

BÖLÜM 2

MATLAB

BÖLÜM 2.1

MATLAB 'in Temel Özellikleri

2.1.1 MATLAB Nedir?

MATLAB (**MAT**rix **LAB**oratory) programlama dili olarak özellikle biliminsanları ve mühendisler için tasarlanmış ileri düzeyli **interaktif** bir yazılımdır.

MATLAB, kolay kullanımı, güçlü, grafiksel gösterimi de içeren kapsamlı bir programlama dilidir.

MATLAB hesaplama ve görsellik içeren bir programdır.

MATLAB komutlar tarafından kontrol edilir ve programlanabilirdir. MATLAB'da önceden tanımlanmış yüzlerce arşiv fonksiyonları bulunur ve bu fonksiyonlar kendinizin yazacakları ile daha da genişletilebilir.

MATLAB, güçlü komutlara sahiptir. Örneğin tek bir komutla bir lineer cebirsel denklem takımını çözebilir, ileri düzeyli matris işlemleri yapabilir.

MATLAB iki- ve üç-boyutlu güçlü grafik donanımlarına sahiptir.

MATLAB diğer programlama dilleri ile birlikte kullanılabilir. Örneğin FORTRAN ile elde edilen hesap sonuçları, MATLAB ile kolayca grafiksel gösterimlere dönüştürülebilir.

2.1.2 Hesap Makinası olarak MATLAB

MATLAB basit hesapları bir hesap makinası gibi yapabilir. Örneğin komut (command) satırına $5+3$ yazıp ENTER tuşuna basarsanız:

```
>> 5+3 ↴
```

```
ans =
```

```
8
```

```
>>
```

elde edilir. Tek bir satırda bir çok işlemi yapabilirsiniz:

```
>> 5+3-4*6+7/2 ↴
```

```
ans =
```

```
-12.5000
```

```
>>
```

2.1.2 Hesap Makinası olarak MATLAB

MATLAB'daki basit matemetiksel işlemler Tablo 2.1'de verilmiştir.

Tablo 2.1

İşlem	MATLAB simbolü	Örnek	Sonuç
Toplama	+	5+3	8
Çıkarma	-	5-3	2
Çarpma	*	6*2	12
Bölme	/	6/2	3
Üs	^	6^2	36

2.1.3 Değişkenlerin Tanımlanması

Değişkenlerin tanımlanması, değişken adının yazılmasından sonra eşitlik işaretini ve bir değer veya matematiksel ifade şeklinde yapılır. MATLAB büyük ve küçük harflere duyarlıdır.

```
>> A=5 ↴
```

```
A =
```

```
5
```

```
>>
```

```
>> a=6 ↴
```

```
a =
```

```
6
```

```
>>
```

```
>> A, a ↴
```

```
A =
```

```
5
```

```
a =
```

```
6
```

```
>>
```

2.1.3 Değişkenlerin Tanımlanması

Değişken adları maksimum 63 karakter uzunluğunda olabilir ve bu adlar harfleri, rakamları alt_çizgi karakteri gibi karakterleri içerebilir. 63 karakterden uzun değişken adları 63. Karakterden sonra kesilir.

```
>> A5=6 ↴
```

```
A5 =
```

```
6
```

```
>>
```

```
>> b17=27.987 ↴
```

```
b17 =
```

```
27.9870
```

```
>>
```

```
>> Aday_No=297.9 ↴
```

```
Aday_No =
```

```
297.9000
```

```
>>
```

2.1.3 Değişkenlerin Tanımlanması

Hesaplarda değişken adları kullanılabilir.

```
>> a=5; ↴  
>> b=3; ↴  
>> c=8; ↴  
>> x=a+b/c ↴  
x =  
    5.3750  
>>
```

Tek bir sayısal değere sahip bir değişken MATLAB'da 1×1 boyutunda bir dizi olarak algılanır. Örneğin 3×4 boyutunda bir dizi aşağıdaki gibi tanımlanabilir:

```
>> m=[1 2 3 4; 66 76 88 44; 567 45 76 0] ↴  
m =  
    1     2     3     4  
   66    76    88    44  
  567   45    76     0  
>>
```

2.1.3 Değişkenlerin Tanımlanması

Herhangi bir anda hafızada saklanan değişkenleri görmek için WHO veya WHOS komutları kullanılabilir.

Hafızadakileri silmek ise CLEAR komutu kullanılabilir.

```
>> clear
```

```
>>
```

```
>> who
```

```
>>
```

$\sin(x)$, $\text{abs}(x)$, fprintf , $\log(x)$ gibi birçok MATLAB komutu fonksiyonlar şeklinde mevcuttur. Örneğin $\sin(1.5)$:

```
>> sin(1.5)
```

```
ans =
```

```
0.9975
```

```
>>
```

2.1.4 Fonksiyonlar

Genel olarak fonksiyonlar birden fazla giriş değerlerine sahip olabilirler ve birden fazla değeri geri döndürebilirler. Örneğin SIN fonksiyonu

```
>> q=sin(1)
```

```
q =
```

```
0.8415
```

```
>>
```

şeklinde bir değeri giriş değeri olarak kabul edip tek bir değeri geri döndürür.
Fonksiyonlarla ilgili geniş bilgi ilerideki bölümlerde ele alınacaktır.

2.1.5 Formatlar

MATLAB'da değerler çeşitli formatlarda gösterilebilirler. En çok kullanılan format tipleri aşağıdaki tabloda verilmiştir.

Tablo 2.2

<u>MATLAB komutu</u>	<u>Görünüm</u>	<u>Açıklama</u>
format short	3.1416	varsayılan 5 hanelik ondalık gösterim
format long	3.14159265358979	15 hanelik ondalık gösterim
format short e	3.1416e+000	5 hane + üslü gösterim
format long e	3.141592653589793e+000	16 hane + üslü gösterim
format bank	3.14	2 ondalık hane (para için)
format +	+	pozitif, negatif, 0
format rat	355/113	oransal gösterim (oran)

Bu formatların hatırlanamaması durumunda komut satırına HELP FORMAT girilirse, MATLAB kullanılabilecek formatların bir listesini gösterecektir.

2.1.6 Ön Tanımlı Değişkenler

MATLAB bir takım önceden tanımlanmış değişkenlere sahiptir. Bunlardan bazıları Tablo 2.3'te verilmiştir.

Tablo 2.3

Değişken	Tanımı	Değeri
pi	(π) çemberin çevresinin çapına oranı	3.141592653589793e+000
eps	1 'e eklendiğinde, üstel gösterim oluşturacak en küçük sayı	2.220446049250313e-016
inf	(∞) sonsuz - örneğin 1/0.	inf
NaN	Tanımsız (Not a Number). 0/0, ∞/∞ ve $\infty \cdot 0$ gibi işlemlerin sonucu	NaN
i	-1 'in karekökü	i
j	-1 'in karekökü	j
realmin	Kullanılabilir en küçük pozitif reel sayı	2.225073858507201e-308
realmax	Kullanılabilir en büyük pozitif reel sayı	1.797693134862316e+308

2.1.6 Ön Tanımlı Değişkenler

```
>> format long  
>> eps  
ans =  
2.220446049250313e-016  
>>
```

```
>> inf  
ans =  
Inf  
>>
```

Dikkat edilirse *inf* değişkeni sayı olarak görülmeyeceği fakat bir sayı olarak algılanmaktadır.

```
>> a=1/0  
Warning: Divide by zero.  
a =  
Inf  
>>
```

2.1.6 Ön Tanımlı Değişkenler

Bu sayede bilgisayarın bir hata mesajı vererek bir programın koşturulmasının durdurulması önlenmiş olur.

Nan sonucu $1/0$, $inf \times 0$, or $inf + inf$ gibi geçersiz işlemlerin oluşması sonucu üretilir. Bu işlemler matematikte tanımlanmamıştır. Bu da yukarıdaki gibi bilgisayarın bir hata mesajı vererek koşmayı durdurmasını önlemek için kullanılmaktadır. ***Nan*** ile yapılan bir işlemin sonucu yine ***Nan***'dır.

```
>> 5+NaN
```

```
ans =
```

```
NaN
```

```
>> a=0*inf
```

```
a =
```

```
NaN
```

```
>>
```

2.1.7 Matrisler ve Vektörler

MATLAB birden fazla sayısal değeri tek bir değişkende depolayabilir. MATLAB bu değişkenleri diziler veya matrisler olarak ele almaktadır. Diziler şeklinde ele alındığında, diğer programlama dillerinde olduğu gibi MATLAB bu diziyi eleman eleman adresler. Matrisler şeklinde ele alındığında ise MATLAB bu kez bu değişkenlere matris operasyonlarını uygular.

Varsayılan olarak MATLAB dizi operasyonlarından ziyade matris cebrini kullanır.

Dizi operasyonları eleman-elemana operasyonlar şeklinde tanımlanır. Örneğin dizi çarpımında, birinci elemanla birinci eleman, ikinci elemanla ikinci eleman, vb. şeklinde yapılır. Matris çarpımında ise birinci matrisin bir satırı ile ikinci matrisin bir sutunu arasında yapılır.

Matris operatörleri, matris çarpımı, bölümü, üs alma, toplama, ve çıkarma ($*$, $/$, $^$, $+$, ve $-$) şeklindedir. Bu operatörler matris cebrine uygun işlev yaparlar.

Dizi operatörleri ise $.*$, $./$, $.^$, $+$, ve $-$ şeklindedir. Dikkat edilirse çarpma, bölme ve üs operatörleri önünde bir nokta (.) bulunmaktadır. Dizi çıkarma ve toplama işlemleri matris çıkarma ve toplama işlemleri ile aynıdır (eleman-elemana).

2.1.7 Matrisler ve Vektörler

MATLAB 'da matrisler ve vektörler köşeli parantezlerle [] oluşturulur. Satırlar noktalı-virgül (;) simbolü ile ayrılır. Örneğin tipik bir A matrisi oluşturmak için $A=[1\ 2\ 3; 4\ 5\ 6; 7\ 8\ 9]$ yazıp ENTER tuşuna basmak yeterlidir. Dikkat edilecek olursa aynı satırdaki büyükler boşluklar ile birbirinden ayrılmaktadır.

```
>> A=[1 2 3; 4 5 6; 7 8 9]
```

```
A =
```

1	2	3
4	5	6
7	8	9

```
>>
```

Aynı satırdaki sayılar aşağıda görüldüğü gibi birbirlerinden virgülerle (,) de ayrılabilir.

```
>> B=[9,8,7;6,5,4;3,2,1]
```

```
B =
```

9	8	7
6	5	4
3	2	1

```
>>
```

2.1.7.1 Matris Çarpımı

Matrisler * operatörü kullanılarak çarpılır:

```
>> c=A*B  
c =  
   30   24   18  
   84   69   54  
 138  114   90  
>>
```

Matris çarpımında değişme özelliği olmadığından $A*B$ nin $B*A$ ya eşit olmadığına dikkat edilmelidir.

```
>> c2=B*A  
c2 =  
   90   114  138  
   54    69   84  
   18    24   30  
>>
```

2.1.7.1 Matris Çarpımı

* operatörü bir matrisin bir skalerle çarpımı için de kullanılabilir:

```
>> d=5*A  
d =  
    5   10   15  
   20   25   30  
   35   40   45  
>>
```

5 skaler bir sayı olduğundan matrisin her elemanı 5 ile çarpılmıştır.

MATLAB ile satır ve sutun vektörleri de oluşturulabilir. Satır vektörü tek bir satırdan oluşan, sutun vektörü de tek bir sutundan oluşan vektördür.

```
>> x=[1,2,3]  
x =  
    1   2   3  
>>
```

2.1.7.1 Matris Çarpımı

```
>> y=[1;2;3]
y =
1
2
3
>>
```

2.1.7.2 Matrislerde Toplama ve Çıkarma

Aynı boyuttaki matrisler toplanabilirler ve çıkarılabilirler:

```
>> q=A-B
q =
-8   -6   -4
-2    0    2
 4    6    8
```

2.1.7.2 Matrislerde Toplama ve Çıkarma

Toplama ve çıkarma işlemleri matrislerle skalerler arasında ya gerçekleştirilebilir:

```
>> A+5  
ans =  
6 7 8  
9 10 11  
12 13 14  
>>
```

2.1.7.3 Bir Matrisin Tersi (invers matris)

Bir kare matrisin tersi **INV** fonksiyonu ile hesaplanır. Örneğin bir A matrisinin tersini almak için **inv(A)** yazıp ENTER tuşuna basmak yeterlidir:

2.1.7.3 Bir Matrisin Tersi (invers matris)

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

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. RCOND = 1.541976e-018.

```
ans =
```

```
1.0e+016 *
```

-0.4504	0.9007	-0.4504
0.9007	-1.8014	0.9007
-0.4504	0.9007	-0.4504

```
>>
```

MATLAB tersi almakta fakat sonucun doğru olmama ihtimalini de hatırlatmaktadır. Bunun nedeni matrisin singüler (tekil) olmaya yakın olmasıdır. Tekillikle ilgili ayrıntı ilerleyen derslerde ele alınacaktır.

