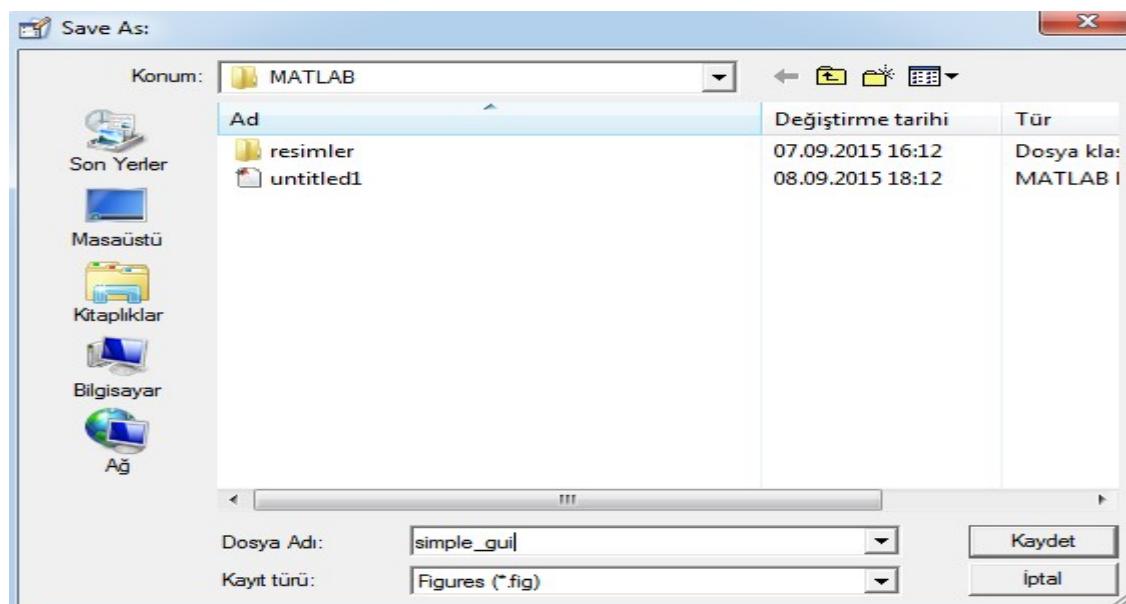


Ayrıca, üç adet butonun her birine sırayla **Surf**, **Mesh** ve **Contour** yazıları **String** özelliklerine eklenmelidir. Bu işlemler gerçekleştirildikten sonra GUI arayüzü penceresi aşağıdaki gibi gözükecektir.

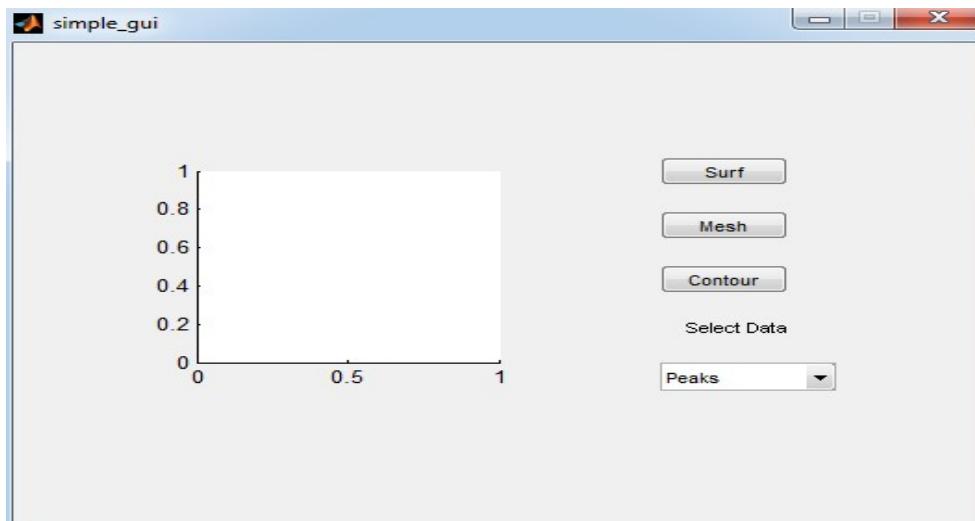


### GUI Tasarımını Kaydetme ve Çalıştırma

Bundan sonra bitmiş olan bu GUI arayüzü çalıştırarak görmek için öncelikle **Tools/Run** yolundan **Run (Çalıştır)** komutu verilir. Daha sonra gelen pencereden çalışmanın **Run** edilebilmesi için kaydedilmesi gerektiğini bildiren bir pencere çıkar. Burada **Yes** butonuna basarız. Bu adımdan sonra **MATLAB GUIDE** bize tasarımın kaydedileceği dosya ismini soran bir pencere getirir. Bu pencereden çalışmamıza bir isim vererek tasarımımızı kaydetmiş oluruz. Ardından karşımıza **Change the MATLAB Directory** gibi bir ekran gelirse burada bu ekranı **OK tuşuna** basarak kapatabilirsiniz.

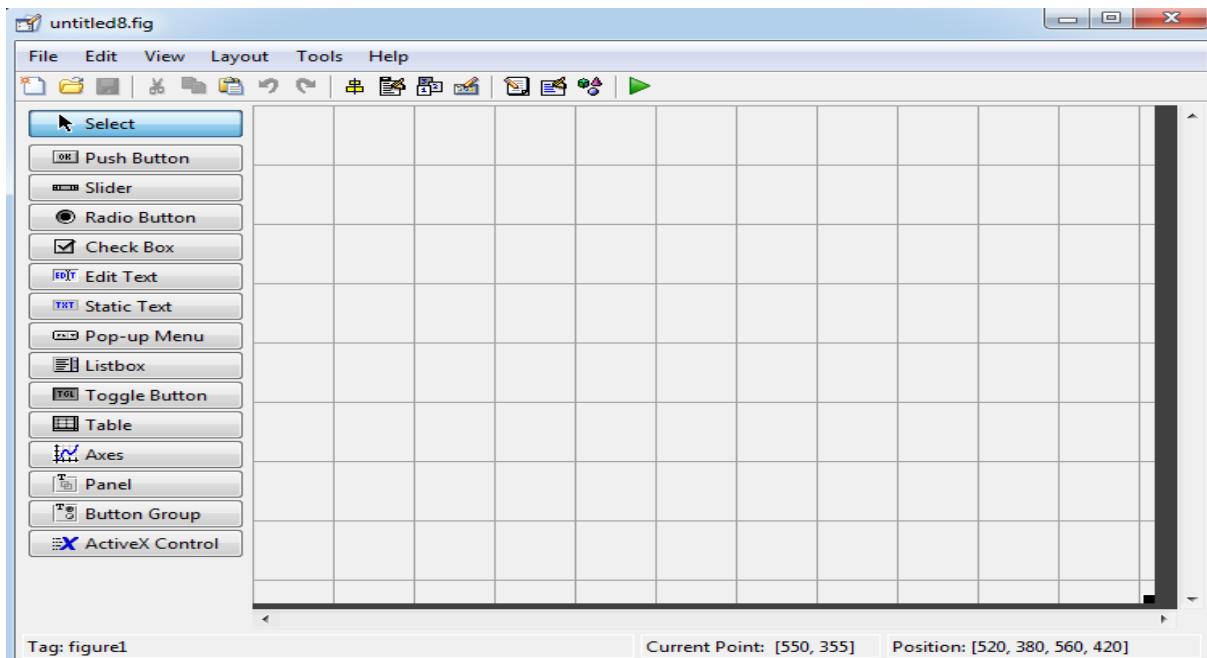


Bu ekran kaydedilen dosya MATLAB tanımlı dizinler dışında bir yere kaydedilme söz konusu olduğunda bizi uyarmaktadır. Sonra da **GUI** tasarımımızın çalışması sonucu gözükecek, uygulama penceresi ekranı karşımıza aşağıdaki gibi bir pencerede gelecektir.



## GUIDE Aracının İncelenmesi

Bir önceki konularımızda da bu aracı kısaca incelemeye çalıştık. Burada GUIDE aracı detaylı olarak incelenecektir. MATLAB komut satırından “**guide**” komutunu yazdığımızda ve gelen pencereden **boş (blank)** bir GUI tasarımını seçtiğimizde aşağıdaki gibi pencere ile karşılaşılır.



Bu ekrandaki araçlar ile ilgili açıklama aşağıda verilmiştir.

**Layout Editor:** GUIDE çalışma alanı ve penceresidir. Bu ekran ile GUI yüzeyine component paletten seçilen ilgili nesneler eklenebilir ya da diğer araçlar ile program kodlarının yazılması, nesnelerin GUI yüzeyi üzerinde hizalanması, tab tuşu geçiş sırasının değiştirilmesi gibi pek çok işlem gerçekleştirilebilir.

**Figure Resize Tab:** Bu araç GUI çalışma alanının boyutlandırılmasını sağlar. Fare işaretçisi bu alan üzerine getirildiğinde konum değiştirecektir. Bu anda farenin sol tuşu tıklanıp ileri geri hareket ettirilerek GUI yüzey alanının boyutları değiştirilebilir.

**Menu Editor:** GUI uygulamasına istenilirse File, Edit... v.b. gibi menü içeren programlarda olduğu gibi bir manü eklenmesi ve eklenen menü ile ilgili işlmelerin yapılması bu araç vasıtasyyla sağlanır.

**Align Objects:** Bu araç sayesinde GUI çalışma alanına eklenilen nesnelerin yatay ya da dikey olarak hizalanması işlemleri gerçekleştirilebilir.

**Tab Order Editor:** Tab Order Editor kullanılarak GUI yüzeyindeki nesnelerin birinden diğerine tab tuşu ile geçiş sırası (örneğin bir buton seçili ve aktif iken bir başka butona ya da bir liste kutusuna tab tuşu kullanılarak geçilmesi gibi.) değiştirilebilir.

**Property Inspector:** Bu pencere sayesinde de GUI uygulamasına eklenen nesnelerin özellikleri değiştirilebilir ya da var olan özelliklerinin ve değerlerinin neler olduğu gözlenilebilir.

**Object Browser:** Bu araç ile tasarımcı GUI uygulamasına eklenmiş olduğu nesnelerin ve isimlerinin neler olduğu genel hali ile bakabilir.

**Run:** Bu buton yardımı ile de hazırlanmış olan bir GUI uygulaması çalıştırılabilir. Bu şekilde tasarımcı hazırlamış olduğu GUI'yi test etme imkânına sahiptir.

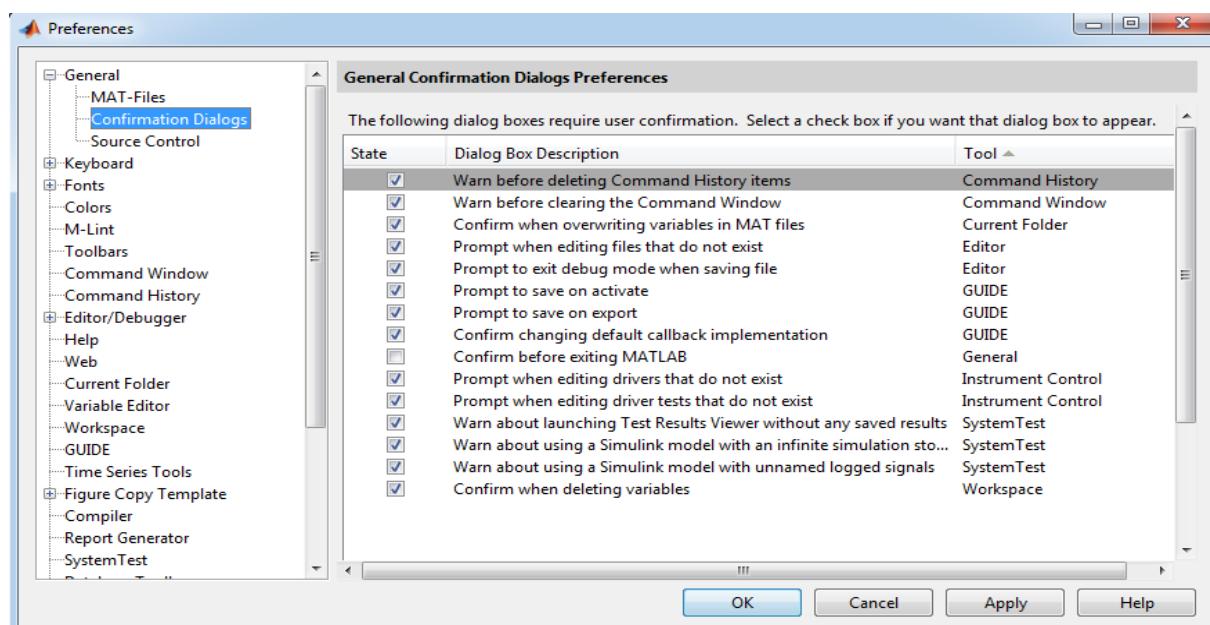
**M-File Editor:** Hazırlanmış olan GUI uygulaması ile ilgili komutları görebilmek ve üzerinde değişiklik yapabilmek için bu araç kullanılır.

## MATLAB GUIDE Tercihleri

Bu tercihleri görebilmek için GUIDE ekranında **File** menüsünden **Preferences komutu** çalıştırılır.

### Doğrulama Seçenekleri

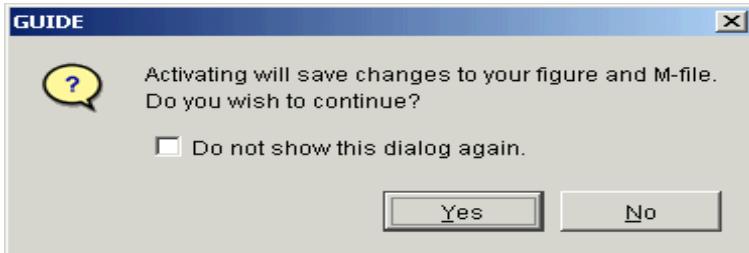
Bu seçeneklere ulaşmak için **General/Confirmatin** yolu izlenmelidir. Karşımıza aşağıdaki gibi bir pencere gelecektir.



Bu penceredeki seçeneklerden bazıları GUIDE ile ilgilidir. Bu seçeneklerin görevleri şu şekildedir:

### Promp to Save on Activate

Bu seçenek seçili ise GUIDE bir GUI uygulaması çalıştırılmadan önce onun kaydedilmesi gerektiğini bildiren Şekil aşağıdaki gibi bir pencere ile kullanıcı uyarır.



