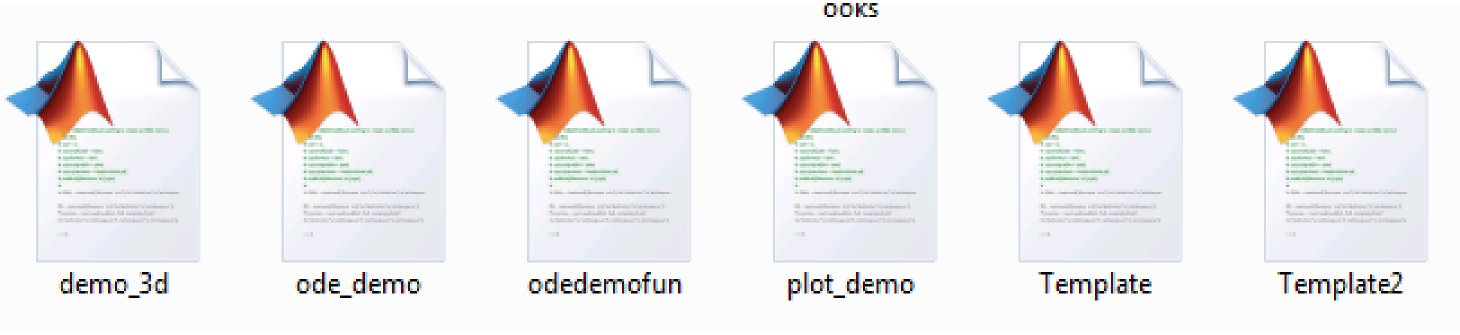
MATLAB Physics - I

- MATLAB and Symbolic Math should be installed on an accessible computer – you will execute the scripts.
- There is a textbook available
  - The book is supplied by UPAS
  - There is a CD with the .m file scripts for the demonstrations used in the course
- Lecture/class notes – email list for communication of the class
- Distribute all scripts by data stick – 200 examples
- Homework and “project”

Programming in MATLAB

- **Templates and demos available - local**

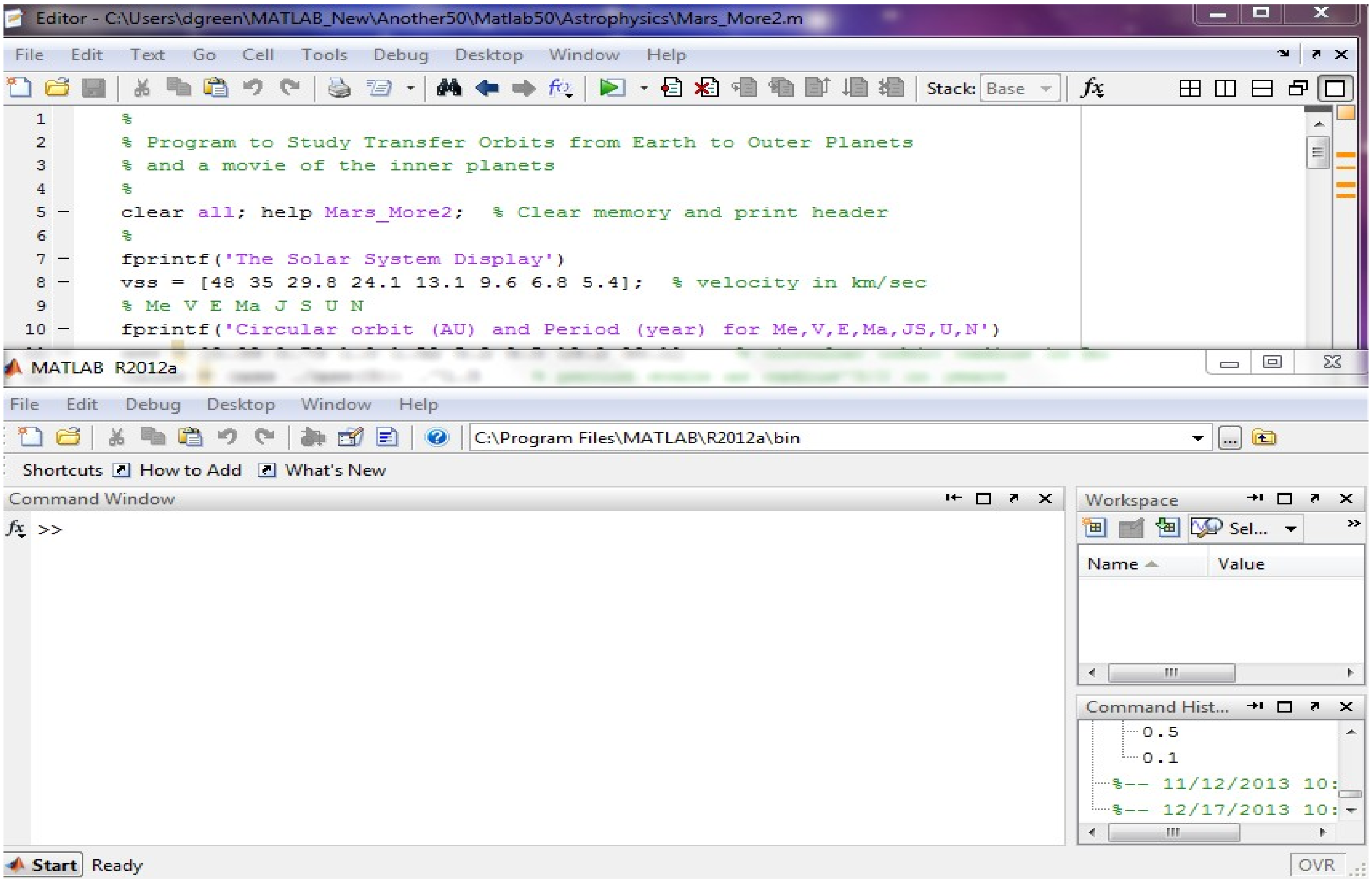


GUI_DEMO	5/16/2014 4:13 PM	MATLAB Figure	6 KB
GUI_DEMO	5/19/2014 9:41 AM	MATLAB Code	6 KB

Matlab Windows

- Workspace, command history, command window and editor – desktop layout
  - Open .m files, editor color coded
  - Debug options
  - In command window – set path to scripts (.m files)
- Setup toolbars, status busy/ready. Help
- Use product help and function browser
- Use demos – e.g. getting started, quick start

