BOSTON UNIVERSITY

Introduction to MATLAB

Shaohao Chen

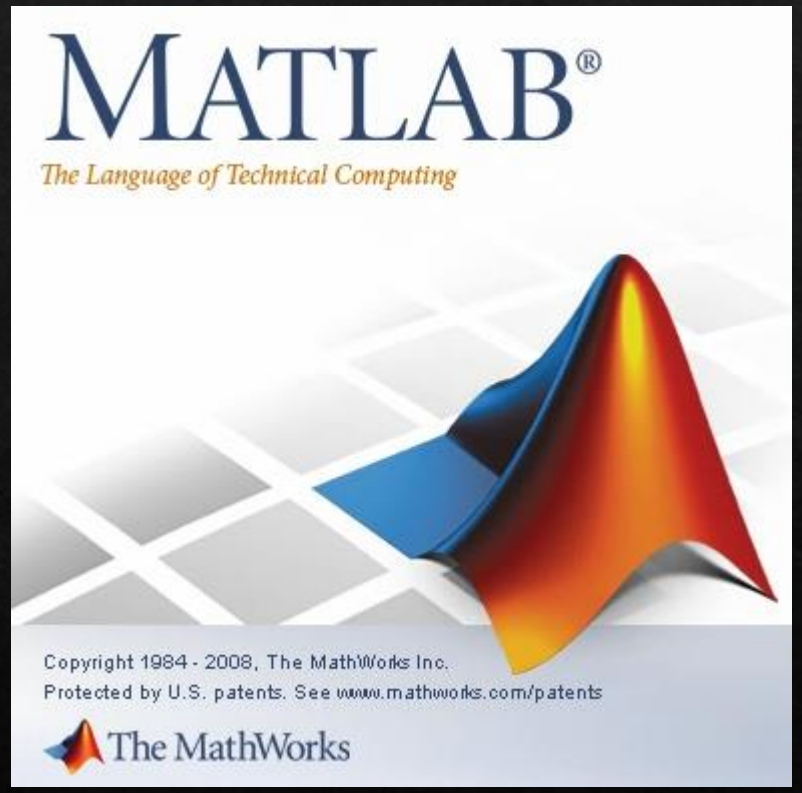
Research Computing

Information Services and Technology

Boston University

Outline

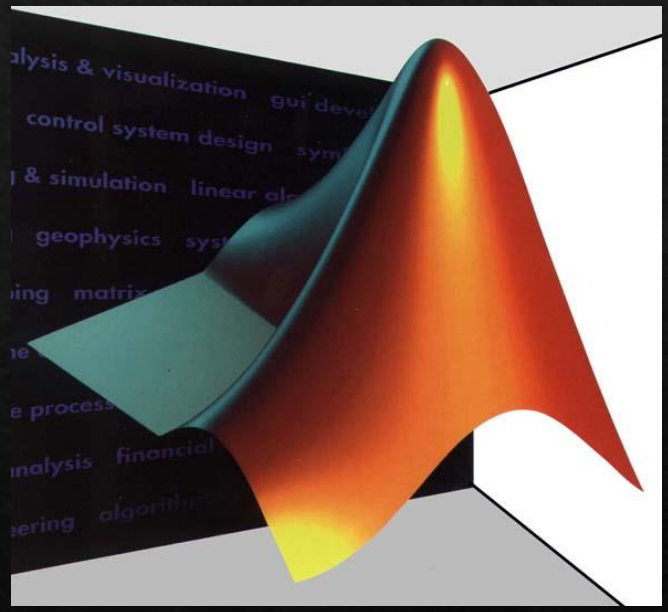
- Overview
- Interface
- Matrix
- Language basics
- Programming
- Plotting figures
- Solve problems:
  - ✓ Regression analysis
  - ✓ Numerical integration



Overview

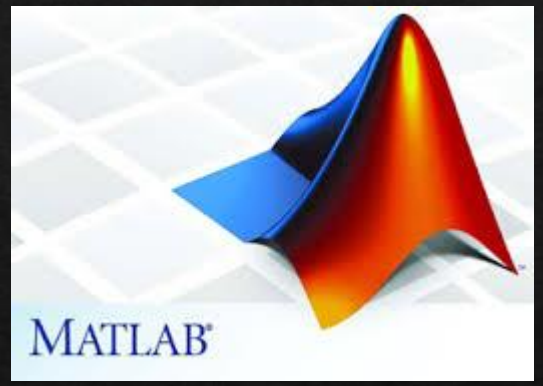
- Matlab is a high-level language and interactive environment for numerical computation, visualization, and programming.
- ✓ Analyze data.
- ✓ Develop algorithms.
- ✓ Create models and applications.

- Millions of engineers and scientists in industry and academia use Matlab.
- The language of technical computing.



Matlab *vs.* C/C++/Fortran

- **Compare Matlab to C/C++ or Fortran**



Pros:

Matlab …

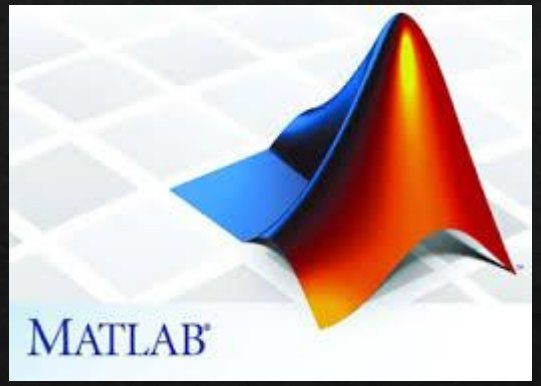
- is a higher level of programming language.
- is easier to start.
- provides convenient tools and built-in functions.
- provides user-friendly graphical interface.
- is good for post data analysis and visualization.

Cons:

- In many cases, the computational speed of Matlab is slower than that of C/C++ or Fortran.

Matlab *vs.* Mathematica

- **Compare Matlab to Mathematica**
  - A lot of built-in math functions in both.
  - Good graphical interface for both.



