CEE 3804

Advanced MATLAB Functions

Drs. Trani and Rakha
Virginia Polytechnic Institute and State University

Spring 2000

Working with Polynomials

Polynomials are expressed in vector form

$$y = 3x^3 + 2x^2 + x + 23$$

in MATLAB nomenclature this will be:

```
 y=[3  2  1  23]
y =
    3     2     1    23
```

Note: if some powers are not represented in the polynomial just set them to zero

Convoluting Polynomials

Define another polynomial such as:

$f = x^2 + 3x + 1$ or f = [1 3 1]

Now multiply both using MATLAB's 'conv' function

```
conv(y,f)
ans =
   3   11   10   28   70   23
```

which is equivalent to,

$$g = 3x^5 + 11x^4 + 10x^3 + 28x^2 + 70x + 23$$

Roots of Polynomials

Take the polynomial,

$$g = 3x^5 + 11x^4 + 10x^3 + 28x^2 + 70x + 23$$

To find the roots we use the ‘roots’ command,

```
roots(g)
ans =
   0.7458 + 1.7309i
   0.7458 - 1.7309i
  -2.6180
  -2.1582
  -0.3820
```

Polynomial Evaluation

Sometimes we would like to evaluate polynomials at particular points. Suppose that we want to find the value of,

$$g = 3x^5 + 11x^4 + 10x^3 + 28x^2 + 70x + 23$$

at point x=1.4. Use the 'polyval' function in MATLAB.

```
polyval(g,1.4)
ans =
  261.7123
```

Deconvoluting Polynomials

Suppose we want to divide,

$$g = 3x^5 + 11x^4 + 10x^3 + 28x^2 + 70x + 23$$

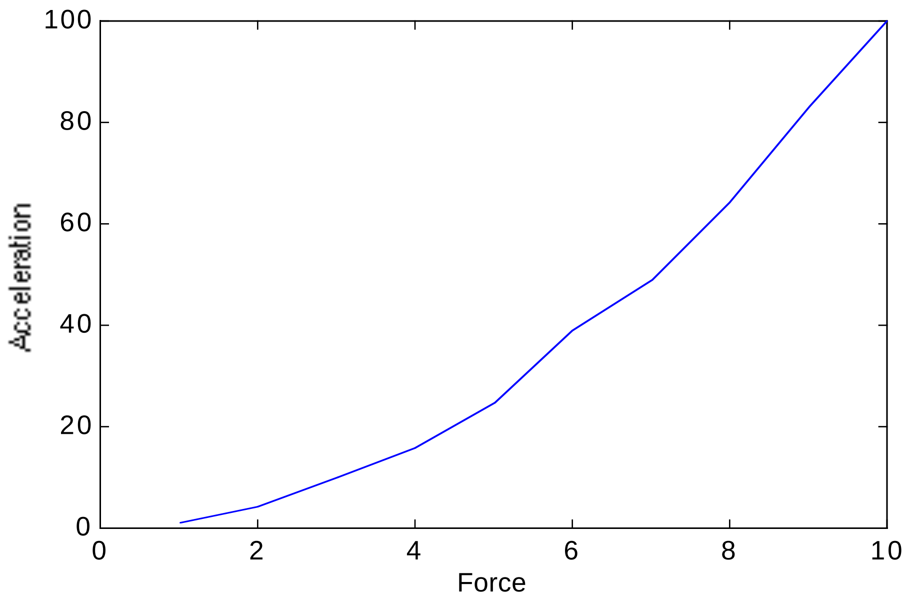
by polynomial $f = x^2 + 3x + 1$ (both have been defined)

```
deconv(g,f)
ans =
    3    2    1    23
```

This is the same as polynomial y previously defined.

Curve Fitting with Polynomials

Suppose the following data is collected in a laboratory



x = [1 2 3 4 5 6 7 8 9 10]

y =[1 4 10 16 25 39 49 64 83 100]

Curve Fitting with Polynomials

Use the 'polyfit' function to approximate the observed behavior. In this case lets try a second degree polynomial.

```
d=polyfit(x,y,2)
d =
  0.9659    0.4477   -0.5500
```

Suppose we want to evaluate values from this resulting polynomial and compare with the original (x,y) values.