Bu gelen ekranın evet denilerek uygulamanın kaydedilmesi ve çalıştırılması sağlanabilir. Eğer ki bu pencerenin her seferinde üzerinde değişiklik yapılan, fakat kaydedilmeyen bir GUI uygulaması olduğunda kullanıcıyı uyarmaması isteniyor ise "**Do not show this dialog again.**" seçeneği işaretlenmelidir.

### Promp to Save on Export

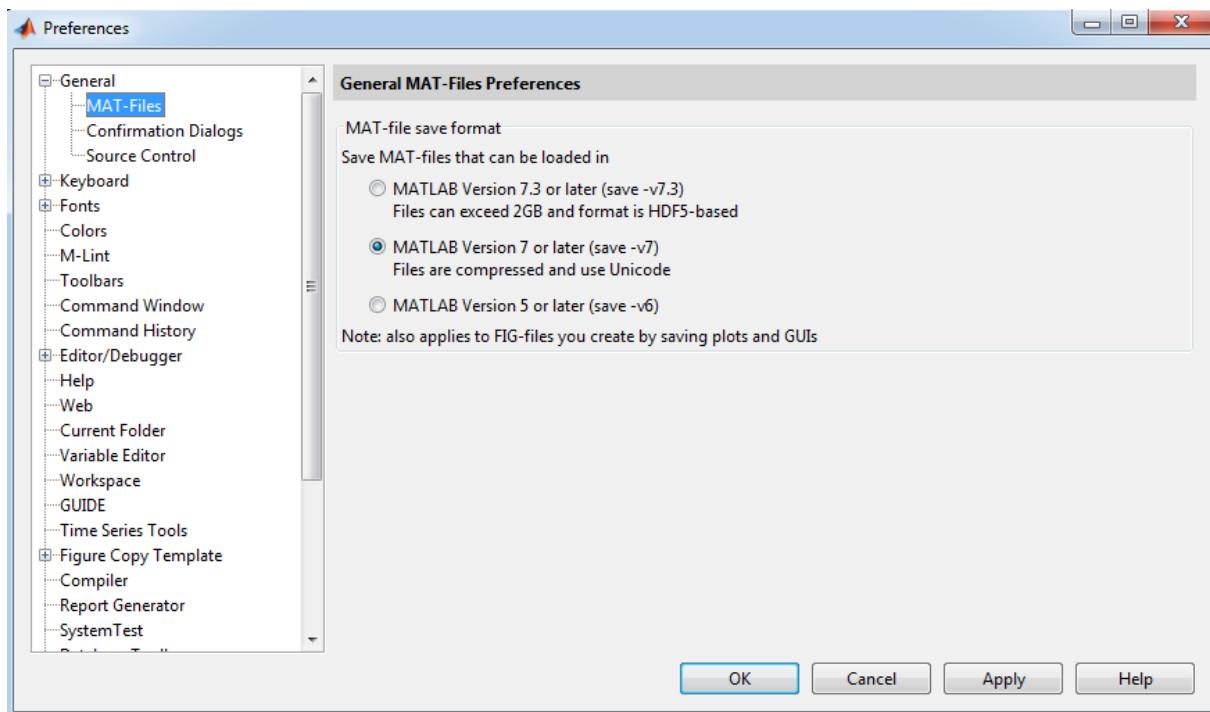
Bu seçenek GUIDE çalışma ekranında iken File menüsünden **Export komutu** verilirse ve tasarlanan GUI uygulaması kaydedilmemiş değişiklikler içeriyorsa kullanıcıya **Export** işlemi öncesinde var olan değişikliklerin kaydedileceği konusunda aşağıdaki pencere ile uyarır.



Bu pencerede evet butonuna tıklanarak işleme devam edilebilir. Eğer ki bu pencerenin sürekli çıkması istenmiyor ise kullanıcı evet demeden önce "**Do not show this dialog again.**" Seçeneğini işaretlemelidir.

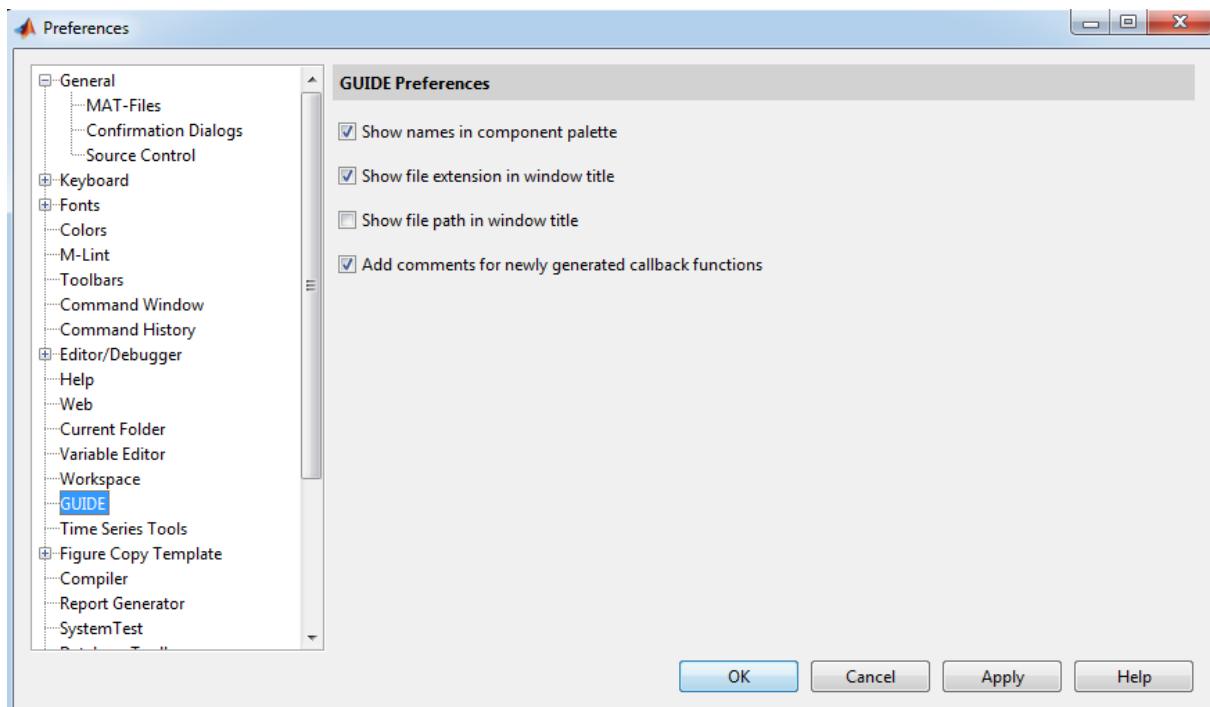
### Geriye Uyumluluk Seçeneği

Seçenekler penceresinde **General>Mat-Files** yolu izlenilerek gelen aşağıdaki pencereden önceki **MATLAB** versiyonları ile uyumlu olacak şekilde dosyaların kaydedilme format ayarı değiştirilebilir.



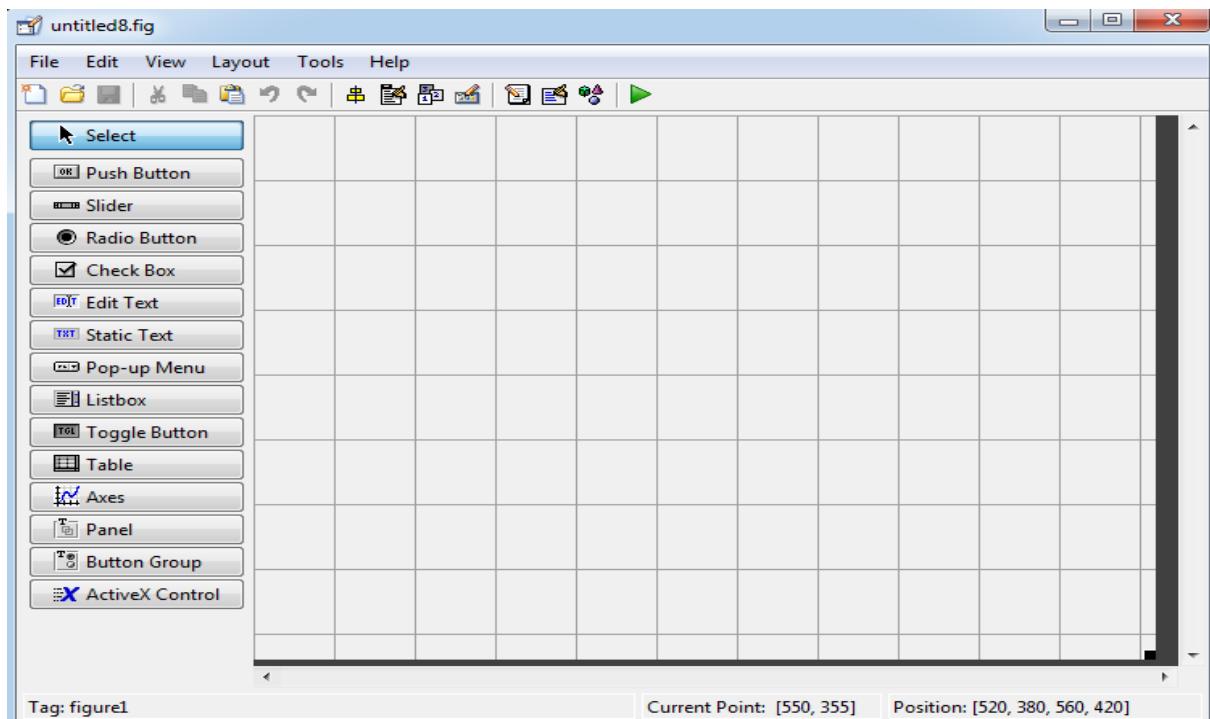
## Diger Tercihler

Preferences ekranında ayrıca GUIDE yolu altında diğer tüm GUIDE seçenekleri yer almaktadır. Karşımıza aşağıdaki gibi bir pencere gelecektir.



Bu penceredeki seçeneklerin işlevleri şu şekildedir:

**Show Names in Component Palette:** GUIDE ekranında aşağıda da görüldüğü gibi component paletinde yer alan butonlarda nesnelerin isimlerini göstermek için bu seçenek işaretlenmelidir.



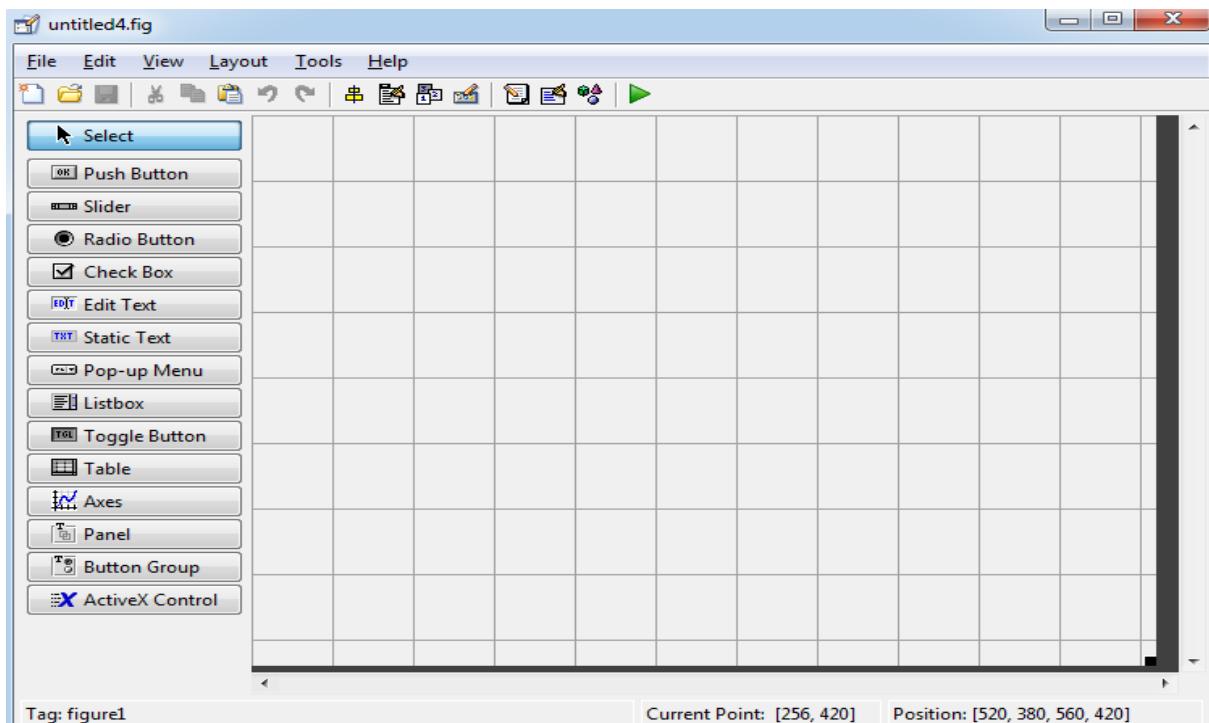
**Show File Extension in Window Title:** GUIDE ekranının başlık çubuğunda üzerinde çalışılan GUI uygulamasının dosya isminin yanında .fig uzantısının da gösterilmesi istenirse bu seçenek işaretlenmelidir.

**Show File Path in Window Title:** GUIDE ekranının başlık çubuğunda üzerinde çalışılan GUI uygulamasının tüm dosya yolu ile birlikte dosya isminin gösterilmesi istenirse bu seçenek işaretlenmelidir.

**Add Comments for Newly Generated Callback Functions:** Boş bir GUI uygulaması (untitled bir döküman) ile GUI tasarıma başlandığında .m uzantılı komut satırlarının olduğu dosyaya her bir callback ve bu dosyada yer alan fonksiyonlarla ilgili otomatik olarak açıklama satırlarının eklenmesi istenirse bu seçenek işaretli olmalıdır. GUI kodlamasında açıklama satırlarının başında % işaretleri yer alır ve varsayılan olarak M-File Editor'de açıklama satırları yeşil renkte gözükmür.

## MATLAB GUIDE Bileşenleri

**MATLAB GUIDE** aracı kullanarak **boş (blank) bir GUI** çalışma ekranını açtığımızda sol tarafta görülen component panel pek çok nesnenin kullanılabileceği görülmektedir. Bu panelde Push Button, Slider, Radio Button, Check Box, Edit Text, Static Text, Pop-up Menu, Listbox, Toogle Button, Table, Axes, Panel, Button Group ve ActiveX Control gibi nesneler yer almaktadır.



Şimdi bu nesnelerin sırasıyla özellikleri ile ilgili bilgiler verilecek ve nasıl kullanılacağı gösterilecektir.