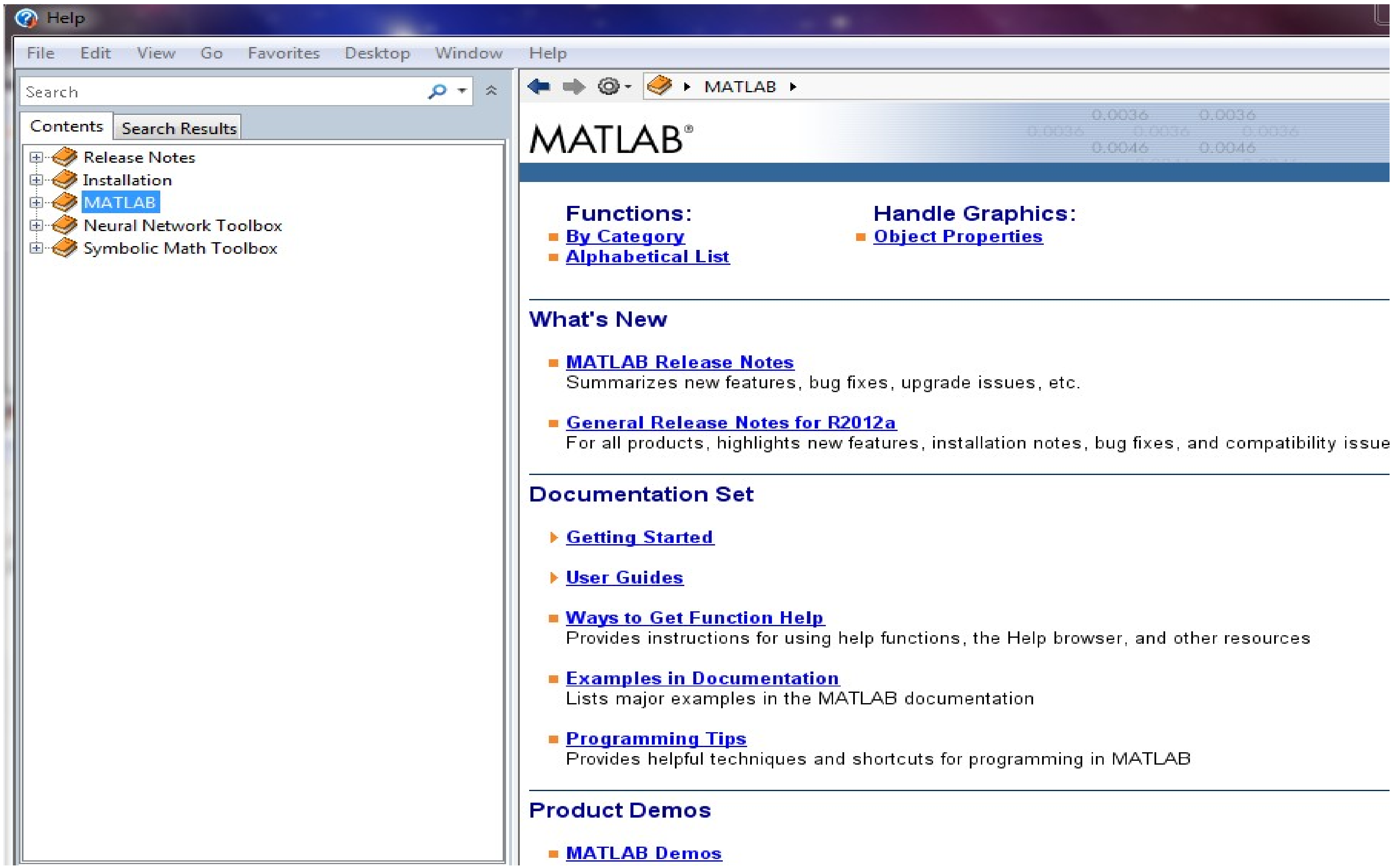
Desktop Layout



MATLAB Startup

- **Getting Started**
  - **Functions**
  - **Examples**
  - **Demos**
- **Quick start**
  - **Language**
  - **Math**
  - **Graphics**
  - **Programming**

Help Page – Top of Tree



Demos and Getting Started

Drill Down in HELP - Examples

Help

File Edit View Go Favorites Desktop Window Help

dsolve

Contents | Search Results

Type	Relevance	Product

dsolve - Ordinary differential equation and syste...
S = **dsolve**(eqn) solves the ordinary ... For example, syms y(x); **dsolve**(diff(y) == y + 1) and
Symbolic Math Toolbox

dsolve Accepts the New Option IgnoreAnalytic...
The **dsolve** command now accepts the option ... The results of the **dsolve** command can differ
Symbolic Math Toolbox

dsolve, expand, int, simple, simplify, and solve A...
dsolve now accepts the
Symbolic Math Toolbox

Solution of Equations
dsolve
Symbolic Math Toolbox

Single Differential Equation
Use **dsolve** to compute symbolic solutions to ... Before using **dsolve**, create the symbolic function
Symbolic Math Toolbox

Solving Algebraic and Differential Equations
differential equations using the **dsolve** function ... **dsolve**(D2y + Dy + y == 0) ... ysol = **dsolve**(D2y
Symbolic Math Toolbox

Several Differential Equations
dsolve can handle several ordinary ... Now use **dsolve** to solve the system ... S = **dsolve**(diff(f) ==
Symbolic Math Toolbox

Computations with Symbolic Functions
dsolve, ezplot, the new odeToVectorField

Search Online Support for dsolve

Symbolic Math Toolbox ▸ Functions ▸ Solution of Equations ▸ dsolve

dsolve

Ordinary differential equation and system solver

Syntax

```
S = dsolve(eqn)
S = dsolve(eqn,cond)
S = dsolve(eqn,cond,Name,Value)
Y = dsolve(eqn1,...,eqnN)
Y = dsolve(eqn1,...,eqnN,cond1,...,condN)
Y = dsolve(eqn1,...,eqnN,cond1,...,condN,Name,Value)
[y1,...,yN] = dsolve(eqn1,...,eqnN)
[y1,...,yN] = dsolve(eqn1,...,eqnN,cond1,...,condN)
[y1,...,yN] = dsolve(eqn1,...,eqnN,cond1,...,condN,Name,Value)
```

Description

`S = dsolve(eqn)` solves the ordinary differential equation `eqn`. Here `eqn` is a symbolic equation containing `diff` to indicate derivatives. Alternatively, you can use a string with the letter `D` indicating derivatives. For example, `syms y(x); dsolve(diff(y) == y + 1)` and `dsolve('Dy = y + 1','x')` both solve the equation `dy/dx = y + 1` with respect to the variable `x`. Also, `eqn` can be an array of such equations or strings.

`S = dsolve(eqn,cond)` solves the ordinary differential equation `eqn` with the initial or boundary condition `cond`.

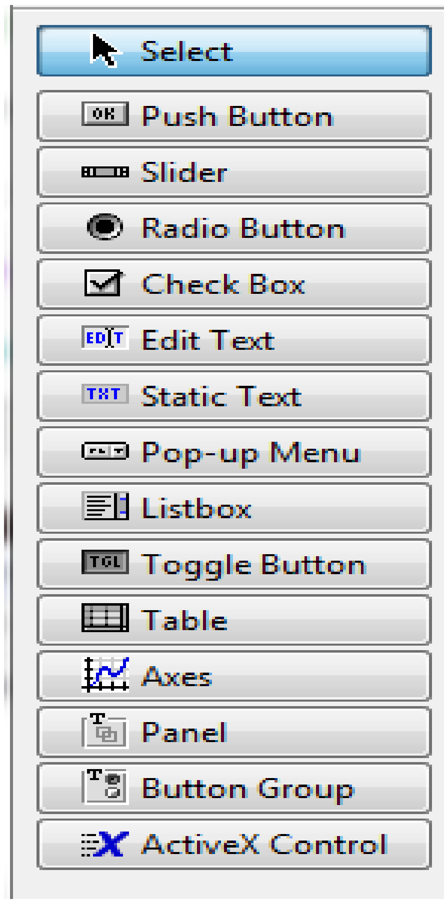
`S = dsolve(eqn,cond,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

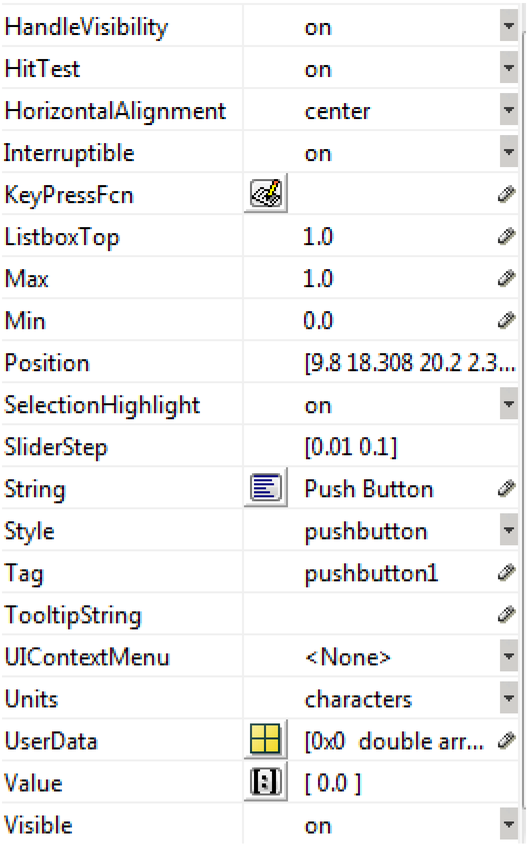
`Y = dsolve(eqn1,...,eqnN)` solves the system of ordinary differential equations `eqn1,...,eqnN` and returns a structure array that contains the solutions. The number of fields in the structure array corresponds to the number of independent variables in the system.

`Y = dsolve(eqn1,...,eqnN,cond1,...,condN)` solves the system of ordinary differential equations `eqn1,...,eqnN` with the initial or boundary conditions `cond1,...,condN`.