Pros: Matlab is …

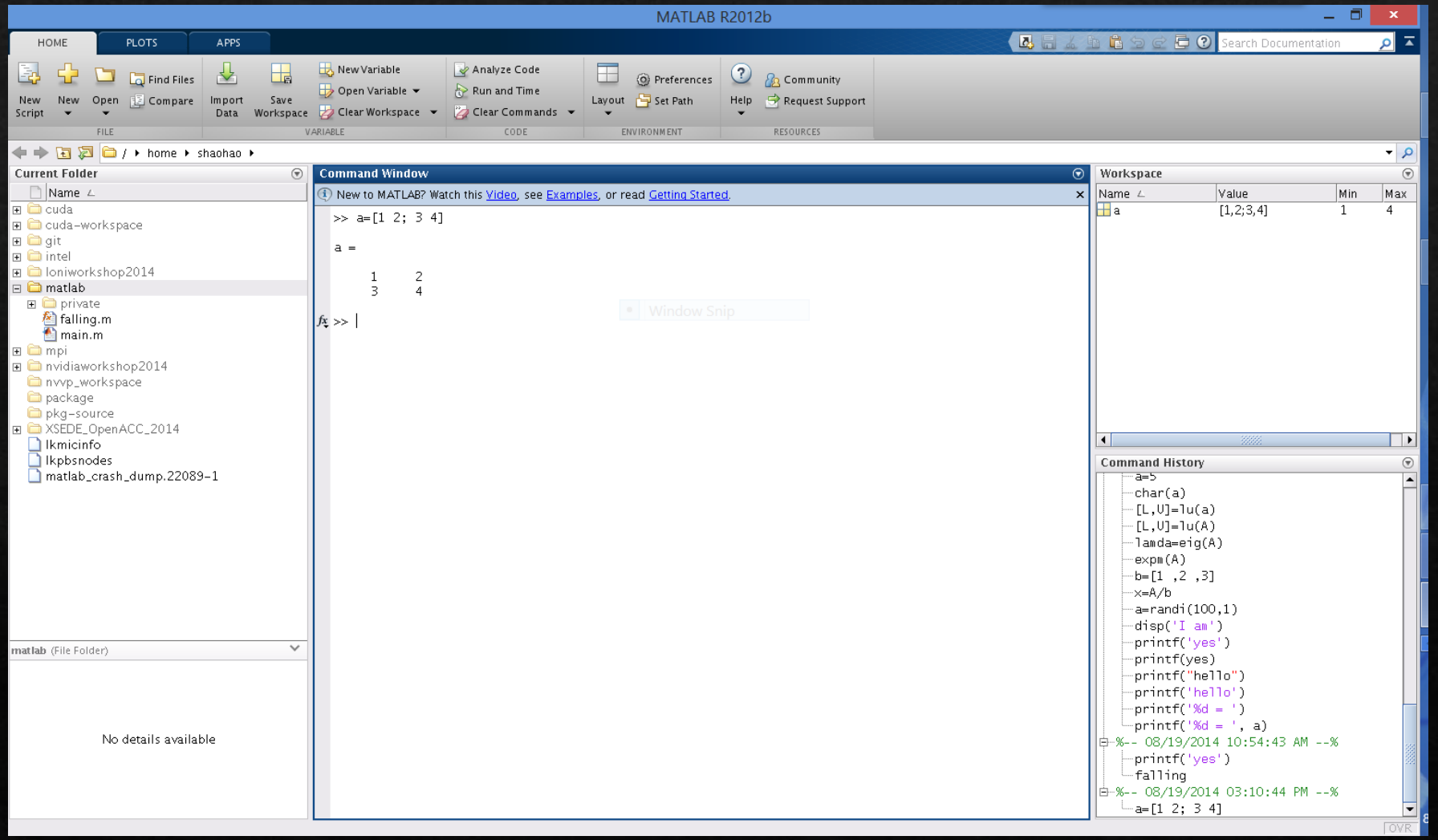
- better for numerical treatments.
- more popular in engineering.

Cons: Mematica is …

- better for analytical treatments.
- more popular in science.

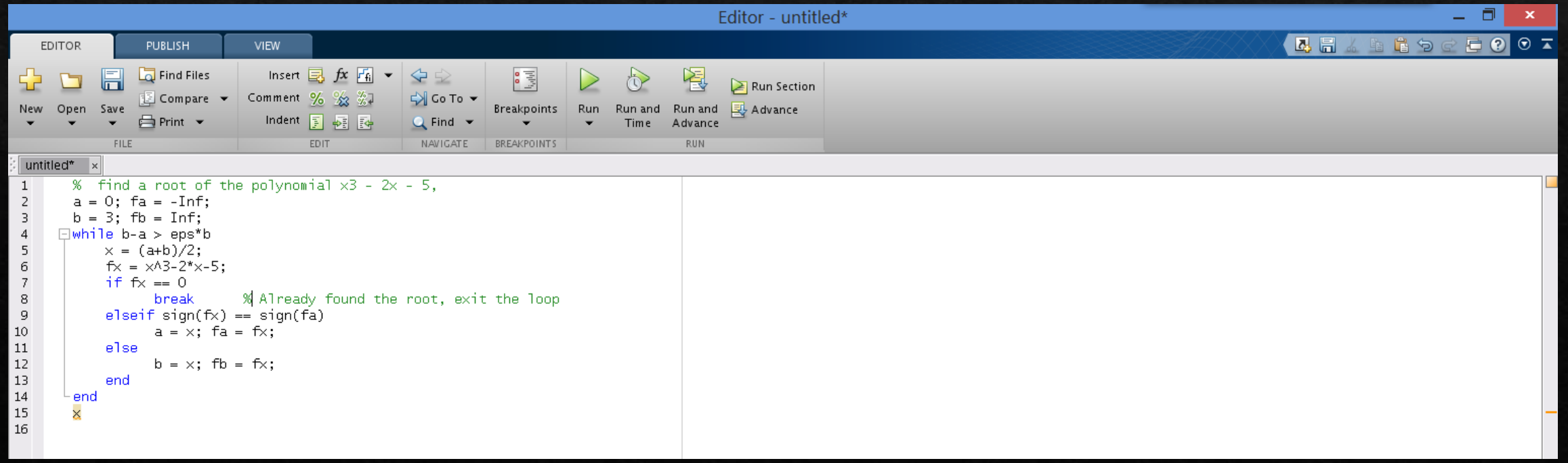
Matlab graphical interface

- Command window
- Workspace
- Navigator
- Toolstrip



M-file

- An m-file, or script file, is a simple text file where you can place MATLAB commands.
- Save your works.
- Convenient for debugging.
- Run directly. No explicit compilation and link.