Şimdi gelişigüzel başka bir matris oluşturup tersini alalım.

2.1.7.3 Bir Matrisin Tersi (invers matris)

```
>> A=rand(3)  
A =  
0.9501 0.4860 0.4565  
0.2311 0.8913 0.0185  
0.6068 0.7621 0.8214  
>>
```

RAND(m,n) fonksiyonu mXn boyutunda gelişigüzel bir matris oluşturur. Örneğin RAND(3) ise 3X3 boyutunda bir kare matris oluşturmak için kullanılabilir.

```
>> A_inv=inv(A)  
A_inv =  
1.6740 -0.1196 -0.9276  
-0.4165 1.1738 0.2050  
-0.8504 -1.0006 1.7125  
>>
```

2.1.7.4 Bir Matrisin Determinantı

Bir matrisin determinantını hesaplamak için DET fonksiyonu kullanılır.

```
>> B  
B =  
    9   8   7  
    6   5   4  
    3   2   1  
  
>>  
  
>> det(B)  
ans =  
    0  
>>
```

2.1.8 Karakterler (Strings)

MATLAB 'da karakter dizileri (strings) kesme işaretleri arasında yazılır. Bir karakter dizisi alfabetik harflerin, sayısal sembollerin, ve diğer sembollerin bir dizisidir. '*abcd*', 'fish', 'x(12)*z{23}', vb. birer karakter dizileridir. Karakter dizisinin bir değişkene atanması

```
>> x='abcd'  
x =  
abcd  
>>
```

şeklinde yapılır. Bir çok karakter, karakter dizisine konulabilir:

```
>> y='abc123@#${ }[ ]::'  
y =  
abc123@#${ }[ ]::  
>>
```

2.1.8 Karakterler (Strings)

Kesme işaretinin kendisinin de karakter dizisinde olması isteniyorsa , iki tane tek kesme işaretti ard arda yazılmalıdır:

```
>> word='didn''t'  
word =  
didn't  
>>
```

Eğer sadece tek bir kesme işaretini karakter olarak tanımlanacaksı:

```
>> x=''''  
x =  
'  
>>
```

2.1.8.1 Karakterler İndeksleme

Karakter dizisinin ilk karakterinin indeksi 1'dir. Karakter dizisinin herhangi bir karakterini adreslemek için, karakter dizisinin atandığı değişken adının yanında parantezle belirtmek yeterlidir. Aşağıda bir karakter dizisinin çeşitli karakterlerinin adreslenmesi görülmektedir:

```
>> a='abcd'
```

```
a =
```

```
abcd
```

```
>>
```

```
>> a(1)
```

```
ans =
```

```
a
```

```
>>
```

```
>> a(3)
```

```
ans =
```

```
c
```

```
>>
```

2.1.8.1 Karakterler İndeksleme

Bir karakter dizisinin belirli bir kısmını da adresleyebiliriz:

```
>> a(2:3)
```

```
ans =
```

```
bc
```

```
>>
```

2.1.8.2 Karakterlerin Eklenmesi

Karakter dizilerini birbirleri ile uc uca eklemek için aşağıdaki notasyon kullanılır:

```
New_string=[string1, string2, string3, ...]
```

Örneğin x_1 , x_2 , and x_3 dizilerini uc uca ekleuerek yeni diziler oluşturalım:

```
>> x1='abcd'
```

```
x1 =
```

```
abcd
```

```
>> x2='efgh'
```

```
x2 =
```

```
efgh
```

```
>> x3='ijklmnop'
```

```
x3 =
```

```
ijklmnop
```

```
>>
```

2.1.8.2 Karakterlerin Eklenmesi

```
>> x=[x1,x2]
```

```
x =
```

```
abcdefgh
```

```
>>
```

```
>> y=[x,x3]
```

```
y =
```

```
abcdefghijklmноп
```

```
>>
```

```
>> z=[y,'qrst']
```

```
z =
```

```
abcdefghijklmнопqrst
```

```
>>
```

```
>> mm=[z,a(1),x1(2:3)]
```

```
mm =
```

```
abcdefghijklmнопqrstabc
```

```
>>
```

2.1.8.3 Karakter Fonksiyonları

Karakterlerle çeşitli işlemler yapmak için MATLAB 'da bir çok fonksiyon bulunmaktadır. Aşağıda bunlardan bazıları verilmiştir:

LENGTH Bir karakter dizisindeki karakter sayısını verir.

```
>> length(mm)  
ans =  
23  
>>
```

strcmp (String Comparison) İki karakter dizisinin karşılaştırılmasında kullanılır.
Eğer karakter dizileri farklı ise 0, aynı ise 1 değerini geri döndürür.
Boşlukların da birer karakter olduğu unutulmamalıdır.

```
>> a='abcd'  
a =  
abcd
```

2.1.8.3 Karakter Fonksiyonları

```
>> b='efgh'  
b =  
efgh  
>> strcmp(a,b)  
ans =  
0  
>>  
  
>> strcmp(a,'abcd')  
ans =  
1  
>>
```

2.1.8.3 Karakter Fonksiyonları

STR2NUM

Bir karakter dizisini bir sayıya çevirir.

```
>> str2num('123')+1
```

ans =

124

>>

STRREP

(String Replacement) Bir karakter dizisindeki bazı karakterlerin farklı karakterlerle değiştirilmesini sağlar.

```
>> xx='replacement'
```

xx =

replacement

```
>> yy=strrep(xx,'ment','s')
```

yy =

replaces

>>

2.1.9 Giriş-Cıkış Deyimleri

Şu ana kadar MATLAB 'ı bir hesap makinası olarak kullandık. Her seferinde bir hesap yaptıktı veya bir değişkene bir değer atadık. MATLAB da otomatik olarak yaptığımız işlemin sonucunu anında gösterdi. Yaptığımız hesaplar genelde tek-satırlık hesaplar türünden idi.

Halbuki bir çok MATLAB programları çoğu zaman çeşitli veri girdileri alır, çok sayıda işlemler gerçekleştirir ve çeşitli sonuçları çıktılar şeklinde verir. Bu sırada ara işlemler görülmez. O halde bunları nasıl gerçekleştirebiliriz?

Yani, ara hesap sonuçlarının görüntülenmemesini, çeşitli tip ve sayıda verilerin programa girilmesini ve yine çeşitli sonuçların programdan alınmasını nasıl sağlارız?

2.1.9.1 MATLAB 'da Noktalı-Virgül (:)

MATLAB 'ın sonucu anında göstermesdini önlemek için satır sonuna bir (;) konulması yeterlidir. Aşağıda örneği görmektesiniz:

```
>> a=5+3
```

```
a =
```

```
8
```

```
>>
```

```
>> a=5+3;
```

```
>>
```

```
>> a
```

```
a =
```

```
8
```

```
>>
```

2.1.9.1 MATLAB 'da Noktalı-Virgül (:)

Noktalı-virgül, aynı zamanda bir satırda birden fazla MATLAB deyiminin yazılmasına da olanak sağlar:

```
>> x=5; y=25; z=x+y/13;  
>>
```

Yukarıdaki satırların aşağıdaki satırlardan hiçbir farkı yoktur:

```
>> x=5;  
>> y=25;  
>> z=x+y/13;  
>>
```

Her ikisinde de MATLAB sonuçların görünmesini engeller.

2.1.9.2 MATLAB 'da Programlanmış Çıktı

MATLAB komut satırına çıktı göndermek için FPRINTF fonksiyonu kullanılır. Aşağıda bu fonksiyonun bazı özellikleri açıklanmaktadır:

Bir yazı (text) satırı oluşturmak için tek yapmamız gereken fprintf komutu ile yazı karakterlerinin birlikte aşağıdaki gibi kullanılmasıdır:

```
>> fprintf('This is a test.\n');
```

This is a test.

```
>>
```

Dikkat edilirse noktalı-virgül işaretini bu kez yazının ekranda görülmemesini engellememektedir. Satırın sonundaki `\n` işaretini bir sonraki yazının yeni satırdan başlayacağını belirtmektedir.

Bir değişkenin değerini yazı içerisinde herhangi bir yerde görüntülemek için ilgili yere bir `%g` format karakterinin konulması yeterlidir. Değişkenin değeri `%g` karakterinin olduğu yerde görülecektir:

```
>> fprintf('The value of pi is %g.\n', pi);
```

The value of pi is 3.14159.

```
>>
```

2.1.9.2 MATLAB 'da Programlanmış Çıktı

Bir çıktı satırına istediğimiz yerine istediğimiz kadar değişken koyabiliriz.

```
>> a=3*pi;  
>> b=pi/3;  
>> fprintf('The value of a is %g, the value of b is %g.\n', a, b);  
The value of a is 9.42478, the value of b is 1.0472.  
>>
```

FPRINTF fonksiyonunun bir diğer uygulaması, karakter dizileri ile ilgilidir. Karakter dizisini FPRINTF fonksiyonunda açıkça yazmak yerine, bir değişkene atanmış olarak da yazabiliriz:

```
>> fprintf('This is a test!\n');  
This is a test!  
>>  
  
>> xx='This is a test!\n';  
>> fprintf(xx)  
This is a test!  
>>
```

2.1.9.3 MATLAB 'da Programlanmış Girdi

Kullanıcıdan istenen giriş bilgisi MATLAB INPUT fonksiyonu ile yapılabilir. Bu deyimin yapısı aşağıdaki gibidir:

```
xx=input('karakter dizisi');
```

Bu deyim kullanıldığında MATLAB önce karakter dizisi 'ni ekrana yansittıktan sonra kullanıcıdan veri girilmesini bekler ve girilen değeri eşitliğin solundaki değişkene atar.

```
>> num=input('Specify a number. ');
```

```
Specify a number. 2
```

```
>> fprintf('\nThe number you typed was %g.\n',num);
```

```
The number you typed was 2.
```

```
>>
```

2.1.9.3 MATLAB 'da Programlanmış Girdi

INPUT fonksiyonunun bir diğer şekli, karakter dizilerinin girilmesine olanak tanır:

```
>> xx=input('Specify a text string. ','s');
```

```
Specify a text string. KTU
```

```
>> xx
```

```
xx =
```

```
KTU
```

```
>>
```

2.1.9.3 MATLAB 'da Grafik Çizimi

MATLAB grafik amaçlı bir çok fonksiyona sahiptir. Bu dersin kapsamında basit matematiksel fonksiyonların grafiklerini oluşturabilmek için kullanabilecek olan PLOT fonksiyonu verilecektir. Örneğin aşağıdaki fonksiyonun grafiğini çizmeye çalışalım:

$$y=|x|\sin(x); \quad -100 < x < 100$$

Yapılacak ilk iş x değişkenlerini oluşturmaktır. Bu amaçla *LINSPACE* fonksiyonunu kullanabiliriz. Bu fonksiyonun yapısı aşağıdaki gibidir.

$$x=linspace(x1, x2, n).$$

Bu fonksiyon $x1$ den $x2$ 'ye kadar eşit aralıklı n tane x değeri üretir ve bunu bir boyutlu dizi olarak saklar. Aşağıda bazı örnekler görülmektedir:

```
>> x=linspace(0,5,6)
x =
    0    1    2    3    4    5
>>
```

2.1.9.3 MATLAB 'da Grafik Çizimi

```
>> x=linspace(0,5,11)
x =
Columns 1 through 10
    0    0.5000   1.0000   1.5000   2.0000   2.5000   3.0000   3.5000   4.0000   4.5000
Column 11
    5.0000
>>
```