**Push Button**(): Normal bir buton özelliği taşımaktadır. Bir buton üzerine tıklanması ile yapılacak komutlar bu buton ile ilgili callbackların altına yazılır.

**Toggle Button**(): Çift durumlu bir buton özelliği taşıyan bu nesne ile iki farklı seçenek içeren durumlarda örneğin bu buton basılı ise bir işlemin, bu buton basılmamış ise başka işlemlerin yapılması gerektiği yerlerde tercih edilen bir nesnedir. Buton grubu nesnesi ile beraber kullanımı tavsiye edilir.

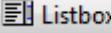
**Radio Button**(): Birden fazla seçenekin olduğu, ancak seçeneklerden sadece herhangi birinin seçilebileceği hallerde bu nesne kullanılır. Buton grupları ile kullanılması genellikle tercih edilir.

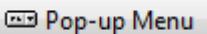
**Check Box**(): Kullanıcıya seçim yapabileceği ve birden fazla sıkı işaretleyebileceği durumlarda bu nesne kullanılır.

**Edit Text**(): Bir kullanıcıdan bilgi girişi ya da bir değerin alınması söz konusu olduğunda giriş elemanı olarak sıkılıkla kullanılan bir nesnedir.

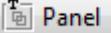
**Static Text**(): Kullanıcıya herhangi bir bilgi verme ya da bulunan bir sonuç veya değeri gösterme amacıyla sıkılıkla kullanılan bir nesnedir.

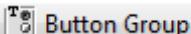
**Slider**(): Kullanıcıdan bir giriş değerini kaydırılmak suretiyle kolaylıkla alınmasına imkân veren bir nesnedir.

**List Box**(  Listbox ): Kullanıcıya bilgi verme amacıyla kullanılabileceği gibi bir değeri listeden seçmek amacıyla da kullanılan sabit bir liste kutusu niteliğinde kullanılan bir nesnedir.

**Pop-Up Menu**(  Pop-up Menu ): Kullanıcıdan alınmak istenilen bilgileri açılan bir listeden seçme özelliği taşıyan bir nesnedir.

**Axes**(  Axes ): Yapılan iş ile ilgili grafik çizimlerinin kullanıcıya gösterilmesini sağlayan bir nesnedir.

**Panel**(  Panel ): GUI yüzeyi nesnelerinin kullanıcıya daha anlamlı ve güzel gözükmesini sağlayan, ayrıca tasarımcıya GUI dizaynında kolaylık sunan bir nense olup, GUI yüzeyi nesnelerinin gruplanması ve bir arada gösterilmesi amacıyla kullanılır.

**Button Group**(  Button Group ): Radio veya toggle tipteki buton nesnelerinin bir arada kullanılarak kullanıcının birden fazla seçenekten sadece bir tanesini seçmesini sağlamak amacıyla kullanılan bir nesnedir.

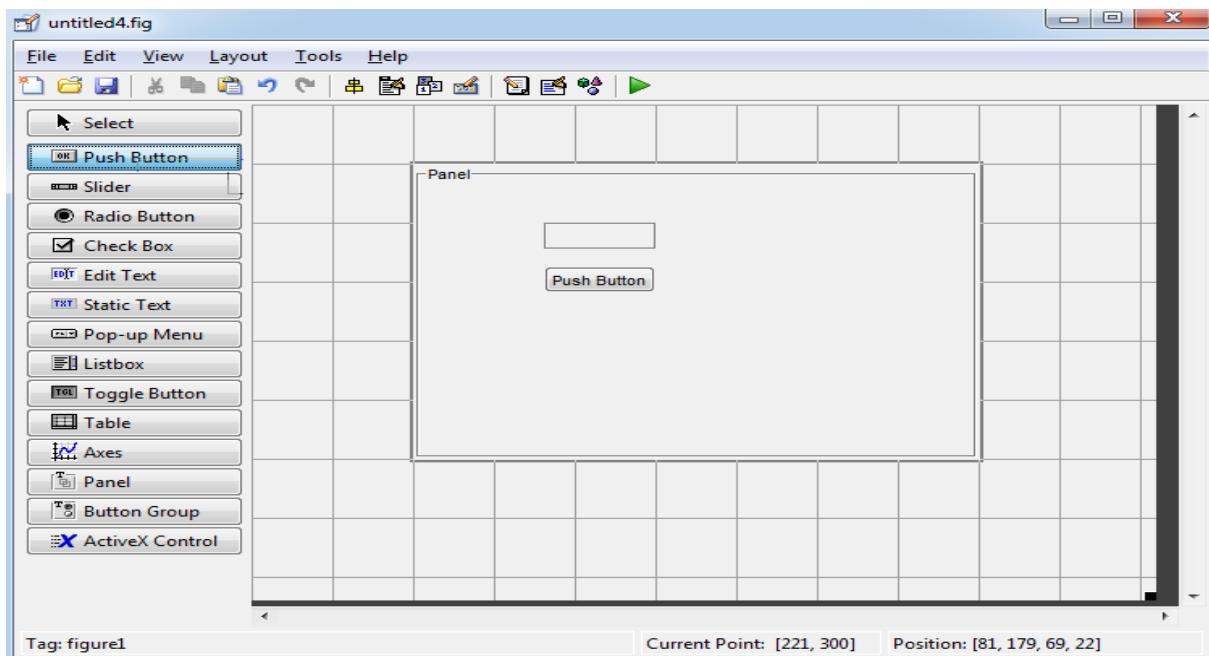
**ActiveX Control**(  ActiveX Control ): MATLAB GUI tasarımları sadece yukarıda belirtilen nesneler ile sınırlı değildir. Tasarımcı ve programcı ayrıca, ActiveX adı verilen ve değişik alternatifleri olan nesnelerin kullanılmasına da imkân verir. Böylece hem tasarımcı hem tasarlanacak GUI arayüzünün kullanımı bakımından kullanıcıya esneklik sağlanmış olunur.

### Nesnelerin GUI Yüzeyine Yerleştirilmesi

Bir nesneyi çalışma alanına eklemek için yapılması gereken sol tarafta yer alan panelden yerleştirilmek istenilen nesnenin butonunu tıklamak ve GUI yüzeyinde yerleştirilmek istenilen yere tıklamak ya da yerleştirilmek istenilen bölgeyi farenin sol tuşu ile basılı tutarak beliren çerçeveyin nesne boyutları olacağını düşünerek yerleştirme işlemi yapılabilir.

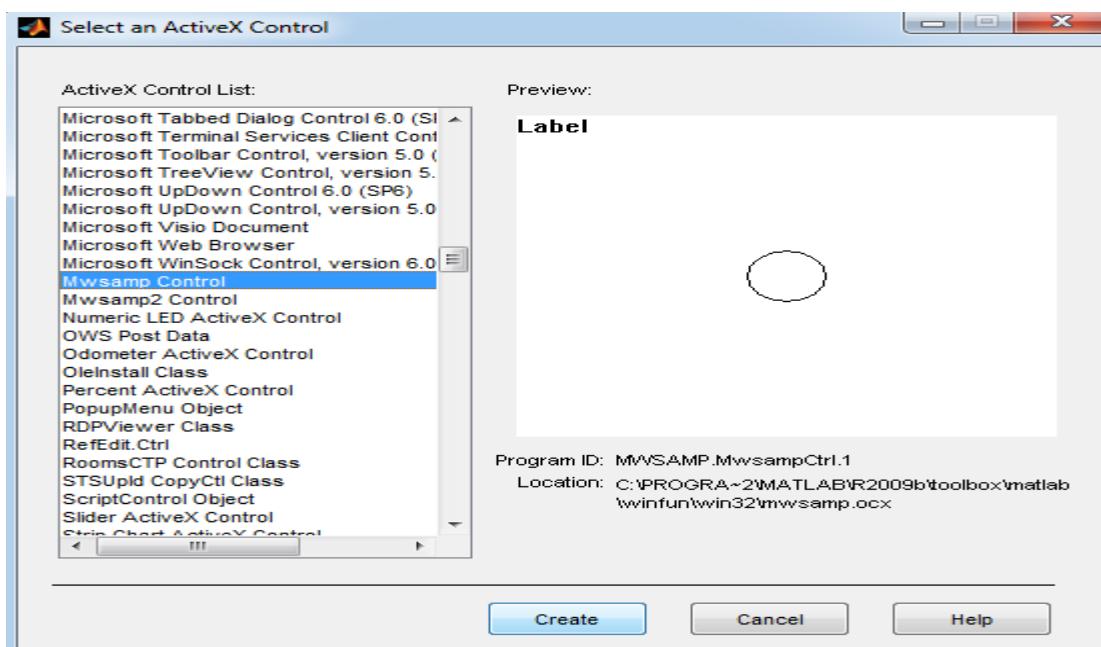
### Bir Nesnenin GUI Yüzeyinde Bulunan Bir Panele ya da Buton Grubuna Yerleştirilmesi

Bir nesne GUI yüzeyinde daha önceden yerleştirilmiş olan bir panele ya da buton grubuna yerleştirmek için öncelikle yerleştirilecek nesne panelden seçilir ve daha sonra fare işaretçisi yerleştirilecek panel ya da buton grubu üzerine götürülür. Bu durumda fare imlecinin üzerinde olduğu bu grup nesnesi bir anda seçili hale gelecektir. Şayet bu işlem ana GUI figure üzerine getirilirse bu sefer grup nesnesinin aktifliği kaybolacak ve figure yüzeyi seçili duruma dönüşecektir. (Bir nesnenin aktif veya seçili olduğu durumu nesnenin kenarında beliren çerçevelerin varlığı ile anlaşılabilir.) Bu durumu GUI yüzeyine yerleştirilmiş olan bir panel nesnesine bir **push buton** nesnesinin yerleştirilmesi örneği aşağıdaki şekilde görüldüğü üzere özetlenebilir.



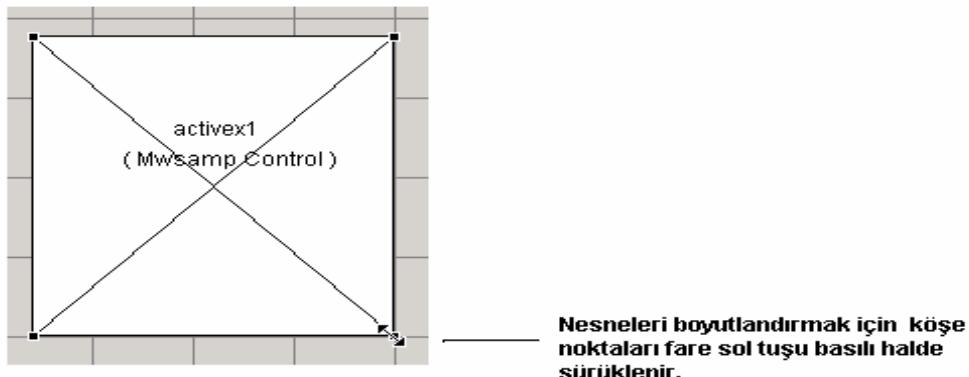
## ActiveX Control Nesnesinin GUI Yüzeyine Yerleştirilmesi

Bir ActiveX nesnesini GUI yüzeyine eklemek için öncelikle panelden seçilir. Daha sonra GUI alanında yerleştirilmesi düşünülen bir yere farenin sol tuşu ile tıklanır. Bu adımdan sonra karşımıza aşağıdaki gibi bir pencere gelecektir. Bu pencerede GUI yüzeyine yerleştirilmek istenilen **ActiveX Control** nesnesi sol taraftaki listeden seçilmeli ve ardından **Create** butonuna basılmalıdır.



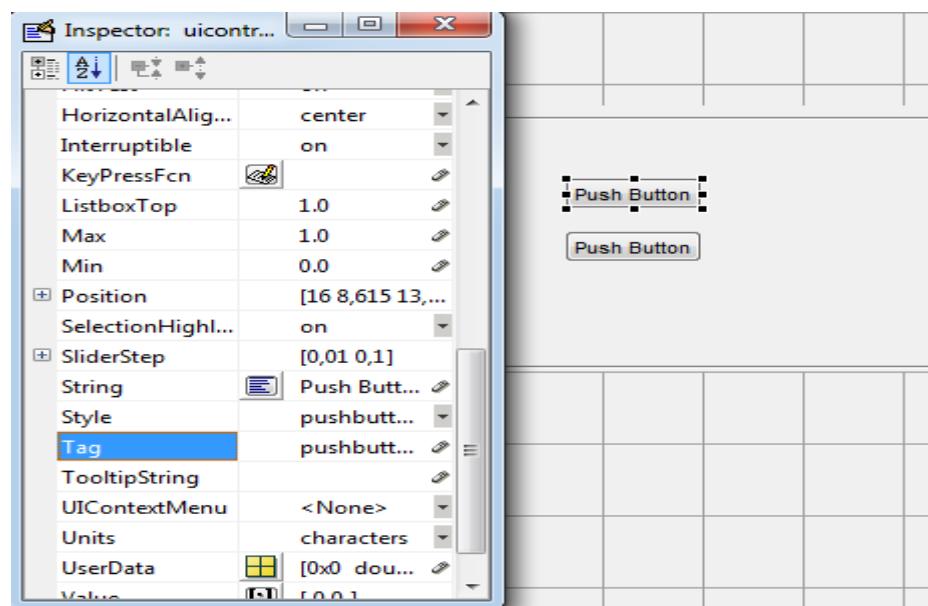
## GUI Yüzeyine Eklenen Nesnelerin Boyutlandırılması

GUI alanına eklenen bir nesnenin boyutlarını değiştirmek için ilgili nesne öncelikle farenin sol tuşu yardımı ile seçilir. Daha sonra da etrafında beliren köşe noktaları kullanılarak boyutları değiştirilebilir. Bu durum aşağıdaki şekilde gösterilmiştir.



## Her Nesneye Tanıtıçı Bir İsim Atamak

Bir nesneye tanıtıçı ve o nesneye özel bir isim vermek için öncelikle o nesne farenin sol tuşu ile GUI tasarım yüzeyinde seçilir. Daha sonra **fare sağ tuş** ve **Property Inspector** komutu verilir. Karşımıza gelen özellikler penceresinden nesnemizin **Tag** özelliğine istenilen bir isim verilebilir. Ayrıca, **Property Inspector** penceresini açmak için nesne üzerinde farenin sol tuşu ile çift tıklanarak da açılabilir. Aşağıda örnek olarak bir buton nesnesi için bu durum görülmektedir.

