

Brief Introduction to Computing System and MATLAB

SAMSI Undergraduate Workshop

Weigang Zhong

May 19th. 2008

Outline

- Basics
 - Development tools and environment
 - Matrices and vectors
 - Operators

Outline

- Basics
 - Development tools and environment
 - Matrices and vectors
 - Operators
- Programming
 - Scripts/Functions
 - Programming syntax

Outline

- Basics
 - Development tools and environment
 - Matrices and vectors
 - Operators
- Programming
 - Scripts/Functions
 - Programming syntax
- Numerical tools
 - Vectors and matrices
 - ODEs solvers
 - Optimization
 - Statistics

Outline

- Basics
 - Development tools and environment
 - Matrices and vectors
 - Operators
- Programming
 - Scripts/Functions
 - Programming syntax
- Numerical tools
 - Vectors and matrices
 - ODEs solvers
 - Optimization
 - Statistics
- Save your work
 - Plotting results
 - Printing graphics
 - Reading/Writing data

Outline

- Basics
 - Development tools and environment
 - Matrices and vectors
 - Operators
- Programming
 - Scripts/Functions
 - Programming syntax
- Numerical tools
 - Vectors and matrices
 - ODEs solvers
 - Optimization
 - Statistics
- Save your work
 - Plotting results
 - Printing graphics
 - Reading/Writing data

Desktop Environment

- Command window
- Command history
- Current directory and workspace

Desktop Environment

- Command window
- Command history
- Current directory and workspace
- Editor/Debugger

Desktop Environment

- Command window
- Command history
- Current directory and workspace
- Editor/Debugger
- Help

Storing Matrices and Vectors

- Row vector

```
>> r=[1 2 3 4];
```

or

```
>> r=[1,2,3,4];
```

- Column vector

```
>> c=[1;2;3;4];
```

or

```
>> c=[1  
      2  
      3  
      4];
```

- Matrix

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

or

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

- To change one element

```
>> M(2,3) = 66;
```

To change multiple

```
>> M(2,1:2) = [11 12];
```

```
>> M(2,:) = [11 12 13];
```

Matrix Arithmetic Operations

- $+$, $-$, $*$, $^{\wedge}$ work as expected

Matrix Arithmetic Operations

- $+$, $-$, $*$, $^{\wedge}$ work as expected

– Examples:

```
>> [1 2 3 4] * [5  
6  
7  
8]
```

ans =

70

```
>> [1  
2  
3  
4] * [5 6 7 8]
```

ans =

```
5 6 7 8  
10 12 14 16  
15 18 21 24  
20 24 28 32
```

Matrix Arithmetic Operations, con't

- Division $/$ works as expected for scalars

Matrix Arithmetic Operations, con't

- Division `/` works as expected for scalars
- Matrix left division `\`

$$Ax = b \rightarrow x = A \setminus b \quad (x = \text{inv}(A) * b)$$

Matrix Arithmetic Operations, con't

- Division / works as expected for scalars
- Matrix left division \

$$Ax = b \rightarrow x = A \setminus b \quad (x = \text{inv}(A) * b)$$

- System doesn't have a solution \rightarrow least squares solution

Array Arithmetic Operations

- Dotted operators ($\cdot*$, $\cdot/$, $\cdot\backslash$, \cdot^{\wedge}) act element-by-element

Array Arithmetic Operations

- Dotted operators ($\cdot*$, $\cdot/$, $\cdot\backslash$, \cdot^{\wedge}) act element-by-element

– Example:

$$\text{Given } A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \text{ and } B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Array Arithmetic Operations

- Dotted operators ($\cdot*$, $\cdot/$, $\cdot\backslash$, \cdot^{\wedge}) act element-by-element

– Example:

$$\text{Given } A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \text{ and } B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
>> A*B
```

```
ans =
```

```
    7    10
```

```
   15    22
```

Array Arithmetic Operations

- Dotted operators ($\cdot*$, $\cdot/$, $\cdot\backslash$, \cdot^{\wedge}) act element-by-element

– Example:

$$\text{Given } A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \text{ and } B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
>> A*B
```

```
ans =  
    7    10  
   15    22
```

```
>> A.*B
```

```
ans =  
    1     4  
    9    16
```

Matrix Transpose

- Matrix/Vector transpose (')

Matrix Transpose

- Matrix/Vector transpose (')

– Examples: Given $x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ and $M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

Matrix Transpose

- Matrix/Vector transpose (')

– Examples: Given $x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ and $M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

```
>> x'
```

```
ans =
```

```
1 2 3 4
```

Matrix Transpose

- Matrix/Vector transpose (')

– Examples: Given $x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ and $M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

```
>> x'
```

```
ans =
```

```
1 2 3 4
```

```
>> M'
```

```
ans =
```

```
1 4
```

```
2 5
```

```
3 6
```