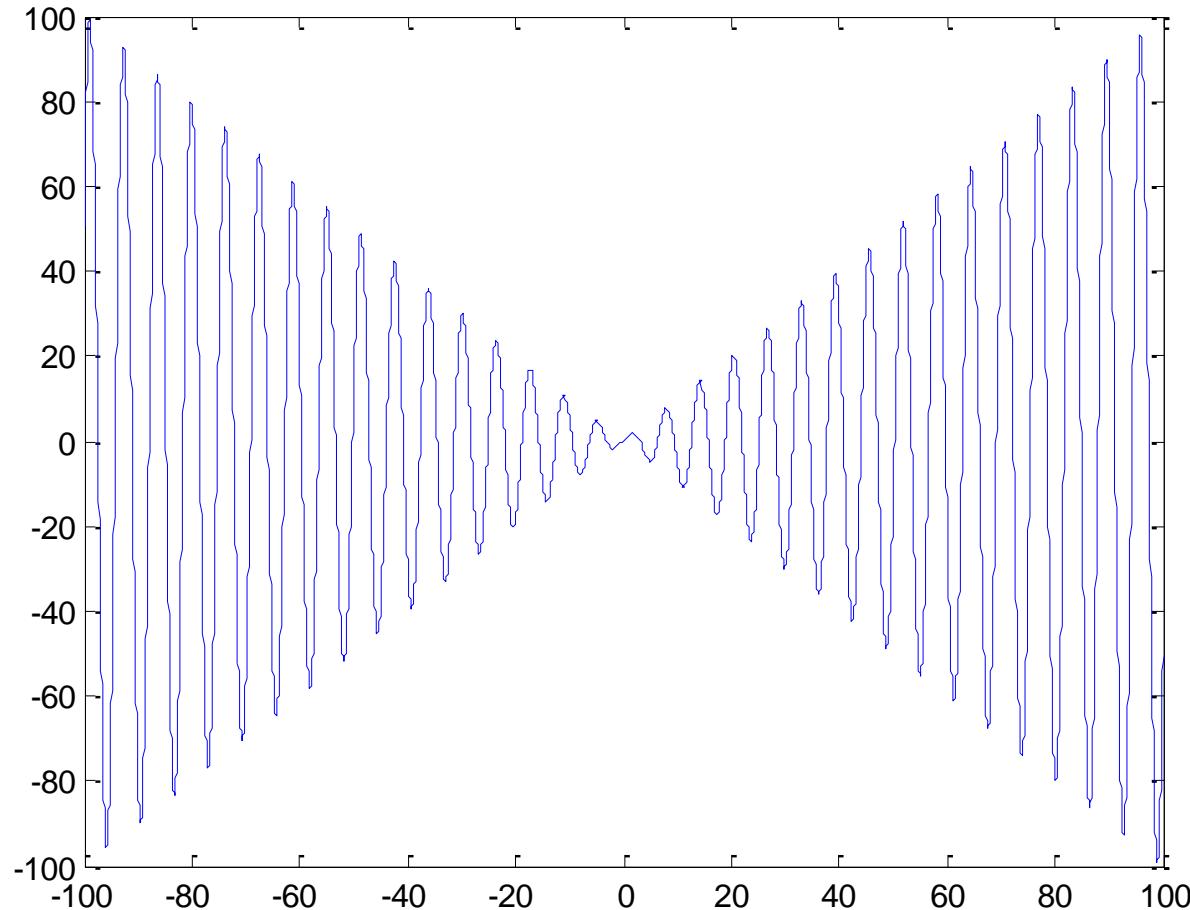
PLOT fonksiyonunun yapısı aşağıdaki gibidir.

Plot(x,y)

Bu fonksiyon *y* dizisinde depolanan verilerin *x* dizisinde depolanan verilere göre değişimini çizer. Her iki dizinin de aynı boyutta olması gereklidir. Aşağıdaki örneklerde temel matematiksel operatörlerin (*, / and ^) dizi versiyonu (.* , ./ and .^) kullanılacaktır.

2.1.9.3 MATLAB 'da Grafik Çizimi

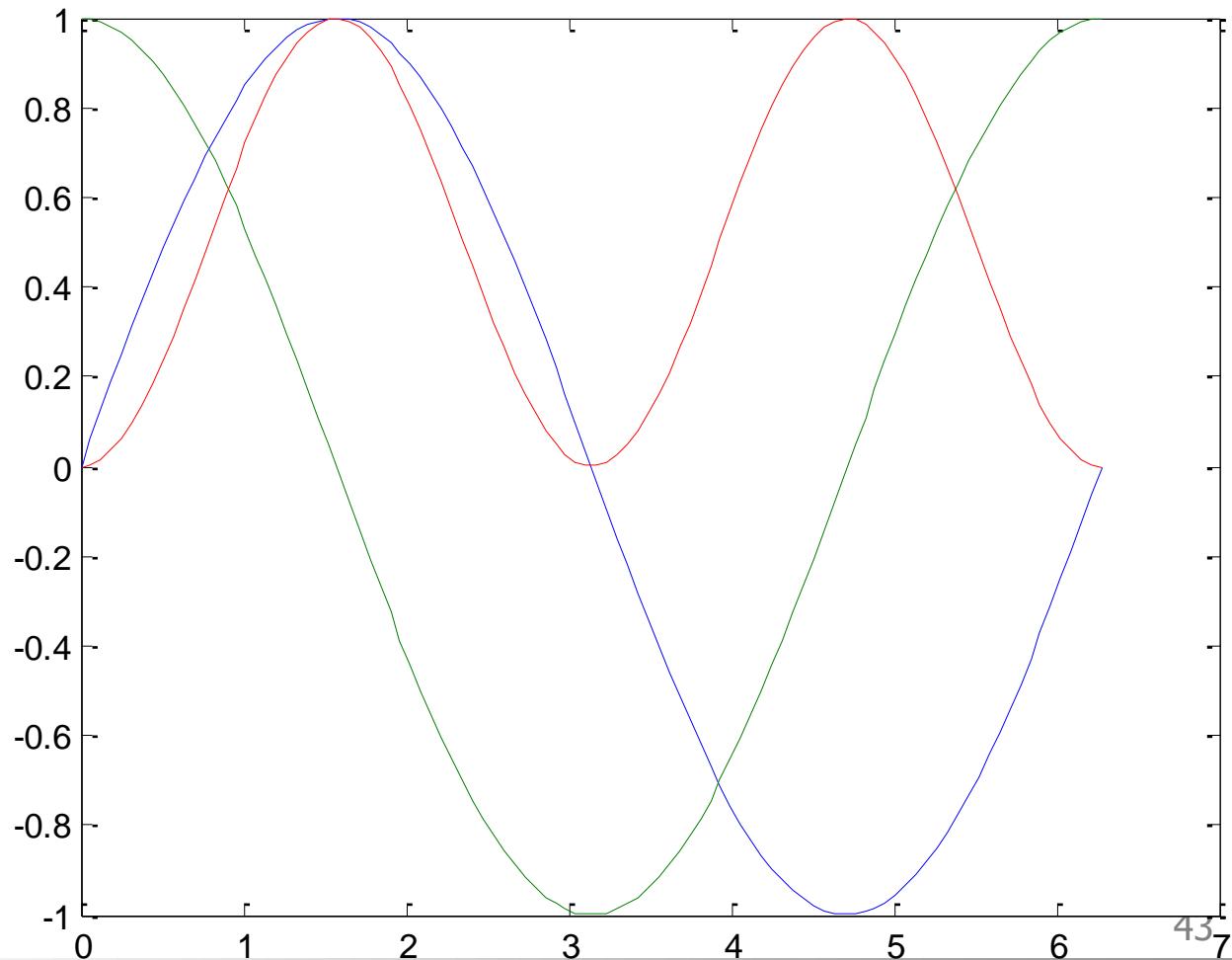
```
>> x=linspace(-100,100,5000);
>> y=abs(x).*sin(x);
>> plot(x,y)
>>
```



2.1.9.3 MATLAB 'da Grafik Çizimi

PLOT fonksiyonu tek bir grafik zemini üzerinde birden fazla fonksiyonun değişimini de çizebilir. Örneğin:

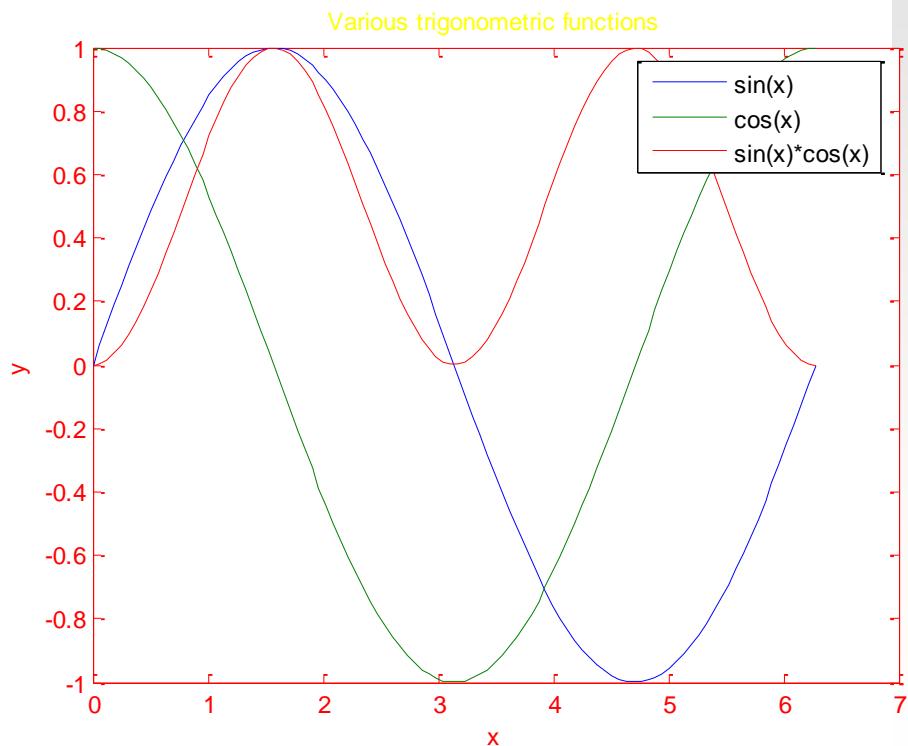
```
>> x=linspace(0,2*pi,100);
>> y1=sin(x);
>> y2=cos(x);
>> y3=sin(x).*sin(x);
>> plot(x,y1,x,y2,x,y3)
>>
```



2.1.9.3 MATLAB 'da Grafik Çizimi

XLABEL, *YLABEL*, *TITLE*, *LEGEND* gibi fonksiyonları kullanarak grafiğimize x-etiketi, y-etiketi, başlık gibi ek bilgileri de koyabiliriz:

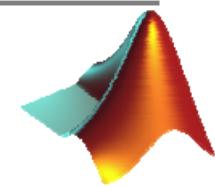
```
>> xlabel('x');
>> ylabel('y');
>> title('Various trigonometric functions');
>> legend('sin(x)', 'cos(x)', 'sin(x)*cos(x)');
>>
```



2.1.9.3 MATLAB 'da Grafik Çizimi

Burada görülen örnekler MATLAB 'ın grafik özelliklerinin sadece çok küçük bir kısmını kapsamaktadır. Diğer yaygın grafik fonksiyonları öğrenciye bırakılmıştır.

MATLAB/Temel Komutlar



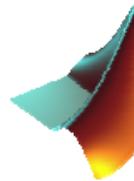
- `clc` Command window'u temizler.
- `clear` İlgili oturumda atanmış tüm değişkenleri siler.
- `clear a` Yalnızca "a" değişkenini siler.
- `demo` Matlab demosunu çalıştırır.
- `date` Gün-Ay-Yıl'ı görüntüler (Örneğin, 17-Oct-2009)
- `exit` Matlab oturumundan çıkar.
- `help` Yardım menüsünü açar.
- `help f_na` f_na fonksiyonu hakkında bilgi verir.
- `save d a` a değişkenini d dosya ismiyle **mat** uzantılı olarak kaydeder.
- `load d` a değişkenini d dosyasından geri çağırır.

Save ve load komutları, matris vb. yapılarının kaydedilmesi için çok önemlidir.

MATLAB/Temel dosya türleri

- * .m MATLAB program dosyaları
- * .fig Grafik dosyaları ve GUI'lerin grafik parçaları
- * .mat Değişken ve matris dosyaları
- * .p pre-parsed pseudo-code dosyaları (bu dosyaların içeriği görüntülenemez ancak program olarak çağrılabılır, yani MATLAB'de çalıştırılabilir!)

