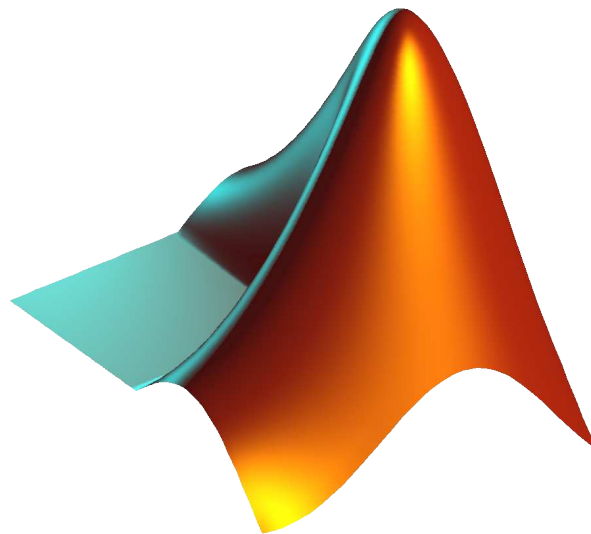


Introduction to MATLAB

SI 507 : Numerical Analysis
Indian Institute of Technology Bombay
August 5, 2016



Introduction

- MATLAB software is used for computation in engineering, science and applied mathematics.
- Unlike Mathematica, Symbolic computations and manipulations are not readily available. (through Symbolic Toolbox).
- Compared to C++ and Fortran, MATLAB is user-friendly, however, the execution speed is slower.
- Yet, its interactive syntax and extensive documentation, and suitability for solving PDEs has made it the bread and butter of applied mathematicians everywhere.

Introduction

- On startup, MATLAB presents a multipaneled desktop including
 - **Command Window** where you can type MATLAB commands at the prompt `>>`.
 - **Current Directory** shows the contents of current path.
 - **Workspace** window shows the variable names currently defined.
 - **Command History** records all the commands ever performed.
- **Help** is your friend. Use `help plot` or `doc plot` to find out information about the plot command.

Exercises

- Use the `doc` command to find out how to find prime factors of an integer. What is the largest prime factor of 20830123?

Exercises

- Use the `doc` command to find out how to find prime factors of an integer. What is the largest prime factor of 20830123?

Ans. 541.

Exercises

- Use the `doc` command to find out how to find prime factors of an integer. What is the largest prime factor of 20830123?

Ans. 541.

- Report the current date and time.

Arrays and Matrices

Arrays and Matrices

To write a matrix, enclose its elements in square brackets, use spaces or commas to separate columns and use semicolons or new lines to separate rows.

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

```
A =
```

```
1     2     3
4     5     6
7     8     9
```


Column Vector

```
>> b = [0;1;0]
```

```
b =
```

```
0
```

```
1
```

```
0
```

Dimensions

| | |
|---------------------|----------------------------------|
| <code>size</code> | dimensions of a matrix |
| <code>ndims</code> | number of dimensions |
| <code>length</code> | size of longest dimension |
| <code>find</code> | indices of the non-zero elements |

Common Matrices

| | |
|---|---|
| <p> <code>[]</code> <code>eye(3)</code> <code>diag([1 2 3])</code> <code>zeros(2,3)</code> <code>ones(2,4)</code> <code>triu(A,-1)</code> <code>tril(A,1)</code> <code>rand,randn</code> <code>i=[0:0.2:10]</code> </p> | <p> empty matrix Identity Matrix of given dimension Diagonal Matrix with given elements Zero Matrix of given dimensions Matrix of ones of given dimensions Upper triangular part of A one below the main diagonal Lower Triangular part of A one above the main diagonal Matrices of given dimensions with random entries Row vector containing entries from 0 to 10 spaced by 0.2 </p> |
|---|---|

Referencing Elements

```
>> A(2,3)    % single element  
ans =  
6
```

```
>> A(2)      % one index from a matrix  
ans =  
4
```