`Y = dsolve(eqn1,...,eqnN,cond1,...,condN,Name,Value)` uses additional options specified

- **Demo exists – many tools supplied**

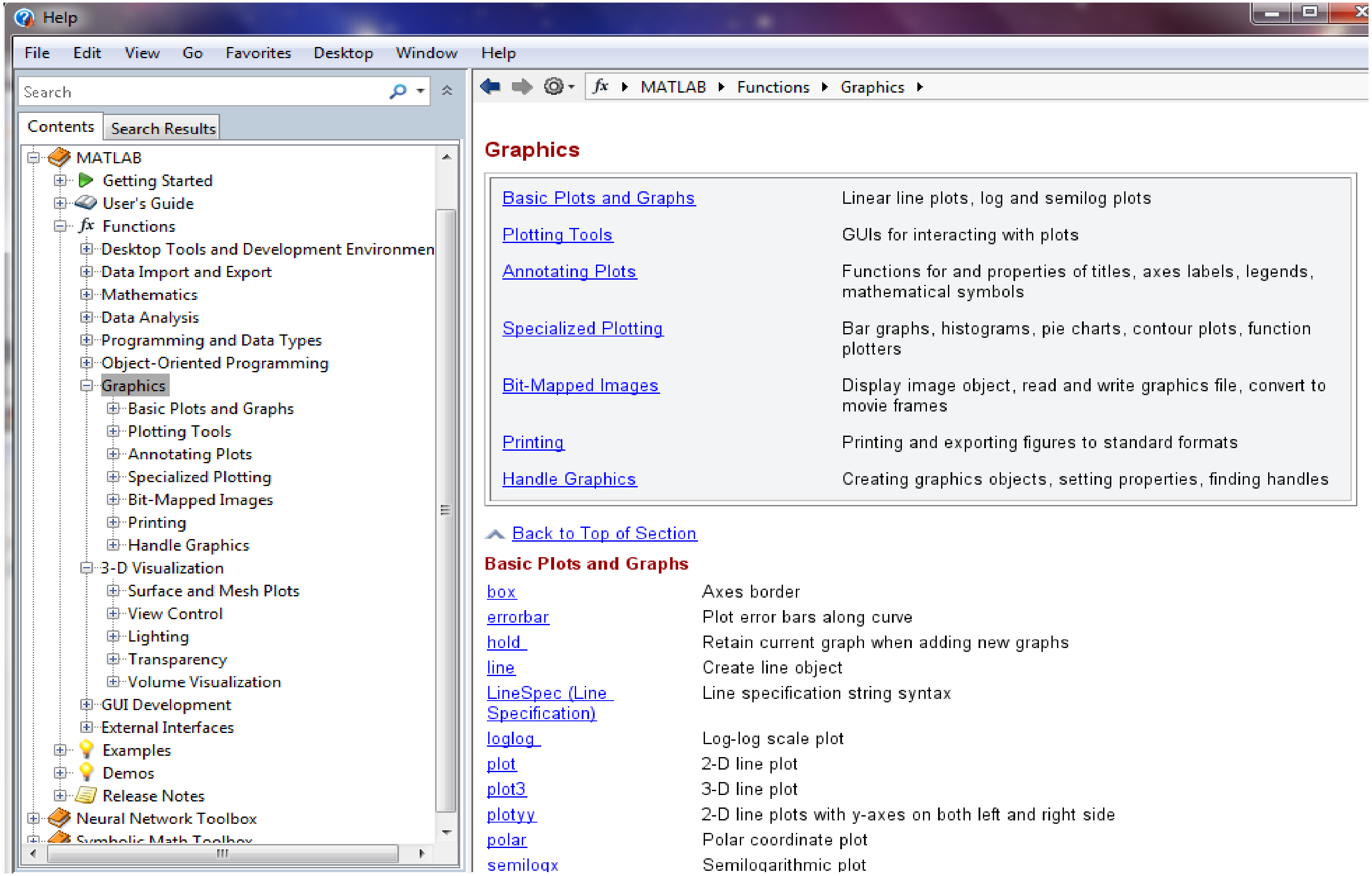




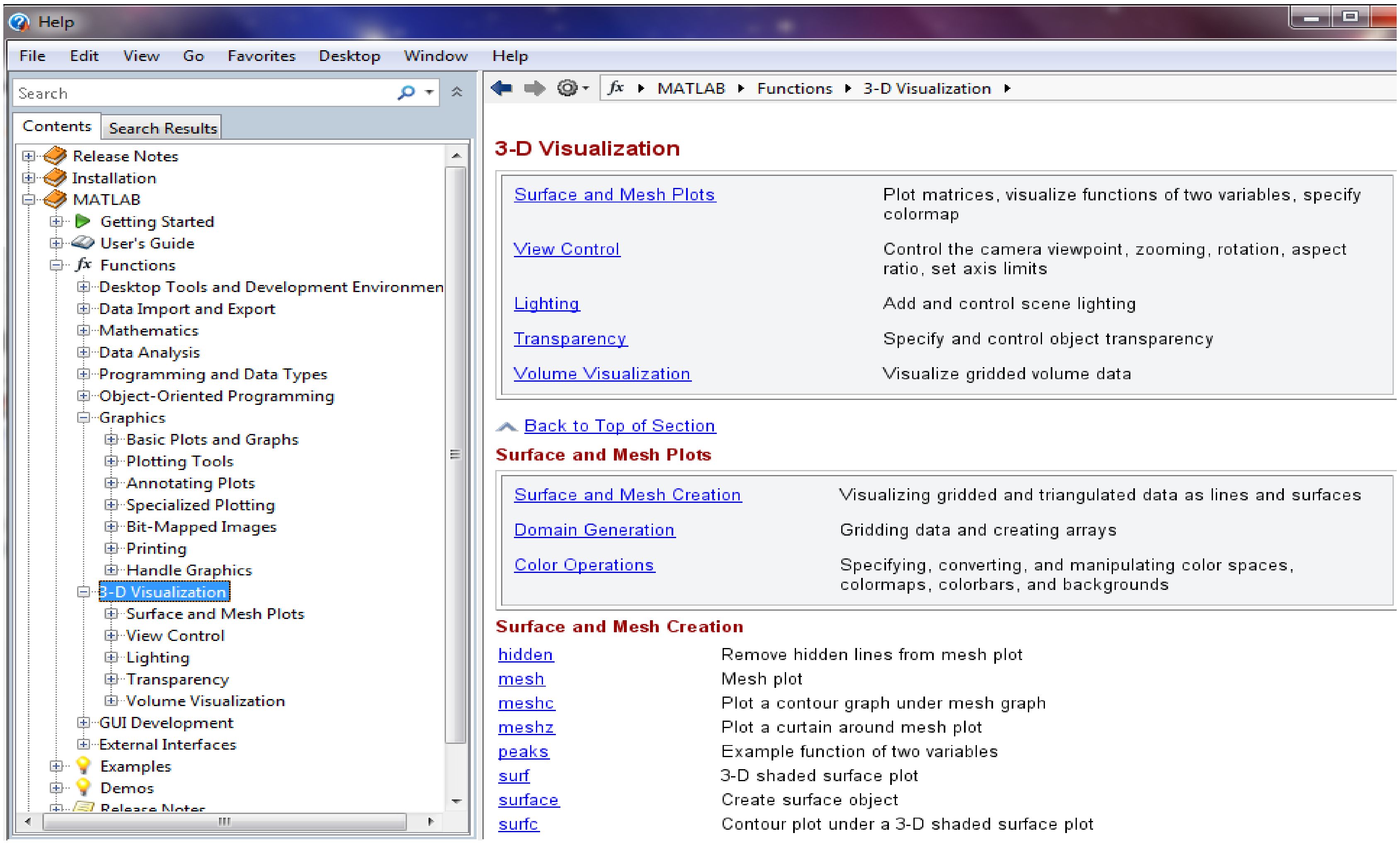
MATLAB Displays

- MATLAB has a suite of displays
- 2d
  - Errorbar, plot, loglog, semilogx, semilogy, polar, bar, pie, hist, scatter
  - title, xlabel, ylabel, legend, grid, hold, axis
- 3d
  - Contour, surface, mesh, quiver

Help - Graphics



Help – 3 d


Help
File Edit View Go Favorites Desktop Window Help
Search
Contents
Search Results
Release Notes
Installation
MATLAB
Getting Started
User's Guide
Functions
Desktop Tools and Development Environmen
Data Import and Export
Mathematics
Data Analysis
Programming and Data Types
Object-Oriented Programming
Graphics
Basic Plots and Graphs
Plotting Tools
Annotating Plots
Specialized Plotting
Bit-Mapped Images
Printing
Handle Graphics
3-D Visualization
Surface and Mesh Plots
View Control
Lighting
Transparency
Volume Visualization
GUI Development
External Interfaces
Examples
Demos
MATLAB ▸ Functions ▸ 3-D Visualization ▸
3-D Visualization
Surface and Mesh Plots — Plot matrices, visualize functions of two variables, specify colormap
View Control — Control the camera viewpoint, zooming, rotation, aspect ratio, set axis limits
Lighting — Add and control scene lighting
Transparency — Specify and control object transparency
Volume Visualization — Visualize gridded volume data
Back to Top of Section
Surface and Mesh Plots
Surface and Mesh Creation — Visualizing gridded and triangulated data as lines and surfaces
Domain Generation — Gridding data and creating arrays
Color Operations — Specifying, converting, and manipulating color spaces, colormaps, colorbars, and backgrounds
Surface and Mesh Creation
hidden — Remove hidden lines from mesh plot
mesh — Mesh plot
meshc — Plot a contour graph under mesh graph
meshz — Plot a curtain around mesh plot
peaks — Example function of two variables
surf — 3-D shaded surface plot
surface — Create surface object
surfc — Contour plot under a 3-D shaded surface plot