MATLAB/Matrislerin Girilmesi



- Matris ve vektörler [] köşeli parantezleri ile tanımlanır.
- Matris ve vektör girmenin 3 farklı yolu vardır:

Örneğin:

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

1.yol

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

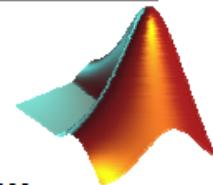
2.yol

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

3.yol

```
A(1,1)=1, A(1,2)=3, A(1,3)=5  
A(2,1)=7, A(2,2)=8, A(2,3)=11  
A(3,1)=100, A(3,2)=1, A(3,3)=4
```

MATLAB/Matrislerin Kaydedilmesi



- Matris ve vektörler *.mat uzantılı olarak `save` komutuyla kaydedilir, `load` ile de istenilen yerden geri çağrırlar.
- Örneğin, girilmiş bir `a` matrisini “D:\yıldız” klasörüne “**katsayılar.mat**” olarak kaydetmek isteyelim: Bunun için aşağıdaki komut dizisi kullanılır;

```
save D:\yıldız\katsayılar a
```

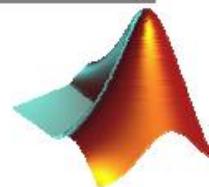
- **katsayılar.mat** olarak kaydedilen `a` matrisinin herhangi bir zamanda geri çağrılmaması için,

```
load D:\yıldız\katsayılar
```

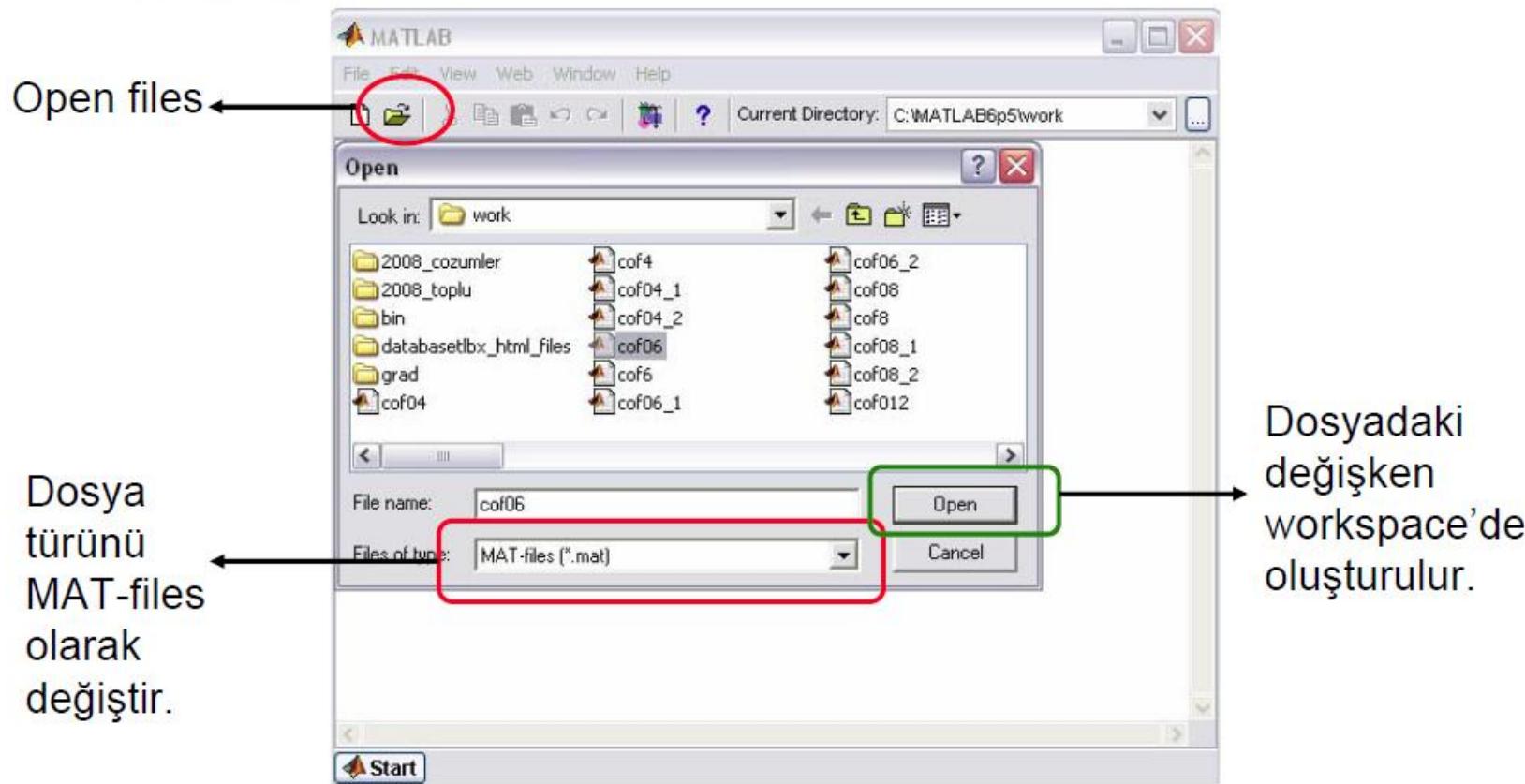
komut dizisi kullanılır. Geri çağrıma işleminden sonra, ilgili matris `a` dizisi olarak `workspace`'de kaydedilir (`workspace`'e kaydetme işleminin geçici olduğunu hatırlayınız!)

Yeni bir matrisi katsayılar.mat olarak kaydettiğimizde, önceki matrisi bir daha görme imkanı kalmaz. Yani `save` overwrite (üzerine yazma) özellikleidir.

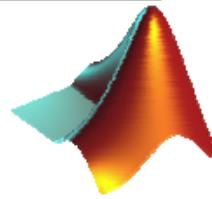
MATLAB/Matrislerin Kaydedilmesi



- *.mat uzantılı dosyalar, ayrıca MATLAB'den open files kısa yolundan da geri çağrılabılır:

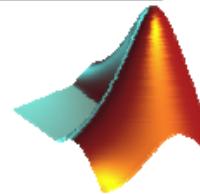


MATLAB/Temel lineer cebir komutları



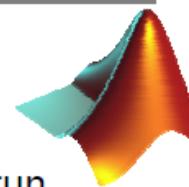
- **inv(a)** Bir a kare matrisinin tersini (inversini) alır.
- **a'** a matrisinin devriğini (transpozesini) alır.
- **det(a)** a matrisinin determinantını hesaplar.
- **a+b** Boyutları aynı olan a ve b matrisini toplar.
- **a-b** Boyutları aynı olan a ve b matrislerinin farkını alır.
- **a*b** Sütun sayısı m olan a matrisiyle satır sayısı m olan b matrisini çarpar.
- **a/b** b düzenli kare bir matrisse (determinantı sıfırdan farklıysa), aynı boyutlu a matrisiyle; **a*inv(b)** işlemini yapar.
- **a .*b** Boyutları aynı olan a ve b matrislerinin elemanlarını karşılıklı olarak çarpar.
- **a ./b** Boyutları aynı olan a ve b matrislerinin elemanlarını karşılıklı oranlar.

MATLAB/Temel lineer cebir komutları



- **trace(a)** Bir a matrisinin izini (köşegen elemanlarının toplamını) hesaplar.
- **diag(a)** Bir kare a matrisinin köşegen elemanlarını bir sütun vektöre atar. Ya da a bir vektör ise köşegenleri bu vektörün elemanlarından oluşan bir köşegen matris oluşturur.
- **sum(a)** a matrisinin her bir sütununun toplamını hesaplar. a bir vektör ise sonuç, vektör elemanlarının toplamı olur.
- **triu(a)** Bir matrisin üst üçgen matrisini oluşturur.
- **tril(a)** Bir matrisin alt üçgen matrisini oluşturur.
- **zeros(m,n)** $m \times n$ boyutlu sıfır matrisi oluşturur.
- **ones(m,n)** $m \times n$ boyutlu elemanları "1" olan matris oluşturur.
- **eye(m)** $m \times m$ boyutlu birim matris oluşturur.

MATLAB/Temel matris operatörleri



- **a(:)** a matrisinin sütunlarının ard arda dizilmesinden oluşan bir sütun vektör oluşturur (vec operatörü)
- **a(:, i)** a matrisinin i. sütununu alır.
- **a(j, :)** a matrisinin j. satırını alır.
- **a(:, [i j])** a matrisinin i ve j. sütununu alır.
- **a([i j], :)** a matrisinin i ve j. satırını alır.
- **e=a:b:n** a, (a+b),...,n sayılarından oluşan bir satır vektör oluşturur.

Örneğin,

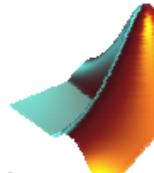
e=1:1:n, 1 ile n arasındaki tam sayılardan oluşan bir vektör.

e=2:2:n, 1 ile n arasındaki çift sayılardan oluşan bir vektör.

e=1:2:n, 1 ile n arasındaki tek sayılardan oluşan bir vektör.

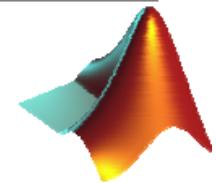
e=-10:0.1:10, -10'dan 0.1 artımla 10'ye kadar olan sayılardan oluşan bir vektör.

MATLAB/Temel matris operatörleri



- **length (a)** a matrisinin sütun sayısını verir. a bir vektör ise sonuç, a vektörünün eleman sayısıdır.
- **[m , n]=size (a)** a matrisinin satır sayısını (m) ve sütun sayısını (n) verir.
- **max (a)** Bir a vektörünün en büyük elemanını gösterir.
- **min (a)** Bir a vektörünün en küçük elemanını gösterir.
- **[m , i]=max (a)** Bir a sütun vektörünün en büyük elemanını (m) ve bunun satır numarasını verir.
- **[m , i]=min (a)** Bir a sütun vektörünün en küçük elemanını (m) ve bunun satır numarasını verir.
- **sort (a)** Bir a vektörünün elemanlarını küçükten büyüğe sıralar.
- **a (: , i)=[]** A'nın i. sütununu siler.
- **a (i , :)=[]** A'nın i. satırını siler.

MATLAB/Temel matris operatörleri



- `sortrows(a,i)` Bir a matrisinin elemanlarını i.sütuna göre sıralar.

Örnek:

a =

1	1000
3	10
2	5
4	1

a =

1	1000
3	10
2	5
4	1

`>> sortrows(a,1)`

ans =

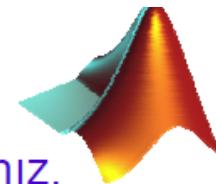
1	1000
2	5
3	10
4	1

`>> sortrows(a,2)`

ans =

4	1
2	5
3	10
1	1000

MATLAB/Uygulama-1



Aşağıdaki işlemleri command window'da yapınız.

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

- 1) A matrisini giriniz.
- 2) A matrisinin determinantını hesaplayınız.
- 3) A matrisinin tersini bulunuz. Çıkan sonucu bir B matrisine atayınız.
- 4) A*B işlemini yapınız. Elde edilen sonucu irdeleyiniz.
- 5) A matrisinin 1. sütununu a1, 3. sütununu a3 vektörlerine atayınız.
- 6) Köşegenleri A matrisinin köşegenlerinden oluşan bir C köşegen matrisi oluşturunuz.
- 7) a1'in devriği ile a3 vektörünü çarpınız.
- 8) a1 ile a3 vektör elemanlarını karşılıklı çarpınız.
- 9) A'nın 3. satırını, diğer satır elemanlarını girmeden, [5 6 7] olarak değiştiriniz.
- 10) A'nın 1 ve 2. satırlarını siliniz.

MATLAB/Uygulama-1:Çözüm

1 >> A=[1 3 5;7 8 11;100 1 4];
2 >> det(A)
ans =
-728
3 >> inv(A)
ans =

-0.0288 0.0096 0.0096
-1.4725 0.6813 -0.0330
1.0893 -0.4107 0.0179
>> B=ans;
4 >> A*B Birim matris
ans =

1.0000 0 0.0000
0 1.0000 0.0000
0 0.0000 1.0000

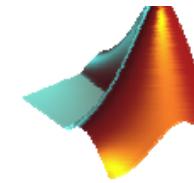
5 >>a1=A(:,1);a3=A(:,3);
6 >>C=diag(diag(A));
7 >>a1'*a3
ans=
482

8 >> a1.*a3
ans =
5
77
400
9 >>A(3,:)=[5 6 7]
A =

1 3 5
7 8 11
5 6 7
10 >> A([1 2],:)=[]
A =

5 6 7
>>

MATLAB/Uygulama-2

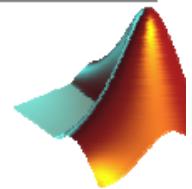


$$B = \begin{bmatrix} 10 & 5 & 5 \\ 70 & 8 & 7 \\ 10 & 1 & 3 \end{bmatrix}$$

Aşağıdaki işlemleri command window'da yapınız.

- 1) B matrisini giriniz.
- 2) B matrisini mevcut çalışma klasörünüze **katsayılar** ismiyle kaydediniz.
- 3) Dosyanın kaydedilip kaydedilmediğini kontrol ediniz.
(Open Files penceresinden)
- 4) MATLAB oturumundaki tüm değişkenleri siliniz (clear)
- 5) Command window'da yazılmış tüm ifadeleri temizleyiniz. (clc)
- 6) B^2 işlemini yapınız.
- 7) B matrisini geri çağırınız.
- 8) B matrisinin üst ve alt üçgen matrislerini oluşturunuz.
- 9) $C=[B \ zeros(3,2)]$ işlemini yapınız.

MATLAB/Uygulama-2:Çözüm



1 >> B=[10 5 5;70 8 7;10 1 3];
2 >> save katsayilar B

4 >> clear

5 >> clc

6 >> B*2
??? Undefined function or Neden?
variable 'B'.