Colon Operator

- Colon operator (:)
 - creates regularly spaced vectors

```
>> j : k
```

is the same as $[j, j + 1, j + 2, \dots, k]$

Colon Operator

- Colon operator (:)

- creates regularly spaced vectors

- >> `j : k`

- is the same as `[j, j + 1, j + 2, ... , k]`

- >> `j : i : k`

- is the same as `[j, j + i, j + 2i, ... , k]`

Colon Operator

- Colon operator (:)

- creates regularly spaced vectors

```
>> j : k
```

is the same as $[j, j + 1, j + 2, \dots, k]$

```
>> j : i : k
```

is the same as $[j, j + i, j + 2i, \dots, k]$

- Picks out selected rows, columns, and elements

```
>> A(:,j)
```

returns the j^{th} column of A

Colon Operator

- Colon operator (:

- creates regularly spaced vectors

```
>> j : k
```

is the same as $[j, j + 1, j + 2, \dots, k]$

```
>> j : i : k
```

is the same as $[j, j + i, j + 2i, \dots, k]$

- Picks out selected rows, columns, and elements

```
>> A(:,j)
```

returns the j^{th} column of A

```
>> A(i,:)
```

returns the i^{th} row of A

Colon Operator

- Colon operator (:

- creates regularly spaced vectors

```
>> j : k
```

is the same as $[j, j + 1, j + 2, \dots, k]$

```
>> j : i : k
```

is the same as $[j, j + i, j + 2i, \dots, k]$

- Picks out selected rows, columns, and elements

```
>> A(:,j)
```

returns the j^{th} column of A

```
>> A(i,:)
```

returns the i^{th} row of A

```
>> A(:, :)
```

is the same as A

Outline

- Basics
 - Development tools and environment
 - Matrices and arrays
 - Operators
- Programming
 - Scripts/Functions
 - Programming syntax
- Numerical tools
 - Vectors and matrices
 - ODEs solvers
 - Optimization
 - Statistics
- Save your work
 - Plotting results
 - Printing graphics
 - Reading/Writing data

Scripts and Functions

- Scripts
 - Execute a sequence of commands in the file

Scripts and Functions

- Scripts
 - Execute a sequence of commands in the file
 - Do not accept input arguments or return output arguments

Scripts and Functions

- Scripts
 - Execute a sequence of commands in the file
 - Do not accept input arguments or return output arguments
 - All data remain in the workspace

Scripts and Functions

- Scripts
 - Execute a sequence of commands in the file
 - Do not accept input arguments or return output arguments
 - All data remain in the workspace
 - No special syntax required

Scripts and Functions

- Scripts

- Execute a sequence of commands in the file
- Do not accept input arguments or return output arguments
- All data remain in the workspace
- No special syntax required
- Example:

```
>> type script1.m  
% Example of MATLAB script file  
% Plot Sine function  
x = -pi:1e-2:pi;  
y = sin(x);  
plot(x,y);
```

Scripts and Functions, con't

- Functions
 - Accept input arguments and return output arguments

Scripts and Functions, con't

- Functions
 - Accept input arguments and return output arguments
 - Internal variables are local to the function

Scripts and Functions, con't

- Functions
 - Accept input arguments and return output arguments
 - Internal variables are local to the function
 - Names of the M-file and of the function should be the same

Scripts and Functions, con't

- Functions

- Accept input arguments and return output arguments
- Internal variables are local to the function
- Names of the M-file and of the function should be the same
- Example:

```
>> type function1.m
```

```
function fout = function1(a,b,c)
```

```
% Example of MATLAB function file
```

```
% Solves quadratic polynomial of the form  $ax^2+bx+c=0$ 
```

```
x1 = (-b+sqrt(b*b-4*a*c))/(2*a);
```

```
x2 = (-b-sqrt(b*b-4*a*c))/(2*a);
```

```
fout = [x1,x2];
```

Flow Control

- Conditional control (if, else, elseif)
Allows different actions based on the state of certain variables

Flow Control

- Conditional control (if, else, elseif)

Allows different actions based on the state of certain variables

– Example:

```
if j < -1
    x = 1;
elseif j > 1
    x = 2;
else
    x = 3;
end
```

Flow Control, con't

- Loop control (for)

Repeats a group of statements a fixed, predetermined number of times

Flow Control, con't

- Loop control (for)

Repeats a group of statements a fixed, predetermined number of times

– Example:

```
x = zeros(100,100);  
y = zeros(100,100);  
for i = 1:100  
    for j = 1:100  
        x(i,j) = cos(j)*sin(i);  
        y(i,j) = sin(j)*sin(i);  
    end  
end
```

Flow Control, con't

- Loop control (while)

Repeats a group of statements an indefinite number of times under control of a logical condition.

Flow Control, con't

- Loop control (while)

Repeats a group of statements an indefinite number of times under control of a logical condition.