Referencing Elements

```
>> A([1 2]) % multiple elements
```

```
ans =
```

```
1     4
```

```
>> A(1:2,2:3) % a submatrix
```

```
ans =
```

```
2     3
```

```
5     6
```

Referencing Elements

```
>> A(:,3)      % third column
ans =
3
6
9
```

```
>> A(2,:)     % second row
ans =
4      5      6
```

Referencing Elements

```
>> A(:, :)      % full matrix
ans =
1      2      3
4      5      6
7      8      9
```

Referencing Elements

```
>> A(:)      % all entries in a column
ans =
1
4
7
2
5
8
3
6
9
```


Relational Operators

| | |
|----|--------------------------|
| == | equal to |
| ~= | not equal to |
| < | less than |
| > | greater than |
| <= | less than or equal to |
| >= | greater than or equal to |

Referencing using Relational Operators

```
>> A>5
```

```
ans =
```

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

```
>> A(ans)
```

```
ans =
```

```
7
8
6
9
```

Matrix Operations

The common matrix operations are $+$, $-$, $*$, $^$, $'$, \backslash .

```
>> A=[1,2,3;4,5,6;7,8,9];  
>> B=[1,0,0;5,8,7;0,0,1];  
>> A+B      % matrix addition
```

```
ans =
```

```
2     2     3  
9     13    13  
7     8     10
```

```
>> A*B      % matrix multiplication
```

```
ans =
```

```
11    16    17  
29    40    41  
47    64    65
```

Solving Linear Systems

```
>> B=[1,0,0;5,8,7;0,0,1];
```

```
>> b=[0;1;-1];
```

```
>> x=B\b
```

```
x =
```

```
0
```

```
1
```

```
-1
```

```
>> b-B*x
```

```
ans =
```

```
0
```

```
0
```

```
0
```

More Matrix Operations

| | |
|-------------------|----------------------|
| <code>\</code> | solve linear systems |
| <code>rank</code> | rank |
| <code>det</code> | determinant |
| <code>norm</code> | 2-norm |
| <code>expm</code> | matrix exponential |
| <code>lu</code> | LU Factorization |
| <code>qr</code> | QR Factorization |

Elementwise Operations

- Scalars are always added elementwise to arrays.

```
>> b=[0,2,1];  
>> b+2  
ans =  
2     4     3
```

- Elementwise Multiplication or Division can be performed by preceding the operator with a dot.

```
>> A.*B  
ans =  
1     0     0  
20    40    42  
0     0     9
```

- Elementary functions like `sin`, `cos` etc. operate elementwise.

Dimension Reducing Operations

| | | | |
|------|------|--------|-----|
| sum | max | mean | any |
| min | diff | median | all |
| sort | prod | std | |

Task 2

- Let A be a random matrix generated by `rand(8)`. Find the maximum values in each column, in each row, overall. Also use `find` to find the indices of all elements that are larger than 0.9.

Task 2

- Let A be a random matrix generated by `rand(8)`. Find the maximum values in each column, in each row, overall. Also use `find` to find the indices of all elements that are larger than 0.9.
- How to add the diagonal elements of a matrix?

Task 2

- Let A be a random matrix generated by `rand(8)`. Find the maximum values in each column, in each row, overall. Also use `find` to find the indices of all elements that are larger than 0.9.
- How to add the diagonal elements of a matrix?
- Use `diag` to create a 16×16 matrix with -2 on the main diagonal, 1 on the super- and the sub-diagonal.

Task 2

- Let A be a random matrix generated by `rand(8)`. Find the maximum values in each column, in each row, overall. Also use `find` to find the indices of all elements that are larger than 0.9.
- How to add the diagonal elements of a matrix?
- Use `diag` to create a 16×16 matrix with -2 on the main diagonal, 1 on the super- and the sub-diagonal.
- How to multiply the third column of a matrix by 2?