Curve Fitting with Polynomials

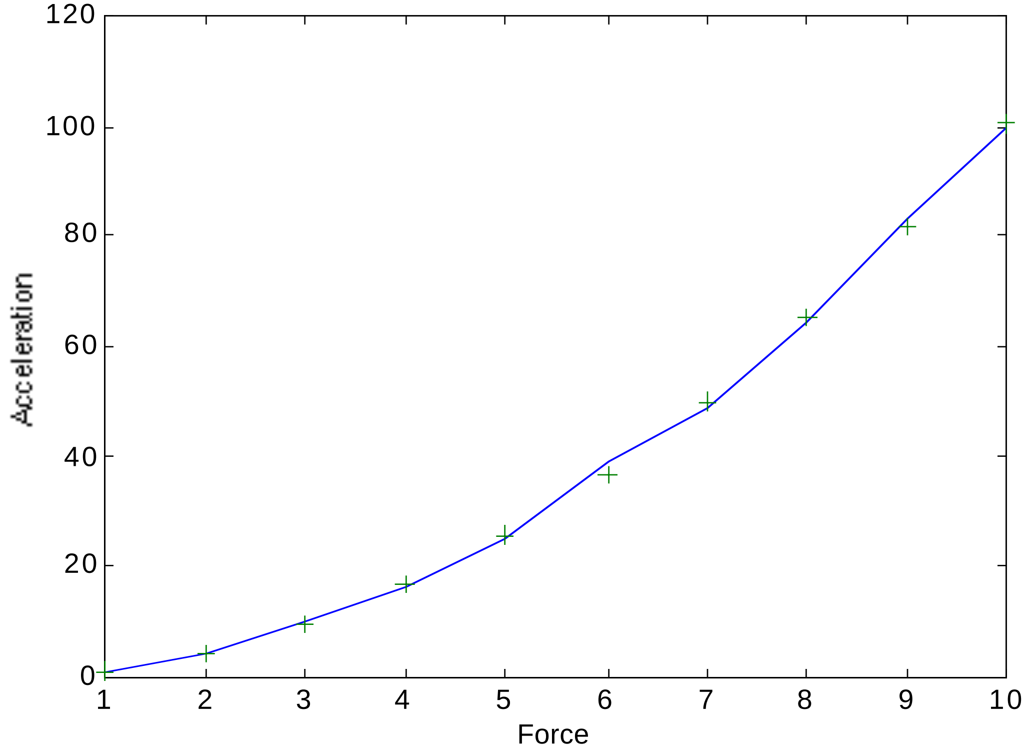
Create a new vector (xnew) with values to be evaluated

```
xnew = 1:1:10;
»s = polyval(d,xnew)
ans =
0.8636    4.2091    9.4864   16.6955   25.8364   36.9091
                                      49.9136   64.8500   81.7182
                                      100.5182
```

Plot the original (x,y) versus (xnew,s)

Curve Fitting with Polynomials

plot(x,y,xnew,s,'+'):xlabel('Force');ylabel('Acceleration')



Interpolation in MATLAB

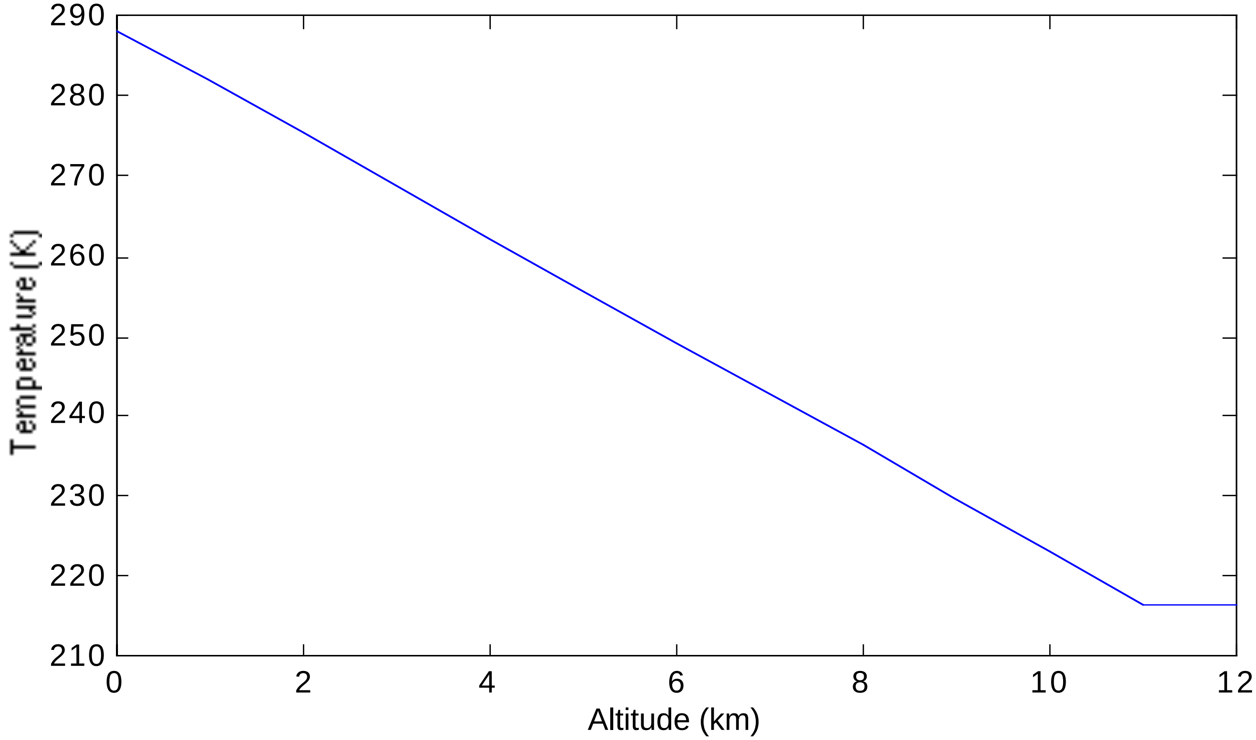
Several interpolation functions exist to facilitate data handling.