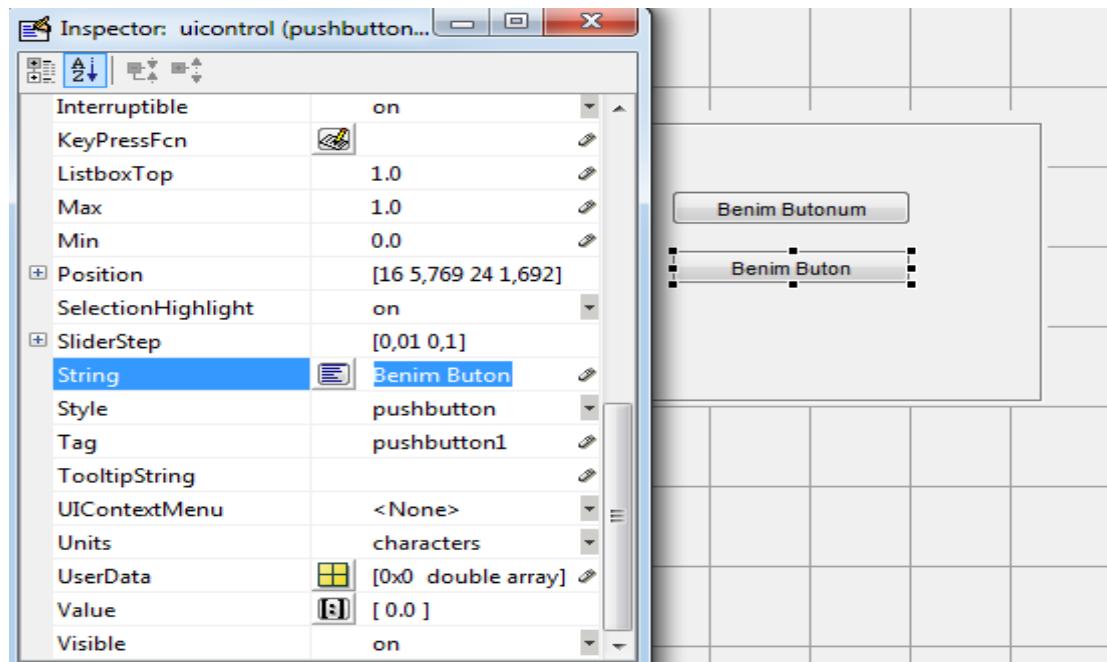


## Nesnelere Yazı EklemeK

Bazı nesneler özellikleri gereği kullanıcıya bilgi vermek veya bir seçenek sunmak amacıyla **string** bilgiler içerirler. Bu nesneler **Push Button**, **Toggle Button**, **Radio Button**, **Check Box**, **Text**, **List Box**, **Popup Menu**, **Panel** ve **Button Group** nesneleridir. Bu nesneler de yapıları gereği değişik özellikler içermektedir ve bu özellikleri üzerinden **string** değerler atamak mümkün olmaktadır. Bu nesnelere yazı bilgilerinin nasıl verildiği aşağıda sırasıyla açıklanmıştır.

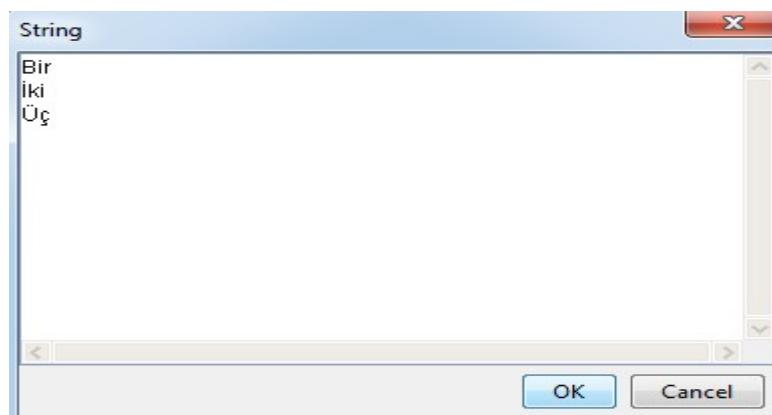
## **Push Button, Toggle Button, Radio Button, Check Box Nesnelerine Yazı Eklenmesi**

Bu nesnelere yazı eklemek için ilgili nesne seçilerek pencerelerinden **String** özelliklerine istenilen bir metin bilgisi girilebilir. Ayrıca, bu özellik programlama komut satırlarıyla da değiştirilebilir. Aşağıdaki örnekte bir push butonun üzerindeki yazının değiştirilmesi gözükmemektedir. Eğer ki bu nesnelere alt alta olacak şekilde birden fazla satır içeren bilgiler girilmek istenirse bir sonraki başlık altında anlatılan teknik kullanılmalıdır.



## **List Box ve Popup Menu Nesnelerine Yazı Eklenmesi**

Bu nesneler çoklu **string** bilgiler içeren liste kutusu tarzı yapılardır. Bu nesnelere **string** bilgiler eklemek istenirse öncelikle nesne seçilir. Ardından **String** özelliğinin yanında yer alan butona tıklanır. Karşımıza aşağıdaki gibi bir pencere gelecektir.



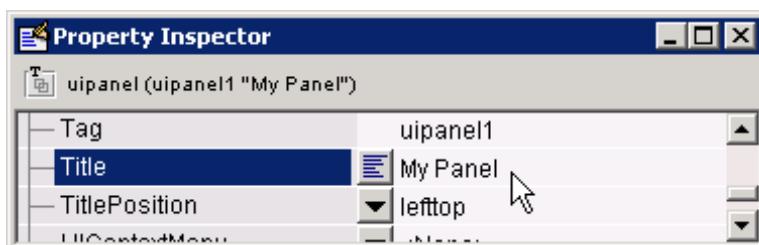
Bu pencereye gerekli bilgiler girildikten sonra **OK** butonuna basılır. Böylece **string** verilerin girilmesi işlemi tamamlanmış olur. İstenirse bu özellik programlama yoluyla da değiştirilebilir. Bu yöntem eğer ki bir önceki başlıkta belirtilen nesnelere birden fazla bilgi girilmesi istendiğinde de bu nesneler için de kullanılabilir.

Yukarıdaki şekilde görülen bilgiler girildiğinde örneğimizdeki pop-up menü nesnemizin görüntüsü aşağıda görüldüğü gibi olacak ve kenarındaki buton tıklandığında da tüm seçenekler kullanıcıya sunulacaktır.

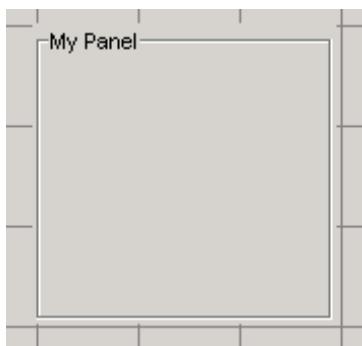


### Panel ve Button Group Nesnelerine Başlık Eklenmesi

Bu nesneler içinde yer alan pek çok nesne gruplandırma imkânına sahip olup bu nesnelere başlık eklemek için farklı bir özellik kullanılmaktadır. Bu nesnelere başlık ekmek için nesne seçili **iken Property Inspector** penceresinden **Title** özelliğine gerekli bilgi girilmelidir. Bu durumu aşağıdaki sekilden takip edilebilir.



Örneğimizde bir panel nesnesi kullanılmış olup başlığı **My Panel** olarak değiştirilmiştir. Bu değişiklik yapıldıktan sonra panelimizin GUI çalışma alanındaki görüntüsü aşağıdaki gibi olacaktır.



### GUI Çalışma Alanında Nesneler ile Çalışma

GUI yüzeyindeki nesnelere tasarım ortamında istenildiği şekilde müdahale edilebilir. GUI çalışma alanındaki nesneler üzerinde kopyalama, silme, taşıma, öne getirme, en arkaya gönderme, hizalama, tab tuşu ile seçim sırasının değiştirilmesi, başka bir noktaya taşınması veya boyutlarının değiştirilmesi, cetvel ve ızgara kullanılarak işlemlerin yapılması, GUI uygulamasında ana menü oluşturmak ya da istenilen bir nesne üzerinde **context** menü oluşturmak, GUI uygulamasına araç çubuğu eklemek, GUI tasarımda kullanılan nesnelerin görülmesi gibi pek çok işlem için GUIDE tasarımcıya pek çok kolaylık sağlamaktadır.

## Nesnelerin Seçilmesi

GUI yüzeyindeki nesneleri teker teker seçmek için fare işaretçisi ilgili nesne üzerine götürüp farenin sol tuşuna basmak yeterlidir. Ancak, tasarımcı birden fazla nesneyi seçmek istiyor ise o zaman **Ctrl ya da Shift** tuşu basılı halde iken ilgili nesneleri birer birer tıklamalıdır. Ayrıca, çoklu seçim işlemi için fare işaretçisi GUI yüzeyinin boş bir yerinde farenin sol tuşu basılı tutularak bu halde hareket ettirilir. Bu esnada oluşan pencere içerisinde seçilmesi istenilen nesneler var olduğunda farenin sol tuşu bırakılır. Bu şekilde de topluca seçim işlemi yapılmış olacaktır.

## Nesneler Üzerinde Kopyalama, Silme, Taşıma, Çoğaltma İşlemlerinin Yapılması

GUI yüzeyine eklenene bir nesneden kısa yoldan yeni bir kopya almak istenirse nesne üzerinde farenin sağ tuşu ile tıklanır ve açılan menüden **Duplicate (Çoğalt) komutu** verilir. Bu şekilde kopya alınan nesnenin tüm görünüm özellikleri aynı yeni bir kopyası oluşturulmuş olacaktır.

Ayrıca, bir nesne kopyalama ve yapıştırma tekniği ile de çoğaltılabılır. Bunun için nesne üzerinde farenin sağ tuşu tıklanır açılan menüden de **Copy (Kopyala) komutu** verilir ya da bu işlem yerine kısaca **Ctrl + C** tuş kombinasyonu kullanılabilir. Daha sonra GUI yüzeyinin istenilen bir noktasına fare işaretçisi götürürler ve ardından farenin sağ tuşu tıklanır. Açılan menüden **Paste (Yapıştır) komutu** verilir. Bu işlem kısaca **Ctrl + V** tuş kombinasyonu ile de yapılabilir.

Bir nesnenin bir noktadan başka bir noktaya taşınması istenirse nesne ya farenin sol tuşu basılı halde istenilen noktaya sürüklenebilir ya da nesne üzerinde farenin sağ tuşu tıklanıp açılan menüden **Cut (Kes) komutu** verilir. Bu işlem kısaca **Ctrl + X** tuş kombinasyonu ile de gerçekleştirilebilir. Ardından istenilen noktaya fare işaretçisi götürürler ve sağ tuş tıklanıp açılan menüden **Paste (Yapıştır) komutu** verilir.

## Bir Nesneyi Diğer Nesneler Arasında Öne veya Arkaya Getirme

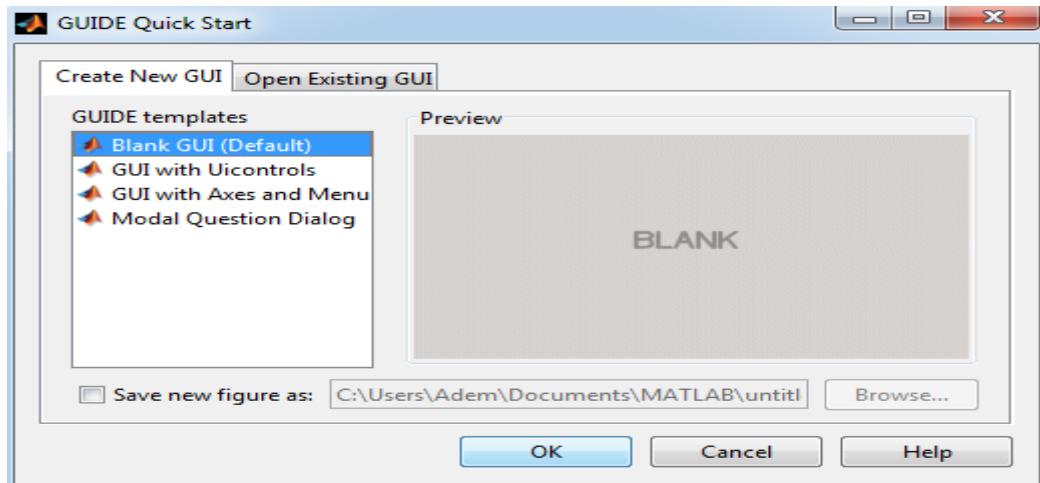
Bir nesnenin diğer nesnelere göre pozisyonunu değiştirmek için önce nesne seçilir. Daha sonra **Layout menüsünden** ilgili komut verilir. Buradaki komutların işlevleri şöyledir:

- ❖ **Bring to Front komutu** seçili nesneyi en öne getirir.
- ❖ **Send to Back komutu** seçili nesneyi en arkaya gönderir.
- ❖ **Bring Forward komutu** seçili nesneyi bir adım öne getirir.
- ❖ **Send Backward komutu** seçili nesneyi bir adım arkaya gönderir.

## ***GUIDE Aracında Şablon Uygulamalar ile Çalışma***

MATLAB GUIDE aracı GUI tasarımcılarına ilk başlayanlar için kendi içinde hazır örnek uygulamalar (templates) sunmaktadır. Bu uygulamaların arayüzü ve nesnelerin callbacklerini kullanan komut satırlarını yazma hususunda programcı ve tasarımcıya önbilgi vermesi bakımından çok yararlıdır.

MATLAB komut satırından **guide** yazıldığında ya da araç çubuğundan GUIDE simgesi tıklandığında karşımıza aşağıdaki gibi bir pencere gelecektir.