Scripts and Functions

Scripts

- A file containing MATLAB commands saved with a *.m* extension is called an **M-file**.
- A collection of consecutive commands stored in an **M-file** is called a **Script**.
- MATLAB comes with an in-built editor.
- You can execute any M-files that are located in the path.

An Example Script

```
>> type('example.m')  
  
% This is an example script  
% that prints a random number  
  
rand(1)  
>>
```

An Example Script

```
>> type('example.m')  
  
% This is an example script  
% that prints a random number  
  
rand(1)  
>>
```

A comment

An Example Script

```
>> type('example.m')  
  
% This is an example script  
% that prints a random number  
  
rand(1)  
>>
```

A comment

A command

Functions

- A function is an **M-file** that starts with a line such as `function [out1,out2] = myfun(in1,in2,in3).`
- It can have any number of input and output variables.
- The *myfun* part should be the name of the file. (myfun.m)
- A function differs from a script in that the variables defined within a function do not exist outside of it.
- Thus, a function has its own local workspace.

An Example Function

```
% This program returns the two  
% roots of a quadratic equation.
```

```
function [x1,x2] = quadsol(a,b,c)  
d = sqrt(b^2-4*a*c);  
x1 = (-b+d)/(2*a);  
x2 = (-b-d)/(2*a);
```

An Example Function

```
% This program returns the two  
% roots of a quadratic equation.
```

```
function [x1,x2] = quadsol(a,b,c)  
d = sqrt(b^2-4*a*c);  
x1 = (-b+d)/(2*a);  
x2 = (-b-d)/(2*a);
```

A comment

An Example Function

```
% This program returns the two  
% roots of a quadratic equation.
```

```
function [x1,x2] = quadsol(a,b,c)  
d = sqrt(b^2-4*a*c);  
x1 = (-b+d)/(2*a);  
x2 = (-b-d)/(2*a);
```

A comment

Function Structure

if

```
if (x==inf) || ~isreal(x)
    disp('Wrong Input')
    y=NaN;
elseif (x == round(x)) && (x>0)
    y = prod(1:(x-1));
else
    y = gamma(x);
end
```

if

```
if (x==inf) || ~isreal(x)
    disp('Wrong Input')
    y=NaN;
elseif (x == round(x)) && (x>0)
    y = prod(1:(x-1));
else
    y = gamma(x);
end
```

Branches of if statements to find the factorial of a number.

switch

```
switch units
  case 'length'
    disp('meters')
  case 'volume'
    disp('liters')
  case 'time'
    disp('seconds')
  otherwise
    disp('I don't know')
end
```

switch

```
switch units
  case 'length'
    disp('meters')
  case 'volume'
    disp('liters')
  case 'time'
    disp('seconds')
  otherwise
    disp('I don't know')
end
```

Commands following the matching case are executed.

for Loop

```
>> f = [1 1];  
>> for n = 3:10  
    f(n) = f(n-1) + f(n-2);  
end
```

for Loop

```
>> f = [1 1];  
>> for n = 3:10  
    f(n) = f(n-1) + f(n-2);  
end
```

A for loop to
generate the Fi-
bonacci Sequence

Another Example

```
>> x = 1:100; s=0;  
>> for j = find(isprime(x))  
    s = s + x(j);  
end
```

Another Example

```
>> x = 1:100; s=0;  
>> for j = find(isprime(x))  
    s = s + x(j);  
end
```

We can use
any row vector
in a for loop.

while Loop

```
x = 0;  
while x < 50  
    x = x + 1;  
end
```

while Loop

```
x = 0;  
while x < 50  
    x = x + 1;  
end
```

A while loop iterates a command as many times as the condition is fulfilled.

Another Example

```
n = 0;
while x > 1
    x = x/2;
    n = n+1;
    if n > 50
        break
    end
end
```

Another Example

```
n = 0;  
while x > 1  
  x = x/2;  
  n = n+1;  
  if n > 50  
    break  
  end  
end  
end
```

A break statement
may be embedded
in a while loop
to prevent
infinite loops.