– Example:

```
count = 0;
tol = 1e-10;
err = 100;
xold = 0;
while (err > tol) && (count <= 500)
    fofx = (xold^2+2*xold-1)*exp(xold);    %(x^2+2x-1)e^x
    fprimeofx = (xold^2+4*xold+1)*exp(xold);
    xnew = xold-fofx/fprimeofx;           % Newton step;
    err = xnew-xold
    count = count+1
    xold = xnew
end ...
```

Outline

- Basics
 - Development tools and environment
 - Matrices and arrays
 - Operators
- Programming
 - Scripts/Functions
 - Programming syntax
- Numerical tools
 - Vectors and matrices
 - ODEs solvers
 - Optimization
 - Statistics
- Save your work
 - Plotting results
 - Printing graphics
 - Reading/Writing data

MATLAB Functions for Linear Algebra

- Create common matrices
 - `zeros(m,n)`: $m \times n$ matrix of all zeros

MATLAB Functions for Linear Algebra

- Create common matrices
 - `zeros(m,n)`: $m \times n$ matrix of all zeros
 - `ones(m,n)`: $m \times n$ matrix of all ones

MATLAB Functions for Linear Algebra

- Create common matrices
 - `zeros(m,n)`: $m \times n$ matrix of all zeros
 - `ones(m,n)`: $m \times n$ matrix of all ones
 - `eye(n)`: $n \times n$ identity matrix

MATLAB Functions for Linear Algebra

- Create common matrices
 - `zeros(m,n)`: $m \times n$ matrix of all zeros
 - `ones(m,n)`: $m \times n$ matrix of all ones
 - `eye(n)`: $n \times n$ identity matrix
 - `rand(m,n)`: $m \times n$ matrix with uniformly distributed random values

MATLAB Functions for Linear Algebra

- Create common matrices
 - `zeros(m,n)`: $m \times n$ matrix of all zeros
 - `ones(m,n)`: $m \times n$ matrix of all ones
 - `eye(n)`: $n \times n$ identity matrix
 - `rand(m,n)`: $m \times n$ matrix with uniformly distributed random values
 - `randn(m,n)`: $m \times n$ matrix with normally distributed random values

MATLAB Functions for Linear Algebra

- Create common matrices
 - `zeros(m,n)`: $m \times n$ matrix of all zeros
 - `ones(m,n)`: $m \times n$ matrix of all ones
 - `eye(n)`: $n \times n$ identity matrix
 - `rand(m,n)`: $m \times n$ matrix with uniformly distributed random values
 - `randn(m,n)`: $m \times n$ matrix with normally distributed random values
 - `linspace(a,b,N)`: a row vector of N points linearly spaced between a and b .

MATLAB Functions for Linear Algebra

- Create common matrices
 - `zeros(m,n)`: $m \times n$ matrix of all zeros
 - `ones(m,n)`: $m \times n$ matrix of all ones
 - `eye(n)`: $n \times n$ identity matrix
 - `rand(m,n)`: $m \times n$ matrix with uniformly distributed random values
 - `randn(m,n)`: $m \times n$ matrix with normally distributed random values
 - `linspace(a,b,N)`: a row vector of N points linearly spaced between a and b .

Compared with “:” operator?

MATLAB Functions for Linear Algebra

- Create common matrices

- `zeros(m,n)`: $m \times n$ matrix of all zeros
- `ones(m,n)`: $m \times n$ matrix of all ones
- `eye(n)`: $n \times n$ identity matrix
- `rand(m,n)`: $m \times n$ matrix with uniformly distributed random values
- `randn(m,n)`: $m \times n$ matrix with normally distributed random values
- `linspace(a,b,N)`: a row vector of N points linearly spaced between a and b .

Compared with “:” operator?

- Determine size of array

- `length(v)`: length of a vector v
- `size(x)`: size of a matrix or vector x

MATLAB Functions for Linear Algebra, con't

- For vectors v
 - $\max(v)/\min(v)$: the largest/smallest element in v

MATLAB Functions for Linear Algebra, con't

- For vectors v
 - $\max(v)/\min(v)$: the largest/smallest element in v
 - $\text{sum}(v)$: sum of elements in v

MATLAB Functions for Linear Algebra, con't

- For vectors v
 - $\max(v)/\min(v)$: the largest/smallest element in v
 - $\text{sum}(v)$: sum of elements in v
 - $\text{norm}(v,p)$: L^p -norm of v

MATLAB Functions for Linear Algebra, con't

- For vectors v
 - `max(v)/min(v)`: the largest/smallest element in v
 - `sum(v)`: sum of elements in v
 - `norm(v,p)`: L^p -norm of v
 - `norm(v,inf)`: L^∞ -norm of v , the largest absolute value

MATLAB Functions for Linear Algebra, con't

- For vectors v
 - `max(v)/min(v)`: the largest/smallest element in v
 - `sum(v)`: sum of elements in v
 - `norm(v,p)`: L^p -norm of v
 - `norm(v,inf)`: L^∞ -norm of v , the largest absolute value
- For matrices M
 - `norm(M,2)`: the largest singular value of M