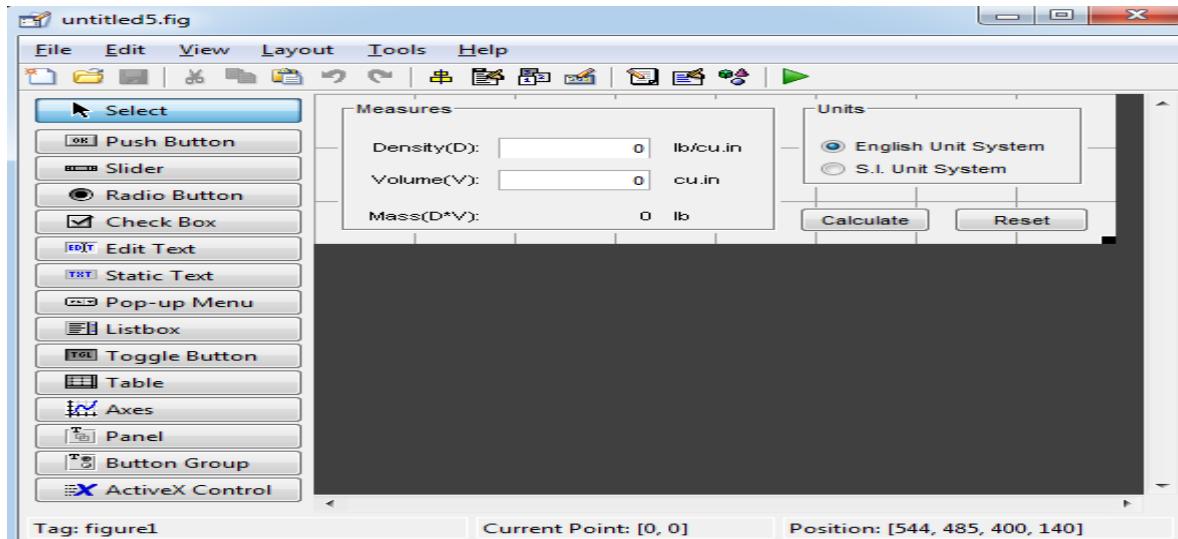
Bu pencereden şablon olarak karşımıza üç seçenek çıkmaktadır. Bunlar:

- ❖ “**GUI with Uicontrols**”
- ❖ “**GUI with Axes and Menu**”
- ❖ “**Modal Question Dialog**”

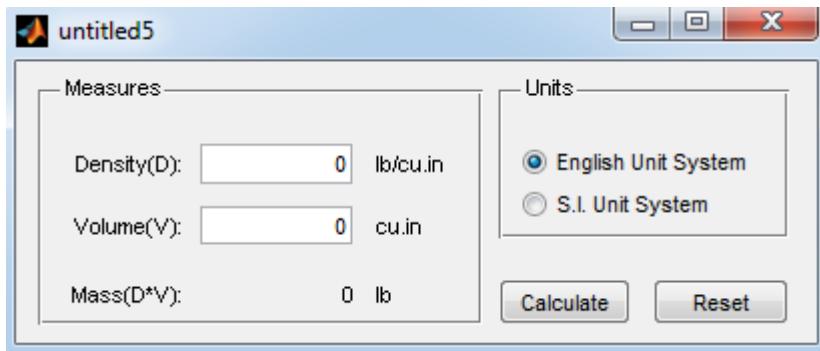
Aşağıda bu uygulamalar hakkında ayrıntılı bilgiler verilmiştir.

### “**GUI with Uicontrols**” Uygulaması

Bu uygulama seçildiğinde aşağıdaki şekilde gözüken GUI tasarımları karşımıza gelecektir.



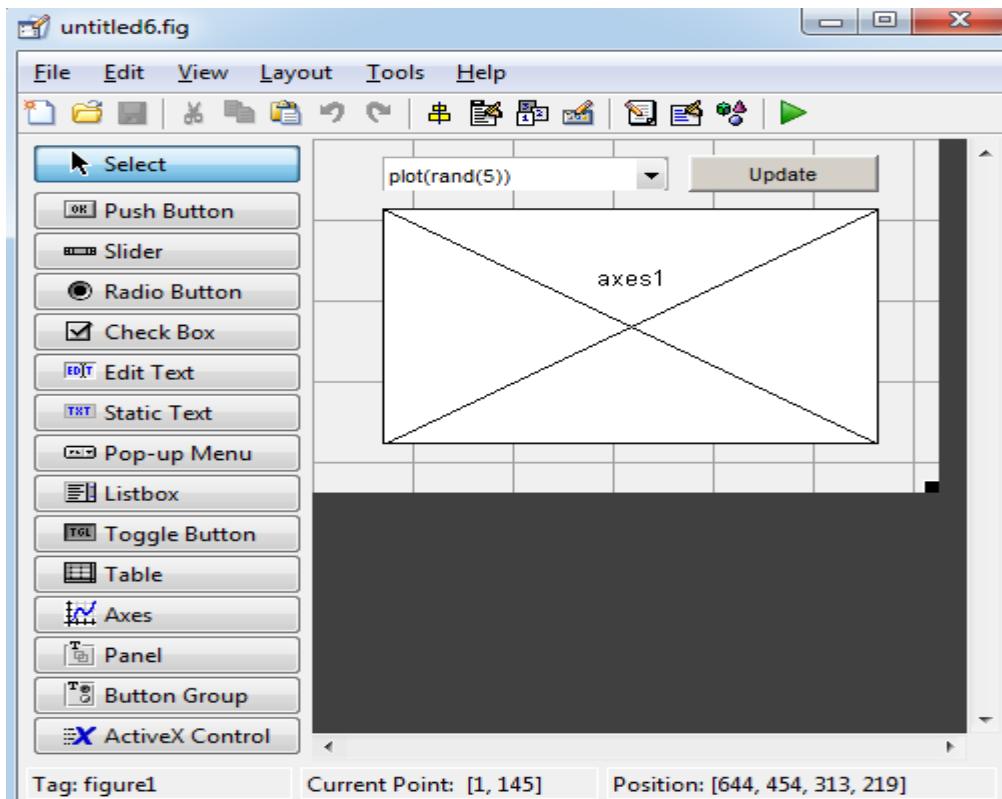
Bu uygulamayı araç çubuğundan Run butonuna basarak çalıştırduğumızda da aşağıdaki gibi bir uygulama arayüzü ekrana gelecektir.



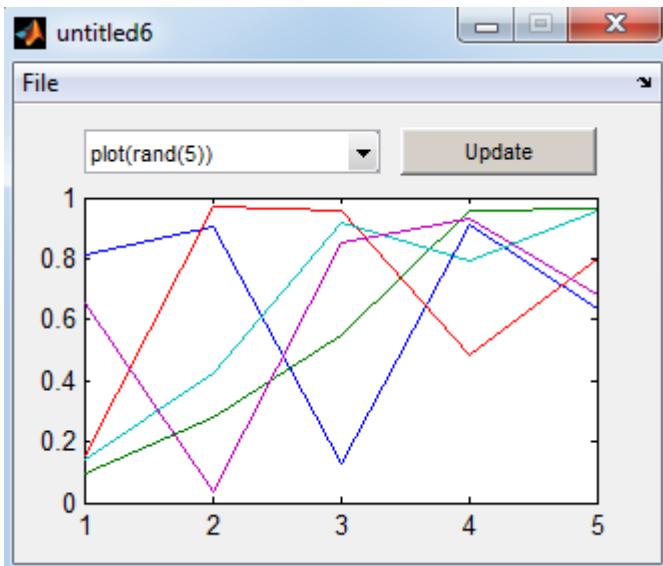
Bu uygulamada amaç seçilen birim sistemine göre yoğunluk ve hacim değerleri girilen bir cismin kütle değerini **Yoğunluk\*Hacim ( $D \cdot V$ )** formülünden yola çıkarak hesaplanması sağlanmak ve kullanıcıya hesaplanan değeri sunmaktır. Burada push butonların ve text kutuların program kodları ve callback parçalarının kullanımına yönelik olarak programcaya bilgiler verilmektedir.

### “GUI with Axes and Menu” Uygulaması

Bu seçenek seçilerek bir GUI şablon uygulaması açıldığında karşımıza aşağıdaki gibi bir tasarım şablonu gelecektir.



Burada da bu GUI uygulamasının çalıştırılması durumunda programcı aşağıdaki gibi bir arayüz ile karşılaşacaktır.



Bu çalışmada amaçlanan liste kutusundaki seçilen elemanlara göre farklı formülasyonlar kullanılarak **update** butonuna tıklandığında grafik nesnesi üzerinde bulunan sonuçlara göre çizimin yapılmasını sağlamaktır.

Bu uygulamada ile programcıya **axes (grafik çizimi)** nesnesinin nasıl kullanıldığı ve de liste kutuları ile ilgili callback satırlarının nasıl işletildiği hususunda bilgiler verilmektedir.

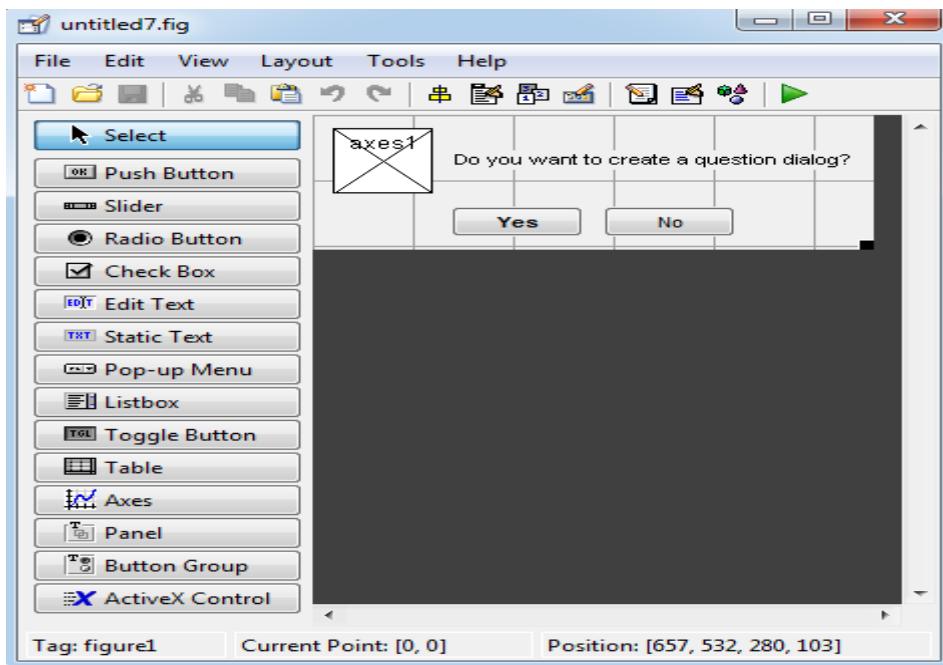
Ayrıca, bu uygulamada “**rand(5)**” komutunun kullanımı ve rasgele değerler içeren dizilerin nasıl üretildiği de gösterilmiştir. Bu uygulamanın bir başka faydalı yönü de uygulamanın menü içermesi ve menülerin programlanması konusunda tasarımcıya önbilgi verilmektedir.

Menülerin kullanımı ile ilgili olarak

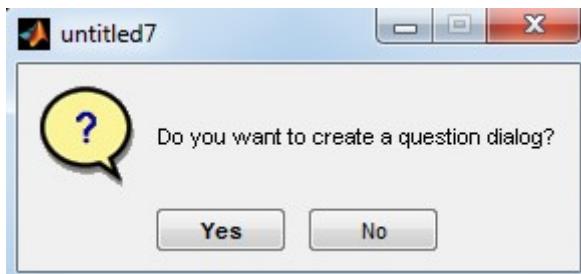
- ❖ **file = uigetfile('\*.fig')** komutu ile bir başka .fig dosyanın bir grafik çizim nesnesinde nasıl aktarılacağı,
- ❖ **printdlg(handles.figure1)** komutu ile de varolan bir çizimin yazıcıdan nasıl çıktı alınacağı
- ❖ **close komutu** ile aktif bir GUI figure ekranının nasıl kapatılacağı konularında programcıya programlama teknikleri sunulmaktadır.

### **“Modal Question Dialog” Uygulaması**

Bu seçenek seçildiğinde karşımıza Şekil 5.34’teki gibi bir template GUI uygulaması çıkacaktır.



Bu uygulamayı çalıştırduğumızda da aşağıdaki gibi bir GUI arayüzü ile karşılaşılır.



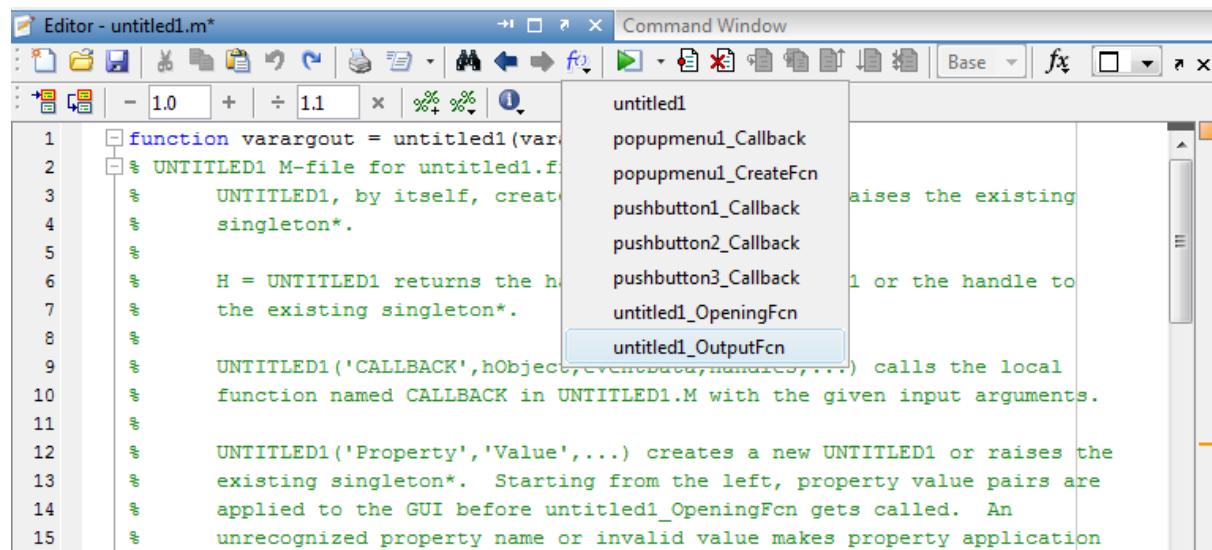
Bu GUI uygulamasında amaçlanan kendisine komut satırından verilen dış parametreleri alarak kendi içerisinde yorumlamak ve buna göre oluşturulacak GUI uygulama ekranının içeriğini (burada pencere başlığı ile text kutusunun string değerini) değiştirmek kullanıcıya sunmaktadır. Ayrıca, bu uygulama ile programcı şu konularda bilgilendirilmektedir:

- ❖ varargin ve varargout komutlarını kullanarak dışarıya bilgi gönderme ve dışarıdan verilen parametreleri yorumlamak
- ❖ uiwait(handles.figure1) komutu ile bir uygulama penceresinin aktif konumunun nasıl bloklandığı ve ne iş yaptığı,
- ❖ uiresume(handles.figure1) komutu ile bloklanan bir GUI penceresinin nasıl eski haline getirildiği,
- ❖ “modal” programlama tekniği ile tasarlanan bir GUI uygulamasında açılan bir pencerenin arka taraftaki bir başka pencerenin aktif kontrolünü ele geçirmesi için gerekli programmanın nasıl yapıldığıdır.