Review Exercises

Store the matrix

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 4 \\ 1 & 7 & 1 \end{pmatrix}$$

in the variable A .

Store the matrix

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 4 \\ 1 & 7 & 1 \end{pmatrix}$$

in the variable A .

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

```
A =
```

```
    1    2    1
    2    3    4
    1    7    1
```

Create row vector

$$(2 \ 4 \ 6 \ 8 \ 10 \ 12 \ 14)$$

using `:` and `linspace`.

Create row vector

$$(2 \ 4 \ 6 \ 8 \ 10 \ 12 \ 14)$$

using `:` and `linspace`.

```
>> [2:2:14]
```

```
ans =
```

```
2    4    6    8   10   12   14
```

```
>> linspace(2,14,7)
```

```
ans =
```

```
2    4    6    8   10   12   14
```

Print the $(2, 2)$ element of the matrix A . Also print the 5th element of A .

Print the (2, 2) element of the matrix A . Also print the 5th element of A .

```
>> A(2,2)
ans =
     3
>> A(5)
ans =
     3
```

Print first row and third column of A .

Print first row and third column of A .

```
>> A(1,:)
ans =
     1     2     1
>> A(:,3)
ans =
     1
     4
     1
```

Use `find` to return all indices containing entries greater than 2, then print those entries.

Use `find` to return all indices containing entries greater than 2, then print those entries.

```
>> find(A>2)
ans =
     5
     6
     8
>> A(find(A>2))
ans =
     3
     7
     4
```

Use `find` to return all indices containing entries greater than 2, then print those entries.

```
>> find(A>2)
ans =
     5
     6
     8
>> A(find(A>2))
ans =
     3
     7
     4
```

```
>> A>2
ans =
     0     0     0
     0     1     1
     0     1     0
>> A(A>2)
ans =
     3
     7
     4
```

Store the row vector

$$(2 \quad 4 \quad 6)$$

in b and the column vector

$$\begin{pmatrix} 2 \\ 5 \\ 1 \end{pmatrix}$$

in c .

Store the row vector

$$(2 \quad 4 \quad 6)$$

in b and the column vector

$$\begin{pmatrix} 2 \\ 5 \\ 1 \end{pmatrix}$$

in c .

```
>> b=[2 4 6]
```

```
b =
```

```
    2    4    6
```

```
>> c=[2;5;1]
```

```
c =
```

```
    2
```

```
    5
```

```
    1
```

Join c to A and b to A along appropriate dimensions.

Join c to A and b to A along appropriate dimensions.

```
>> D = [ A c ]
```

```
D =
```

```
    1    2    1    2
    2    3    4    5
    1    7    1    1
```

```
>> C = [ A ; b ]
```

```
C =
```

```
    1    2    1
    2    3    4
    1    7    1
    2    4    6
```


Assuming that all the entries of A are non-zero, multiply each column by a number so that the sum of the column is 1. Do not use loops. Use the `sum` and `diag` commands.

Assuming that all the entries of A are non-zero, multiply each column by a number so that the sum of the column is 1. Do not use loops. Use the `sum` and `diag` commands.

```
>> A=A*diag(1./sum(A))  
A =  
    0.2500    0.1667    0.1667  
    0.5000    0.2500    0.6667  
    0.2500    0.5833    0.1667
```

Modify the previous command to allow for zero columns.

Modify the previous command to allow for zero columns.

```
>> B=[ 0 1 0;0 2 1; 0 3 4]
```

```
B =
```

```
0    1    0
```

```
0    2    1
```

```
0    3    4
```

```
>> p=diag(sum(B))
```

```
p =
```

```
0    0    0
```

```
0    6    0
```

```
0    0    5
```

```
>> p(p~=0)=1./p(p~=0)
```

```
p =
```

```
0    0    0
```

```
0    0.1667    0
```

```
0    0    0.2000
```

```
>> B*p
```

```
ans =
```

```
0    0.1667    0
```

```
0    0.3333    0.2000
```

```
0    0.5000    0.8000
```

Write an M-file containing a function that prints your name.