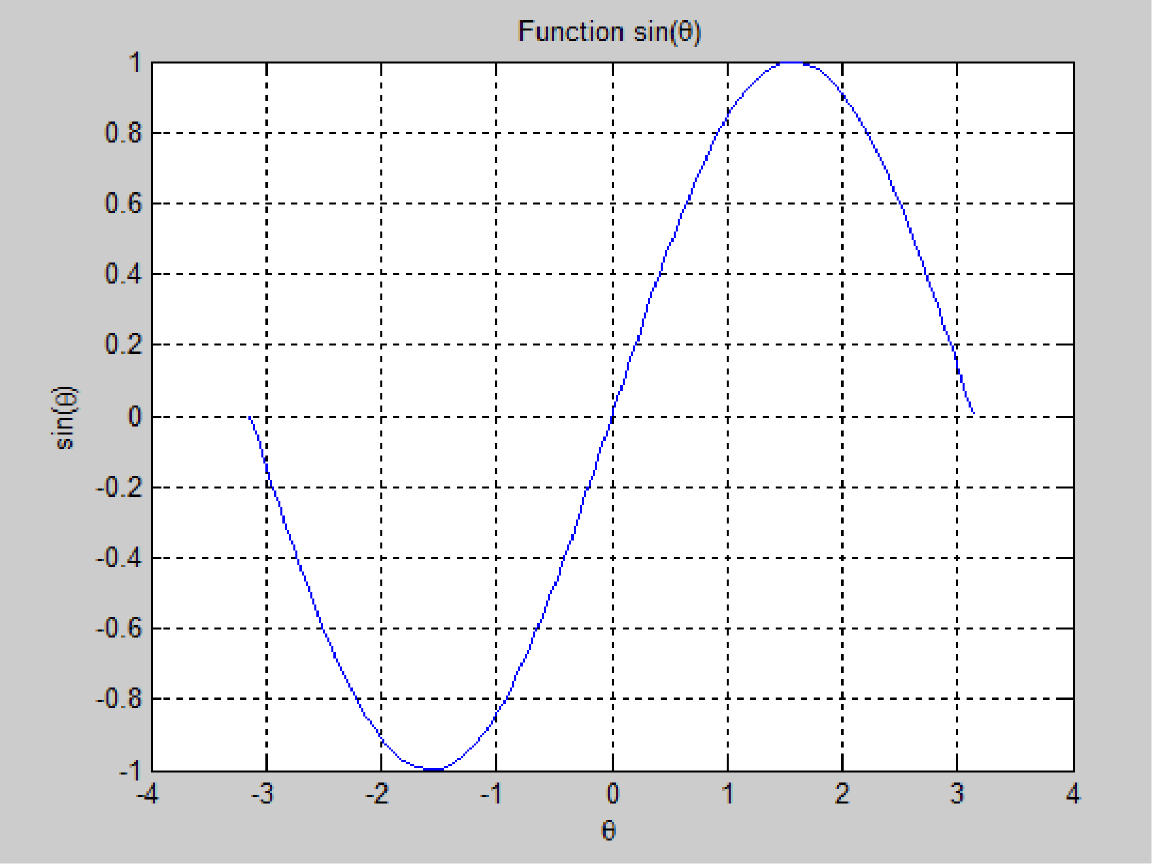
Plot details, Axis definition

```
Various line types, plot symbols and colors may be obtained with
plot(X,Y,S) where S is a character string made from one element
from any or all the following 3 columns:

       b     blue          .     point              -     solid
       g     green         o     circle             :     dotted
       r     red           x     x-mark             -.    dashdot
       c     cyan          +     plus               --    dashed
       m     magenta       *     star             (none)  no line
       y     yellow        s     square
       k     black         d     diamond
       w     white         v     triangle (down)
                           ^     triangle (up)
                           <     triangle (left)
                           >     triangle (right)
                           p     pentagram
                           h     hexagram

For example, plot(X,Y,'c+:') plots a cyan dotted line with a plus
at each data point; plot(X,Y,'bd') plots blue diamond at each data
```

```
>> help axis
 axis  Control axis scaling and appearance.
    axis([XMIN XMAX YMIN YMAX]) sets scaling for the x- and y-axes
       on the current plot.
```

plot(x,y) –

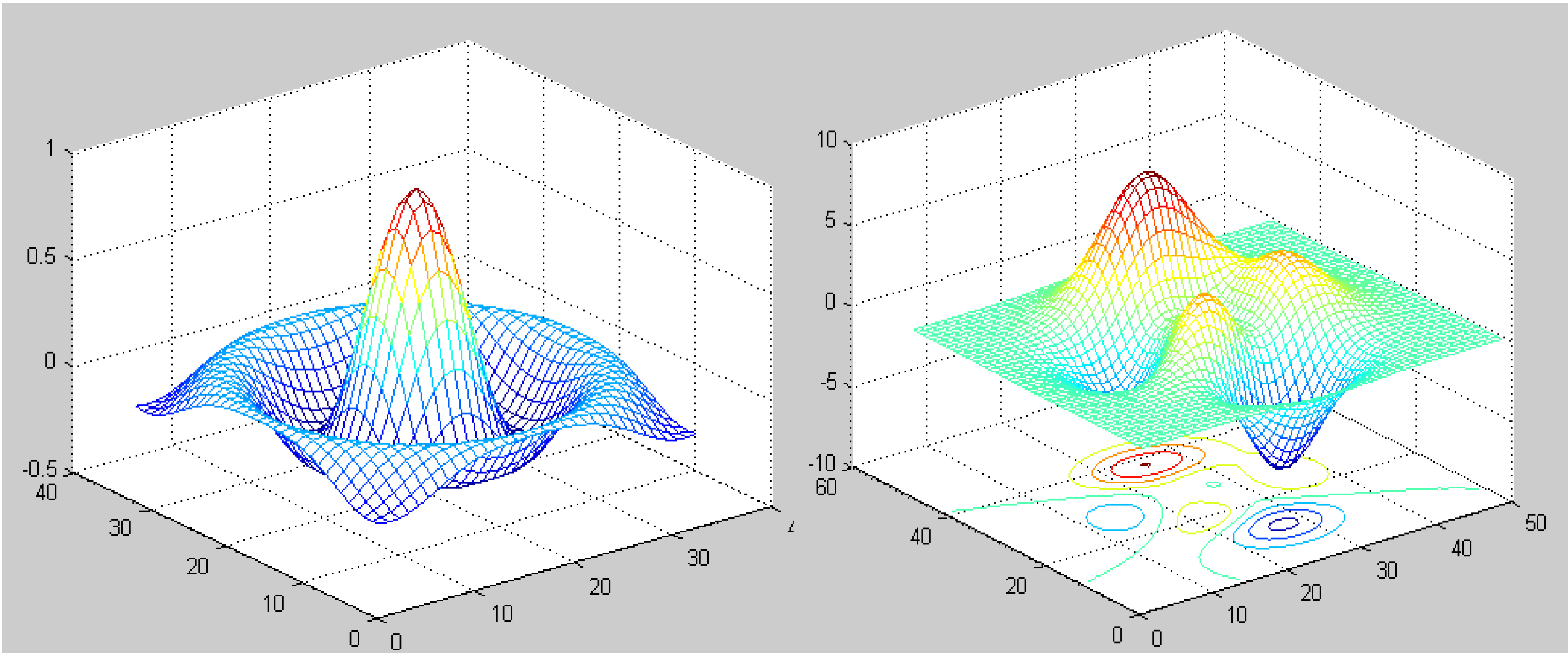
```
%
% plot demo
%
x = linspace(-pi,pi);
y = sin(x);
plot(x,y)
title('Function sin(\theta)')
xlabel('\theta')
ylabel('sin(\theta)')
grid
```



