Entering Matrices

Entering the numbers in these two ways creates the same matrix A.

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


A =


    1     2     3

    4     5     6

    7     8     9


EDU>> A = [

     1 2 3

     4 5 6

     7 8 9 ]


A =


    1     2     3

    4     5     6

    7     8     9
```

Entering complex matrices can also be done in two ways.

```
EDU>> A = [1 2; 3 4] + i*[5 6; 7 8]


A =
```

```
   1.0000 + 5.0000i   2.0000 + 6.0000i
   3.0000 + 7.0000i   4.0000 + 8.0000i
```

```
EDU>> A = [1+5i 2+6i; 3+7i 4+8i]

A =

   1.0000 + 5.0000i   2.0000 + 6.0000i
   3.0000 + 7.0000i   4.0000 + 8.0000i
```

Matrix Operations, Array Operations

To use matrix operations, the size of the matrices must be compatible. The operations that can be used are:

+	addition
-	subtraction
*	multiplication
^	power
‘	conjugate transpose
\	left division
/	right division

Operations can be used to make calculations entry-wise by putting a period in front of the desired operation.

```
EDU>> [1,2,3,4].*[1,2,3,4]

ans =
```

```
     1     4     9    16

EDU>> [1,2,3,4].^2

ans =

     1     4     9    16
```

Matrix Building Functions

Some convenient functions that can be used to populate a matrix include:

eye	identity matrix
zeros	matrix of zeros
ones	matrix of ones
diag	create or extract diagonals
triu	upper triangular part of a matrix
tril	lower triangular part of a matrix
rand	randomly generated matrix
hilb	Hilbert matrix
magic	magic square

```
EDU>> n=2

n =

     2

EDU>> zeros(m,n)
```

```
ans =

     0     0
     0     0
     0     0

EDU>> zeros(n)

ans =

     0     0
     0     0

EDU>> A = [1 2; 3 4]

A =

     1     2
     3     4

EDU>> zeros(size(A))

ans =
```

```
    0     0
    0     0

EDU>> x= [1 2]

x =

    1     2

EDU>> diag(x)

ans =

    1     0
    0     2

EDU>> A = [1 2; 3 4]

A =

    1     2
    3     4

EDU>> diag(A)
```

```
ans =

     1
     4

EDU>> diag(diag(A))

ans =

     1     0
     0     4

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

A =

     1     2     3
     4     5     6
     7     8     9

EDU>> B =[A, zeros(3,2); zeros(2,3), eye(2)]

B =

     1     2     3     0     0
```

4 5 6 0 0

7 8 9 0 0

0 0 0 1 0

0 0 0 0 1

For, while, if –and relations

The functions primer and primers will produce the same vectors while primers will produce the same vectors but in reverse order.

```
function primer(n)

x=[]; for i=1:n, x=[x,i^2],end
end
```

EDU>> primer(3)

x =

1

x =

1 4

x =

1 4 9

```
function primers(n)
```

```
x=[];
for i=1:n
    x=[x,i^2]
end
end
```

EDU>> primers(3)

x =

1

x =

1 4

x =

1 4 9

```
function primerss(n)

x=[]; for i=n:-1:1, x=[x,i^2], end
end
```

EDU>> primerss(3)

x =

9

x =

9 4

x =

9 4 1

The hil function produces a m-by-n Hilbert matrix. The semicolon inside the for statement stops the program from printing unwanted intermediate results.

```
function hil(m,n)

for i=1:m
    for j=1:n
        H(i,j)=1/(i+j-1);
    end
end
H
end
```

EDU>> hil(3,2)

H =

1.0000 0.5000

0.5000 0.3333

0.3333 0.2500

Using the for statement allows for any matrix to be used instead of simple using 1:n.

```
function s=su(A)

s=0;
for c=A
    s = s + sum(c);
end

end
```

EDU>> A= [1 2 3; 4 5 6]

A =

1 2 3

4 5 6

EDU>> su(A)

ans =

21

The function uses the while operator so that the statement will be executed only if the relation remains true and will stop execution once the relation is no longer true.

```
function onlyy(a)

n=0;
while 2^n<a
    n = n + 1;
end
n
end
```

EDU>> onlyy(5)

n =

    3

The function uses the if operator so that the statements will be executed only if the relation is true.