Write an M-file containing a function that prints your name.

```
function [] = myname();  
disp('Vivek');
```

Write an M-file containing a function that takes two inputs and adds them and returns it as output.

Write an M-file containing a function that takes two inputs and adds them and returns it as output.

```
function [x] = adding(a,b);  
x=a+b;
```


Write a function that inputs a number and returns a logical true value if it is even. Use `if` and `mod` command.

Write a function that inputs a number and returns a logical true value if it is even. Use `if` and `mod` command.

```
function [x] = evenodd(a);  
if mod(a,2)==0  
    x = 1;  
else  
    x = 0;  
end
```

Task 2.1

- Write a function `allfacts` to find all the factors of an integer.

Task 2.1

- Write a function `allfacts` to find all the factors of an integer.
- Use the above function to write another function `isperfect` to test whether a number is *perfect*. Recall that a number is said to be perfect if it is the sum of its factors, not including itself.

Task 2.1

- Write a function `allfacts` to find all the factors of an integer.
- Use the above function to write another function `isperfect` to test whether a number is *perfect*. Recall that a number is said to be perfect if it is the sum of its factors, not including itself.
- Now use `allfacts` to find the largest prime factor of an integer.

Solutions

- **allfacts:**

```
function [x] = allfacts(a);  
m=[1:a];  
x=m(mod(a,m)==0);
```

Solutions

- `isperfect`:

```
function [x] = isperfect(a);  
if 2*a==sum(allfacts(a))  
    x=1;  
else  
    x=0;  
end
```

Solutions

- largest prime factor of an integer:

```
function [x] = largestprimefactor(a);  
p=allfactors(a);  
x=max(p(isprime(p))));
```


Advanced
Functions

Function Handles

- MATLAB distinguishes between the call to a function for evaluation and the call to the function as an abstract entity.
- Some functions require another function as their input, for example, `fzero`.

```
>> fzero(sin,1) %error
```

- The correct invocation is

```
>> fzero(@sin,1) %error
```

Anonymous Functions

- Other than storing a function in an **M-file**, MATLAB allows creation of a *function-handle* directly.

```
>> sincos = @(x) sin(x)+cos(x);
```

- This prevents unnecessary proliferation of M-files.
- This function can now be used for evaluation as well as a handle.
- An anonymous function can have multiple inputs and outputs.

Inline Functions

- Older versions of MATLAB allows creation of an *inline function*.

```
>> f = inline('3*sin(2*x.^2)')  
f =  
Inline function:  
f(x) = 3*sin(2*x.^2)
```

- These can be invoked within a function as

```
>> fzero(f,1);
```

Bisection Method

Define a function handle f using `inline`. We can use the bisection method to compute the zeros of a continuous function. In this method, we take two endpoints a and b such that $f(a)f(b) < 0$. This ensures that there is a zero of the function between a and b . Then, we compute their midpoint m and depending on whether $f(a)f(m) < 0$ or $f(m)f(b) < 0$, we assign two new endpoints. We repeat this procedure till the length of the interval made by the endpoints goes below a certain tolerance tol . Now, define a function `bisect(f,a,b,tol)` for finding zero of f .

```
function [x] = bisection(f,a,b,tol);
x1=a;x2=b;len=b-a;
while(len>2*tol)
    m=(x1+x2)/2;y1=f(x1)*f(m);y2=f(x2)*f(m);
    if y1<0
        x2=m;
    elseif y2<0
        x1=m;
    else
        x=m;
        break;
    end
    len=x2-x1;
end
x=m;
```