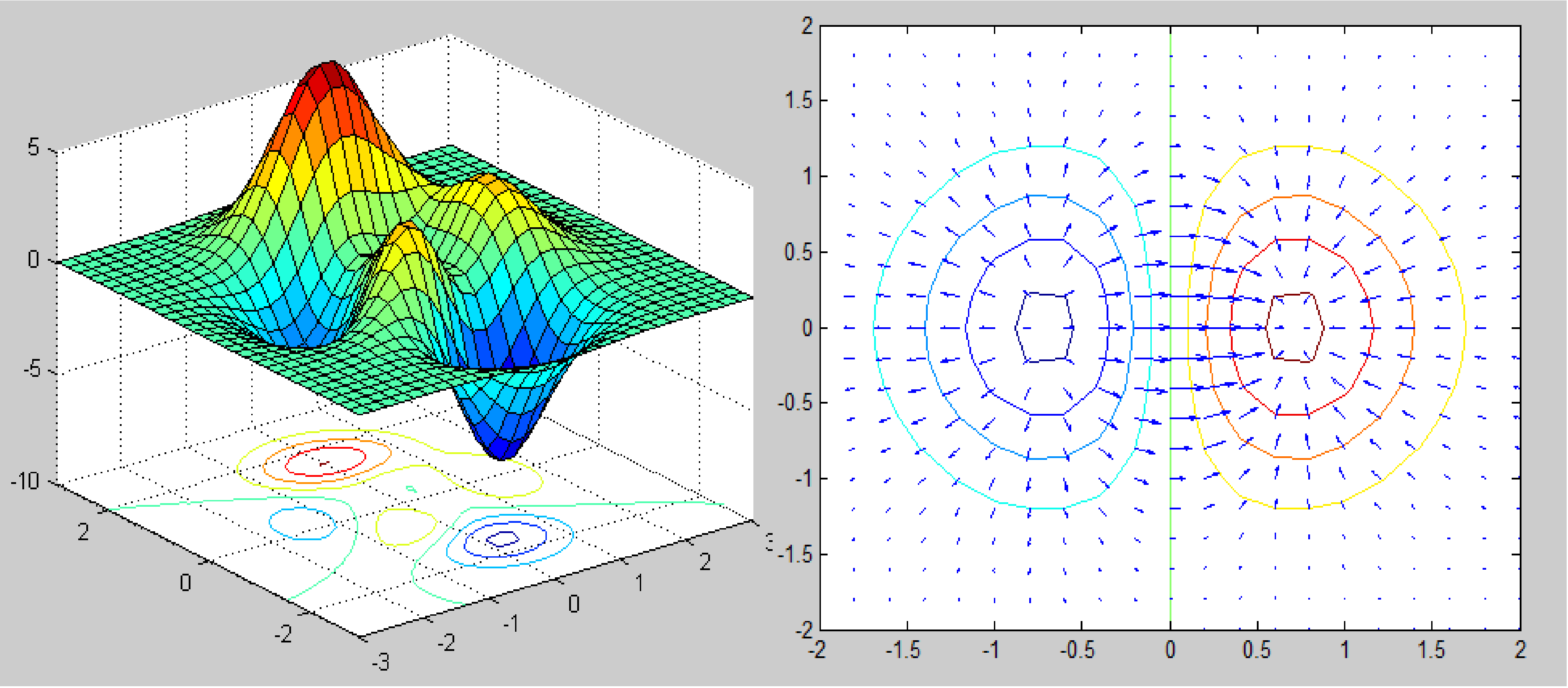
3-d Plots

```
%
% demo for simple 3-d plots
%
figure(1)
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
mesh(Z);
%
figure(2)
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshc(Z);
%
figure(3)
[X,Y,Z] = peaks(30);
surfc(X,Y,Z);
axis([-3 3 -3 3 -10 5]);
%
figure(4)
[X,Y] = meshgrid(-2:.2:2);
Z = X.*exp(-X.^2 - Y.^2);
[DX,DY] = gradient(Z,.2,.2);
contour(X,Y,Z)
hold on
quiver(X,Y,DX,DY)
hold off
%
```

Mesh and Meshc



Surfc and quiver

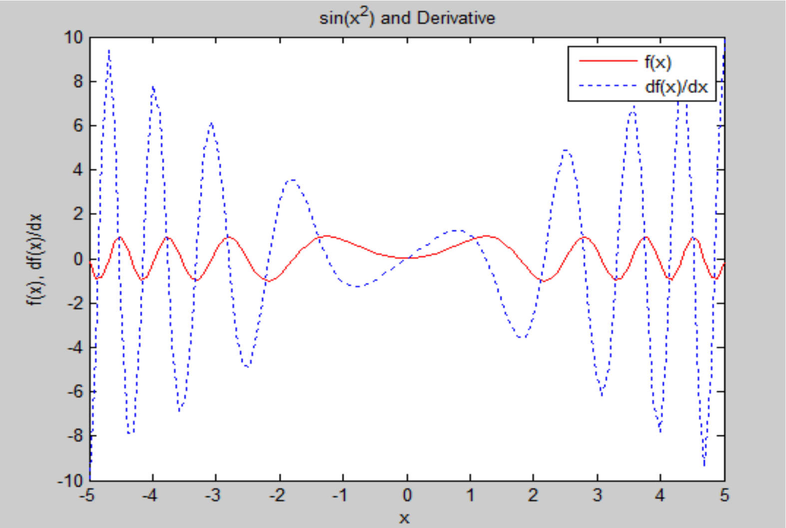


MATLAB Symbolic Tools

- MATLAB has many, many tools. You will have to browse through the options.
  - Diff, int (calculus)
  - Matrix inversion
  - Eigenvalues
  - Taylor
  - Fourier – our scripts
- Other examples and script in the textbook

SM_Diff and SM_Int

- **Use MATLAB diff, int you need never do another integral**




```
>> SM_Int
  Symbolic Math - Integration


Symbolic Math Indefinite Integration: enter f(x), + - * / ^ cos sin tan sqrt,
 An Example, Plot Function and Integral

fin =

tanh(x)


  log(cosh(x))
```

SM_Diff Script “loop” -blue

- ~ standard “loop”
- Use “while”

  “break”

  “for”

  “if”

  “end”

```
while irun > 0
    krun = menu('Another Function?','Yes', 'No');
    if krun == 2
        irun = -1;
        break
    end
    %
    if krun == 1;
        clear( 'x', 'fin', 'fpr')
        syms  x a b n fin fpr Y N yy
        iloop = iloop + 1;
        fin = input(' Enter f(x): ');
        %
        fpr = diff(sym(fin));
        yy = simple(fpr);
        pretty(yy)
        %
        xx = linspace(-5,5);
        for i = 1:100
            x = xx(i);
            fx(i) = eval(fin);
            dfx(i) = eval(fpr);
        end
        figure(iloop)
        plot(xx,fx,'r-',xx,dfx,':b')
        xlabel('x');
        ylabel('f(x), df(x)/dx')
        title(' Function and Derivative')
        legend('f(x)','df(x)/dx')
    %
    end
end
```

Eigenvalues

```
>> SM_Eigen
  Symbolic Math - Square Matrices: Determinants, Inverses Eigenvalues, Eigenve


Symbolic Matrices: Eignevalues and Eigenvectors
  An Example - Rotation by x
A =

[ cos(x), -sin(x)]
[ sin(x),  cos(x)]

Determinint of A

  1
Inverse of A

  +-                 -+
  |   cos(x), sin(x)  |
  |                   |
  |  -sin(x), cos(x)  |
  +-                 -+
```

```
Eigenvalues of A, Diagonals

  +-                       -+
  |  exp(-x i),      0      |
  |                         |
  |      0,      exp(x i)   |
  +-                       -+
Eigenvectors of A, Columns

  +-        -+
  |  -i, i   |
  |          |
  |   1, 1   |
  +-        -+
 Enter A, [A11, A12; A21, A22]: [sin(x)^2, cos(x); -cos(x) , sin(x)^2]
```

You need never solve another eigenvalue problem

Eigen Script

- Used diff and int for the 2 previous examples
- Now use MATLAB matrix tools, det, inv and eig