MATLAB Functions for Statistics

- `mean(x)` : Average/Mean value

MATLAB Functions for Statistics

- `mean(x)` : Average/Mean value
- `median(x)` : Median value

MATLAB Functions for Statistics

- `mean(x)` : Average/Mean value
- `median(x)` : Median value
- `std(x)` : Standard Deviation

MATLAB Functions for Statistics

- `mean(x)` : Average/Mean value
- `median(x)` : Median value
- `std(x)` : Standard Deviation
- `var(x)` : Variance

MATLAB Functions for Statistics

- `mean(x)` : Average/Mean value
- `median(x)` : Median value
- `std(x)` : Standard Deviation
- `var(x)` : Variance
- `cov(x)` : Covariance

MATLAB Functions for Statistics

- `mean(x)` : Average/Mean value
- `median(x)` : Median value
- `std(x)` : Standard Deviation
- `var(x)` : Variance
- `cov(x)` : Covariance
- Note: If you think MATLAB should have a certain function, it probably does. Use help

MATLAB ODE Solvers

- $[t, y] = \text{ode45}(@\text{fun}, [t_0, t_f], y_0, \text{options}, a_1, \dots, a_N);$
Solves initial value problems for ODE.

MATLAB ODE Solvers

- $[t, y] = \text{ode45}(@\text{fun}, [t_0, t_f], y_0, \text{options}, a_1, \dots, a_N);$
Solves initial value problems for ODE.
 - Input
 - * $@\text{fun}$: a function handle evaluates RHS of differential equations

MATLAB ODE Solvers

- $[t, y] = \text{ode45}(@\text{fun}, [t_0, t_f], y_0, \text{options}, a_1, \dots, a_N);$
Solves initial value problems for ODE.
 - Input
 - * $@\text{fun}$: a function handle evaluates RHS of differential equations
 - * $[t_0, t_f]$: time span

MATLAB ODE Solvers

- $[t, y] = \text{ode45}(@\text{fun}, [t_0, t_f], y_0, \text{options}, a_1, \dots, a_N);$
Solves initial value problems for ODE.

– Input

- * $@\text{fun}$: a function handle evaluates RHS of differential equations
- * $[t_0, t_f]$: time span
- * y_0 : initial condition

MATLAB ODE Solvers

- $[t, y] = \text{ode45}(@\text{fun}, [t_0, t_f], y_0, \text{options}, a_1, \dots, a_N);$
Solves initial value problems for ODE.
 - Input
 - * $@\text{fun}$: a function handle evaluates RHS of differential equations
 - * $[t_0, t_f]$: time span
 - * y_0 : initial condition
 - * options : optional parameters change the integration properties

MATLAB ODE Solvers

- $[t, y] = \text{ode45}(@\text{fun}, [t_0, t_f], y_0, \text{options}, a_1, \dots, a_N);$
Solves initial value problems for ODE.
 - Input
 - * `@fun` : a function handle evaluates RHS of differential equations
 - * `[t0, tf]` : time span
 - * `y0` : initial condition
 - * `options` : optional parameters change the integration properties
 - created by `odeset`

MATLAB ODE Solvers

- $[t, y] = \text{ode45}(@\text{fun}, [t_0, t_f], y_0, \text{options}, a_1, \dots, a_N);$
Solves initial value problems for ODE.

– Input

- * $@\text{fun}$: a function handle evaluates RHS of differential equations
- * $[t_0, t_f]$: time span
- * y_0 : initial condition
- * options : optional parameters change the integration properties
 - created by `odeset`
 - changes properties of error control, output, step-size, Jacobian ,...

MATLAB ODE Solvers

- $[t, y] = \text{ode45}(@\text{fun}, [t_0, t_f], y_0, \text{options}, a_1, \dots, a_N);$
Solves initial value problems for ODE.

– Input

- * $@\text{fun}$: a function handle evaluates RHS of differential equations
- * $[t_0, t_f]$: time span
- * y_0 : initial condition
- * options : optional parameters change the integration properties
 - created by `odeset`
 - changes properties of error control, output, step-size, Jacobian ,...
- * a_1, \dots, a_N : arguments to be passed to your function

MATLAB ODE Solvers

- $[t, y] = \text{ode45}(@\text{fun}, [t_0, t_f], y_0, \text{options}, a_1, \dots, a_N);$
Solves initial value problems for ODE.

– Input

- * $@\text{fun}$: a function handle evaluates RHS of differential equations
- * $[t_0, t_f]$: time span
- * y_0 : initial condition
- * options : optional parameters change the integration properties
 - created by `odeset`
 - changes properties of error control, output, step-size, Jacobian ,...
- * a_1, \dots, a_N : arguments to be passed to your function

– Output

- * t : times at which solutions were calculated
- * y : solutions at given times

MATLAB ODE Solvers, con't

- Solves ODEs in the form $y' = fun(t, y(t))$
@fun is the function handle used in ODE45.