Subfunctions

It is possible to have multiple functions defined within an M-file. Only the first function can be accessed from the workspace. The rest of the functions can only be accessed by the main function.


```
function [x1,x2] = quadsol(a,b,c)
d = discrim(a,b,c);
x1 = (-b+d)/(2*a);
x2 = (-b-d)/(2*a);
end
function D = discrim(A,B,C)
D = sqrt(B^2-4*A*C);
end
```

Nested Functions

It is also possible to nest functions within an M-file. The parent function and the nested functions can share variables.

```
function [x1,x2] = quadsol(a,b,c)
function D = discrim(A,B,C)
D = sqrt(B^2-4*A*C);
end

d = discrim(a,b,c);
x1 = (-b+d)/(2*a);
x2 = (-b-d)/(2*a);
end
```

Graphics

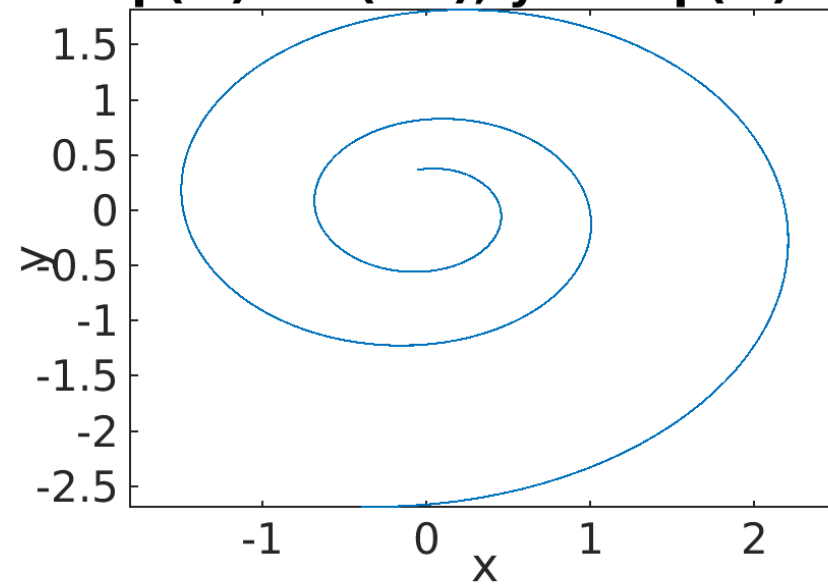
ez plots

- The command `ezplot` makes two-dimensional plots of explicit, implicit or parametric functions.

```
>> ezplot('sin(x)', [0, 2*pi])  
>> ezplot(@(x,y) x.^8+y.^8-1, [-1 1])  
>> ezplot('exp(-t).*cos(8*t)', 'exp(-t).*sin(8*t)', [-1 1])
```