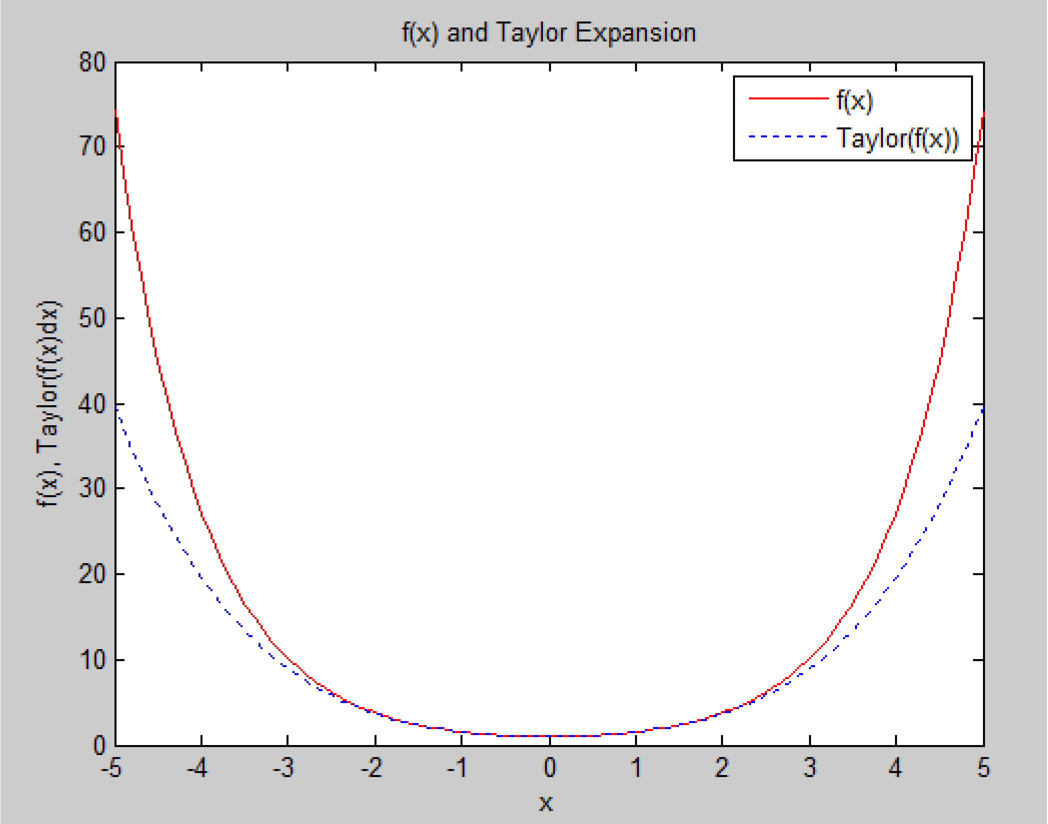
```
fprintf(' An Example - Rotation by x')
A = [cos(x),-sin(x) ; sin(x), cos(x)]
fprintf('Determinint of A \n ')
Adet = det(A);
y = simple(Adet);
pretty(y)
%
% next the inverse;
%
fprintf('Inverse of A \n ')
Ainv = inv(A);
y = simple(Ainv);
pretty(y)
%
% then the eigenvalues and eigevectors ;
%
fprintf('Eigenvalues of A, Diagonals \n ')
 [Veig,Aeig]  = eig(A);
 y = simple(Aeig);
 pretty(y)
 fprintf('Eigenvectors of A, Columns \n ')
 y = simple(Veig);
 pretty(y)
```

Taylor Series

- **See convergence of the series graphically – select the number of terms**

```
ftay = taylor(sym(fin),x,aoff,'Order',nterm);
      yy = simple(ftay);
      pretty(yy)
```

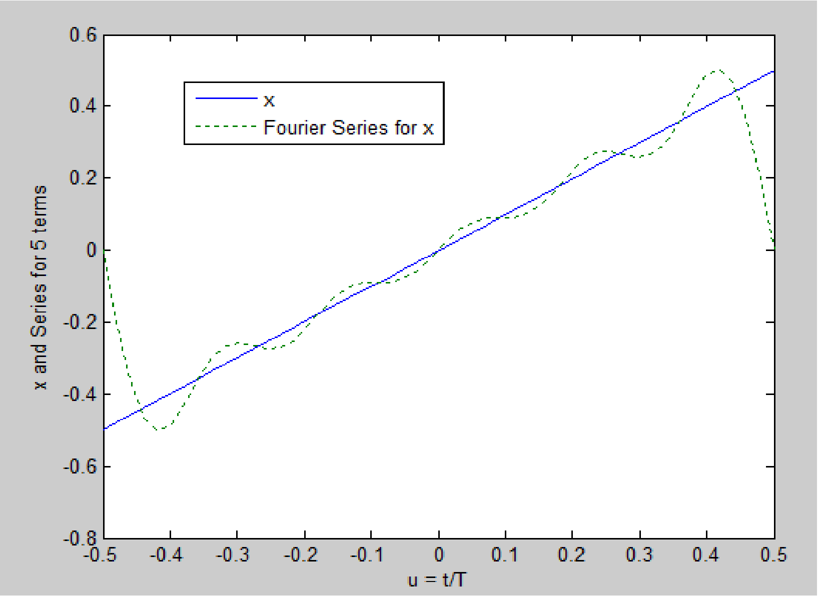
```
 Enter f(x): cosh(x)
 Enter Number of Terms: 5
Enter a, Expansion About x = a: 0

   4    2
  x    x
  -- + -- + 1
  24   2
```



Fourier - 1

- **Examples worked out already – square, triangle and sawtooth. See convergence with number of terms**

```
>> SM_fourier_ex
  Fourier - fourier series for functions, period = [-1/2,1/2]


x = ao/2 + sumk( ak*cos(kwt) + bk*sin(kwt) )
ak = 2*int(x(u)*cos(2*pi*k*u)du, bk = 2*int(x(u)*sin(2*pi*k*u)d
u = t/T, [-1/2,1/2], w = 2*pi/T, w*t = 2*pi*u
z = pi*k, bk = 0, ak = 2*(1-cosz)/z^2
Cos Coefficients

a =

     0     0     0     0     0

Sin Coefficients

b =

    0.3183   -0.1592    0.1061   -0.0796    0.0637
```



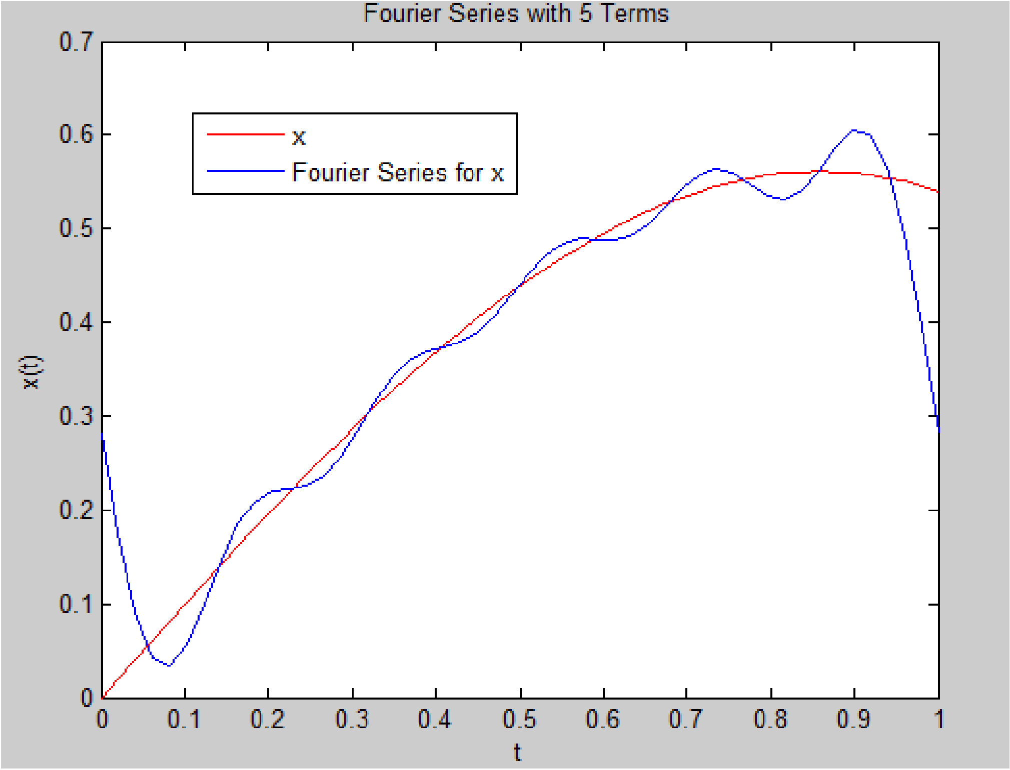
Fourier - 2

- Use int, sym and eval to do a general Fourier series. Any analytically solvable series!

```
>> SM_fourier_try
   Fourier - fourier series for analytic function - input


Period is T = 1, Enter x(t) for First and Second 1/2 Period
Enter x(t) for t < 1/2 as an equation; e.g., t*cos(t)
: t*cos(t)
Enter x(t) for t > 1/2 as an equation; e.g., t*cos(t)
: t*cos(t)
Enter Number of Terms in the Series : 5
Even and Odd Fourier Coefficients

a =

[ 2*cos(1) + 2*sin(1) - 2, (2*(2*cos(1) + 2*cos(1/2) + 2*sin(1) + sin(1/2) + pi^2*(8*cos(1)


b =

[ 0, (2*(pi*(2*cos(1) + cos(1/2) - 4*sin(1) - 4*sin(1/2)) - pi^3*(8*cos(1) + 4*cos(1/2))))/

Full Fourier Series

xx =

cos(1) + sin(1) - sin(2*pi*t)*((2*(pi*(cos(1/2) - 4*sin(1/2)) - 4*pi^3*cos(1/2)))/(4*pi^2 -
```