7 >> load katsayilar

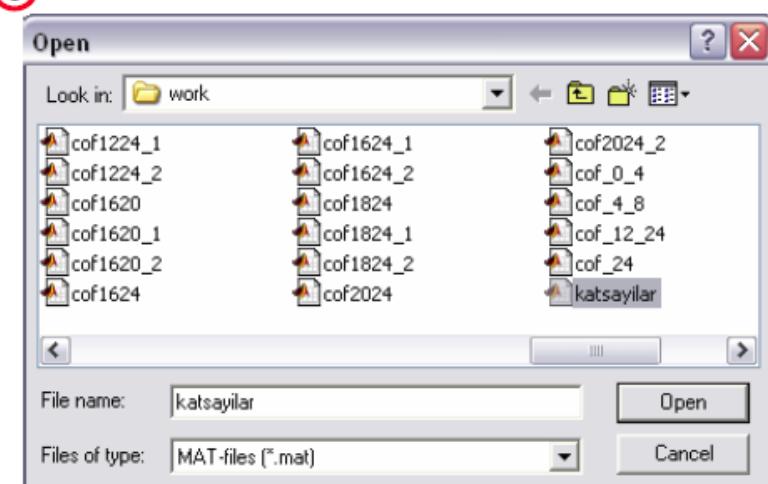
8 >> triu(B)

```
ans =  
10      5      5  
0      8      7  
0      0      3
```

>>tril(B)

```
ans =  
  
10      0      0  
70      8      0  
10      1      3
```

3



9

>> C=[B zeros(3,2)]

C =

10	5	5	0	0
70	8	7	0	0
10	1	3	0	0

Aşağıdaki işlemleri command window'da yapınız.

1. Sonraki işlemlerde kullanılacak bir a sayı değerini, inputdlg fonksiyonu ile girdiren komutu yazınız.
2. a değerinin bir sayı olup olmadığını irdeleyiniz.
3. a^2 işlemini yapınız. Bu işlemin neden sonuç vermediğini irdeleyiniz.
4. a değerini, gerekli ise, sayı dizisine dönüştürünüz.

1

```
>> a=inputdlg('Bir sayı giriniz','YTU-2009')
```



```
a =
```

```
'150.123135465'
```

inputdlg ile karakter hücre dizisi oluşturulur.
Bu nedenle, girilen verinin sayı yapılması
gerekir.

2

```
>>isnumeric(a)
```

```
ans =
```

```
0
```

3

```
>>a*2
```

```
>>?? Error using ==> *
```

```
Function '*' is not defined for values of class 'cell'.
```

4

```
>>a=str2num(char(a))
```

```
>>
```

```
a =
```

```
150.1231
```

BÖLÜM 2.2

Akış Kontrolü

2.2.1 IF-END

IF deyimi bir koşulun yerine gelmesi durumunda bir program bloğunun icra edilmesine olanak tanıyan bir deyimdir. Üç tipi vardır:

IF-END

IF-ELSE-END

ve

IF-ELSEIF-ELSE-END

2.2.1 IF-END

IF-END deyiminin yapısı aşağıdaki gibidir:

IF *koşul*

İfade 1;

İfade 2;

.

.

.

İfade n;

END

Eğer *koşul*'un sayısal değeri 1 veya *true* oluyorsa, IF ve END deyimleri arasındaki ifadeler icra edilir. Eğer tersine olarak koşul 0, veya *false* değerini alıyorsa bu kez program IF ve END arasındaki ifadelerden oluşan bloğu *atlayıp* END deyiminden sonra icraya devam eder. *koşul* tipik olarak aşağıdaki gibi olabilir:

değer ilişkি_operatörü değer

2.2.1 IF-END

burada *değer* olarak adlandırılan büyülük *sabit*, *değişken* veya *fonksiyon sonucu* olabilir, veya *cebirsel bir ilişki* olabilir.

İlişki_operatörü olarak adlandırılan terim aşağıdaki listedeki *operatörlerden* biri olabilir.

Tablo 3.1

<u>İlişki_operatörü</u>	Tanım	Örnek
>	büyük	$A>B$
\geq	büyük veya eşit	$\text{abs}(y)\geq 7$
<	küçük	$\text{real}(y)<3$
\leq	küçük veya eşit	$a\leq B$
\equiv	eşit	$\text{abs}(x)\equiv \text{real}(y)$
$\sim=$	eşit değil	$A\sim=27$

Aşağıda konuya çeşitli örnekler verilmektedir (if1):

2.2.1 IF-END

```
reel_sayı=10*rand(1);
sayı=ceil(reel_sayı);
%sayı
tahmin=input('1 ile 10 arasında bir sayı tahmin edin: ');
if tahmin==sayı
fprintf('Tebrikler bildiniz.\n');
end
if tahmin~=sayı
fprintf('Hatalı tahmin.\n')
end
```

Yukarıdaki programda, şu ana kadar görmediğimiz *CEIL* deyimi kullanılmıştır. *ceil(x)* şeklindeki bu deyim, reel bir sayıyı en yakın (pozitif yönde) bir büyük tam sayıya yuvarlar. Örneğin

```
>> ceil(2.1)
```

```
ans =
```

```
3
```

sonucunu verir.

2.2.1 IF-END

IF deyimindeki koşul mantıksal operatörler kullanılarak daha da genişletilebilir. Aşağıdaki Tablo 3.2 'de örnekler görülmektedir:

Tablo 3.2

Operatör	Tanım	Örnek
& AND	Mantıksal VE. Eğer iki taraf da true ise true.	$(a>b) \& (x < 5)$
OR	Mantıksal VEYA. En az bir taraf true ise true	$(x \sim= 5) (y <= 1)$
~ NOT	Mantıksal DEĞİL. Koşulun sonucu false ise true, Koşulun sonucu true ise false (örn:if2.m)	$\sim(a == b)$
xor EXCLUSIVE OR	özel Mantıksal VEYA. koşul1 true veya koşul2 true ise true, her iki koşul da true ise veya false ise false	$xor((x > 2), y == 5))$

2.2.2 IF-ELSE-END

IF deyiminin bir diğer şekli *else* koşulunu da içerebilir. Bu durumda IF koşulunun gerçekleşmesi halinde program *ELSE* bloğunu dikkate almaz, aksi takdirde, yani IF koşulu gerçekleşmez ise program ELSE bloğunu icra eder.

IF condition

statement $t_1;$

statement $t_2;$

:

.

statement $t_n;$

ELSE

statement $f_1;$

statement $f_2;$

:

.

statement $f_m;$

END

2.2.3 IF-ELSEIF-ELSE-END

IF deyiminin üçüncü şekli *elseif* deyimini de içerir. IF bloğundan sonra gelen *elseif* bloklarının sayısı istenildiği kadar olabilir.

IF condition1

statement $t_1;$

statement $t_2;$

:

statement $t_n;$

ELSEIF condition i

statement $k_1;$

statement $k_2;$

:

statement $k_s;$

ELSE

statement $f_1;$

statement $f_2;$

:

statement $f_m;$

END

Örnek: Girilen bir sayının negatif olması durumunda, sayıyı doğal logaritmasıyla değiştiren bir kod düşünelim:

```
a=input(' bir sayı giriniz= ');
if a<0
    a=log(a);
else
    a=a;
end
a
```

“Diğer durumda”
anlamındadır:
Burada, $a > 0$
koşulunu temsil eder.

Else yapısı kullanılmaması

```
a=input(' bir sayı giriniz= ');
if a<0
    a=log(a);
end
if a>0
    a=a;
end
a
```

2.2.3 SWITCH-CASE

SWITCH-CASE deyimi bir ifadenin farklı değerlere eşit olup olmadığını denenmesinde kullanılan yaygın bir deyimdir. *Eğer $a \geq 5$ ise* gibi karşılaştırmalar için değil de, *eğer $a=1$ ise* bir şey yap, *eğer $a=2$, ise* başka bir şey yap, vb. gibi uygulamalar için uygundur.

SWITCH-CASE deyiminin yapısı aşağıdaki gibidir:

SWITCH ifade

CASE değer₁

 deyim 1a;

 deyim 1b;

 deyim 1c;

...

 deyim 1n;

CASE değer₂

 deyim 2a;

 deyim 2b;

 deyim 2c;

...

 deyim 2n;

2.2.3 SWITCH-CASE

```
:  
.CASE değer;  
    deyim īa;  
    deyim īb;  
    deyim īc;  
. . .  
    deyim īn;  
.OTHERWISE  
    deyim īa;  
    deyim īb;  
    deyim īc;  
. . .  
    deyim īn;  
.END
```

2.2.3 SWITCH-CASE

değer bir skaler veya karakter dizisi olabilir. Ayrıca SWICH-CASE deyiminde her bir CASE aşağıdaki gibi birden fazla *değere* sahip olabilir.

CASE {Value₁, value₂, value₃, ...}

Bu durumda eğer *ifade*, listedeki herhangi bir *değere* eşit ise *a*'dan *n*'e kadar olan deyimler icra edilir.

Aşağıda birim değiştirme ile ilgili örnekte SWICH-CASE deyiminin uygulaması görülmektedir (units1.m)

2.2.3 SWITCH-CASE

%

% Bu program bir sayı ve birimini istemekte, birim inç veya metre ise santimetreye
% çevrilmektedir.

```
Sayi=input('Sayiyi giriniz: ');
Birim=input('Sayinin birimini giriniz: ', 's');
hata=0;

switch Birim
    case{'inç', 'in', 'İnç', 'İNÇ', 'İN', 'İN'}
        Yeni_sayı=Sayı*2.54;
    case{'Metre', 'metre', 'METRE', 'm', 'M'}
        Yeni_sayı=Sayı*100;
    otherwise
        hata=1;
end
...
```

2.2.3 SWITCH-CASE

```
...
if hata
    fprintf(Birim);
    fprintf(' birimi bu program tarafından tanınmıyor.\n');
else
    fprintf('%g %s', Sayi,Birim);
    fprintf(' =%g cm.\n', Yeni_sayı);
end
```

2.2.4 FOR Döngüsü

FOR döngüsü, bir takım ifadelerin belirli bir sayıda tekrarlanarak icra edilmesini sağlar.
Deyimin yapısı aşağıdaki gibidir:

FOR değişken=ifade

 deyim₁

 deyim₂

 deyim₃

...

 deyim_n

END

Örneğin yukarıdaki program parçası aşağıdaki gibi olabilir:

FOR i=1:10

 deyim₁

...

 deyim_n

END

2.2.4 FOR Döngüsü

Bu örnekte *i*'nin 1 den 10'a kadar olan değerleri için döngü içindeki deyimler icra edilir. 10. İcradan sonra program *end* deyiminden devam eder.

Aşağıdaki basit bir programla deyimin işleyişini görelim (FOR_loop1):

```
fprintf('Testing the FOR loop structure.\n\n');
for i=1:4
    fprintf('Testing the FOR loop. i=%g\n',i);
end
fprintf('\nAfter end of loop. i=%g\n',i);
```

FOR döngüleri geriye doğru da saydırılabilir (FOR_loop2):

2.2.4 FOR Döngüsü

```
fprintf('Testing the FOR loop structure.\n\n');
for i=4:-1:1
    fprintf('Testing the FOR loop. i=%g\n',i);
end
fprintf('\nEnd of loop.\n');
```

Yukarıdaki iki program karşılaştırıldığında, *varsayılan (default)* adının *1* olduğu anlaşılabilir.

Adının bir *tam sayı (integer)* olması gerekmez. Aşağıdaki örnekte adım boyutu *0.25* dir (FOR_loop3):

```
fprintf('Testing the FOR loop structure.\n\n');
for i=1:0.25:4
    fprintf('Testing the FOR loop. i=%g\n',i);
end
fprintf('\nEnd of loop.\n');
```

2.2.4 FOR Döngüsü

Bir döngünün içinde herhangi bir yerde **BREAK** komutunu kullanarak döngünün icrasına son verilerek döngüden çıkışılabilir.

FOR döngüleri iç içe olabilir (FOR_loop4):

```
fprintf('Demonstrating nested FOR loop structure.\n\n');
for i=1:5
    fprintf(' i =%g - j = ',i);
    for j=1:10
        fprintf('%g ', j);
    end
    fprintf('\n');
end
fprintf('\nEnd of demonstration.\n');
```

2.2.4 FOR Döngüsü

Yukarıda FPRINTF deyiminde **%g** formatı kullanılarak yazdırma işlemi yapıldı. Bunun yerine yazdırduğumuz sayının toplam basamak sayısını ve ondalık kısmın basamak sayısını da kendimizin belirleyebileceğim bir format olan **%f** formatı da kullanılabilir. Aşağıda **%f** formatının çeşitli kullanımları ve elde edilen sonuçlara örnekler verilmiştir:

```
>> fprintf('Testing - %0.0f\n',pi);  
Testing - 3  
>>
```

Yukarıda **pi** sayısı **0.0f** formatıyla yazdırılmaya çalışıldı. Buradaki ilk sıfır, yazılacak sayının toplam basamak sayısını, ikinci kısmı ise ondalık kısımdaki basamak sayısını göstermektedir. Dikkat edilirse, pi sayısının ondalık kısmı gerçekten yazılmamıştır, ancak tam kısmı aynen yazılmıştır. Diğer bir deyişle, tam kısmı bizim belirttiğimiz formattan büyük ise MATLAB sayının tam kısmını olduğu gibi yazmaktadır. MATLAB'ın bu özelliği, tam kısmının basamak sayısını tahmin edemediğimiz sayıların yazdırılmasında pratik olarak kullanılabilir.