Matrix and vectors

- Matlab = Matirx laboratory
- Ojects (e.g. data, text, color) in Matlab can be represented by matrices.

- **Scalar:** s = 5
- **Vector:** a = [1 2 3] ; % row vector
  b = [4; 5; 6] ; % column vector
  c = 1:5 % row
- **Matrix:** A = [1 2 3; 4 5 6; 7 8 9]

% Use percent sign for comments.

% Suppress output by adding a semicolon at the end of a command line

- **Functions to create matrices**

```
zeros(5,5)      % All zeros
ones(5,5)       % All ones
I=eye(5)        % Unit matrix
rand(5,5)       % Uniformly distributed random elements, between 0 and 1.
randn(5,5)      % Normally distributed random elements, with mean 0 and variance 1.
```

- **Vector operations:**
  - a+3 % add a scalar to a vector
  - a+b % element-by-element addition
  - a-b % element-by-element subtraction
  - a*3 % multiply a vector and a scalar
  - a.*b % element-by-element multiplication
  - a*c % dot product
  - dot(a,c) % dot product
  - a' % transpose

- **Vector operations (continued)**
  - cross(a,b) % cross product (only for arrays with 3 elements)
  - pinv(a) % Moore-Penrose pseudoinverse of an vector: a*pinv(a)=1
  - a./b % element-by-element division
  - a/b % equivalent to a*pinv(b)
  - norm(b) % norm

Matrix operations

- A+3 % a matrix plus a scalar
- A*3 % a matrix multiply a scalar
- A*a % a matrix multiplies a vector
- sin(A) % element-by-element sine of a matrix
- exp(A) % element-by-element exponential of a matrix
- A + B % matrix addition
- A*B % matrix multiplication
- A.*B % element-by-element multiplication.

Matrix operations (continued)

- A.^3 % element-by-element power
- A' % transpose or complex conjugate transpose of a matrix
- inv(A) % inverse of a matrix
- A./B % element-by-element division.
- A/B % equivalent to A*inv(B)
- A\B % backslash operator, returns inv(A)*B, with better performance.
- det(A) % determinant of a matrix
- isequal(A,B) % return 1 if A=B or 0 if otherwise.

- **Matrix indexing**

Index starts from 1 (not from 0).

Column-major convention.

- A(3,2) % the element of 3rd row and 2nd column
- A(:,1) % the 1st column
- A(2,2:3) % through 2nd to 3rd elements of 2nd row
- sum(A(2,:)) % sum all elements of the 2nd row
- max(A(3,:)) % maximum element of the 3rd row
- find(isprime(A)) % return the indexes of prime numbers among all elements

Exercise 1

- Vector and matrix operations

  i) Create length-3 vectors and 3 by 3 matrices.

  ii) Practice the vector operations listed above.

  iii) Practice the matrix operations listed above.

Cell array

- **Create a Cell Array**

```
cell(size1, size2, …)        % Create a multidimensional cell array
myCell = {2, [7 8 9], [1 2 3; 4 5 6]; 'text', rand(5,5), {11; 22; 33}}  % Initialize a 2*3 cell array
% The elements in a cell array can be of different types, for example, the element can be a number, a vector, a matrix, text, or itself can be a cell array too.
```

- Access Data in a Cell Array

```
myCell{1,3}  % The cell at the first row and the third column.
myCell{2,1:2}      % The first and second cells in the second row.
myCell{1,:}     % All cells in the first row.
```

```
iscell(myCell)        %  Determine whether input is cell array
```

Language basics

- **Variables**
  - No declaration
  - n = 25 % Integer
  - a = 6.2 % Real number
  - t = ‘hello’ % Text

- **Data type**
  - double(n) % Convert to double precision
  - single(n) % Convert to single precision
  - int8(a), int16(a), int32(a), int64(a) % Convert to 8-bit, 16-bit, 32-bit, 64-bit signed integer
  - eps % Floating-point relative accuracy for double precision
  - realmax % Largest Double-Precision Values

Complex number

- i % imaginary unit
- j % imaginary unit
- sqrt(-1) % imaginary unit
- x=3+4i % complex number
- x=complex(a,b) % real part is a, imaginary part is b
- complex(x) % convert to complex number
- real(x) % real part of x
- imag(x) % imaginary part of x
- angle(x) % argument of x
- abs(x) % amplitude of x
- conj(x) % conjugate of x
- isreal(x) % x is real or not

- Math expressions
  - a^3 + b^2 - 3*c + d/6 + 9
  - abs(x) % absolute value
  - sin(x); cos(x); tan(x); % triangle functions
  - asin(x); acos(x); atan(x); atan2(y,x); % inverse triangle functions
  - sqrt(x) % square root
  - exp(x) % exponential
  - log(x) % natural logarithm, inverse of exp(x)
  - log10(x) % base-10 logarithm, inverse of 10^x

- Long statement
  - s = 1 -1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ... % use … to combine two lines
    - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;

Programming I : Control Flow

- Condition — if, else

```
a = randi(100, 1);
if a < 30
    fprintf('%d is smaller than 30. \n', a);
elseif a > 80
    fprintf('%d is larger than 80. \n', a);
else
    X=[num2str(a), ' is between 30 and 80.'];
    disp(X)
end
```