Ayrıca, bu uygulama MATLAB GUI tasarımlarında birden fazla form ile nasıl çalışılabileceği konusunda bir ipucu niteliği taşımaktadır. Bu uygulama ile giriş ve çıkış parametreleri kullanan GUI uygulama pencereleri vasıtasyyla kendi içinde birden fazla GUI arayüzü içeren GUI uygulamalarının programlama tekniği tanıtmaktadır.

## MATLAB GUI Arayüz Programlama

Bir GUI arayüzünün programlanması demek o çalışmanın kaydedildiği isimle aynı zamanla oluşturulan .m uzantılı dosya içerisinde kodlama satırlarının eklenmesi demektir. Bu dosyanın içine görebilmek, değişiklik yapabilmek için GUIDE çalışma ekranı penceresinden **View/M-File Editor komutu** işletilebilir. Ardından karşımıza aşağıdaki gibi bir pencere gelecektir.



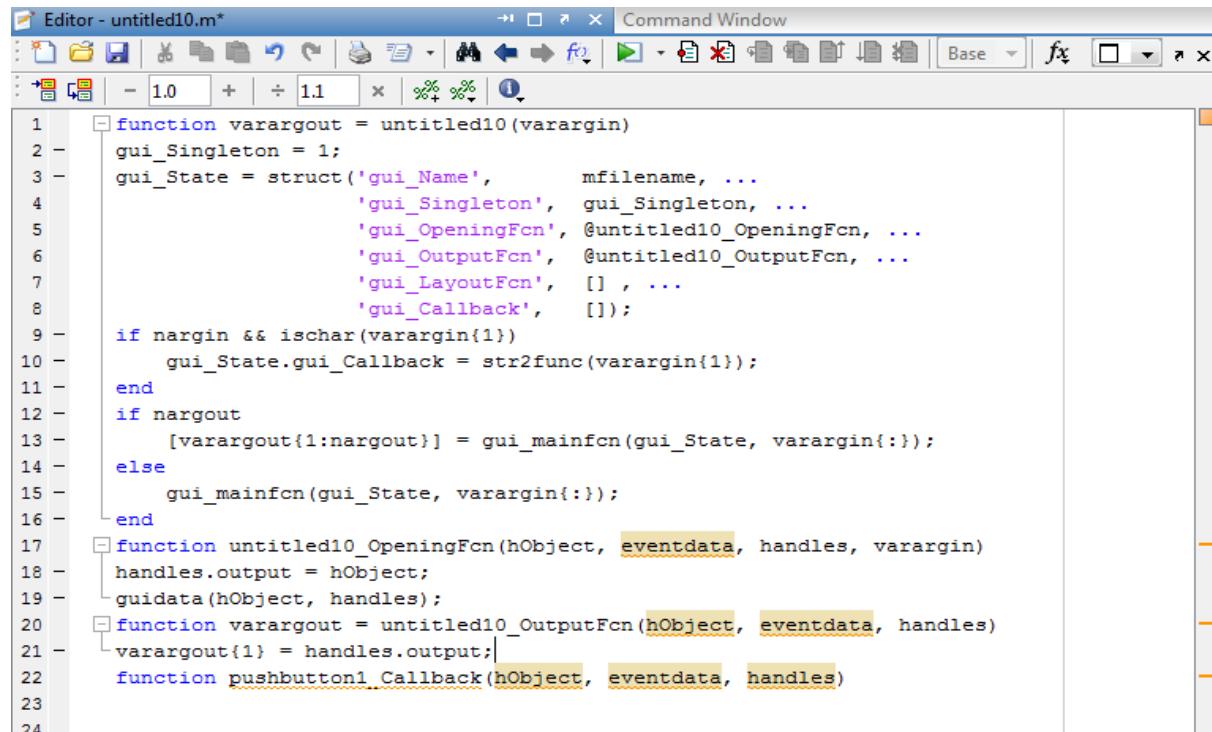
The screenshot shows the MATLAB Editor window with the file 'untitled1.m' open. The code defines a function 'untitled1' which creates a singleton GUI. A tooltip is displayed over the line 'untitled1\_Callback' in the code editor, providing documentation for the callback function:

```
untitled1
popupmenu1_Callback
popupmenu1_CreateFcn
pushbutton1_Callback
pushbutton2_Callback
pushbutton3_Callback
untitled1_OpeningFcn
untitled1_OutputFcn

UNTITLED1('CALLBACK', hObject, eventdata, handles, ...) calls the local
function named CALLBACK in UNTITLED1.M with the given input arguments.
```

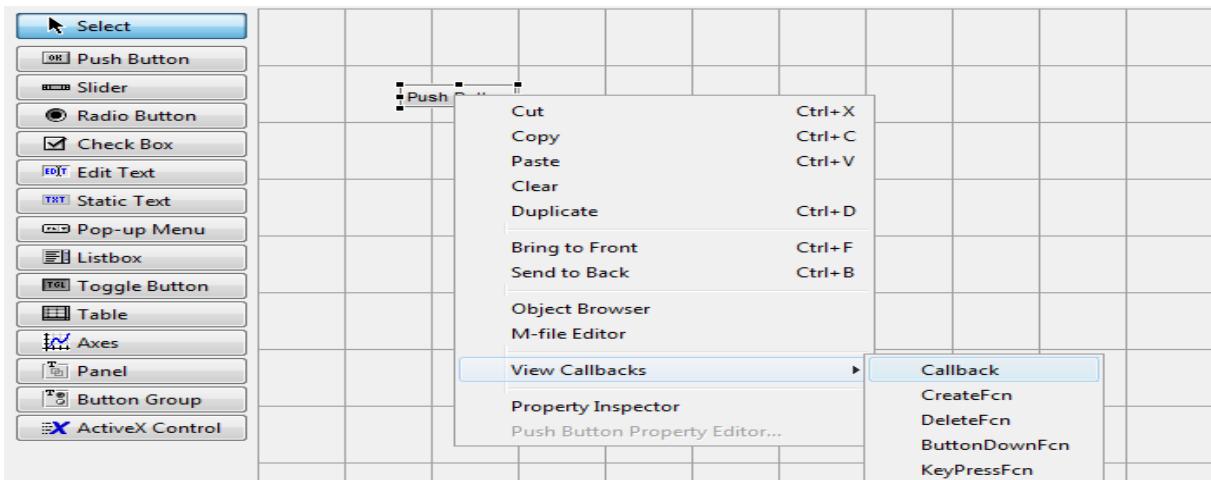
Yukarıdaki pencerede hazırlamış olduğumuz GUI tasarımlına ait kodlar gözükmektedir. Burada pek çok kodun hazır eklenmiş olduğu görülecektir. Bu kodlar otomatik olarak **MATLAB GUIDE** tarafından eklenmiştir.

Boş bir GUI arayüzüne MATLAB GUIDE tarafından eklenen hazır kodların içerisindeki açıklama satırlarının çıkarılmış hali aşağıdadır.



The screenshot shows the MATLAB Editor window with the file 'untitled10.m' open. The code is significantly simplified compared to the previous screenshot, as all explanatory comments and documentation have been removed. The code still defines a function 'untitled10' that creates a singleton GUI, but it lacks the detailed explanations provided in the first screenshot.

Biz burada ilgili butonlara ve liste kutularına ya da istenilen bir nesneye ait **callback** isimli alt program parçalarına ilgili kodları yazacağız. Bir nesneye ait **callback** in bulunduğu satırda gitmek için araç çubuğunda yer alan **f simgeli butona** tıklanır ve açılan listeden ilgili nesneye ait **callback** in ismi seçilir. Bu durum yukarıdaki pencerede de görülmektedir. Ayrıca, **GUIDE** çalışma ekranından da direk istenilen bir **callback** satırına gidilebilir. Bunun için ilgili nesne üzerinde sağ tıklanır ve açılan pencereden **View Callbacks** menüsünden ilgili **callback** tıklanması ya da ilgili nesne seçili **View/View Callbacks** yolu üzerinden açılan listeden gidilmek istenilen **callback** tıklanması yeterlidir.



Şimdi GUI arayüzünde yer alan tüm nesneler için **View/M-File Editor** yolundan kodlama penceresi açılıp, aşağıda yer alan tüm kodlar写字楼ın.

```
function varargout = untitled_ilk(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
'gui_Singleton', gui_Singleton, ...
'gui_OpeningFcn', @untitled_ilk_OpeningFcn, ...
'gui_OutputFcn', @untitled_ilk_OutputFcn, ...
'gui_LayoutFcn', [], ...
'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function untitled_ilk_OpeningFcn(hObject, eventdata, handles, varargin)
    handles.output = hObject;
    guidata(hObject, handles);
***
```

```

***  

handles.peaks=peaks(35);  

handles.membrane=membrane;  

[x,y] = meshgrid(-8:.5:8);  

r = sqrt(x.^2+y.^2) + eps;  

sinc = sin(r)./r;  

handles.sinc = sinc;  

handles.current_data = handles.peaks;  

guidata(hObject, handles);  

surf(handles.current_data)  

function varargout = untitled_ilk_OutputFcn(hObject, eventdata, handles)  

varargout{1} = handles.output;  

function pushbutton1_Callback(hObject, eventdata, handles)  

surf(handles.current_data);  

function pushbutton2_Callback(hObject, eventdata, handles)  

mesh(handles.current_data);  

function pushbutton3_Callback(hObject, eventdata, handles)  

contour(handles.current_data);  

function popupmenu1_Callback(hObject, eventdata, handles)  

str = get(hObject, 'String');  

val = get(hObject, 'Value');  

switch str{val};  

case 'Peaks' % User selects peaks.  

handles.current_data = handles.peaks;  

case 'Membrane' % User selects membrane.  

handles.current_data = handles.membrane;  

case 'Sinc' % User selects sinc.  

handles.current_data = handles.sinc;  

end  

guidata(hObject,handles)  

function popupmenu1_CreateFcn(hObject, eventdata, handles)  

if ispc  

set(hObject,'BackgroundColor','white');  

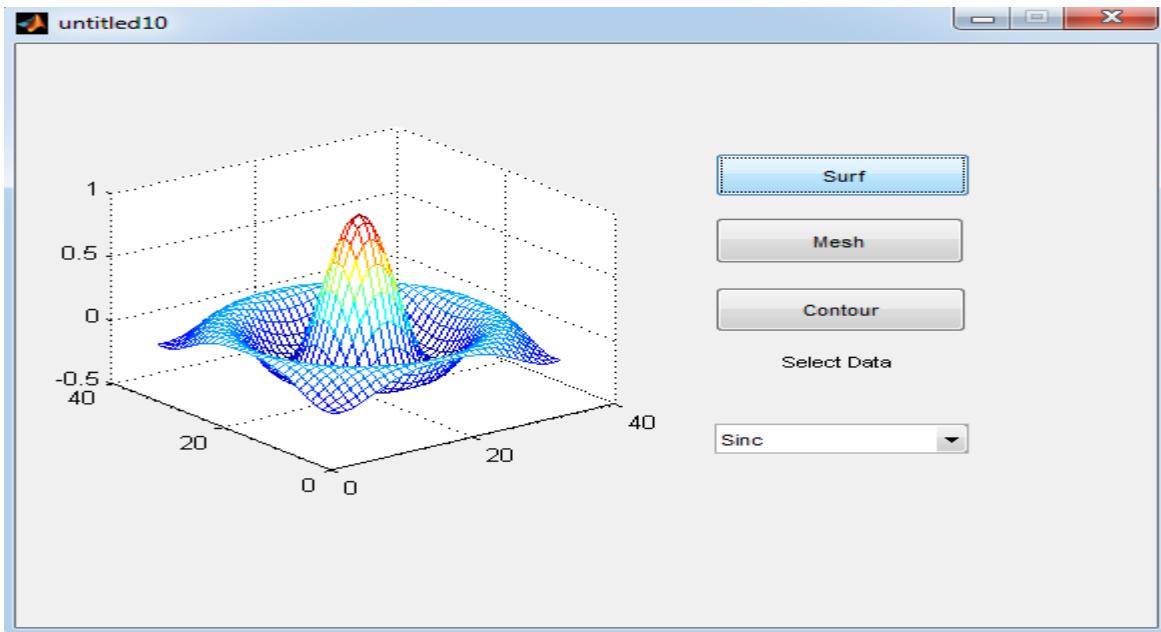
else  

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));  

end

```

Burada teker teker nesnelerin üzerinde sağ tıklayıp **View Callback** ve ilgili **Callback** satırına gidilip ayrı ayrı da yazılabildi. Burada göstermek amaçlı olduğu için kodlar bu şekilde direk verilmiştir. Kodlama satırlarında % işaretü ile başlayan satırlar açıklama satırları olup, bu satırlar herhangi bir komut olarak görülmeler sadece açıklama amacı taşırlar. Tüm işlemler tamamladığını göre **Tools/Run** komutu ile **GUI** uygulamamızı çalıştırduğumızda aşağıdaki gibi bir ekran ile karşılaşılır. Aşağıdaki ekrandaki gibi görüntü elde etmek istiyorsak ilk olarak boş olan GUIDE ekranına gerekli olan axes, push button, static text ve pop-up menu nesnelerini eklemiş olmamız gereklidir. Aksi halde ekranada sadece grafik görünür.



Burada yazılan kod parçalarını (callback rutinlerini) kısaca açıklayalım.

```
function varargout = untitled_ilk(varargin)
```

Yukarıdaki function bloğu GUIDE tarafından otomatik olarak oluşturulur. Burada GUI uygulamasına komut satırından gönderilen parametrelerin alınması ve GUI uygulaması çalıştırıldıktan sonra bir fonksiyon olarak dışarıya gönderilecek parametrelerin tanımlanması ile ilgili kod satırları mevcuttur.

```
function untitled_ilk_OpeningFcn(hObject, eventdata, handles, varargin)
```

Bu fonksiyon GUI arayüzü ekrana gelmeden (visible olmadan) hemen önce çalıştırılacak kodları içerir. Örneğin böyle bir callback bir GUI uygulaması çalışmadan önce initialization işlemlerinin yapılması ya da bazı GUI nesne özelliklerinin değiştirilmesi istendiğinde kullanılabilir. Ayrıca, varargin giriş parametresi kullanılarak da MATLAB komut satırından girilen parametre değerleri GUI uygulaması içinde kullanılmak üzere bu blokta alınır.

```
function varargout = untitled_ilk_OutputFcn(hObject, eventdata, handles)
```

Bu fonksiyon bloğu bir GUI uygulaması hafızadan silinip programı sonlandırılmadan hemen önce (destroy edilmeden önce) çalıştırılacak komutlar içerir. Ayrıca, komut satırına gönderilecek çıkış parametre değerleri de bu blok tarafından varargout değişkeni kullanılarak işleme konulur.

```
function pushbutton1_Callback(hObject, eventdata, handles)
```

Bu callback bloğu pushbutton1 isimli buton (istenilse bu buton ismi butonun Tag özelliğine özellikler penceresinden yeni bir isim verilerek de değiştirilebilir.) ki burada Surf stringine sahip olan buton tıklandığı zaman çalıştırılacak komutları içerir.

```
function pushbutton2_Callback(hObject, eventdata, handles)
```

Bu callback bloğu da benzer şekilde pushbutton2 isimli buton ki burada Mesh stringine sahip olan buton tıklandığı zaman çalıştırılacak komutları içerir.

```
function pushbutton3_Callback(hObject, eventdata, handles)
```

Bu callback bloğu da benzer şekilde pushbutton3 isimli buton ki burada Contour stringine sahip olan buton tıklandığı zaman çalıştırılacak komutları içerir.

```
function popupmenu1_Callback(hObject, eventdata, handles)
```

GUI arayüzüne eklenmiş olan popup\_menu nesnesinden herhangi bir eleman tıklanıp seçildiği zaman çalışması istenilen kod parçaları bu callback altında yazılır.