2.2.4 FOR Döngüsü

```
>> fprintf('Testing - %0.1f\n',pi);  
Testing - 3.1  
>>
```

Buradaki formatta aslında yazdırılmak istenen format "tam kısmı ne kadar basamaktan oluşursa oluşsun, sayıyı ondalık kısmı 1 basamak olacak şekilde yaz" olarak algılanabilir.

```
>> fprintf('Testing - %10.1f\n',pi);  
Testing -      3.1  
>>
```

Bu örnekte görüldüğü gibi, ondalık kısım 10 basamaktan küçük olduğu için, yazım sırasında 9 basamaklı bir boşluk kalmaktadır.

2.2.4 FOR Döngüsü

Aşağıda FOR döngüsünün bazı özel durumları görülmektedir (FOR_loop5):

```
for i=[1, 5, 99, -23, 4]
fprintf('The value of i is %g.\n',i);
end
fprintf('\n');
```

```
for i='abc123xyz'
fprintf('The value of i is %c.\n',i);
end
fprintf('\n');
```

```
for i='a':'e'
fprintf('The value of i is %c.\n',i);
end
```

Bu program parçası koşturulduğunda:

2.2.4 FOR Döngüsü

>> FOR_loop5

The value of i is 1.

The value of i is 5.

The value of i is 99.

The value of i is -23.

The value of i is 4.

The value of i is a.

The value of i is b.

The value of i is c.

The value of i is 1.

The value of i is 2.

The value of i is 3.

The value of i is x.

The value of i is y.

The value of i is z.

....

2.2.4 FOR Döngüsü

The value of i is a.

The value of i is b.

The value of i is c.

The value of i is d.

The value of i is e.

>>

Elde edilir. Yukarıda gördüğünüz özellikler tamamen MATLAB'a özgüdür.

2.2.5 WHILE Döngüsü

WHILE döngüsünün FOR döngüsünden farkı, önceden belirli olmayan sayıda döngüyü tekrarlamasıdır. Diğer bir deyişle **koşul** gerçekleştiği sürece döngü tekrarlanır.

WHILE koşul

 deyim₁

 deyim₂

 ...

 deyim_n

END

Eğer **koşul true** veya **1** ise **1**'den **n**'e kadar olan deyimler icra edilir. **Koşulun false** veya **0** olması durumunda ise MATLAB **end** deyimine sıçrar.

2.2.5 WHILE Döngüsü

Aşağıda WHILE döngüsüne çeşitli örnekler görülmektedir (WHILE_loop1):

```
fprintf('Testing the WHILE loop structure.\n\n');
i=1;
while i<=3
    fprintf('Testing the WHILE loop. i=%g\n',i);
    i=i+1;
end
fprintf('\nEnd of WHILE loop. i=%g\n',i);
```

Programın koşturulmasından sonra aşağıdaki sonuçlar elde edilir:

2.2.5 WHILE Döngüsü

>> WHILE_loop1

Testing the WHILE loop structure.

Testing the WHILE loop. i=1

Testing the WHILE loop. i=2

Testing the WHILE loop. i=3

End of WHILE loop. i=4

>>

Dikkat edilirse döngü tamamlandığında *i*'nin aldığı değer FOR döngüsünden farklı olarak döngü sayacının son değerinden bir fazlası (4) olmaktadır. Bu duruma aşağıdaki gibi pratik bir çözüm getirilebilir (WHILE_loop2):

2.2.5 WHILE Döngüsü

```
fprintf('Testing the WHILE loop structure.\n\n');
i=0;
while i<=3
    i=i+1;
    fprintf('Testing the WHILE loop. i=%g\n',i);
end
fprintf('\nEnd of WHILE loop. i=%g\n',i);
```

Programın koşturulmasından sonra bu kez:

2.2.5 WHILE Döngüsü

>> WHILE_loop2

Testing the WHILE loop structure.

Testing the WHILE loop. i=1

Testing the WHILE loop. i=2

Testing the WHILE loop. i=3

Testing the WHILE loop. i=4

End of WHILE loop. i=4

>>

Şeklinde istenilen sonuç elde edilmiş olur.

BÖLÜM 2.3

Fonksiyonlar

2.3.1 Giriş

Fonksiyonlar (Function) sıkılıkla yerine getirilmesini istediğimiz görevler için kullandığımız altprogramlardır. Özellikle kapsamlı programlarda sıkılıkla yerine getirilmesi istenen görevler için benzer program parçalarının çok kere tekrarlanması yerine, bunların altprogram haline getirilerek gerektiği yerde bir deyim ile çağrılması, programın hantalliktan kurtulmasını ve daha kolay anlaşılabilir bir duruma gelmesini sağlayacaktır. Fonksiyonların yararları kısaca şöyle özetlenebilir: (1) Fonksiyonlar bir kez yazıldıkları defalarca kullanılabilirler, (2) Bir fonksiyon derlenerek doğru çalıştığı test edildikten sonra, artık diğer kullanımlarda hata yapma şansımız yoktur, (3) Siz diğerleri tarafından yazılmış paraprogramları kolaylıkla kullanabilirsiniz, veya diğerleri sizin yazdığınız fonksiyonları kolaylıkla kullanabilirler.

2.3.1 Bir Fonksiyonun Genel Yapısı

Bir **fonksiyonun** (*function*) genel yapısı aşağıdaki gibidir:

FUNCTION *y=fonksiyon_adı* (giriş argümanları)

% Fonksiyonun kullanımına ilişkin açıklamalar

%

...

deyimler

...

y = birşey

function kelimesi bir MATLAB deyimidir ve kullanılması zorunludur. *y* değişkeni ise fonksiyonun geri döndürdüğü değerdir. *y* tek bir değere sahip olabileceği gibi farklı birden çok değer de alabilir. **Fonksiyon_adı** fonksiyonun çağrılmırken ve bir dosya olarak kaydedilirken kullanılacağı addır. Örneğin

function *y* = fact1(*n*)

fonksiyonu *n* gibi bir tam sayının faktöriyelini hesaplayan bir programdır.

2.3.1 Bir Fonksiyonun Genel Yapısı

Fonksiyonun adı *fact1*'dir. Bu fonksiyon hard diske kaydedilirken *fact1.m* adıyla kaydedilmiştir.

Bir fonksiyon altprogramının hiçbir giriş değeri olmayabileceği gibi, bir veya birden fazla giriş değerine de sahip olabilir. Örneğin *fact(n)* fonksiyonu tek bir giriş değerine sahiptir. Aşağıda üç giriş değerine sahip bir fonksiyon adı görülmektedir:

```
function q = zeta(a,b,c)
```

Fonksiyonun geri döndürdüğü değer burada tek bir *q* değişkenine atanmaktadır. Burada *a*, *b*, *c* veya *q* ler, sabit, değişken veya dizi olabilirler. Genel olarak birden fazla değişkeni geri döndüren fonksiyon yapısı aşağıdaki gibidir:

```
function [x,y,z] = beta (a,b,c,d)
```

Bu fonksiyon dört giriş değişkenine ve üç çıkış değişkenine sahiptir.

2.3.1 Bir Fonksiyonun Genel Yapısı

Unutulmaması gereken bir durum, fonksiyon alt programı içerisinde çıkış değişkenlerine atama yapılması gerekliliğidir. Yukarıdaki örneklerden birinde fonksiyonun ilk satırı `y`'yi çıkış değişkeni olarak belirtmekte, son satırda da `y`ye bir değer atanmaktadır. Eğer bu atamanın yapılması unutulursa, fonksiyon alt programı anlamsız bir değeri geri döndürecektir. Hem programı daha iyi anlamak, hem de unutulmanın önlenmesi için, bir zorunluluk olmamakla birlikte, çıkış değişkenleri ile ilgili atamaların, alt programın sonlarındamasına çalışılır.

Dikkat edilmesi gereken bir diğer konu da, fonksiyonun ilk birkaç satırının programı tanıtıcı nitelikte açıklama satırı olmasıdır. Komut satırına `help` ve ardından `program adı` yazıldığında MATLAB otomatik olarak `function y = ...` şeklindeki ilk satırдан sonra açıklama satırlarını ekrana getirmektedir. Bu açıklama satırları kullanıcılar açısından önem taşımaktadır.

Aşağıda faktöriyel hesabı için kullanılabilcek olan basit bir programın yapısı görülmektedir (`fact1`):

2.3.1 Bir Fonksiyonun Genel Yapısı

```
function x=fact1(n)
% function x=fact1(n) - This function calculates the factorial of an integer n.
% The function checks to see if n is a positive integer.
% The function returns the following values depending on the input variable n.
%
% x=n! - if n is a valid input.
% x=-1 - if n is not an integer.
% x=-2 - if n is not positive.
% x=-3 - if n is not an integer and n was negative.

error_flag=0;
if floor(n)~=n %Not an integer
    error_flag=1;
    x=-1;
    fprintf('The number you entered was not an integer')
end
...
```

2.3.1 Bir Fonksiyonun Genel Yapısı

```
...
if n<0
    if error_flag
        x=-3; %Not positive and not an integer
        fprintf(' and not positive')
    else
        x=-2;
        error_flag=1; %Not positive
        fprintf('The number you entered was not positive')
    end
end
```

2.3.1 Bir Fonksiyonun Genel Yapısı

```
if error_flag==0 %The number entered was a good number.  
if n==0  
    factorial=1;  
else  
    factorial=1;  
    for i=n:-1:2  
        factorial=factorial*i;  
    end  
end  
x=factorial;  
end
```

Yukarıdaki programda n için bir hata kontrolu yapılarak, eğer n pozitif tam sayı ise faktöriyeli hesaplanmaktadır. n 'nin hatalı girilmesi durumunda (-1, -2, or -3) şeklinde hata kodları ekrana gönderilmektedir.

2.3.1 Bir Fonksiyonun Genel Yapısı

Help komutunun nasıl çalıştığını deneyelim:

```
>> help fact1
```

function x=fact1(n) - This function calculates the factorial of an integer n.

The function checks to see if n is a positive integer.

The function returns the following values depending on the input variable n.

x=n! - if n is a valid input.

x=-1 - if n is not an integer.

x=-2 - if n is not positive.

x=-3 - if n is not an integer and n was negative.

```
>>
```

Bu şekilde oluşturulan bir function programı artık diğer programlar içerisinde istenildiği kadar çağrılarak kullanılabilir.

2.3.2 Değişkenlerin Tanımlı Oldukları Program Bölümleri (scope)

MATLAB'da aksi belirtilmediği sürece bir değişken tanımlanmış olduğu program parçasında geçerlidir. Örneğin aşağıdaki fonksiyonu düşünelim (scope):

```
function z=scope(x)
fprintf('Inside function scope.\n');
a=5;
b=6;
c=7;
z=10;
fprintf('a=%g, b=%g, c=%g, z=%g\n',a,b,c,z);
fprintf('Leaving function scope.\n');
```

Bu foksiyon içerisinde *a*, *b*, *c* ve *z* değişkenleri tanımlanmaktadır. Şimdi bu fonksiyonu bir diğer fonksiyon içerisinde kullanarak sonucu görelim (scope2):

2.3.2 Değişkenlerin Tanımlı Oldukları Program Bölümleri (scope)

```
a=27;  
b=53;  
c=100;  
p=scope(5);  
fprintf('Back in the main program.\n');  
fprintf('a=%g, b=%g, c=%g\n',a,b,c);
```

scope2 programı koşturulduğunda aşağıdaki çıktıyı elde ederiz:

```
>> scope2  
Inside function scope.  
a=5, b=6, c=7, z=10  
Leaving function scope.  
Back in the main program.  
a=27, b=53, c=100  
>>
```

2.3.2 Değişkenlerin Tanımlı Oldukları Program Bölümleri (scope)

Görüldüğü gibi *function scope*'de tanımlı olan *a*, *b* ve *c* değişkenleri *5*, *6* ve *7* değerlerini alırken, *scope2*'de aynı adla tanımlı değişkenler farklı değerler alabilmekte ve bunlar birbirlerini etkilememektedirler.