- Condition — switch

```
[dayNum, dayString] = weekday(date, 'long', 'en_US');
switch dayString
    case 'Monday'
        disp('Start of the work week')
    case 'Tuesday'
        disp('Day 2')
    case 'Wednesday'
        disp('Day 3')
    case 'Thursday'
        disp('Day 4')
    case 'Friday'
        disp('Last day of the work week')
    otherwise
        disp('Weekend!')
end
```

- Loop — for

```
% sum of an array
    s=0;
    b=rand(100,1)
    for i = 1:1:100
       s=s+b(i);      % not allow +=
    end
    s
```

```
%  nested loop
    m=50;
    n=100;
    for i = 1:1:m      % Stripe 1
       for j = 1:2:n   % Stripe 2
          H(i , j) = 1/(i+j);
       end
    end
```

- Loop — while, break

```
% find a root of the cubic polynomial x^3 - 2x – 5 using Newton's method
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b - a > eps * b
    x = (a + b) / 2;
    fx = x^3 - 2*x - 5;
    if fx == 0
        break      % Already found the root, exit the loop
    elseif sign(fx) == sign(fa)      % This algorithm works only when the polynomial
        a = x; fa = fx;              % is increasing at the vicinity of the root.
    else
        b = x; fb = fx;
    end
end
x
```

Programming II : Functions

- Anonymous Functions

  f = @(arglist) expression

  - One argument

    ```
    my_fun = @(x)  x.^2+exp(x)+5;
    my_fun(5)
    ```

  - Two arguments

    ```
    my_fun = @(x,y)  x.^3+6*sqrt(y);
    my_fun(3,4)
    ```

- Functions

  function y = my_fun(x)

```
% The code for function (falling.m) . File name should be the same as function name.
function height = falling(t)
    global GRAVITY
    height = 1/2*GRAVITY*t.^2;      % The height of a freely falling object
end

% The code for main program (main.m).  Should be in the same path with the function code.
global GRAVITY
GRAVITY = 32;
y = falling((0:.2:5)')
```

Programming III : I/O with external files

- Input data from a file

```
% text format
load(filename,'-ascii')
% Matlab format
load(filename,'-mat')
```

- Output data to a file

```
% open a file
fileID = fopen(filename, permission)
% print data to the file
fprintf(fileID, formatSpec, A1,...,An)
```

- An example for I/O files

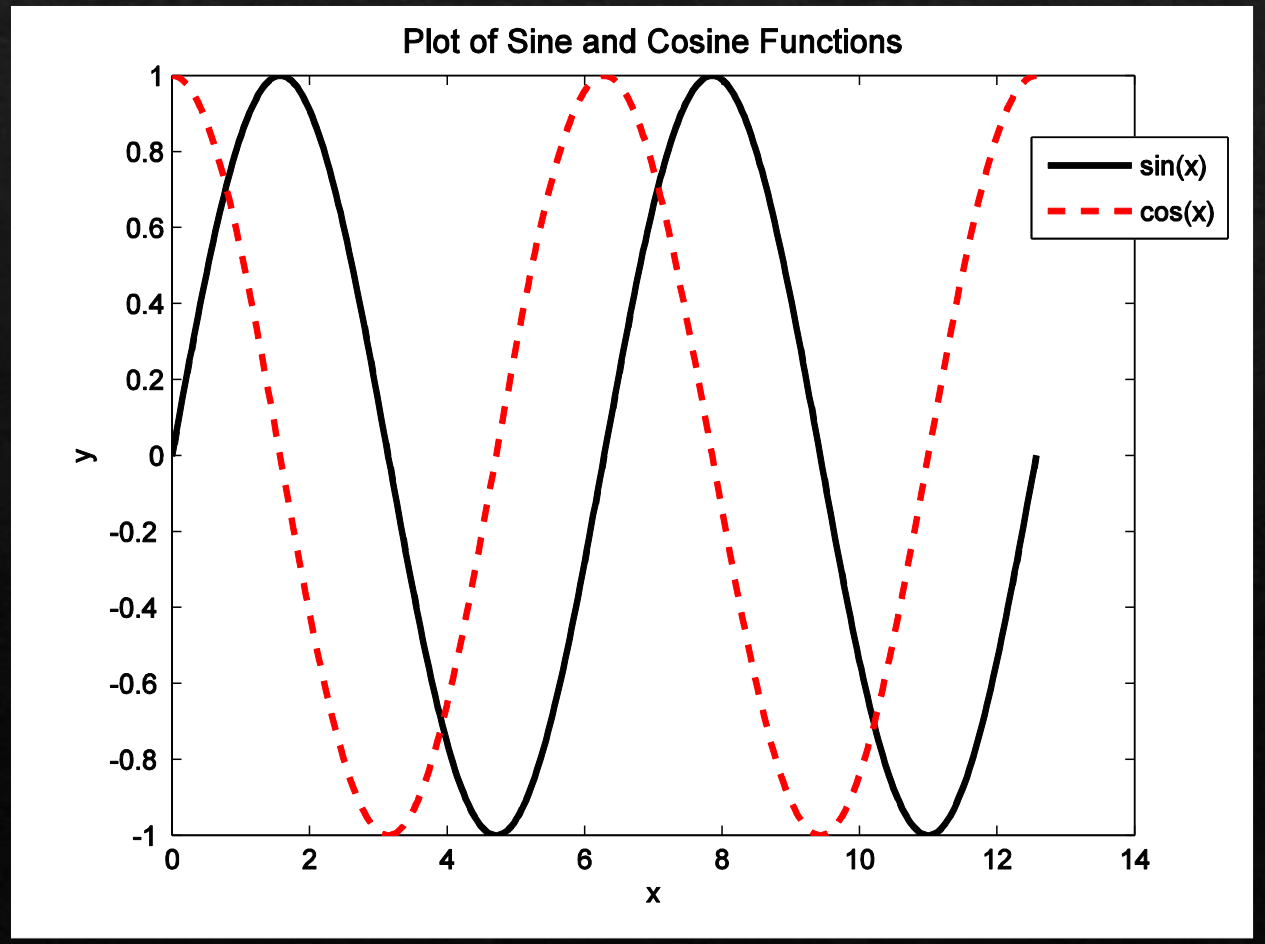
```
x = 0:.1:1;
A = [x; exp(x)];

fileID = fopen('exp.txt', 'w');     % open a writable file
fprintf(fileID, '%6.2f %12.8f\n', A);   % print real data
fclose(fileID);      % close the file

B=load('exp.txt', '-ascii')    % load data from the file
```

Plotting figures

- Two ways to plot figures: mouse-operation *vs.* scripting.