```
function parity=relt(n)

if n<0
    parity=0;
elseif rem(n,2)==0
    parity=2;
else
    parity=1;
end

end
```

EDU>> relt(1)

ans =

    1

EDU>> relt(-2)

ans =

    0

EDU>> relt(2)

ans =

2

EDU>> relt(0)

ans =

2

The relational operators that can be used in MATLAB include:

<	less than
>	greater than
<=	less than of equal
>=	greater than or equal
==	equal
~=	not equal

IT is important to remember that = is an assignment while == is a relation.

Relations may be connected or quantified by logical operators:

&	and
|	or
~	not

When it is applied to a scalar, a relation is just the scalar 1 or 0 depending on if the relation is true or false.

EDU>> 3<5

ans =

```
     1

EDU>> 3>5

ans =

     0

EDU>> 3==5

ans =

     0

EDU>> 3==3

ans =

     1

EDU>> a=rand(5)

a =

    0.8147    0.0975    0.1576    0.1419    0.6557
```

```
    0.9058    0.2785    0.9706    0.4218    0.0357

    0.1270    0.5469    0.9572    0.9157    0.8491

    0.9134    0.9575    0.4854    0.7922    0.9340

    0.6324    0.9649    0.8003    0.9595    0.6787


EDU>> b=triu(a)


b =


    0.8147    0.0975    0.1576    0.1419    0.6557

         0    0.2785    0.9706    0.4218    0.0357

         0         0    0.9572    0.9157    0.8491

         0         0         0    0.7922    0.9340

         0         0         0         0    0.6787


EDU>> a==b


ans =


     1     1     1     1     1

     0     1     1     1     1

     0     0     1     1     1

     0     0     0     1     1

     0     0     0     0     1
```

<u>Scalar functions</u>

Some functions operate solely on scalars or element-wise on matrices. They include:

SIN	ASIN	EXP	ABS	ROUND
COS	ACOS	LOG	SQRT	FLOOR
TAN	ATAN	REM	SIGN	CEIL

Vector functions

Some functions operate solely on a vector. They include:

max	sum	median	any
min	prod	mean	all
sort		std	

The following function finds the maximum entry in matrix A. max(max(A)) is used because a function operates in a column-by-column fashion and produces a row vector.

```
function maxx(A)

max(max(A))

end
```

EDU>> A = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]

A =

```
     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16
```

EDU>> maxx(A)

ans =

16

Matrix functions

The reason why Matlab is so useful is because of the matrix functions. Some common ones include:

eig	eigenvalues and eigenvectors
inv	inverse
lu	LU factorization
rref	reduced row echelon form
det	determinant
size	size

```
A =

     1     2     3
     4     5     6
     7     8     9

EDU>> eigg(A)

ans =

   16.1168
   -1.1168
   -0.0000

EDU>> eiggg(A)
```

U =

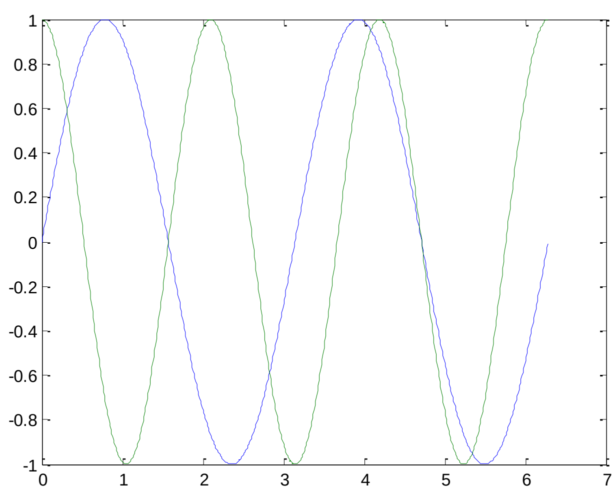
```
   -0.2320   -0.7858    0.4082
   -0.5253   -0.0868   -0.8165
   -0.8187    0.6123    0.4082
```

D =

```
   16.1168         0         0
         0   -1.1168         0
         0         0   -0.0000
```

Command line editing and recall

```
function plott()
m=2; n=3; x=0:.01:2*pi; y=sin(m*x); z=cos(n*x); plot(x,y,x,z)
end
```



Submatrices and colon notation

EDU>> 0.2:0.2:1.2

ans =

    0.2000   0.4000   0.6000   0.8000   1.0000   1.2000

EDU>> 5:-1:1

ans =

     5   4   3   2   1

```
function onetwo()

x=[0.0:0.1:2.0];
y=sin(x);
[x y]

end
```

EDU>> onetwo()

ans =

  Columns 1 through 7

         0   0.1000   0.2000   0.3000   0.4000   0.5000   0.6000

  Columns 8 through 14