2.3.3 Değişkenlerin Farklı Program Bölümlerine Aktarılması

Aşağıdaki *passing* adlı fonksiyonu kullandığımızı varsayıalım:

```
function z=passing(a, b, c)
fprintf('Inside function passing.\n');
fprintf('While entering function passing, ');
fprintf('a=%g, b=%g, c=%g.\n\n',a,b,c);
a=a/10;
b=b/10;
c=c/10;
z=a+b+c;
fprintf('After performing a few calculations, ');
fprintf('a=%g, b=%g, c=%g, z=%g\n',a,b,c,z);
fprintf('Leaving function passing.\n\n');
```

Bu fonksiyona üç değer (*a*, *b* ve *c*) aktarılmakta ve fonksiyona ilk girildiğinde bu değerler ekrana yazdırılmaktadır. Daha sonra bu değişkenlerin değerleri değiştirilmekte ve fonksiyondan çıkmadan önce tekrar yazdırılmaktadır.

2.3.3 Değişkenlerin Farklı Program Bölümlerine Aktarılması

Şimdi bu fonksiyonu aşağıdaki program parçasında kullanalım (*passing2*):

```
x=1;  
y=2;  
w=3;  
fprintf('In the main program, ');\nfprintf('x=%g, y=%g, w=%g.\n\n',x,y,w);\nanswer=passing(x, y, w);\nfprintf('Back in the main program, ');\nfprintf('x=%g, y=%g, w=%g, answer=%g.\n\n',x,y,w,answer);
```

Program koşturulduğunda aşağıdaki çıktıyı üretecektir:

2.3.3 Değişkenlerin Farklı Program Bölümlerine Aktarılması

>> passing2

In the main program, $x=1$, $y=2$, $w=3$.

Inside function passing.

While entering function passing, $a=1$, $b=2$, $c=3$.

After performing a few calculations, $a=0.1$, $b=0.2$, $c=0.3$, $z=0.6$

Leaving function passing.

Back in the main program, $x=1$, $y=2$, $w=3$, $answer=0.6$.

>>

Ana programda x , y , and w değerleri tanımlanmakta ve $x=1$, $y=2$, $w=3$ olarak yazdırılmaktadır. Daha sonra MATLAB $answer=passing(x, y, w)$ deyimini icra etmekte ve bu değişken değerlerini *passing function* alt programına a , b ve c değişkenleri olarak aktarmaktadır. Passing alt programının tanımlanması $z=passing(a, b, c)$ şeklindedir. Alt programda a , b ve c değerleri 10'a bölünerek değerleri değiştirilmekte ve z değeri hesaplanmakta, sonra bunlar yazdırılmaktadır. Programın icrası alt programı terkederken $z=passing(a, b, c)$ deyi̇mi ile z değeri ana programa aktarılmaktadır. Ancak bu sırada a , b ve c değerleri, değişmiş halleri ile tekrar ana programa aktarılmamaktadır. z değeri ana programa *answer* değişkeni olarak aktarılmaktadır.

2.3.4 Global Değişkenler

Bazan bir değişkenin herhangi bir anda sahip olduğu değerin birden fazla function alt programlarında geçerli olmasını isteyebiliriz. Bu tür değişkenlere *global değişken* adı verilir. Aşağıda buna ilişkin bir örnek görülmektedir (*global1*):

```
function z=global1(in)
global a;
fprintf('Inside the function the value of a is %g\n', a);
a=1;
```

Bu function *a*'nın değerini önce yazdırınmakta ve sonra değerini *1* olarak değiştirmektedir. Bu alt programı aşağıdaki program parçası içerisinde kullanalım (*global2*):

```
global a;
a=5;
global1(7);
fprintf('In the code segment, a=%g.\n', a);
```

2.3.4 Global Değişkenler

global2 program parçasını koşturduğumuzda aşağıdaki çıktıyı görürüz:

```
>> global2  
Inside the function the value of a is 5  
In the code segment, a=1.  
>>
```

Görüldüğü gibi ana programda a 'nın değeri 5 olarak tanımlandıktan sonra alt programda değeri yazdırılmakta, daha sonra alt programda değeri 1 olarak değiştirilmekte ve son değer tekrar ana programda yazdırılmaktadır.

2.3.5 RETURN Deyimi

Normalde bir function kendisini çağıran ana programa kendinin son satırının içrasından sonra döner. Ancak **RETURN**

Deyimi kullanılarak bu dönüşün istenildiği anda olması sağlanabilir (*return1*):

```
function y=return1(choice)
%
fprintf('Entering function return1.\n');
if choice==1
    fprintf('Exiting early.\n');
    y=0;
    return
end
fprintf('After the IF statement.\n');
y=1;
fprintf('Executing the last line of the function.\n');
```

2.3.5 RETURN Deyimi

Eğer function 1 giriş argümanı ile çağrılırsa, IF deyiminin içindeki komutlar icra edilecek ve **RETURN** deyimi kullanılarak alt programdan erken çıkış olacaktır. Eğer 1 'den farklı bir argümanla function çağrılırsa, bu kez alt programda en son satır icra edildikten sonra function 'dan çıkarılır.

Örneğin ilk olarak 1 argümanı ile function 'u çağıralım:

```
>> return1(1)
Entering function return1.
Exiting early.
ans =
 0
>>
```

2.3.5 RETURN Deyimi

Şimdi de 1 'den farklı (örneğin sıfır) bir değerle function 'ı çağıralım:

```
>> return1(0)
```

Entering function return1.

After the IF statement.

Executing the last line of the function.

```
ans =
```

```
1
```

```
>>
```

2.3.6 Dinamik Diziler

Bir çok programlama dilinde dizi boyutları sabittir ve bu dizilerle ilgili işlemler yapılmadan önce bu boyutların tanımlanması gereklidir. Farklı uygulamalarda farklı boyutlara sahip dizilerle çalışıldığında, boyut uyumsuzluğu problemi olmaması için genellikle olası en büyük boyutun kullanılması gereklidir.

MATLAB 'da dizi elemanlarına erişim çok kolay olup, dizi boyutları programın herhangi bir yerinde tanımlanabilir veya boyutları değiştirilebilir. Örneğin aşağıdaki program parçasında dizi önce tek elemana sahip olacak şekilde tanımlanmakta, daha sonra diziye ikinci bir eleman eklenmektedir:

```
>> x=zeros(1);  
>> x(2)=1;
```

```
>> x  
x =  
0 1  
>>
```

2.3.6 Dinamik Diziler

Benzer işlemi iki- ve üç-boyutlu diziler için de aşağıdaki örneklerde olduğu gibi gerçekleştirebiliriz:

```
>> x=zeros(2)
```

```
x =
```

```
0 0  
0 0
```

```
>>
```

Yukarıda x dizisinin elemanları sıfır olan 2×2 boyutunda bir kare matris olduğu belirtiliyor.

Şimdi bu matrisin boyutunu $3 \times 3 \times 3$ 'lük bir matrise yükselten aşağıdaki eleman atamasını yapalım ve sonucu inceleyelim:

2.3.6 Dinamik Diziler

```
>> x(3,3,3)=1
```

```
x(:,:1) =
```

```
0 0 0
```

```
0 0 0
```

```
0 0 0
```

```
x(:,:2) =
```

```
0 0 0
```

```
0 0 0
```

```
0 0 0
```

```
x(:,:3) =
```

```
0 0 0
```

```
0 0 0
```

```
0 0 1
```

```
>>
```

Yukarıda matrisin üçüncü satırındaki, üçüncü sutunundaki ve üçüncü derinlikteki elemanına 1 değerinin atandığı görülmüþdir. MATLAB yeni oluþan $3 \times 3 \times 3$ boyutundaki matrisin elemanlarını iki boyutlu matrisler şeklinde karþımıza getirmektedir.

2.3.6 Dinamik Diziler

$x(:,:,1) =$

0	0	0
0	0	0
0	0	0

gösterimi matrisin birinci derinlik düzeyine sahip elemanlarını,

$x(:,:,2) =$

0	0	0
0	0	0
0	0	0

gösterimi matrisin ikinci derinlik düzeyine sahip elemanlarını,

$x(:,:,3) =$

0	0	0
0	0	0
0	0	1

de matrisin üçüncü derinlik düzeyine sahip elemanlarını oluşturmaktadır.

2.3.6 Dinamik Diziler

`x(:,:,3) =`

```
0 0 0
0 0 0
0 0 1
```

Görüldüğü gibi $x(3,3,3)$ elemanının değeri 1 dir. Yukarıdaki atama ile matris boyutu $3 \times 3 \times 3$ 'e çıkarılmıştır.

2.3.6 Dinamik Diziler

Satır veya sutun dizileri aşağıdaki gibi işlemlerle adım adım genişletilebilir. Örneğin satır dizilerinin genişletilmesi:

```
>> x=[ ]  
x =  
[ ]  
>> x=[x,0]  
x =  
0  
>> x=[x,1]  
x =  
0 1  
>> x=[x,2]  
x =  
0 1 2  
>>
```

2.3.6 Dinamik Diziler

Sutun dizilerinin genişletilmesi:

```
>> x=[ ]  
x =  
[]  
>> x=[x;0]  
x =  
0  
>> x=[x;1]  
x =  
0  
1  
>> x=[x;2]  
x =  
0  
1  
2  
>>
```

BÖLÜM 2.4

Veri Dosyaları ile
Veri Alış-Verisi

2.4.1 Dosyaların Açılıp Kapanması

MATLAB'da bir dosyadan okuma veya bir dosyaya yazma yapabilmemiz için bu dosyanın önce açılması gereklidir. Bu işlem *FOPEN* komutu ile yapılabilir.

```
fid=fopen(filename, permissions)
```

filename değişkeni dosya adını tanımlayan bir karakter dizisidir. Eğer bir dosyayı okuma amaçlı olarak açıyorsanız, dosyanın zaten varolması gereklidir. Eğer bir dosyayı yazmak için açıyorsanız ve bu dosya henüz yok ise, *filename* adlı yeni bir dosya oluşturulur.

permissions değişkeni bir dosyanın nasıl açıldığını belirten bir koddur. Bu kodlar Tablo 6.1 'de verilmiştir.

2.4.1 Dosyaların Açılıp Kapanması

Tablo 6.1

Permission Text String	Action
' r '	Dosyayı okumak (<i>reading</i>) için aç. Açılabilmesi için dosyanın zaten var olması gereklidir. Eğer dosya yok ise hata mesajı verilir.
' r+ '	Dosyayı okumak ve yazmak için aç. Açılabilmesi için dosyanın zaten var olması gereklidir. Eğer dosya yok ise hata mesajı verilir.
' w '	Dosyayı yazmak (<i>writing</i>) için aç. Eğer dosya zaten var ise içeriğini sil. Eğer yok ise, yeni bir dosya oluşturur.
' w+ '	Dosyayı okumak ve yazmak için aç. Eğer dosya zaten var ise içeriğini sil. Eğer yok ise, yeni bir dosya oluşturur.
' a '	Dosyayı yazmak için aç. Eğer dosya zaten var ise yeni verileri dosyanın sonuna ekle. Eğer yok ise, yeni bir dosya oluşturur.
' a+ '	Dosyayı okumak ve yazmak için aç. Eğer dosya zaten var ise yeni verileri dosyanın sonuna ekle. Eğer yok ise, yeni bir dosya oluşturur.

2.4.1 Dosyaların Açılıp Kapanması

Bir kerede birden çok dosya açılabilir. `fid=fopen(filename, permissions)` deyiminde `fid` yeni açılan dosyayı tanımlayan bir koddur. Açılan her dosya ayrı bir kod ile tanımlanır. `-1` den `2`'ye kadar olan kodlar MATLAB'da özel bir anlama sahip olduklarından, `FOPEN` function'ı ile `açılan her dosya için` geri döndürülen `fid` kodu `3`'ten başlar.

- `-1` kodu, dosya açılırken bir hatanın olduğunu belirtir. Okumak için bir dosya açılmaya çalışılır ve fakat dosya yok ise genellikle bu kod geri gönderilir.
- `0` kodu standart giriş kodunu ifade eder. Bu giriş genellikle klavyedir ve her zaman yazma için açıktır (`permission 'r'`).
- `1` kodu standart çıkış kodudur. Bu çıkış MATLAB komut satırı olup, her zaman çıktılarının sona eklenme izini (`append permission`) açıktır. Ekleme izni (append permission), önceki veriler silinmeden yeni verilerin her zaman dosyanın sonuna eklenme izni olduğunu gösterir.
- `2` kodu standart hata kodudur ve eklenme izni ile her zaman çıkış için açıktır. Standart giriş, çıkış ve hata, `FOPEN` ve `FCLOSE` function'ları ile açılıp kapatılamaz. Bir dosya ile veri alış-verişi yapıldığında, MATLAB sonlandırılmadan önce dosya her zaman kapatılmalıdır. Bu `FCLOSE` function'ı ile yapılır. Yapısı:

```
status=fclose(fid)
```

2.4.1 Dosyaların Açılıp Kapanması

fid değişkeni kapatmak istediğiniz dosyanın koduna sahiptir. Eğer açık olarak birden fazla dosyaya sahipseniz, *fid* kapatmak istediğiniz dosyayı gösterir. *status* değişkeni *FCLOSE* function'ı tarafından geri döndürülen ve dosyanın uygun bir şekilde kapatılıp kapatılmadığını gösteren bir koddur. *-1* kodu işlemin başarısız olduğunu, *0* kodu dosyanın uygun bir şekilde kapatıldığını gösterir.