- **plot**

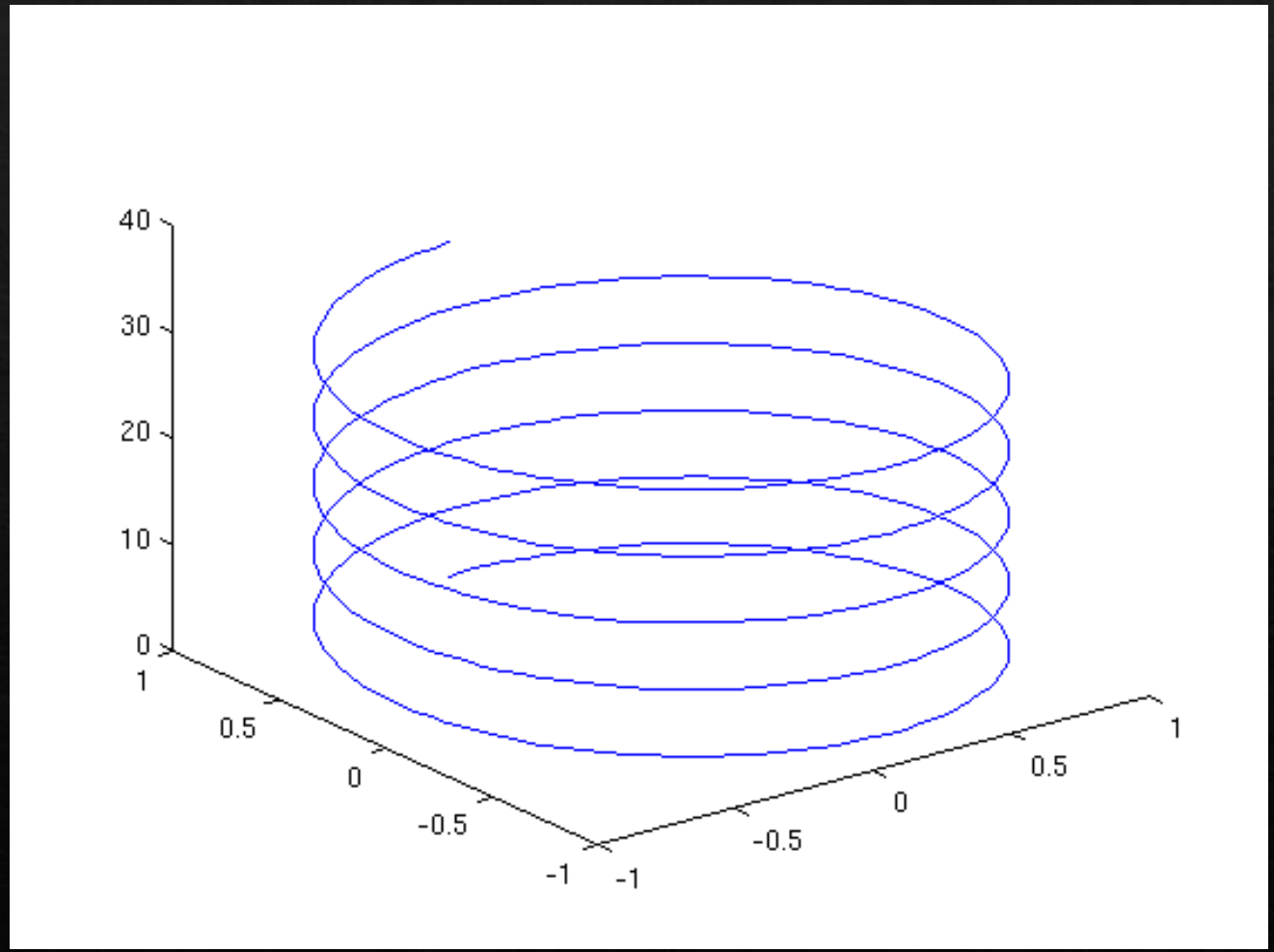
```
x = 0:pi/100:4*pi;
y = sin(x);
y2 = cos(x);
plot(x,y,'black',x,y2,'red--','linewidth',2)
xlabel('x')
ylabel('y')
axis([0  4*pi  -1  1])
title('Plot of Sine and Cosine Functions', 'FontSize', 12)
legend('sin(x)','cos(x)')
```