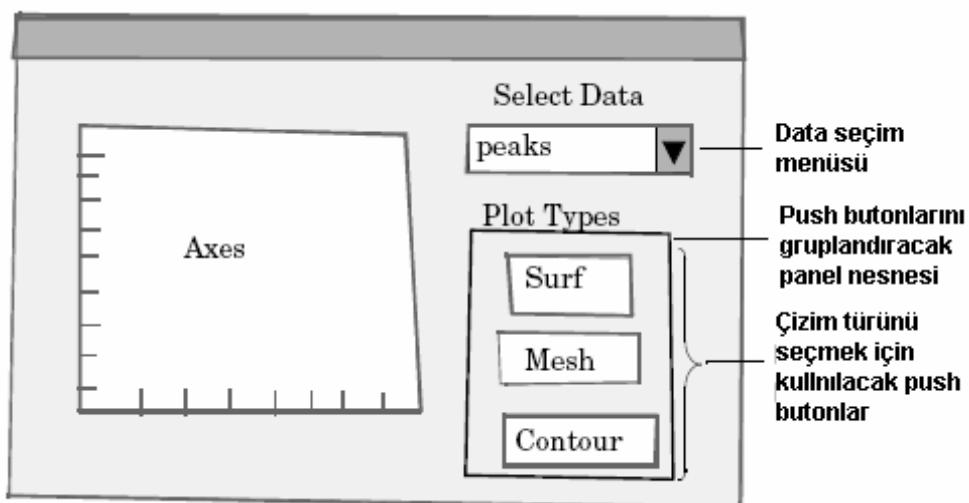
```
function popupmenu1_CreateFcn(hObject, eventdata, handles)
```

Bu callback GUIDE tarafından otomatik olarak oluşturulmuş olup, popup\_menu nesnesi uygulama ekranına gelmeden (visible olmadan) ve de oluşturulmadan önce koşturulacak program satırlarını içerir.

## **M-FILE PROGRAMLAMA YÖNTEMİ İLE GUI TASARIMI OLUŞTURMA**

Burada GUIDE gibi bir tasarım aracı kullanılmaz. Sadece kod satırları yazılarak hem GUI arayüzü hem de bu arayüzün koşturduğu komut satırları aynı dosya içerisinde yazılır. Bu dosyalar .m uzantısına sahiptirler.

Bir GUI arayüzüne bu yöntemle oluşturabilmek için öncelikle tasarım öncesi arayüzün bir planı taslak halinde bir kâğıt üzerine çizilmelidir. Çünkü burada tüm işlemlerin yapılması muazzam bir çalışma ve ölçeklendirme ile belirlenen nesnelerin uygun yerlere kullanışlı bir GUI arayüzü çıkarmak üzere bir araya gelmesi tamamı ile GUI tasarım ve programcısının yazdığı kodlar ile gerçekleştirilecektir.



Yukarıdaki pencerede görüldüğü üzere bir önceki sayfalarda anlatılan örnek GUI tasarımlının taslak görüntüsü görülmektedir. Bu şekilde nesnelerin yerleri tespit edildikten sonra bir GUI uygulaması oluşturulmak üzere programlama yöntemi ile tasarıma geçilebilir.

Şimdi MATLAB komut satırından “**edit**” komutunu verelim. Karşımıza boş bir m file dosya gelecektir. Genel olarak programlama yolu ile tasarlanacak GUI uygulaması komut satırları aşağıda belirtilen yapıda olmalıdır. Burada örneğin GUI uygulamamızın adı MYGUI olsun.

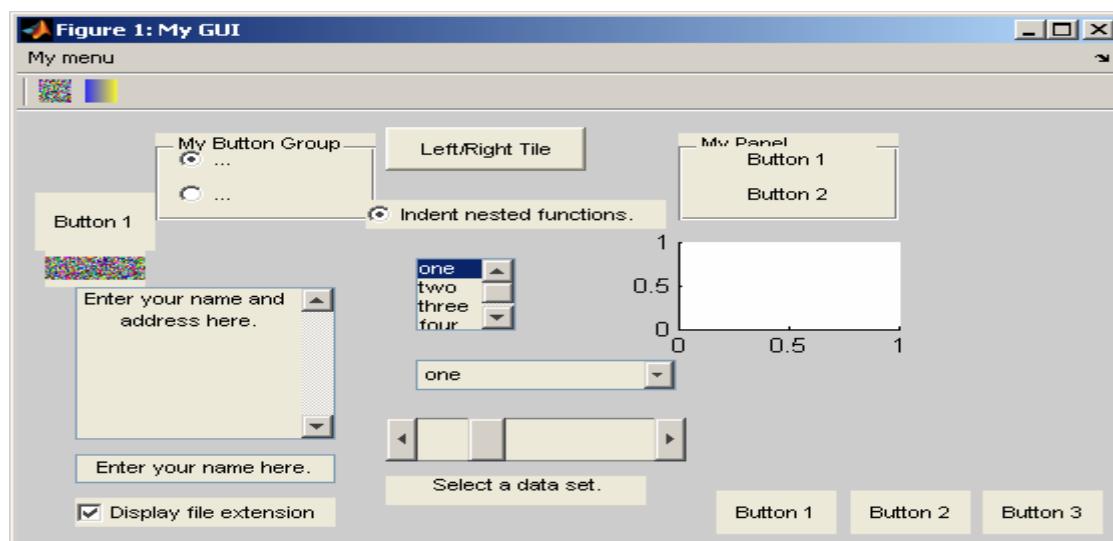
**function varargout = mygui(varargin)** : MYGUI uygulaması için **mygui.m** dosyası ilk satırı

- ❖ MYGUI Kısa bir GUI uygulaması ile ilgili açıklayıcı bilgi
- ❖ Bir adet boş açıklama satırına kadar bu satır ve sonra gelen satırlar MATLAB komut satırında GUI uygulamasını, açıklayıcı ve **help komutu** ile kullanıcıya sunulan yardım satırlarını içerir.
- ❖ Burada **help** satırlarını kod satırlarından ayırmak için bir adet boş açıklama satırı konulur.
- ❖ GUI uygulamasının giriş parametre alınması ve ilk ön hazırlık işlemleri bloğu
- ❖ GUI nesnelerinin oluşturulması ile ilgili satırlar bloğu
- ❖ **Callback’ler** öncesi ön hazırlık işlemleri bloğu
- ❖ MYGUI için gerekli **callback** fonksiyonları
- ❖ MYGUI için kullanılacak fonksiyonlar bloğu

**end**: Bu komut fonksiyon bloğunun sonunu belirtmek için konulmuştur.

Yukarıdaki yapıyı oluşturacak şekilde komutlar MYGUI isimli GUI uygulaması için **mygui.m** isimli dosyaya kaydedilir. Bu GUI uygulamasını çalıştmak için de MATLAB komut satırından sadece “**mygui**” komutunun verilmesi yeterlidir. Bu şekilde uygulama penceresi karşımıza gelecektir. Şu aşamada herhangi bir kod yazılmadığı herhangi bir şey olmayacağından emin olmak isterseniz, **mygui.m** dosyasını çalıştırın. Ancak, yazılmış olsayıdı GUI penceresi görülecekti.

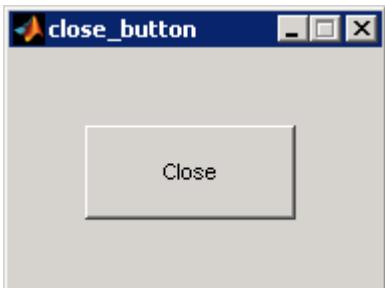
Yukarıda bahsedilen **mygui.m** dosyasının içeriğine gerekli kodlar eklendiğinde GUI arayüzü aşağıdaki şekilde görülebilir.



## **GUI Nesnelerinin Programlanması**

Burada her bir GUI nesnesinin programlama esnasında sıkılıkla hangi özelliklerinin ve nasıl bir teknik ile kullanıldığı anlatılacaktr.

**Push Button:** Aşağıdaki gibi bir arayüz için butona tıklandığında GUI uygulamasını kapatacak bir örnekle bir push buton kullanımını açıklayalım.



Böyle bir GUI çalışma alanında fare işaretçisi **push buton** üzerinde iken **View Callbacks/Callback komutunu** verelim ve açılan **callback** bloğuna aşağıdaki komutları ekleyelim.

```
function pushbutton1_Callback(hObject, eventdata, handles)
display Goodbye
close(handles.figure1);
```

Böylece Close butonu tıklandığında GUI uygulaması sonlandırılmış olacaktır.

Başa bir örnekte de bir push butonun üzerine nasıl resim eklendiğini inceleyelim. Yukarıdaki örnekte **Callback** bloğuna önceki program satırları yerine aşağıdaki kodları yazalım.

```
a(:,:,1) = rand(16,64);
a(:,:,2) = rand(16,64);
a(:,:,3) = rand(16,64);
set(hObject,'CData',a)
```

Bu GUI uygulamasını Tools menüsünden Run komutunu kullanarak çalışıralım ya da araç çubuğundan Run düğmesine basalım. Gelen GUI uygulamasında butona tıkladığımızda üzerine rasgele renklerden oluşan bir resmin atandığı görülecektir. Bu durum Şekil 5.72'de de görülmektedir.



**Toggle Buton:** Bir toggle buton çift durumlu çalışır. Bir kere tıklandığında basılı kalır. Bir daha tıklanırsa basılı kalmayıp eski konumuna geri döner. Böyle bir nesnede geçerli buton konumunu öğrenmek ve kullanabilmek için aşağıdaki komut satırları kullanılmalıdır.

```
function togglebutton1_Callback(hObject, eventdata, handles)
button_state = get(hObject, 'Value');
if button_state == get(hObject, 'Max')
    % Toggle buton basıldığında yapılacak işlemler
...
elseif button_state == get(hObject, 'Min')
    % Toggle buton basılmadığı durumda yapılacak işlemler
...
end
```

**Radio Buton:** Radio buton Buton Group nesnesi ile birlikte kullanıldığından daha etkili sonuçlar alınır. Ancak, kodlama yolu ile de radio butonların konumu kontrol edilebilir. Bir radio butonun basılıp basılmadığının kontrolü için şu kodlar kullanılabilir:

```
if (get(hObject, 'Value') == get(hObject, 'Max'))
    % Radio buton basıldığında yapılacak işlemler
else
    % Radio buton basılmadığı durumda yapılacak işlemler
end
```

**Check Box:** Check Box nesnesinin konum kontrolü de radio butonlarındaki benzer şekildedir.

```
if (get(hObject, 'Value') == get(hObject, 'Max'))
    % Checkbox nesnesi işaretlendiğinde yapılacak işlemler
else
    % Checkbox nesnesi işaretlenmediği durumda yapılacak işlemler
end
```

**Edit Text:** Bilgi girişi amacıyla sıkılıkla kullanılan edit text nesnesinin string içerik bilgisini almak için ilgili komut satırları şöyledir:

```
function edittext1_Callback(hObject, eventdata, handles)
user_string = get(hObject, 'String');
% Callback bloğunun devamı ve diğer komutlar
```

Ancak, burada alınan bilgiler string tiptedir ve sayısal olarak kullanılamazlar. Sayısal olarak kullanabilmek için öncelikle edit box içerikleri sayısal dönüştürülmelidir. Daha sonra eğer ki hatalı bir giriş söz konusu ise hata kontrol deyimlerinin kullanılması ile bu durum giderilmelidir. Böyle bir durum için kullanılabilen komut satırları aşağıda gösterilmiştir.

```
function edittext1_Callback(hObject, eventdata, handles)
user_entry = str2double(get(hObject,'String'));
if isnan(user_entry)
errordlg('Sayısal bir değer girilmelidir!..','Hatalı Giriş','modal')
return
end
% Callback bloğunun devamı ve diğer komutlar
```

Bu komut satırlarında ayrıca bir hata durumu oluştuğunda errordlg komutu ile kullanıcıya hatalı bir giriş yaptığı uyarı diyalog penceresi gösterilmektedir.

Edit Text nesnesinin callback satırları ancak kullanıcı baska bir nesneye ya da gUI yüzeyine tıkladığı veya edit nesnesi içinde iken Enter tuşuna bastığı (çoklu giriş kutusu ise Ctrl + Enter tuş kombinasyonu kullanıldığı) zaman icra edilecektir. Aksi takdirde kullanıcı bir değer edit veya text nesnesine girerken bu callback satırları çalışmayaacaktır.

**Static Text:** Edit Text nesnesi ile benzer özelliklere sahiptir. Bu nesnenin edit text ten tek farkı kullanıcıdan bilgi girişi alınamamasıdır, sadece bilgi göstermek amaçlı kullanılır. Kodlama mantığı edit text nesnesi ile aynıdır.

**Slider:** Bir kaydırıcı (slider) nesnesinin geçerli değerini program yoluyla okumak için gerekli komut satırları şöyle olmalıdır.

```
function slider1_Callback(hObject, eventdata, handles)
slider_value = get(hObject,'Value');
% Callback bloğunun devamı ve diğer komutlar
```

Bir kaydırıcı nesnesinin en küçük ve en büyük değerlerinin de ayarlanması bir programcı için gereklidir. Bunun için bu nesnenin Max ve Min özellikleri kullanılmalıdır.