MATLAB ODE Solvers, con't

- Solves ODEs in the form $y' = fun(t, y(t))$
@fun is the function handle used in ODE45.

– Example: We try to solve the ODE

$$t^2 y' = \sin(at).$$

MATLAB ODE Solvers, con't

- Solves ODEs in the form $y' = fun(t, y(t))$
@fun is the function handle used in ODE45.

– Example: We try to solve the ODE

$$t^2 y' = \sin(at).$$

```
>> type fun.m
```

```
function dy = fun(t,y,a)
```

```
% RHS function of ODE
```

```
dy = sin(a*t)/t^2;
```

MATLAB ODE Solvers, con't

- Solves ODEs in the form $y' = fun(t, y(t))$
@fun is the function handle used in ODE45.

– Example: We try to solve the ODE

$$t^2 y' = \sin(at).$$

```
>> type fun.m
```

```
function dy = fun(t,y,a)
```

```
% RHS function of ODE
```

```
dy = sin(a*t)/t^2;
```

```
>> ode45(@fun,[t0,tf],y0,options,a)
```

MATLAB ODE Solvers, con't

- ODE of 2^{nd} order or higher

MATLAB ODE Solvers, con't

- ODE of 2^{nd} order or higher

– Example:

$$\begin{cases} m\ddot{y}(t) + c\dot{y} + ky(t) = \cos(t). \\ y(0) = 0 \\ \dot{y}(0) = 0 \end{cases}$$

MATLAB ODE Solvers, con't

- ODE of 2^{nd} order or higher

– Example:

$$\begin{cases} m\ddot{y}(t) + c\dot{y} + ky(t) = \cos(t). \\ y(0) = 0 \\ \dot{y}(0) = 0 \end{cases}$$

Let $y_1 = y$ and $y_2 = \dot{y}_1$, then $\dot{y}_2 = \ddot{y}_1 = \ddot{y}$

MATLAB ODE Solvers, con't

- ODE of 2nd order or higher

– Example:

$$\begin{cases} m\ddot{y}(t) + c\dot{y} + ky(t) = \cos(t). \\ y(0) = 0 \\ \dot{y}(0) = 0 \end{cases}$$

Let $y_1 = y$ and $y_2 = \dot{y}_1$, then $\dot{y}_2 = \ddot{y}_1 = \ddot{y}$

$$\begin{bmatrix} \dot{y}_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \cos(t) \end{bmatrix}.$$

MATLAB ODE Solvers, con't

- ODE of 2^{nd} order or higher

- Example:
$$\begin{cases} m\ddot{y}(t) + c\dot{y} + ky(t) = \cos(t). \\ y(0) = 0 \\ \dot{y}(0) = 0 \end{cases}$$

Let $y_1 = y$ and $y_2 = \dot{y}_1$, then $\dot{y}_2 = \ddot{y}_1 = \ddot{y}$

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \cos(t) \end{bmatrix}.$$

- RHS function defining the ODE reads

```
function dy = fun(t,y,m,c,k)
```

```
A = [0 1; -k/m -c/m];
```

```
b = [0; cos(t)/m];
```

```
dy = A*y+b;
```

MATLAB Functions for Optimization

- `[x, fval] = fminsearch(@fminfun, x0, options)`
Finds a **local** minimum of a scalar function

MATLAB Functions for Optimization

- `[x, fval] = fminsearch(@fminfun, x0, options)`
Finds a **local** minimum of a scalar function
 - multidimensional unconstrained nonlinear optimization

MATLAB Functions for Optimization

- `[x, fval] = fminsearch(@fminfun, x0, options)`
Finds a **local** minimum of a scalar function
 - multidimensional unconstrained nonlinear optimization
 - `@fminfun` : function handle for a **scalar** function
 - `x0` : initial guess, can be a vector
 - `options` : change the optimization parameters: max number of iterations or function evaluation, termination tolerance ...

MATLAB Functions for Optimization

- `[x, fval] = fminsearch(@fminfun, x0, options)`
Finds a **local** minimum of a scalar function
 - multidimensional unconstrained nonlinear optimization
 - `@fminfun` : function handle for a **scalar** function
 - `x0` : initial guess, can be a vector
 - `options` : change the optimization parameters: max number of iterations or function evaluation, termination tolerance ...
 - `x` : local minimum.
 - `fval` : value of the objective function at `x`

MATLAB Functions for Optimization

- $[x, fval] = \text{fminsearch}(@\text{fminfun}, x0, \text{options})$
Finds a **local** minimum of a scalar function
 - multidimensional unconstrained nonlinear optimization
 - $@\text{fminfun}$: function handle for a **scalar** function
 - $x0$: initial guess, can be a vector
 - options : change the optimization parameters: max number of iterations or function evaluation, termination tolerance ...
 - x : local minimum.
 - $fval$: value of the objective function at x
- Note: A minimum will be reached, but most likely not be global.