Plotting three-dimensional curves

- **plot3**

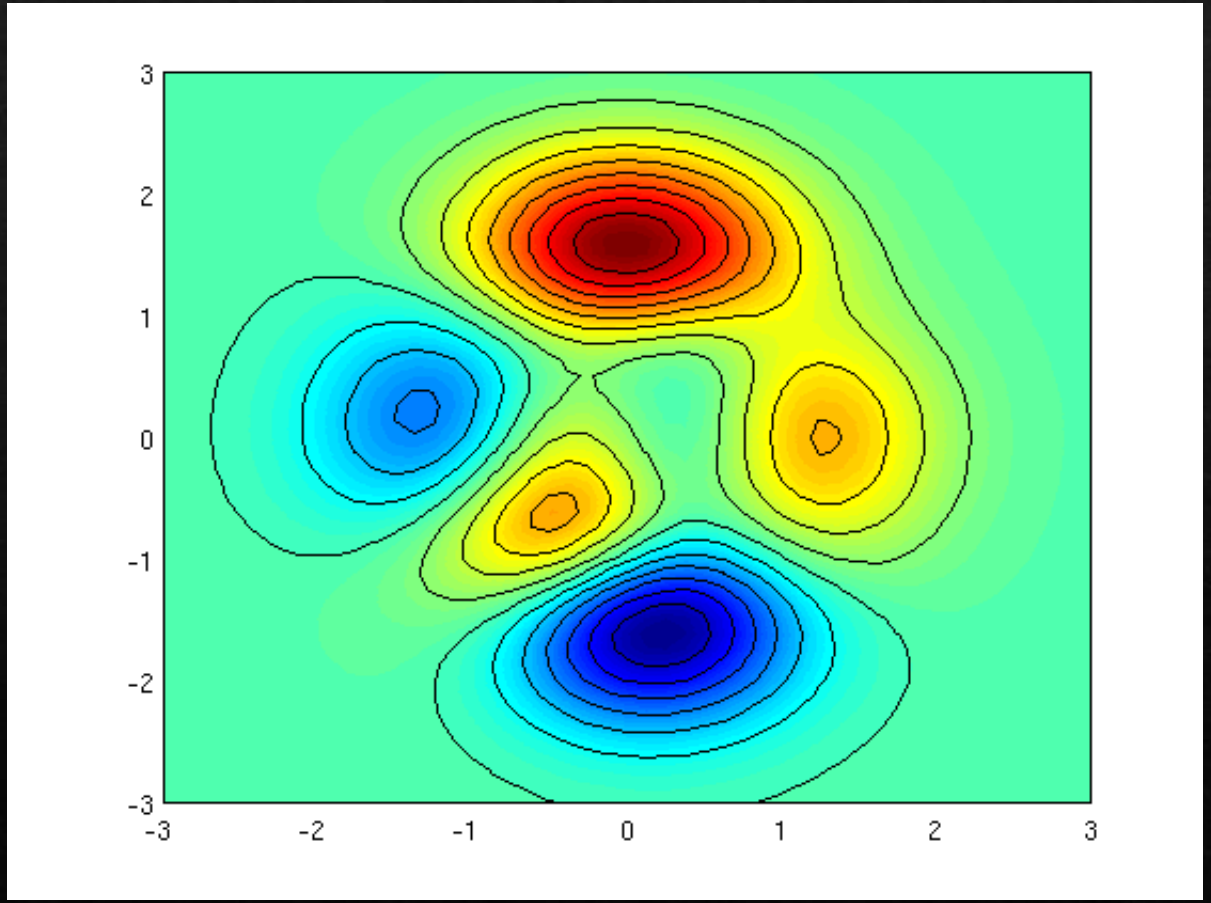
```
t = 0:pi/50:10*pi;  % z
st = sin(t);        % x
ct = cos(t);        % y

figure
plot3(st,ct,t)
```



Contour Plot

- **contour, pcolor**

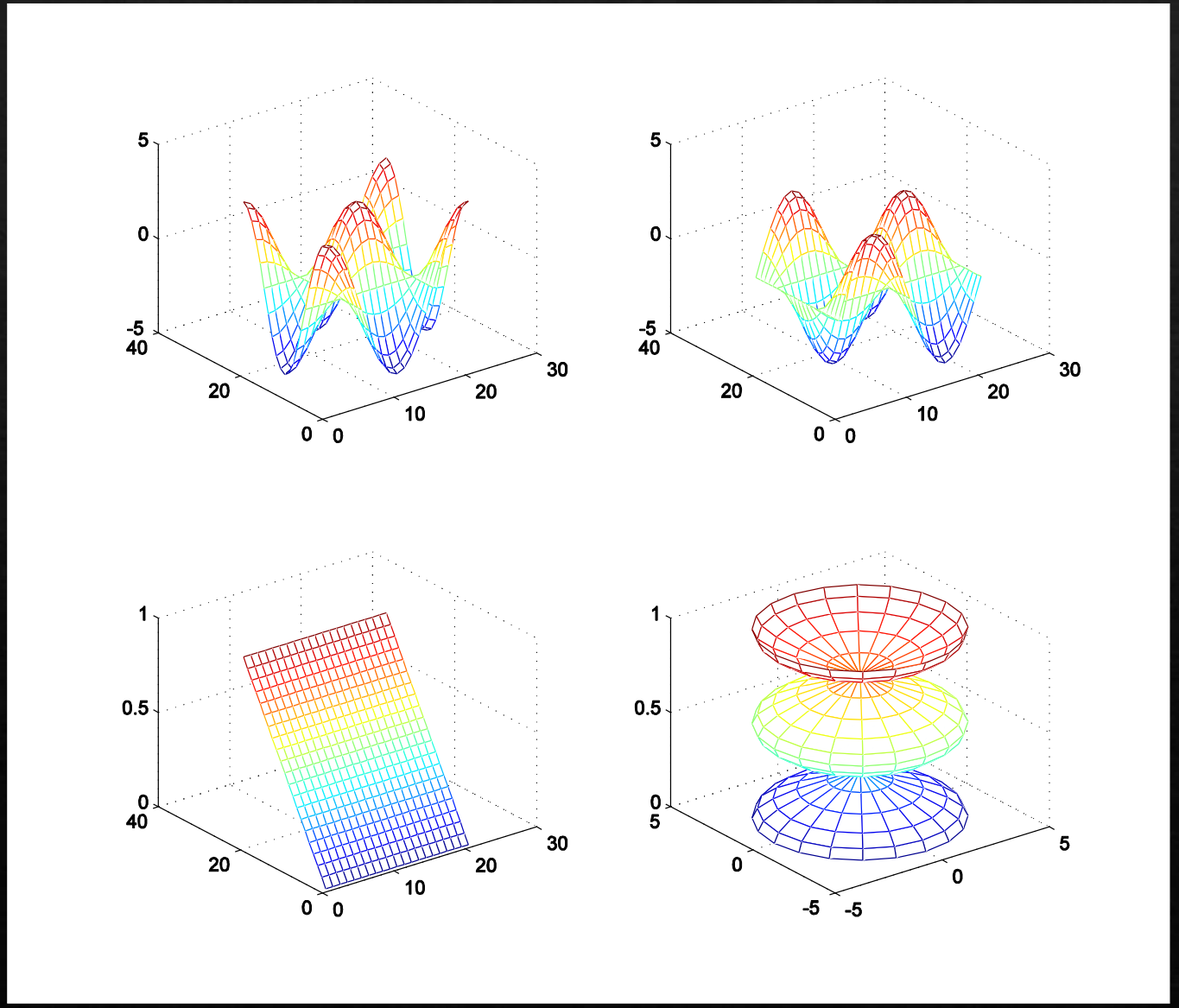
```
% Obtain data from evaluating peaks function
[x,y,z] = peaks;
% Create pseudocolor plot
pcolor(x,y,z)
% Smooth the colors
shading interp
% Hold the current graph
hold on
% Add the contour graph to the pcolor graph
contour(x,y,z,15,'k')    % 15-level, black line
% Return to default
hold off
```