**List Box:** List Box nesnelerinin liste tipindeki string içeriğinin kullanılabilmesi için bu nesnelerin Value ve String özellikleri birlikte kullanılır. Kodlama mantığı şu şekilde olacaktır:

```
function listbox1_Callback(hObject, eventdata, handles)
index_selected = get(hObject,'Value');
list = get(hObject,'String');
item_selected = list{index_selected}; % Hücre dizisinden çevirme işlemi
% to string
```

**Pop-Up Menu:** Popup menü nesnelerinde seçilen bir öğenin hangisi olduğu anlamak için bu nesnelerin Value özelliğinden yararlanılır. Kodlama örneği aşağıda gösterilmiştir.

```

val = get(hObject,'Value');
switch val
    case 1
        % Birinci öğe seçili iken yapılacak işlemler
    case 2
        % İkinci öğe seçili iken yapılacak işlemler
    % Callback bloğunun devamı ve diğer komutlar

```

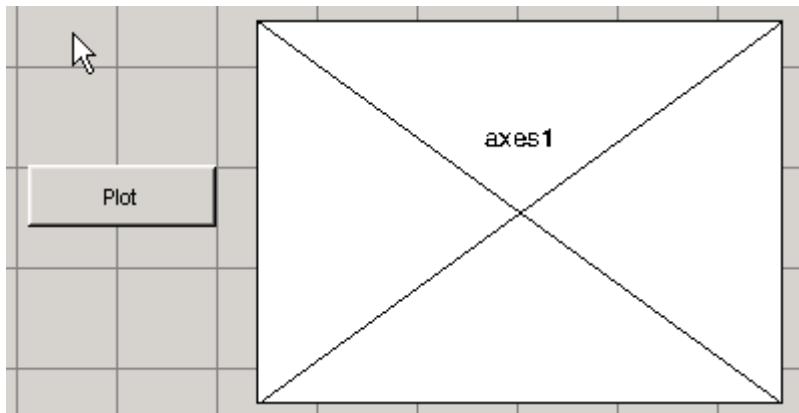
Eğer ki programcı seçilen ögenin stringini öğrenmek isterse şu komut satırları kullanılabilir:

```

function popupmenu1_Callback(hObject, eventdata, handles)
val = get(hObject,'Value');
string_list = get(hObject,'String');
selected_string = string_list{val}; % Hücre dizisinden çevirme işlemi
% to string
% Callback bloğunun devamı ve diğer komutlar

```

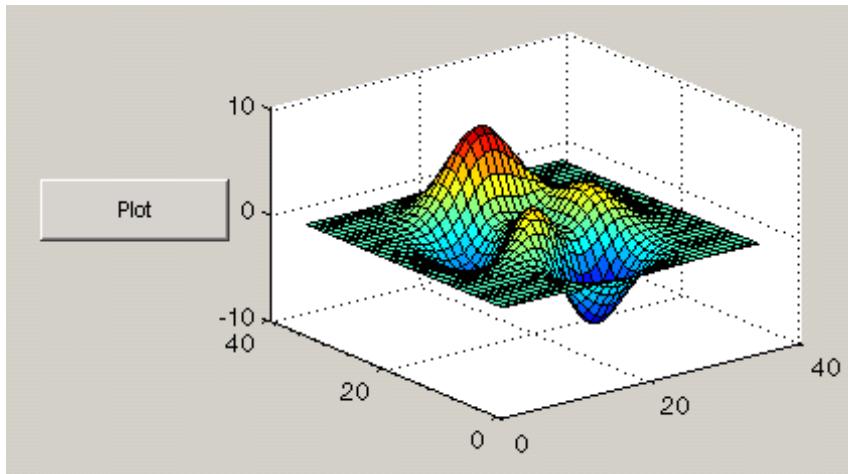
**Axes:** Grafik çizimlerinin kullanıcıya sunulmasında sıkılıkla kullanılan bir nesne olan axes için aşağıdaki gibi bir GUI arayüzüne sahip olunduğu düşünülmüş olsun.



Burada Plot butonunun callback bloğuna şu komut eklensin ve ardından Run komutu ile bu GUI uygulamasını çalıştırılsın.

```
surf(peaks(35));
```

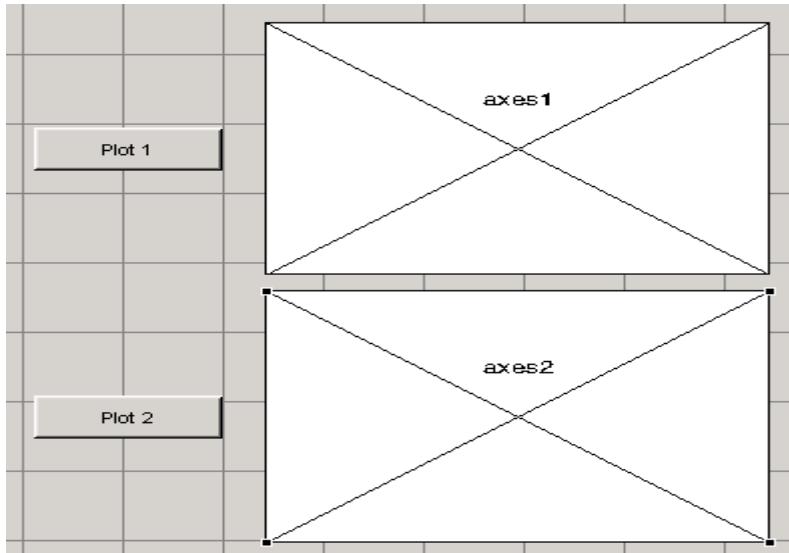
Ekrana gelen GUI uygulama penceresinde Plot düğmesini tıkladığımızda aşağıdaki gibi bir ekran görüntüsü ile karşılaşılır.



Göründüğü üzere tek axes nesnesi içeren GUI tasarımlarını programlamak kolay gözükmemektedir. Ancak, eğer ki bir GUI arayüzü birden fazla axes nesnesine sahip ise bu takdirde hangi axes nesnesi o an aktif ise o nesne üzerinde çizimimiz gözükecektir. Aktif axes nesnesinin hangisi olacağı şu komutla belirtilir:

### **axes(handles.axes1)**

Bir sonraki axes nesnesi ile ilgili örneğimizde çoklu axes nesnelerinin programlanması ait bilgilere yer verilmiştir. Bu örnek için GUI arayüzünün aşağıdaki gibi olduğu kabul edilsin.



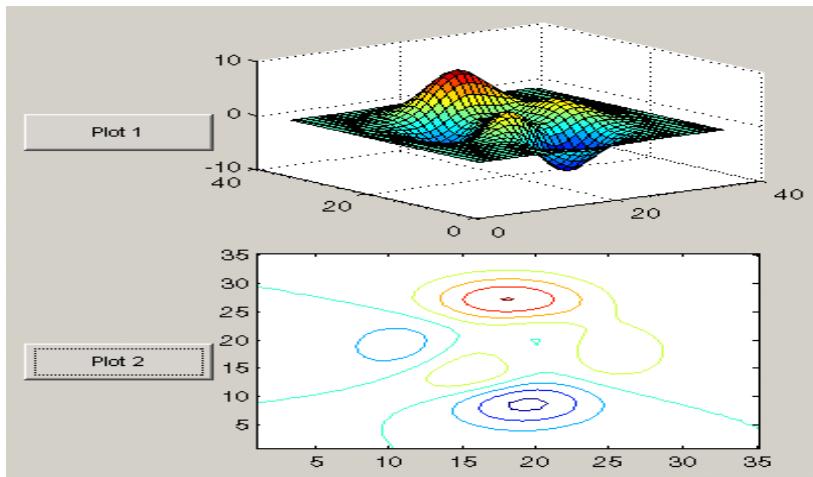
Bu uygulamada Plot 1 butonuna ait callback satırlarına

```
surf(handles.axes1, peaks(35));
```

Plot 2 butonuna ait callback satırlarına da

```
contour(handles.axes2, peaks(35));
```

komutları eklensin. Daha sonra da bu GUI uygulaması çalıştırılsın. Gelen uygulama penceresinde ayrı ayrı Plot1 ve Plot 2 butonlarını tıklansın. aşağıdaki gibi bir görüntü ile karşılaşılır:



Bu uygulama bize istenilen sonucu vermektedir. Ancak, aynı işlev farklı yöntem kullanılarak da gerçekleştirilebilir.

Plot 1 butonuna ait callback satırlarına önceden yazılan komut satırları yerine

Plot 2 butonuna ait callback satırlarına önceden yazılan komut satırları yerine

```
axes(handles.axes2);
contour(peaks(35));
```

satırları yazısın ve bu uygulama tekrar çalıştırılsın. Butonlar tıklandığında sonuç değişimeyecektir. Ancak, axes() komutunun kullanılması bir programcı olarak daha kolay ve sade bir kodlama anlamı taşıması bakımından genellikle tercih edilir.

**Panel:** Bu nesneler programlama amacı taşımayan yapıdadır. Panellerin kullanım amacı görsel olarak GUI uygulamasını zengin kılmak ve kullanıcıya yapacağı işlemler ile ilgili kullanımını kolaylaşturmaktır. Ancak, bir GUI uygulamasının koşturulması anında GUI figure alanının boyutlarının değiştirilmesi söz konusu ise, bu takdirde panel nesnelerinin ResizeFcn metodu kullanılabilir.

**Button Group:** Örnek olarak aşağıdaki gibi görülen bir GUI arayüzüne sahip olunduğu düşünülsün. Böyle bir durumda Buton Group nesnesi özelliğinden ötürü bu dört nesne toggle durumlu olarak (yani herhangi bir anda yalnızca biri seçili olabilir şekilde) çalışacaktır. Böyle bir uygulamada hangi buton seçili ise yaptırılmak istenilen programlama işlemleri şöyle olacaktır:

```

function uibuttongroup1_SelectionChangeFcn(hObject,...  

eventdata,handles)  

switch get(hObject, 'Tag') % Seçili nesnenin Tag bilgisini alma  

case 'radiobutton1'  

% radiobutton1 seçili ise yapılacak işlemler  

case 'radiobutton2'  

% radiobutton2 seçili ise yapılacak işlemler  

case 'togglebutton1'  

% togglebutton1 seçili ise yapılacak işlemler  

case 'togglebutton2'  

% togglebutton2 seçili ise yapılacak işlemler  

% Daha fazla sayıda buton varsa koşulların kontrolü böylece devam eder.  

otherwise  

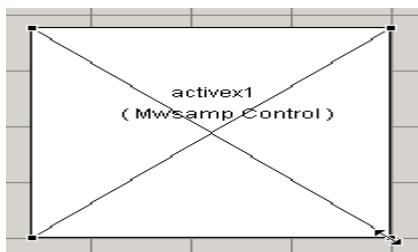
% Hiçbir buton seçili değilse yapılacak işlemler  

end

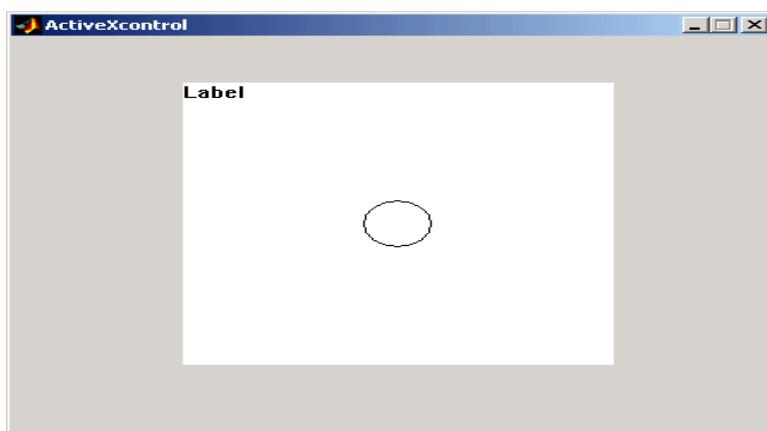
```

**ActiveX Component:** Active X nesneleri ile MATLAB GUI uygulamalarının esnekliği arttırmıştır. Böylece sadece GUI'nın kendi nesneleri ile tasarımcı sınırlı kalmamış olup, pek çok farklı nesneyi de alıp kullanabilir.

GUI yüzeyine önce bir ActiveX nesnesi ekleyelim ve karşımıza gelen component listesinden **“Mwsamp Control”** ü tıklayıp **Create** butonuna basalım. Eklenen bu ActiveX nesnesinin köşesinde boyutlarını değiştirmek ve büyütmek mümkündür. GUI çalışma alanındaki görüntü aşağıdaki gibi olacaktır:



Bu GUI uygulamasını çalıştırıldığında karşılaşılan arayüz ekranı aşağıda gösterilmiştir.



Tekrar tasarım ortamına geçilsin, **Activex** nesnesi seçili **Property Inspector** penceresinden özelliklerine bakılsın.



Bu ActiveX nesnesinin iki özelliği vardır. Bunların işlevi şu şekildedir:

- ❖ Label özelliği, kullanıcıya ekranda sunulacak bilgidir.
- ❖ Radius özelliği, ekranda gözükecek dairenin yarıçap uzunluğuudur.

Şimdi de bu ActiveX nesnesi ile ilgili programlama teknikleri açıklayalım. Öncelikle ActiveX nesnemizin **Click Callback** satırlarına aşağıdaki komutları ekleyelim.

```
hObject.radius = .9*hObject.radius;
hObject.label = ['Yaricap = ' num2str(hObject.radius)];
refresh(handles.figure1);
```

Böylece fare işaretçisi ile her ActiveX nesnesi üzerinde tıkladığımızda bu callback'teki komutlar icra edilecektir. Burada yapılan her tıklanmada daire yarıçapının artırılıyor olmasıdır. Ayrıca, nesnemizin Label özelliğinden faydalalarak dairemizin yarıçap değeri ekrana yazdırılmaktadır. Burada unutulmaması gereken nokta bir ActiveX nesnesi ile ilgili özellikler değiştirildiği zaman mutlaka bunların o nesneye uygulanabilmesi ve uygulama arayüzünde yapılan değişikliklerin gözlenebilmesi için geçerli GUI figure yüzeyinin refresh edilmesi gerekliliğidir. Bunun yanında burada Label özelliği string tipi verileri tuttuğu için yarıçap sayısal değerden num2str komutu kullanılarak string tipi bilgiye dönüştürülmemekte ve ardından Label özelliğine atanmaktadır. Aksi takdirde programımızın çalıştırılması sırasında hata ile karşılaşılacaktır.

ActiveX nesnelerinin program yoluyla özelliklerinin değiştirilmesi gerekiğinde direkt olarak ActiveX nesnelerinin özellikleri get metodu kullanılarak alınabilir. Bu durumla ilgili olarak şu komut satırları incelenebilir:

```
handles.activex1.radius = ...
get(hObject,'Value')*handles.default_radius;
handles.activex1.label = ...
['Radius = ' num2str(handles.activex1.radius)];
refresh(handles.figure1);
```