Outline

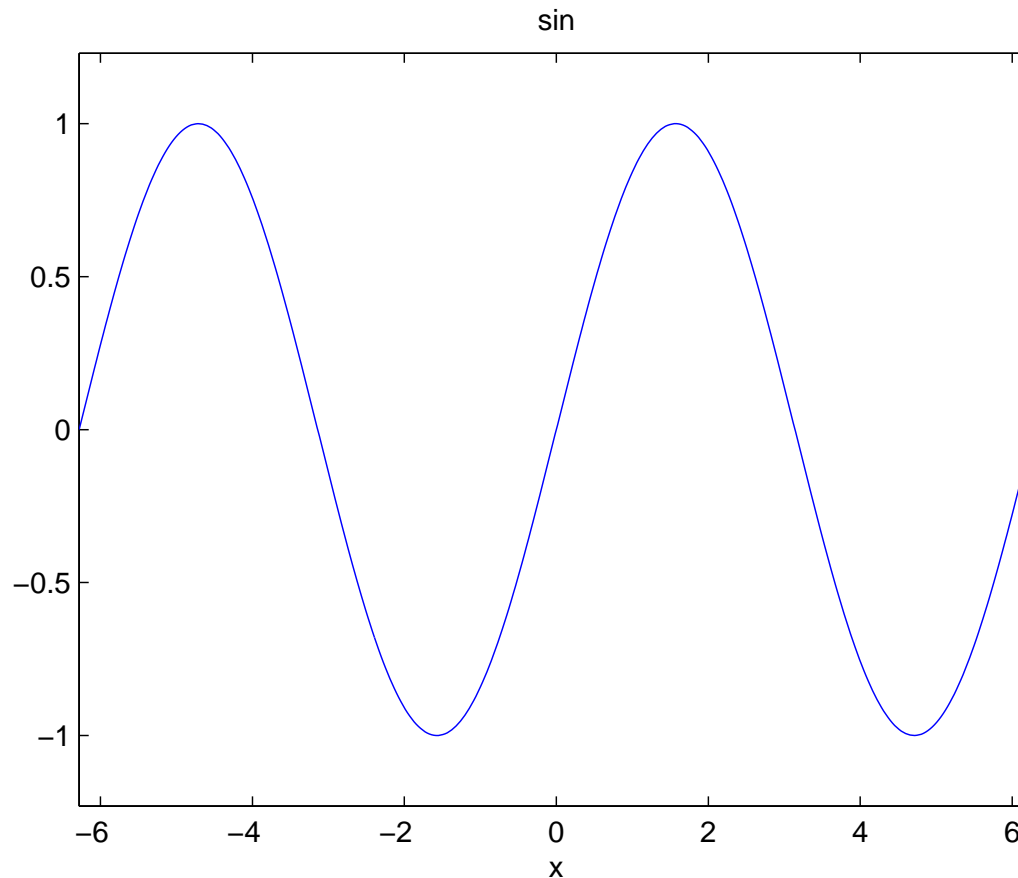
- Basics
 - Development tools and environment
 - Matrices and arrays
 - Operators
- Programming
 - Scripts/Functions
 - Programming syntax
- Numerical tools
 - Vectors and matrices
 - ODEs solvers
 - Optimization
 - Statistics
- Save your work
 - Plotting results
 - Printing graphics
 - Reading/Writing data

Displaying Results

- `ezplot('fun')`, `ezplot(@fun)`
 - creates a plot for $y=\text{fun}(x)$ over domain $[-2\pi, 2\pi]$

Displaying Results

- `ezplot('fun')`, `ezplot(@fun)`
 - creates a plot for $y=\text{fun}(x)$ over domain $[-2\pi, 2\pi]$
- Example: `>> ezplot('sin');`