Subfigures and layout

- **subplot, mesh**

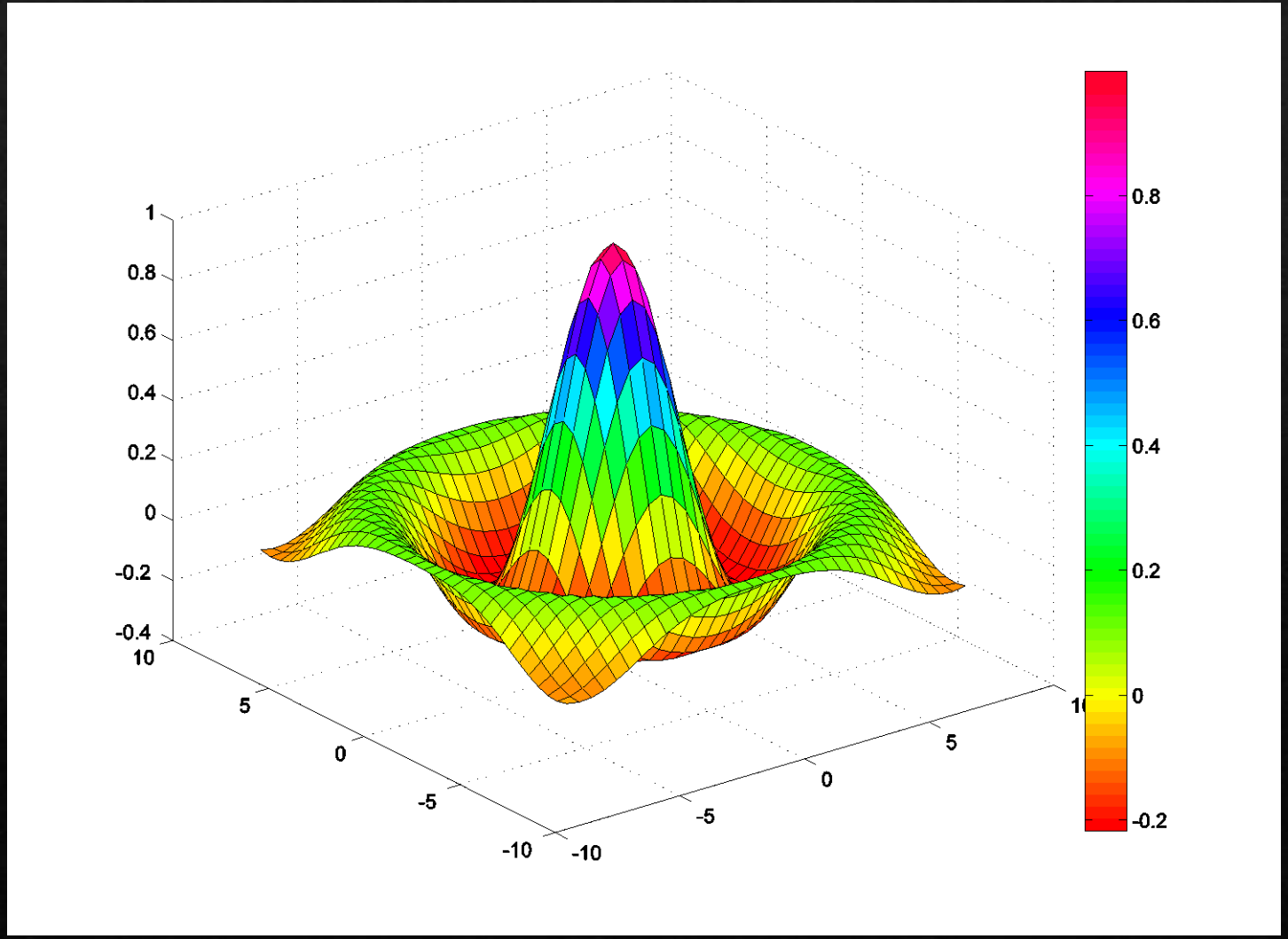
```
t = 0:pi/10:2*pi;
% cylinder with a self-defined profile
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1);   % left-up
mesh(X)
subplot(2,2,2);   % right-up
mesh(Y)
subplot(2,2,3);   % left-down
mesh(Z)
subplot(2,2,4);   % right-down
mesh(X,Y,Z)
```