Fourier - 3

- **Example for f(t) = t*cos(t), 5 terms**



$$x = a_o/2 + \sum_k [a_k \cos(k\omega t) + b_k \sin(k\omega t)]$$

$$a_k = 2\int x(u)\cos(2\pi\, ku)du$$

$$b_k = 2\int x(u)\sin(2\pi\, ku)du$$

$$u = t/T,\ [-1/2, 1/2]$$

$$\omega = 2\pi/T,\ \omega t = 2\pi u$$

Numeric Tools - I

- Most problems must be solved numerically
- Numeric tools for a vector x
  - Length
  - Min
  - Max
  - Mean
  - Std
  - sort

Numeric Tools- II

- **MATLAB utility "quad" does numeric integration. Try "int" first – default to "quad" if it fails.**

quad

Numerically evaluate integral, adaptive Simpson quadrature

Syntax

```
q = quad(fun,a,b)
q = quad(fun,a,b,tol)
q = quad(fun,a,b,tol,trace)
[q,fcnt] = quad(...)
```

Description

Quadrature is a numerical method used to find the area under the graph of a function, that is, to compute a definite integral.

$$q = \int_a^b f(x)dx$$

`q = quad(fun,a,b)` tries to approximate the integral of function `fun` from `a` to `b` to within an error of `1e-6` using recursive adaptive Simpson quadrature. `fun` is a function handle. See Function Handles in the MATLAB Programming documentation for more information. Limits `a` and `b` must be finite. The function `y = fun(x)` should accept a vector argument `x` and return a vector result `y`, the integrand evaluated at each element of `x`.

MATLAB Equation Solvers

- **MATLAB has several ways to solve equations**
  - **Symbolic solutions should be tried first**
  - **Define symbolic variables using “syms”**
  - **“solve” is for algebraic equations**
  - **“dsolve” is for ordinary differential equations**
  - **“ode45” is for numerical solutions to ODE**
  - **“pde” is for partial differential equations in 1 space (x) and 1 time (t) dimension**

Help for solve – has examples

fx ▸ Symbolic Math Toolbox ▸ Functions ▸ Solution of Equations ▸ solve

solve

Equations and systems solver

Syntax

```
S = solve(eqn)
S = solve(eqn,var,Name,Value)
Y = solve(eqn1,...,eqnN)
Y = solve(eqn1,...,eqnN,var1,...,varN,Name,Value)
[y1,...,yN] = solve(eqn1,...,eqnN)
[y1,...,yN] = solve(eqn1,...,eqnN,var1,...,varN,Name,Value)
```

Description

`S = solve(eqn)` solves the equation `eqn` for the default variable determined by `symvar`. You can specify the independent variable. For example, `solve(x + 1 == 2, x)` solves the equation $x + 1 = 2$ with respect to the variable x.

`S = solve(eqn,var,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments. If you do not specify `var`, the solver uses the default variable determined by `symvar`.

`Y = solve(eqn1,...,eqnN)` solves the system of equations `eqn1,...,eqnN` for the variables determined by `symvar` and returns a structure array that contains the solutions. The number of fields in the structure array corresponds to the number of independent variables in a system.

`Y = solve(eqn1,...,eqnN,var1,...,varN,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments. If you do not specify `var1,...,varN`, the solver uses the default variables determined by `symvar`.

`[y1,...,yN] = solve(eqn1,...,eqnN)` solves the system of equations `eqn1,...,eqnN` for the variables determined by `symvar` and assigns the solutions to the variables `y1,...,yN`.

`[y1,...,yN] = solve(eqn1,...,eqnN,var1,...,varN,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments. If you do not specify `var1,...,varN`, the solver uses the default variables determined by `symvar`.

- **Use help and search for solve**
- **Symbolic solution**
- **You need never solve another ODE**

solve -2

- Use command window to define a simple quadratic

```
>> syms a b c x root
>> root = solve(a*x^2 + b*x + c == 0)

root =

 -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
 -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

- Use “simple” and “pretty for visualization

```
>> pretty(root)

  +-                             -+
  |            2          1/2     |
  |      b + (b   - 4 a c)        |
  |   -  ------------------       |
  |              2 a              |
  |                               |
  |            2          1/2     |
  |      b - (b   - 4 a c)        |
  |   -  ------------------       |
  |              2 a              |
  +-                             -+
```

dsolve

- **Solves ODE symbolically with IC**

Symbolic Math Toolbox ▸ Functions ▸ Solution of Equations ▸ dsolve

dsolve

Ordinary differential equation and system solver

Syntax

```
S = dsolve(eqn)
S = dsolve(eqn,cond)
S = dsolve(eqn,cond,Name,Value)
Y = dsolve(eqn1,...,eqnN)
Y = dsolve(eqn1,...,eqnN,cond1,...,condN)
Y = dsolve(eqn1,...,eqnN,cond1,...,condN,Name,Value)
[y1,...,yN] = dsolve(eqn1,...,eqnN)
[y1,...,yN] = dsolve(eqn1,...,eqnN,cond1,...,condN)
[y1,...,yN] = dsolve(eqn1,...,eqnN,cond1,...,condN,Name,Value)
```

Description

`S = dsolve(eqn)` solves the ordinary differential equation `eqn`. Here `eqn` is a symbolic equation containing `diff` to indicate derivatives. Alternatively, you can use a string with the letter `D` indicating derivatives. For example, `syms y(x); dsolve(diff(y) == y + 1)` and `dsolve('Dy = y + 1','x')` both solve the equation `dy/dx = y + 1` with respect to the variable `x`. Also, `eqn` can be an array of such equations or strings.

`S = dsolve(eqn,cond)` solves the ordinary differential equation `eqn` with the initial or boundary condition `cond`.

`S = dsolve(eqn,cond,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

`Y = dsolve(eqn1,...,eqnN)` solves the system of ordinary differential equations `eqn1,...,eqnN` and returns a structure array that contains the solutions. The number of fields in the structure array corresponds to the number of independent variables in the system.

`Y = dsolve(eqn1,...,eqnN,cond1,...,condN)` solves the system of ordinary differential equations `eqn1,...,eqnN` with the initial or boundary conditions `cond1,...,condN`.

`Y = dsolve(eqn1,...,eqnN,cond1,...,condN,Name,Value)` uses additional options specified

Dsolve - example

- Use symbolically for ODE. Initial conditions

```
>> syms a y(t)
>> Dy = diff(y);
>> D2y = diff(y, 2);
>> dsolve(D2y == -a*y)

ans =

C7*exp((-a)^(1/2)*t) + C8*exp(-(-a)^(1/2)*t)

>> dsolve(D2y == -a*y , Dy(0) == 0, y(0) ==1)

ans =

exp((-a)^(1/2)*t)/2 + exp(-(-a)^(1/2)*t)/2

>> simple(ans)

simplify:

cosh((-a)^(1/2)*t)
```

SM_ODE3

- **SM_ODE2 allows a choice of a few well known equations. SM_ODE3 is free form.**

```
>> SM_ODE3
  Program to symbolically solve ODE


Enter Single Differential Eq to Solve y(t); e.g., D2y+a*y=0
: Dy+a*y^3=0
Enter Condition on Function; e.g. y(a) = b,
: y(0)=b
Symbolic Solution, y(t) and v(t)

y =

  (2^(1/2)*(1/(a*t + 1/(2*b^2)))^(1/2))/2
 -(2^(1/2)*(1/(a*t + 1/(2*b^2)))^(1/2))/2


v =

 -(2^(1/2)*a)/(4*(1/(a*t + 1/(2*b^2)))^(1/2)*(a*t + 1/(2*b^2))^2)
  (2^(1/2)*a)/(4*(1/(a*t + 1/(2*b^2)))^(1/2)*(a*t + 1/(2*b^2))^2)
```

ode45

- **Numerical solution of ODE**

MATLAB ▸ Functions ▸

ode23, ode45, ode113, ode15s, ode23s, ode23t, ode23tb

Solve initial value problems for ordinary differential equations

Syntax