```
   0.7000    0.8000    0.9000    1.0000    1.1000    1.2000    1.3000

  Columns 15 through 21

   1.4000    1.5000    1.6000    1.7000    1.8000    1.9000    2.0000

  Columns 22 through 28

        0    0.0998    0.1987    0.2955    0.3894    0.4794    0.5646

  Columns 29 through 35

   0.6442    0.7174    0.7833    0.8415    0.8912    0.9320    0.9636

  Columns 36 through 42

   0.9854    0.9975    0.9996    0.9917    0.9738    0.9463    0.9093
```

Text strings, error messages, input

```
function show()

s='This is a test'

end
```

```
EDU>> show()


s =
```

This is a test

```
function showw()

disp('this message is hereby displayed')

end
```

EDU>> showw()

this message is hereby displayed

```
function showww()

error('sorry, the matrix must be symmetric')

end
```

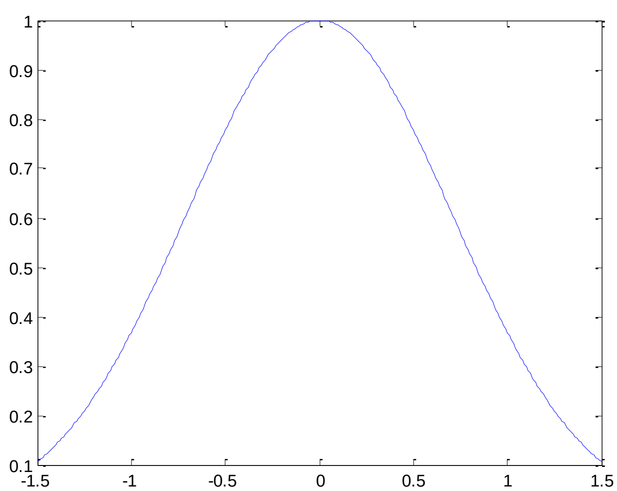
EDU>> showww()

??? Error using ==> showww at 3

sorry, the matrix must be symmetric

<u>Planar plots</u>

```
function plt()

x=-1.5:.01:1.5; y=exp(-x.^2); plot(x,y)

end
```

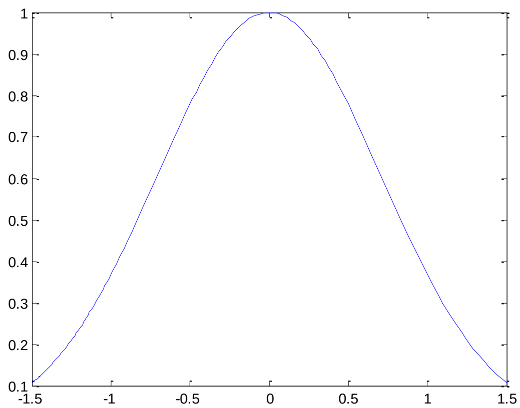


M file written:

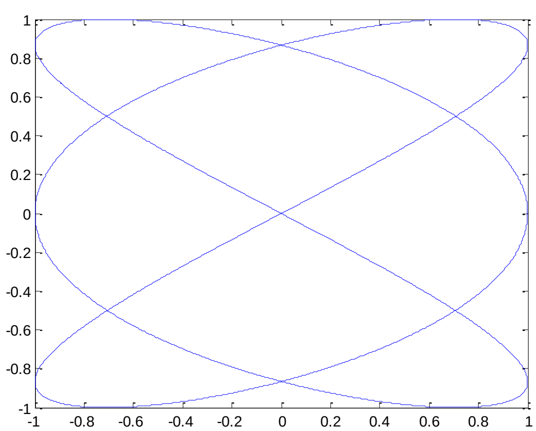
```
function y=expnormal(x)
y=exp(-x.^2)
```

Call M file in program:

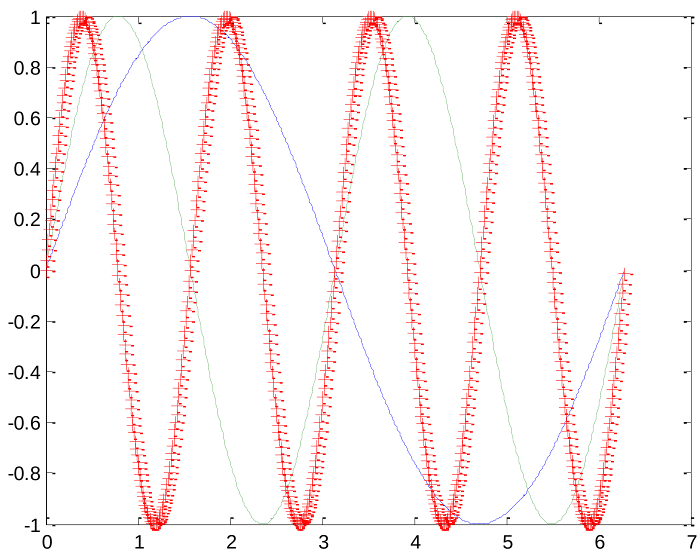
```
function lott()