Displaying Results, con't

- `plot(X1,Y1,X2,Y2,...XN,YN)`
 - Combined plots of Y_i vs X_i , $i=1,2,\dots,N$

Displaying Results, con't

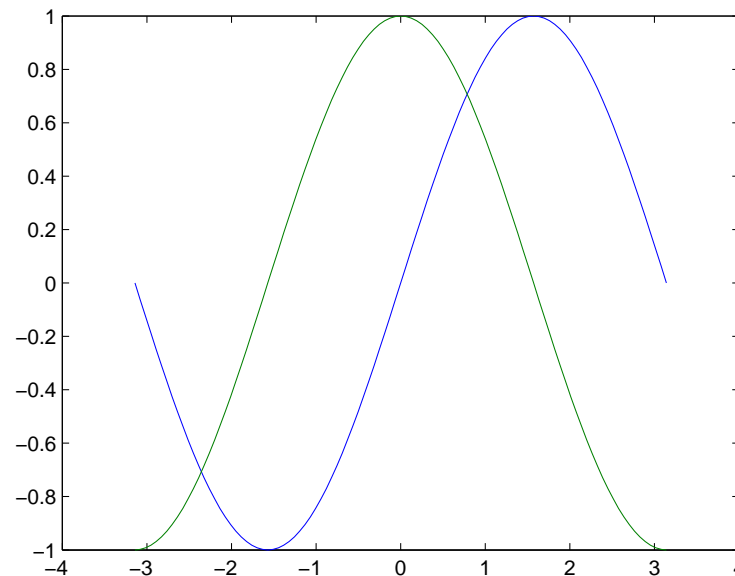
- `plot(X1,Y1,X2,Y2,...XN,YN)`

- Combined plots of Y_i vs X_i , $i=1,2,\dots,N$

- Example:

```
>> x = linspace(-pi,pi,101); % a row vector of 101 points
```

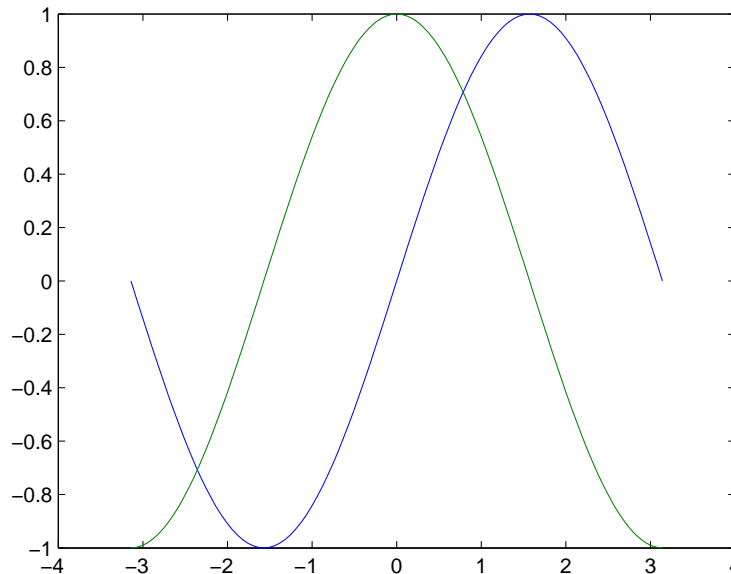
```
>> plot(x,sin(x),x,cos(x));
```



Displaying Results, con't

- `plot(X1,Y1,X2,Y2,...XN,YN)`
 - Combined plots of Y_i vs X_i , $i=1,2,\dots,N$
 - Example:

```
>> x = linspace(-pi,pi,101); % a row vector of 101 points
>> plot(x,sin(x),x,cos(x));
```



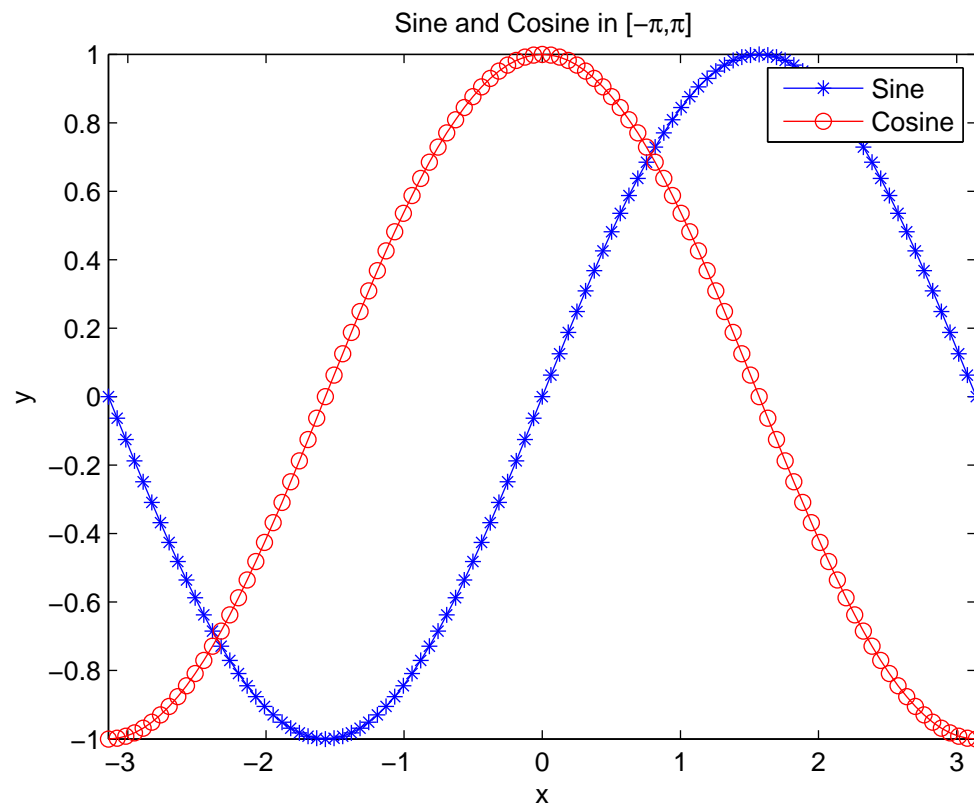
- More fancy line types, plot symbols, and colors ...