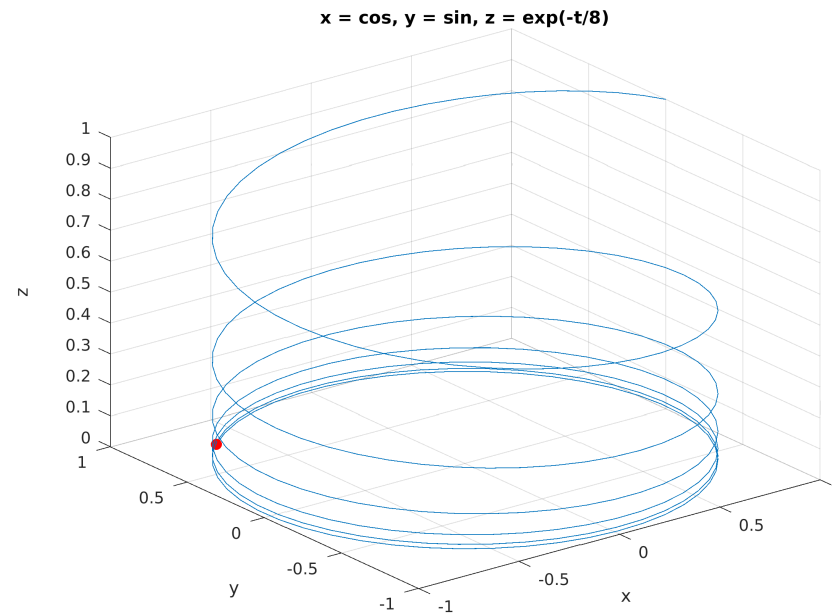
An example output

$$x = \exp(-t) \cos(8 t), y = \exp(-t) \sin(8 t)$$



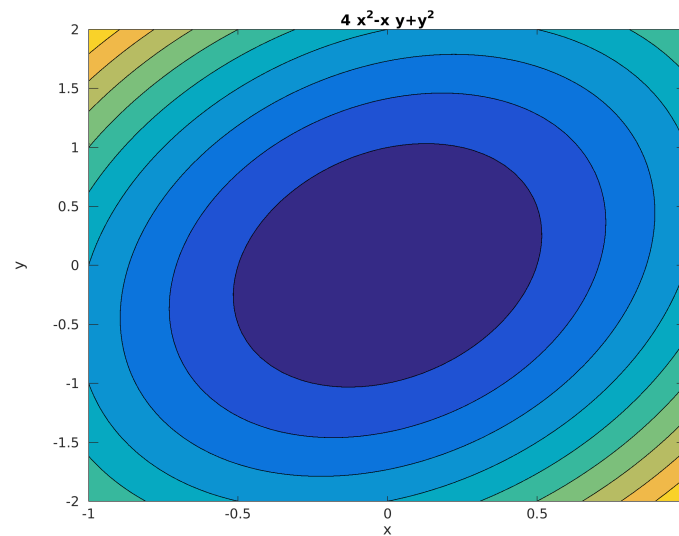
ezplot3

```
>> ezplot3(@cos,@sin,@(t) exp(-t/8),[0 40], 'animate')
```



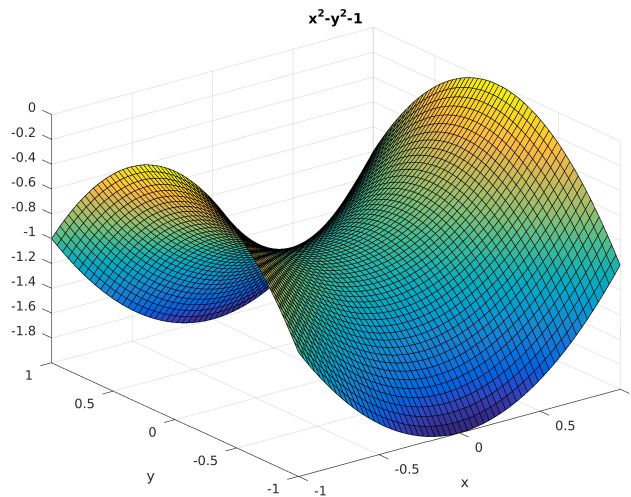
ezcontour

```
>> ezcontour(@(x,y) 4*x.^2-x.*y+y.^2, [-1 1 -2 2])  
>> ezcontourf(@(x,y) 4*x.^2-x.*y+y.^2, [-1 1 -2 2])
```



ezsurf

```
>> ezsurf(@(x,y) x.^2-y.^2-1, [-1 1])  
>> x=@(u,v) cosh(u).*cos(v);  
>> y=@(u,v) sinh(u).*cos(v);  
>> z=@(u,v) sin(v);  
>> ezmesh(x,y,z, [-1 1 0 2*pi])
```



Data Plots

- MATLAB can plot data from files.
- Data may be stored in a file in the form of columns such as

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |
| 7 | 8 |
- `load datafile` loads the data into MATLAB.
- Store columns into variables by issuing commands such as
`y = datafile(:,2), x = datafile(:,1).`
- Plot the data by issuing a command such as `plot(x,y,'rs-')`.