Bir dosyayı açarken dosya adresini (*entire pathname*) tamamen tanımlayabilirsiniz, veya sadece dosya adını (*file name*) belirtebilirsiniz. Eğer sadece dosya adını belirtirseniz, dosya o anki (*current directory*) klasöre kaydedilir. Bulunulan klasörü görmek için *PWD* komutu kullanılır.

```
>> pwd  
ans =  
C:\MATLAB7\work  
>>
```

Klasörün içeriğini görmek için *DIR* komutu kullanılır.

2.4.1 Dosyaların Açılıp Kapanması

Şimdi okutmak için bir dosya açmaya çalışalım:

```
>> file_id_1=fopen('Text1.txt','r');  
>> file_id_1  
file_id_1 =  
    -1  
>>
```

Dikkat edilirse *file_id_1* nin değeri *-1* olmuştur. Bu, dosyanın açılışında bir hata olduğunu gösterir. Bunun oluşma nedeni '*r*' izinin açılma için dosyanın varolmasını gerektirmesidir. Geçerli klasörde böyle bir dosya olmadığından FOPEN function'ı tarafından *-1* hata kodu üretimiştir.

2.4.1 Dosyaların Açılıp Kapanması

Şimdi 'w' iznini kullanarak dosya açalım. Hatırlanacağı üzere bu izin, yazmak için yeni bir dosya yaratılmasını sağlar. Eğer dosya zaten var ise içeriğini siler.

```
>> file_id_1=fopen('Text1.txt','w');
>> file_id_1
file_id_1 =
    3
>>
```

Göründüğü gibi şimdi *file_id_1* 'nin değeri 3'tür. Bu, dosyanın doğru bir şekilde açıldığını bildirir.

2.4.1 Dosyaların Açılıp Kapanması

Bir başka dosya açalım:

```
>> file_id_2=fopen('Text2.txt','w');
>> file_id_2
file_id_2 =
    4
>>
```

Görüldüğü gibi yeni *fid* kodu **4** değerini almıştır. Çünkü halihazırda 3 koduna sahip bir başka dosya açılmış ve henüz kapatılmamıştır. Benzer şekilde yeni dosyalar açılsa bu kodlar 5, 6, vb. şeklinde devam edecektir. Yukarıdaki dosyalar 'w' izin kodu ile açıldığından, bu dosyaların herhangi birini çıktı yazdırma için kullanabiliriz. MATLAB'dan çıkmadan önce bütün açık dosyaların FCLOSE function'ı ile kapatılması gereklidir:

```
>> status1=fclose(file_id_1);
>> status1
status1 =
    0
>>
```

2.4.1 Dosyaların Açılıp Kapanması

status (burada *status1*) 'un *sıfır* değer alması dosyanın başarılı bir şekilde kapatıldığını göstermektedir. Benzer şekilde diğer dosya da kapatılabilir:

```
>> status2=fopen(file_id_2);
>> status2
status2 =
    0
>>
```

Açık olan bir çok dosyayı tek bir kerede kapatmak istiyorsak *fclose('all')* deyimini kullanabiliriz.

```
>> fclose('all')
ans =
    0
```

2.4.2 Formatlı Çıktıların Dosyalara Yazılması

Bir dosyaya yazmak için kullanacağımız ilk yöntem *FPRINTF* (*Formatted PRINT Functions*) fonksiyonudur. Bu fonksiyonu MATLAB'ın ekranına yazdırma için yaygın olarak kullandık. Bir dosya tanımlayıcısı (file identifier (*FOPEN* function'ı tarafından döndürülen *fid*)) belirlemediğimizde, *FPRINTF* fonksiyonu çıktıyı otomatik olarak ekrana gönderir. Buna karşın, bir *fid* belirlersek, çıktı *fid* ile belirlenen dosyaya gönderilir.

Şimdi bir örnek görelim. Fakat yazmak için önce dosyalar açmalıyız. Yeni dosyalar yaratacağımızdan, '*a*' or '*w*' izinlerinden birini kullanmalıyız.

```
>> fid1=fopen('file1.txt', 'w');  
>>
```

Önce *FPRINTF*'te bir *fid* belirtmeden komutu kullanalım:

```
>> fprintf('This is a test. No fid specified.\n');  
This is a test. No fid specified.  
>>
```

2.4.2 Formatlı Çıktıların Dosyalara Yazılması

Şimdi de *file1.txt* dosyasına yazalım:

```
>> fprintf(fid1,'This is text sent to the file indicated by fid1.\n');  
>> fprintf(fid1,'The value of fid is %g.\n',fid1);  
>>
```

Word pad veya *microsoft word* gibi kelime işlemcilerle dosyaya bakılabilmesi için önce dosyanın kapatılması gereklidir.

```
fclose(fid1);  
>>
```

MATLAB 'ın içinden veya word pad ile .txt uzantılı çıktı dosyalarına bakılabilir.

Aşağıda $m=[1\ 2\ 3\ 4\ 5\ 6;10\ 20\ 30\ 40\ 50\ 60]$ matrisini yazdırma ile ilgili bir başka örnek verilmektedir:

2.4.2 Formatlı Çıktıların Dosyalara Yazılması

```
>> m=[1 2 3 4 5 6;10 20 30 40 50 60]
```

```
m =
```

1	2	3	4	5	6
10	20	30	40	50	60

Yukarıda matris tanımlanıyor.

```
>> fid=fopen('veriler.dat','w')
```

```
fid =
```

```
3
```

veriler.dat adlı dosya yazılmak üzere açılıyor. '*w*' izni gereği dosya hiç yok ise ilk kez yaratılacak, var ise içeriği silinip yeni veriler yazılacak. *fid* değişkenine *fopen* fonksiyonu tarafından *3* değerinin atanması dosyanın doğru bir şekilde açıldığını göstermektedir.

2.4.2 Formatlı Çıktıların Dosyalara Yazılması

```
>> fprintf(fid,'%10f %10f\n',m);
>> fclose(fid)
ans =
0
```

Açılan dosyaya **10f** formatında (**10** hanelik alana, ondalık nokta bulunduğu yerde olacak şekilde) iki sutun halinde matris elemanları yazdırılmıştır. Dosyanın kapmasını doğru bir şekilde gerçekleştiği **fclose** fonksiyonunun geri gönderdiği **sıfır** değerinden anlaşılmaktadır. Komut satırına **type veriler.dat** yazılarak yazdırılan dosyanın içeriğine bakılabilir:

```
>> type veriler.dat
1.000000 10.000000
2.000000 20.000000
3.000000 30.000000
4.000000 40.000000
5.000000 50.000000
6.000000 60.000000
>>
```

2.4.2 Formatlı Çıktıların Dosyalara Yazılması

```
>> type veriler.dat
1.000000 10.000000
2.000000 20.000000
3.000000 30.000000
4.000000 40.000000
5.000000 50.000000
6.000000 60.000000
>>
```

Burada dikkati çeken önemli bir durum, matrisin yazdırılırken sutun sutun (yani orijinal matrisin transpozunun) yazdırılmasıdır. Örneğin aynı veriler

```
fprintf(fid,'%10f %10f %10f\n',m);
>>
```

Şeklinde üç sutuna yazdırılırsa sonucun nasıl görüldüğüne dikkat edelim:

2.4.2 Formatlı Çıktıların Dosyalara Yazılması

```
>> type veriler.dat
1.000000 10.000000 2.000000
20.000000 3.000000 30.000000
4.000000 40.000000 5.000000
50.000000 6.000000 60.000000
>>
```

Aşağıda $f(x)=1/x$ fonksiyonunun, x 'in 1 ile 4 aralığındaki değerlerine karşılık $f(x)$ değerlerini hesaplayan ve 4×2 boyutunda bir matris şeklinde yazdırın bir program parçası görülmektedir (fprintf1.m).

```
x=1:4; y=zeros(4,2);
```

$y(:,1)=x'$; % x satır vektörünün transpozu y matrisinin ilk sutunu atıyor.

$y(:,2)=1./x'$; % y matrisinin ikinci sutunu atıyor.

```
fid=fopen('tab.txt','w+'); %tab.txt dosyası açılıyor.
```

```
fprintf(fid,'%4.0f \t %6.3f \n',y');
```

```
fclose(fid);
```

2.4.2 Formatlı Çıktıların Dosyalara Yazılması

```
x=1:4; y=zeros(4,2);  
y(:,1)=x'; %x satır vektörünün transpozu y matrisinin ilk sutununa atanıyor.  
y(:,2)=1./x'; % y matrisinin ikinci sutunu atanıyor.  
fid=fopen('tab.txt','w+'); %tab.txt dosyası açılıyor.  
fprintf(fid,'%4.0f \t %6.3f \n',y);  
fclose(fid);
```

Yazdırma komutundaki `\t` formatı *tab (sekme)* anlamına gelmektedir. Ayrıca *y matrisinin transpozunun (y')* yazdırılması, *y* nin çıktıda satır satır gözükmesini sağlamaktadır.

Program koşturulup sonuçlara bakılırsa:

```
>> fprintf  
>> type tab.txt  
1      1.000  
2      0.500  
3      0.333  
4      0.250  
>>
```

2.4.2 Formatlı Verilerin Dosyalardan Okunması

Formatlı verilerin dosyadan okunması için *FSCANF* komutu kullanılmaktadır. Bu komutun yapısı

```
[A,c]=fscanf(fid, fstr, s)
```

şeklindedir. Burada *A* okunan değerlerin adresleneceği matris, *c* de okunan değer sayısıdır.

Eğer *s* bir skaler ise, o zaman *A* bir *sutun* vektörü olmaktadır. Eğer *s=[m,n]* ise, *A* bu kez *fid* dosyasından *sutun sutun* okunan *mxn* boyutunda bir *matris* olmaktadır. *fstr* okunacak olan verinin formatını ifade etmektedir.

Şimdi, yukarıdaki bir örnekte formatlı olarak veriler.dat dosyasına yazdırılan m matrisinin elemanlarını okutmaya çalışalım.

2.4.2 Formatlı Verilerin Dosyalardan Okunması

Hatırlanacağı üzere bu elemanlar aşağıdaki formatta dosyaya yazdırılmıştır:

```
1.000000 10.000000  
2.000000 20.000000  
3.000000 30.000000  
4.000000 40.000000  
5.000000 50.000000  
6.000000 60.000000
```

Bu verileri okutmak için fscanf1.m program parçasını kullanalım:

```
fid=fopen('veriler.dat','r');  
[A,c]=fscanf(fid,'%f',[2,6])  
fclose(fid);
```

Program koşturulduğunda ekranda:

2.4.2 Formatlı Verilerin Dosyalardan Okunması

```
>> fscanf1  
A =  
1 2 3 4 5 6  
10 20 30 40 50 60  
c =  
12  
>>
```

çıktısı elde edilir. Görüldüğü gibi okutma işlemi doğru bir şekilde gerçekleştirilmiş, toplam 12 eleman okutulmuştur.

Bir diğer örnek okutma ise yine önceki örneklerde oluşturulan tab.txt dosyasındaki verilerin okutulmasına ilişkin olacaktır.

Bu dosyadaki verileri tekrar hatırlayalım:

2.4.2 Formatlı Verilerin Dosyalardan Okunması

```
>> fprintf1  
>> type tab.txt
```

1	1.000
2	0.500
3	0.333
4	0.250

Şimdi bu verileri okutmak için aşağıdaki gibi bir program parçası kullanalım (fscanf2.m):

```
fid=fopen('tab.txt','r');  
[A,c]=fscanf(fid,'%f',[2,4]);  
A=A'  
fclose(fid);
```

Şimdi programı koşturalım:

2.4.2 Formatlı Verilerin Dosyalardan Okunması

```
>> fscanf2  
A =  
1.0000 1.0000  
2.0000 0.5000  
3.0000 0.3330  
4.0000 0.2500  
>>
```

Dosyanın düzgün bir şekilde okutıldığı görülmektedir.

Skaler bir sayının bir dosyaya yazdırılıp okutulması da örnek olarak fprintf2.m ve fscanf3 program parçalarının koşturulması ile sağlanabilir.