fplot('expnormal',[-1.5,1.5])

end
```




```
function lotta()

t=0:.001:2*pi; x=cos(3*t); y=sin(2*t); plot(x,y)

end
```

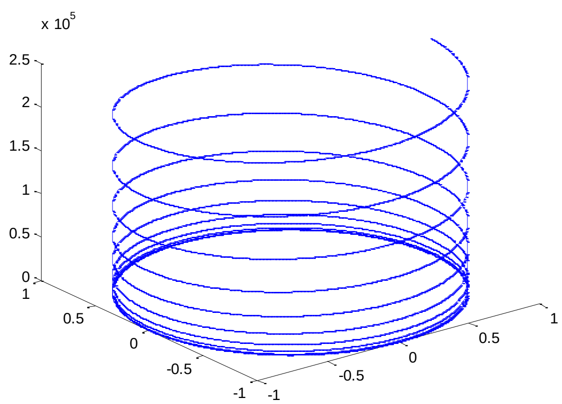


```
function lotts()

x=0:.01:2*pi; y1=sin(x); y2=sin(2*x); y3=sin(4*x);
plot(x,y1,'--',x,y2,':',x,y3,'+')

end
```



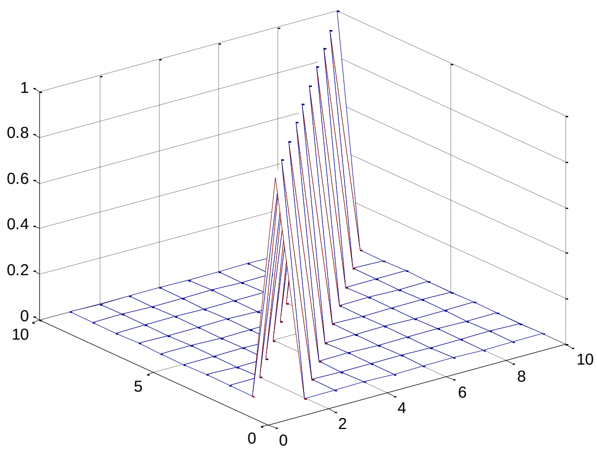
3-D plots

```
function threed()

t=.01:.01:20*pi; x=cos(t); y=sin(t); z=t.^3; plot3(x,y,z)

end
```

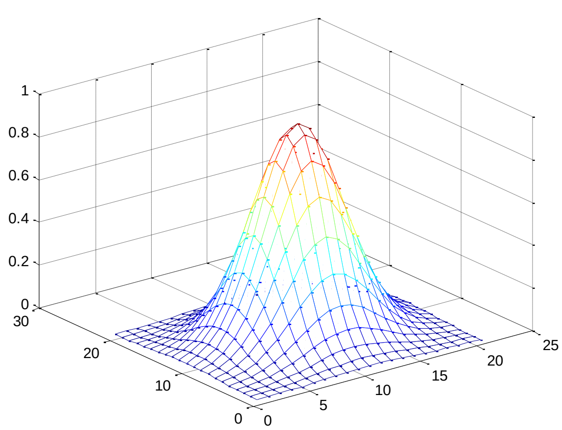


3-D mesh and surface plots

```
function mee()

mesh(eye(10))

end
```




```
function meee()

xx=-2:.2:2;
yy=xx;
[x,y]=meshgrid(xx,yy);
z=exp(-x.^2-y.^2);
mesh(z)

end
```



Sparse Matrix Computations

```
function spr()

F=floor(10*rand(6)); F=triu(tril(F,1),-1);
S=sparse(F)

end
```

EDU>> spr()

S =

```
(1,1)    8
(2,1)    9
(1,2)    2
(2,2)    5
(3,2)    9
(2,3)    4
(3,3)    8
(4,3)    1
(3,4)    6
(5,4)    8
(4,5)    3
(5,5)    6
(6,5)    1
(6,6)    8
```

```
function sprr()

m=6; n=6; e=ones(n,1); d=-2*e;
T=spdiags([e,d,e],[-1,0,1],m,n)

end
```

```
EDU>> sprr()

T =

   (1,1)      -2
   (2,1)       1
   (1,2)       1
   (2,2)      -2
   (3,2)       1
   (2,3)       1
   (3,3)      -2
   (4,3)       1
   (3,4)       1
   (4,4)      -2
   (5,4)       1
   (4,5)       1
   (5,5)      -2
   (6,5)       1
   (5,6)       1
   (6,6)      -2
```