```
[T,Y] = solver(odefun,tspan,y0)
[T,Y] = solver(odefun,tspan,y0,options)
[T,Y,TE,YE,IE] = solver(odefun,tspan,y0,options)
sol = solver(odefun,[t0 tf],y0...)
```

where `solver` is one of `ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t`, or `ode23tb`.

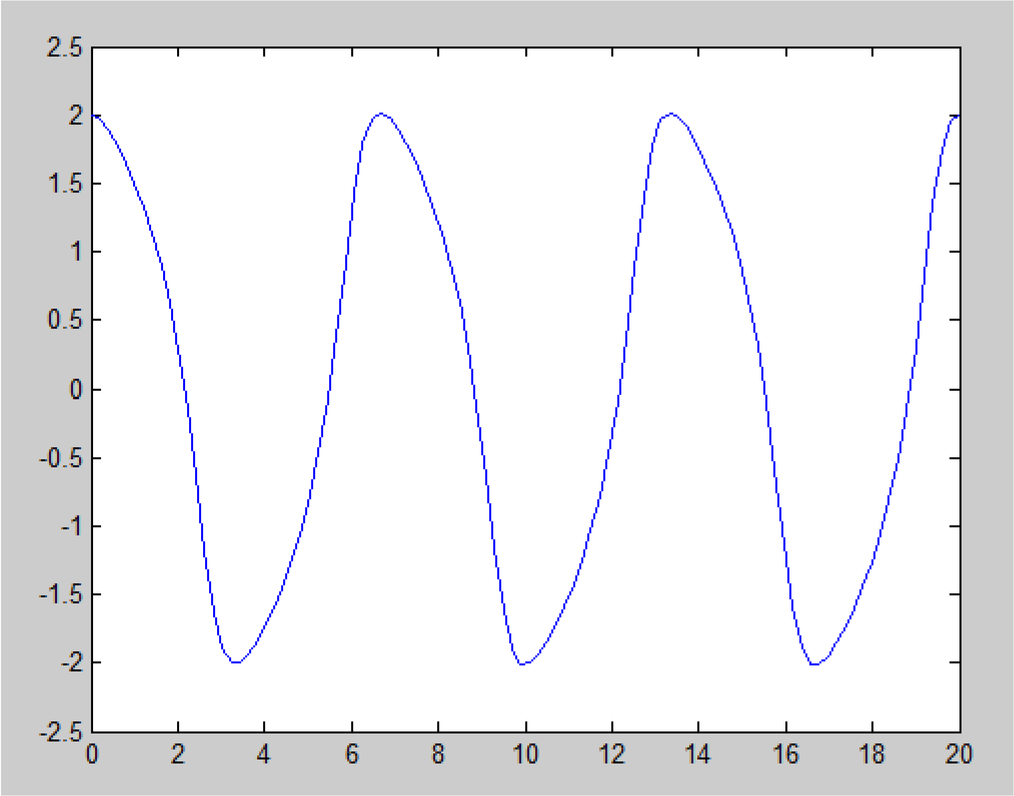
Arguments

The following table describes the input arguments to the solvers.

`odefun`	A function handle that evaluates the right side of the differential equations. See Function Handles in the MATLAB Programming documentation for more information. All solvers solve systems of equations in the form $y' = f(t,y)$ or problems that involve a mass matrix, $M(t,y)y' = f(t,y)$. The `ode23s` solver can solve only equations with constant mass matrices. `ode15s` and `ode23t` can solve problems with a mass matrix that is singular, i.e., differential-algebraic equations (DAEs).

ode45 - 2

Need to define a function which specifies the ODE




```
>> sol = ode45(@vdp1,[0 20],[2 0]);
>> x = linspace(0,20);
>> y = deval(sol,x,1);
>> plot(x,y);
```

ode45 - 3

- **The physics is all in the function specified. The boundary conditions are specified in the ode45 call as is the time evolution of the system. We do examples later.**

An example of a nonstiff system is the system of equations describing the motion of a rigid body without external forces.

$$
\begin{aligned}
y_1' &= y_2 y_3 & y_1(0) &= 0 \\
y_2' &= -y_1 y_3 & y_2(0) &= 1 \\
y_3' &= -0.51\, y_1 y_2 & y_3(0) &= 1
\end{aligned}
$$

To simulate this system, create a function `rigid` containing the equations

```
function dy = rigid(t,y)
dy = zeros(3,1);    % a column vector
dy(1) = y(2) * y(3);
dy(2) = -y(1) * y(3);
dy(3) = -0.51 * y(1) * y(2);
```

• PDE solver in MATLAB

PDE Solver Syntax

The basic syntax of the solver is:

```
sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan)
```

The input arguments are

`m`	Specifies the symmetry of the problem. `m` can be `0` = slab, `1` = cylindrical, or `2` = spherical. It corresponds to *m* in Equation 10-2.
`pdefun`	Function that defines the components of the PDE. It computes the terms *c*, *f*, and *s* in Equation 10-2, and has the form `[c,f,s] = pdefun(x,t,u,dudx)` where `x` and `t` are scalars, and `u` and `dudx` are vectors that approximate the solution *u* and its partial derivative with respect to *x*. `c`, `f`, and `s` are column vectors. `c` stores the diagonal elements of the matrix *c*.
`icfun`	Function that evaluates the initial conditions. It has the form `u = icfun(x)` When called with an argument `x`, `icfun` evaluates and returns the initial values of the solution components at `x` in the column vector `u`.

pde-2

• Example from 1-d Quantum Mechanics

```
sol = pdepe(m,@Sch_pde,@Sch_ic,@Sch_bc,x,t);
%
psixt = sol(:,:,1);
%

%-----------------------------------------------------------------------
function [c,f,s] = Sch_pde(x,t,u,DuDx)
%
global xm dxo k Vo a
hbar = 0.666; % units eV,A, 10^-15 sec
mec2 = 511000.0;  % eV - electron mass
hbarc = 2000.0;   % ev*A
c = j .*hbar;
f = -(hbarc .*hbarc .*DuDx) ./(2.0 .*mec2); % kinetic energy
%s = 0.0;   % free particle
if x < 0 | x > a
    s = 0;
else
    s = Vo .*u; % constant potential energy
end
%
```

Special Functions

- You need never again look up a special function! Search MATLAB help (mfun)
- Run script DG/SM ” Matlab_Functions”

Function Name	Definition	mfun Name	Arguments
Bernoulli numbers and polynomials	Generating functions: $\frac{e^{xt}}{e^t - 1} = \sum_{n=0}^{\infty} B_n(x) \cdot \frac{t^{n-1}}{n!}$	bernoulli(n) bernoulli(n,t)	$n \geq 0$ $0 < \lvert t \rvert < 2\pi$
Bessel functions	BesselI, BesselJ—Bessel functions of the first kind. BesselK, BesselY—Bessel functions of the second kind.	BesselJ(v,x) BesselY(v,x) BesselI(v,x) BesselK(v,x)	v is real.
Beta function	$B(x, y) = \frac{\Gamma(x) \cdot \Gamma(y)}{\Gamma(x+y)}$	Beta(x,y)	
Binomial coefficients	$\binom{m}{n} = \frac{m!}{n!(m-n)!}$ $= \frac{\Gamma(m+1)}{\Gamma(n+1)\Gamma(m-n+1)}$	binomial(m,n)	
Complete elliptic integrals	Legendre's complete elliptic integrals of the first, second, and third kind. This definition uses modulus k. The numerical ellipke function and the MuPAD functions for computing elliptic integrals use the parameter $m = k^2 = \sin^2 \alpha$.	EllipticK(k) EllipticE(k) EllipticPi(a,k)	a is real, $-\infty < a < \infty$. k is real, $0 < k < 1$.
Complete elliptic integrals with complementary modulus	Associated complete elliptic integrals of the first, second, and third kind using complementary modulus. This definition uses modulus k. The numerical ellipke function and the MuPAD functions for computing elliptic integrals use the parameter $m = k^2 = \sin^2 \alpha$.	EllipticCK(k) EllipticCE(k) EllipticCPi(a,k)	a is real, $-\infty < a < \infty$. k is real, $0 < k < 1$.

functions

Categories

Special Functions

airy	Airy functions
besselh	Bessel function of third kind (Hank...
besseli	Modified Bessel function of first kind
besselj	Bessel function of first kind
besselk	Modified Bessel function of second...
bessely	Bessel function of second kind
beta	Beta function
betainc	Incomplete beta function
betaincinv	Beta inverse cumulative distributio...
betaln	Logarithm of beta function
ellipj	Jacobi elliptic functions
ellipke	Complete elliptic integrals of first a...
erf	Error function
erfc	Complementary error function
erfcinv	Inverse complementary error functi...
erfcx	Scaled complementary error function
erfinv	Inverse error function
expint	Exponential integral
gamma	Gamma function
gammainc	Incomplete gamma function
gammaincinv	Inverse incomplete gamma function
gammaln	Logarithm of gamma function
legendre	Associated Legendre functions