Color Surface Plot

- **surfc**

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;   % sinc function
surfc(X,Y, Z)
colormap hsv     % color map
colorbar    % show color scaling
view([1 1 1])  % view angle
```



- colormap

- Color is represented by a vector

Red [1 0 0]

Green [0 1 0]

Blue [0 0 1]

Black [0 0 0]

White [1 1 1]

% define your own color

- View angle represented by a vector

From x axis: [1 0 0]

From y axis: [0 1 0]

From z axis: [0 0 1]

From diagonal line: [1 1 1]

Use Matlab to solve mathematical problems

- ✓ Regression analysis
- ✓ Numerical integration

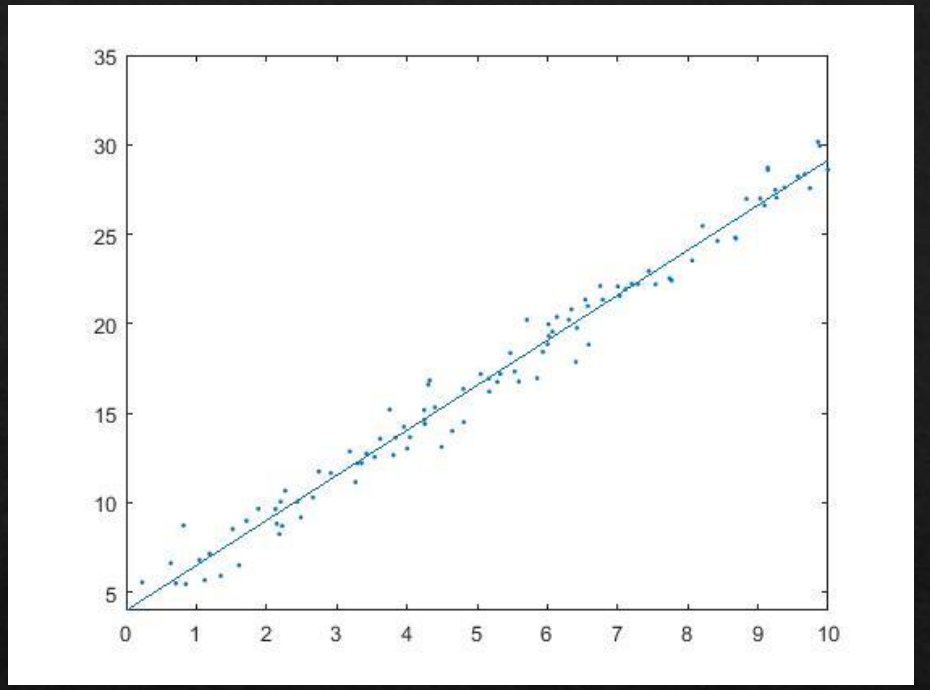
Regression analysis

- Given a set of data x and y, predict p coefficients $b_0, b_1, b_2, \ldots, b_p$, to best fit the data set with the *p-th* order polynomial $y = b_0 + b_1 * x + b_2 * x^2 \ldots + b_p * x^p$.
- A common method is the Least Squares fit.

```
b = regress(y, X)
% Return the predicted values of the coefficients b0 , b1, b2, … bp .
% y and X are input data.
% y is a length-n vector
% X is an n-by-p matrix, filled with [ones, x, x.^2, … , x.^p].

lsline                % Plot the predicted (least-squares) line.
```

Linear regression

- Given a set of raw data x and y, predict the slope b_1 and the intercept b_0 to best fit the data set with the linear equation $y = b_0 + b_1 * x$.




```
n=100;    %  Give data size
x = rand(n,1)*10;    % Create random data x between 0 and 10
beta0 = 4.;          % Give an intercept value to create data y
beta1 = 2.5;         % Give a slope value to create data y
noise = randn(n,1);   % Create normally distributed noise for data y
y = beta0 + beta1 * x + noise;     % Create raw data points that are in the vicinity of a line.
plot(x,y,'.')      % Plot the raw data
lsline             % Plot the predicted (least-squares) line.
X = [ones(size(x))  x];   % Prepare the input matrix for the regress function. Add ones to obtain b0.
b = regress(y, X)      % Return the predicted values of the intercept b0 and the slope b1.
```

Exercise 2

- Regression analysis

  i) Create random *x*-*y* data points that are in the vicinity of a quadratic curve.

  ii) Predict coefficients b_0, b_1, and b_2 to fit the data set with the quadratic polynomial $y = b_0 + b_1 * x + b_2 * x^2$. (Hint: use the *regress* function)

  iii) Plot the *x*-*y* data and the fitting curve. (Hint: use the *plot* function)

Numerical integration

- One-dimensional integration

q = integral(fun,xmin,xmax)

% approximates the integral of function from xmin to xmax using global adaptive quadrature and default error tolerances.

fun = @(x) exp(-x.^2).*log(x).^2; % define a function $f(x) = e^{-x^2}[\ln(x)]^2$

p = integral(fun,0,0.5) % proper integral

q = integral(fun,0, Inf) % improper integral

fun = @(x) log(x); % logarithm function

format long % output long digits

q = integral(fun,0,1) % integral with singularity at the lower limit

Exercise 3

- Plot a 3D curve and compute the length of the curve

  Consider the curve parameterized by the following equations:

  $x(t) = \sin(2t)$, $y(t) = \cos(t)$, $z(t) = t$,

  where $t \in [0, 3\pi]$.

  i) Create a three-dimensional plot of this curve. Hints: use the *plot3* function.

  ii) Compute the arc length of this curve. Hints: Use the following arc length formula:

$$\int_0^{3\pi} \sqrt{4\cos(2t)^2 + \sin(t)^2 + 1}\, dt.$$

References

- Matlab document center: http://www.mathworks.com/help/
- Matlab: Primer
  http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf
- Matlab: Programming Fundamentals
  http://www.mathworks.com/help/pdf_doc/matlab/matlab_prog.pdf
- Matlab: Mathematics
  http://www.mathworks.com/help/pdf_doc/matlab/math.pdf

- Matlab basics on BU Research Computing Services (RCS) web site:
  http://www.bu.edu/tech/support/research/software-and-programming/common-languages/matlab/

- RCS help: help@scv.bu.edu , shaohao@bu.edu