Displaying Results, con't

- ```
>> x = linspace(-pi,pi,101);
>> plot(x,sin(x),'-*',x,cos(x),'r-o');
>> xlabel('x'); ylabel('y');
>> title('Sine and Cosine in [-\pi,\pi]');
>> legend('Sine','Cosine');
>> axis([-pi pi -1 1]);
```

# Displaying Results, con't

- ```
>> x = linspace(-pi,pi,101);  
>> plot(x,sin(x),'-*',x,cos(x),'r-o');  
>> xlabel('x'); ylabel('y');  
>> title('Sine and Cosine in [-\pi,\pi]');  
>> legend('Sine','Cosine');  
>> axis([-pi pi -1 1]);
```



Uploading Data

- Given a data file "data.in" looks like

1 10

1 20

3 30

4 40

5 50

Uploading Data

- Given a data file "data.in" looks like

1 10

1 20

3 30

4 40

5 50

- Load this data file into MATLAB with

```
>> load data.in
```

Uploading Data

- Given a data file "data.in" looks like

1 10

1 20

3 30

4 40

5 50

- Load this data file into MATLAB with

```
>> load data.in
```

- Data will be stored in a 5×2 matrix named "data"

Uploading Data

- Given a data file "data.in" looks like

1 10

1 20

3 30

4 40

5 50

- Load this data file into MATLAB with

```
>> load data.in
```

- Data will be stored in a 5×2 matrix named "data"

- More delicate control over input file by file identifier

```
>> fid = fopen('data.in','a+');
```

Saving and Clearing Data

- Save current workspace to mat-file
`>> save mystuff;`

Saving and Clearing Data

- Save current workspace to mat-file

```
>> save mystuff;
```

- All variables currently in memory are saved to "mystuff.mat"
- mat-file can be uploaded by the command load

- Clear variables and functions from memory

```
>> clear; % clears all variables from the workspace
```

Saving and Clearing Data

- Save current workspace to mat-file

```
>> save mystuff;
```

- All variables currently in memory are saved to "mystuff.mat"
- mat-file can be uploaded by the command load

- Clear variables and functions from memory

```
>> clear; % clears all variables from the workspace
```

```
>> clear GLOBAL; % removes all global variables
```

Saving and Clearing Data

- Save current workspace to mat-file

```
>> save mystuff;
```

- All variables currently in memory are saved to "mystuff.mat"
- mat-file can be uploaded by the command load

- Clear variables and functions from memory

```
>> clear; % clears all variables from the workspace
```

```
>> clear GLOBAL; % removes all global variables
```

```
>> clear a*; % clears variables starting with "a"
```

Saving and Clearing Data

- Save current workspace to mat-file

```
>> save mystuff;
```

- All variables currently in memory are saved to "mystuff.mat"
- mat-file can be uploaded by the command load

- Clear variables and functions from memory

```
>> clear; % clears all variables from the workspace
```

```
>> clear GLOBAL; % removes all global variables
```

```
>> clear a*; % clears variables starting with "a"
```

```
>> clear all; % removes EVERYTHING !!
```

Saving and Clearing Data

- Save current workspace to mat-file

```
>> save mystuff;
```

- All variables currently in memory are saved to "mystuff.mat"
- mat-file can be uploaded by the command load

- Clear variables and functions from memory

```
>> clear; % clears all variables from the workspace
```

```
>> clear GLOBAL; % removes all global variables
```

```
>> clear a*; % clears variables starting with "a"
```

```
>> clear all; % removes EVERYTHING !!
```

- Close figure

```
>> close; % closes the current figure window
```

Saving and Clearing Data

- Save current workspace to mat-file

```
>> save mystuff;
```

- All variables currently in memory are saved to "mystuff.mat"
- mat-file can be uploaded by the command load

- Clear variables and functions from memory

```
>> clear; % clears all variables from the workspace
```

```
>> clear GLOBAL; % removes all global variables
```

```
>> clear a*; % clears variables starting with "a"
```

```
>> clear all; % removes EVERYTHING !!
```

- Close figure

```
>> close; % closes the current figure window
```

```
>> close all; % closes all the open figure windows
```

Getting Help

- help function

Getting Help

- help function

- Get more information about a specific function

Example:

```
>> help plot;
```

Getting Help

- help function
 - Get more information about a specific function
- Example:
- ```
>> help plot;
```
- Full documentation
    - Click "Help" menu

The End

Questions?