3D Plots

- To create a 3D plot from an explicit function, we first need to create a `meshgrid`.

```
>> x=pi*[0:0.02:1];  
>> y=2*x;  
>> [X Y] = meshgrid(x,y);
```

- Finally, define the function

```
>> Z = sin(X.^2+Y);
```

- and plot

```
>> surf(X,Y,Z)
```

- Other relevant commands are `mesh`, `waterfall`, `contour`, etc.

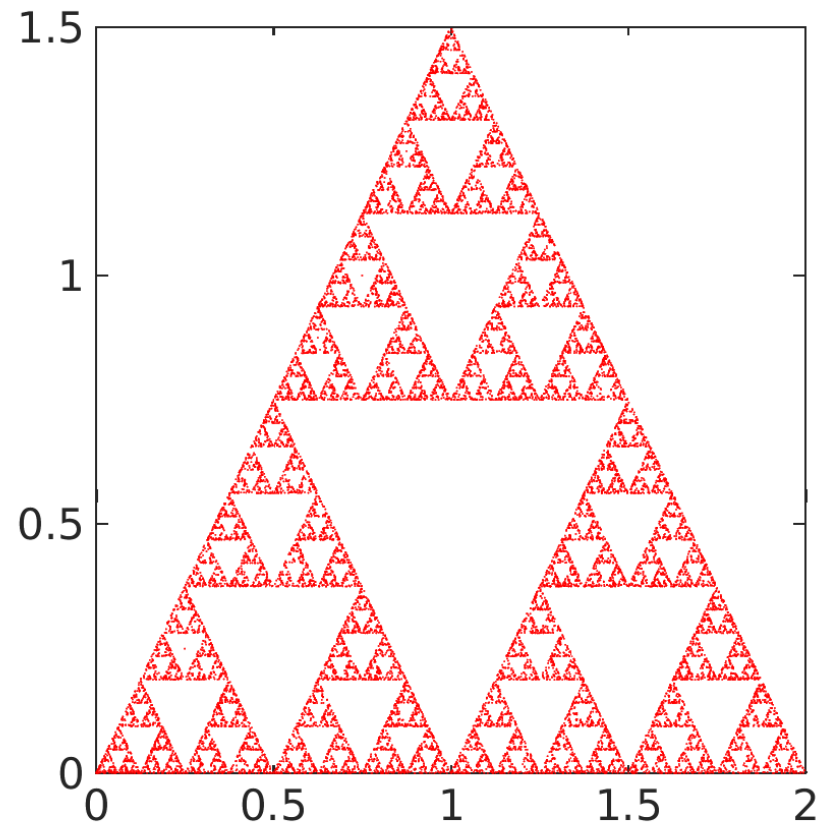
Annotations

```
>> ezplot(@(t) exp(-t/5).*sin(t), [0 6*pi])  
>> xlabel('time')  
>> ylabel('amplitude')  
>> title('Damped Oscillator')
```

Chaos Game

Define a function that picks an equilateral triangle and any point inside it. Then it picks one of the vertices of the triangle at random and plots the point midway between this point and the initial point. Repeat this procedure a large number of times and observe that a pattern emerges. You can use `plot(z)` where z is any complex number.

```
function []=chaosgame();  
x1=0+0i;x2=2+0i;x3=1+1.5i;  
x=[x1 x2 x3];  
loc=0.5+0.5i;  
for i=1:10000  
    pos = randi(length(x));  
    card = x(pos);    %midway between current  
    loc=(loc+card)/2;    %location and random vertex  
    plot(loc,'r.','markersize', 1);  
    hold on  
end
```



The programs and exercises in this presentation have been adapted from the book: **Driscoll, Tobin A. Learning MATLAB. Vol. 115. SIAM, 2009.**

Thank you