Suppose the following data represent temperatures measured in a standard atmosphere as a function of altitude. Altitude (h) in km and temperature (t) in degrees Kelvin.

```
h=[0  1  2  3  4  5  6  7  8  9  10  11  12]
t=[288.2  281.7  275.2  268.7  262.2  255.7  249.2  242.7
                                    236.2  229.7  223.2  216.7]
```

Interpolation in MATLAB

The following plot represents the observed behavior,



Interpolation in MATLAB

Suppose we want to include the temperature data in a program and want to evaluate the temperature in Denver (1.58 km above mean sea level).

Define a variable called h_denver representing its altitude,

```
h_denver =
   1.5800
»a=interp1(h,t,h_denver)
a =
  277.9300
```

Numerical Integration

Some background information is necessary to expose the student to various techniques available to execute numerical integration.

Several numerical methods to be reviewed:

- Standard numerical integration
- Numerical differentiation methods
- Differential equation solvers (document 4.2)

Matlab offers several procedures and built-in functions to address these methods

Standard Numerical Integration Methods

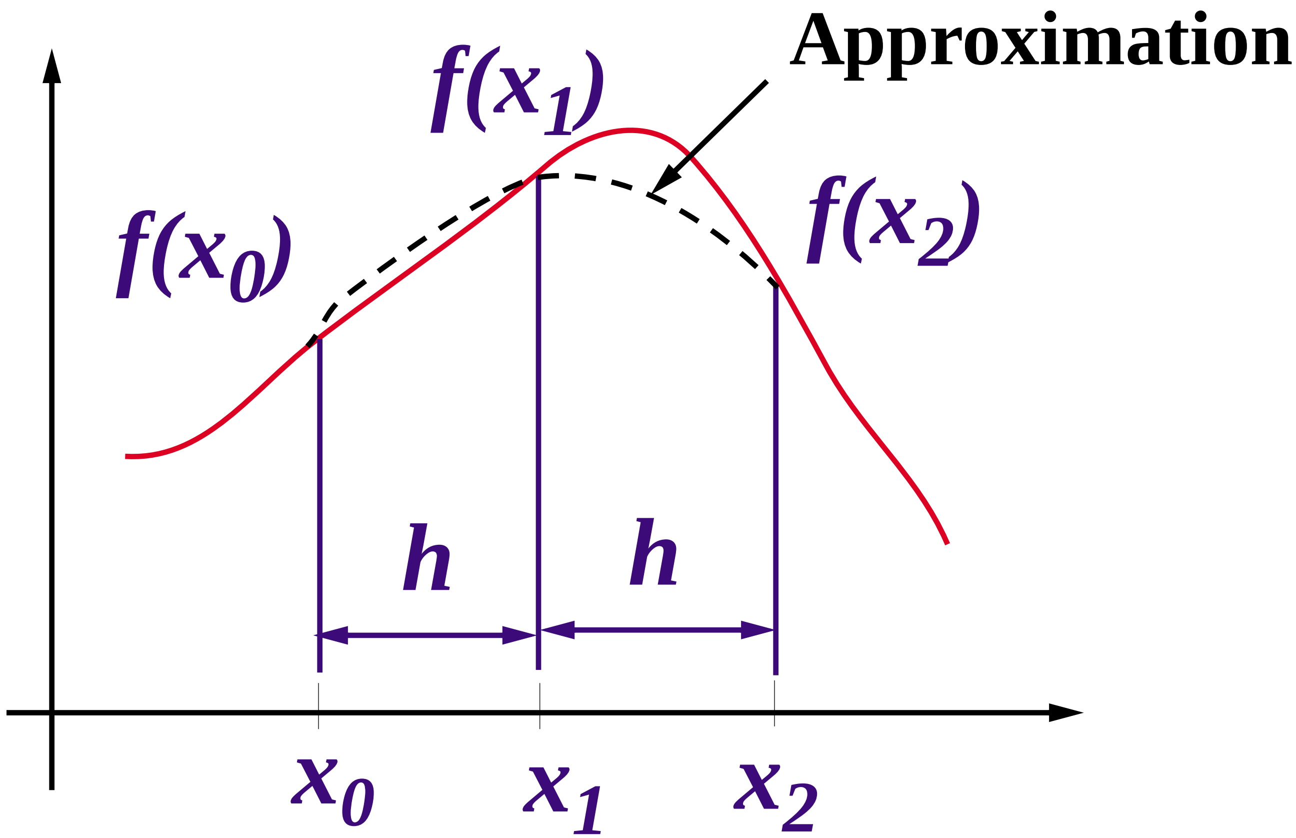
Goal is to evaluate definite integrals of the form:

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

Several integration rules are possible:

- Trapezoidal
- Simpson’s rule
- Newton-Cotes

Simpson's Rule



Simpson's Rule

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}(f_0 + f_1 + f_2) \text{ for each interval pair}$$

$$\int_a^b f(x)dx = \frac{h}{3}(f_1 + 4f_2 + 2f_3 + \ldots + f_{n+1})$$

where n is the number of pair intervals and

$$h = (b - a)/(n)$$

n is an even number of intervals.

Composite Simpson's Rule

In vector form this rule is,

$$\int_a^b f(x)dx = \frac{h}{3} c\, \boldsymbol{f}^T$$

where,

$$c = \begin{bmatrix} 1 & 4 & 2 & \ldots & 2 & 4 & 1 \end{bmatrix}$$

and $\boldsymbol{f} = \begin{bmatrix} f_1 & f_2 & f_3 & \ldots & f_{n+1} \end{bmatrix}$

Composite Simpson's Rule

Truncation error of this evaluation is approximated by (Penny and Lindfield),

$$E_t \approx (b-a)h^4 f^{IV} \frac{t}{180}$$

where, $a \leq t \leq b$

Matlab Built-in Functions

Matlab uses Newton-Cotes numerical techniques

Use higher degree polynomials (*n*th order)

$$\int_a^b f(x)dx = \frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3)$$

Newton-Cotes formula (*n*=3)

Truncation error is, $\frac{3h^5}{80} f^{IV}(t)$ where, $a \le t \le b$

Matlab Function 'Quad'

quad('func',a,b)

% 'func' is the function to be integrated
% a and b are the lower and upper limits of integration

- Uses a 2-panel, adaptive recursive Newton Cotes integration method
- Good compromise in accuracy and speed

Example of 'Quad' Function

```
% Matlab quad function use
%
t=clock; flops(0);

quadeval = quad('fsim',0,1.0)    % invokes function

fprintf('Integral value %15.8f\n',quadeval)
fprintf('\ntime = %4.2f ...
     seconds flops = %6.0f\n',etime(clock,t),flops);
```

```
Integral value      0.33333799
time = 0.42 seconds        flops =   2969
```

Differential Eqn. Background

Matlab offers several procedures and built-in functions to address these methods:

- Standard ODE solvers
- Stiff ODE solvers

Differential Equations

We want to solve dynamic systems of the form,

$$\frac{df}{dt} = f(y, t)$$

Use a Taylor series expansion,

$$y(t_0 + h) = y(t_0) + y'(t_0)h + y''(\Phi)\frac{h^2}{2}$$

The term $y''(\Phi)\frac{h^2}{2}$ is the reminder (includes all others)

Euler Method

Simplest of all methods of solving an ODE

Considers two terms in Taylor series expansion

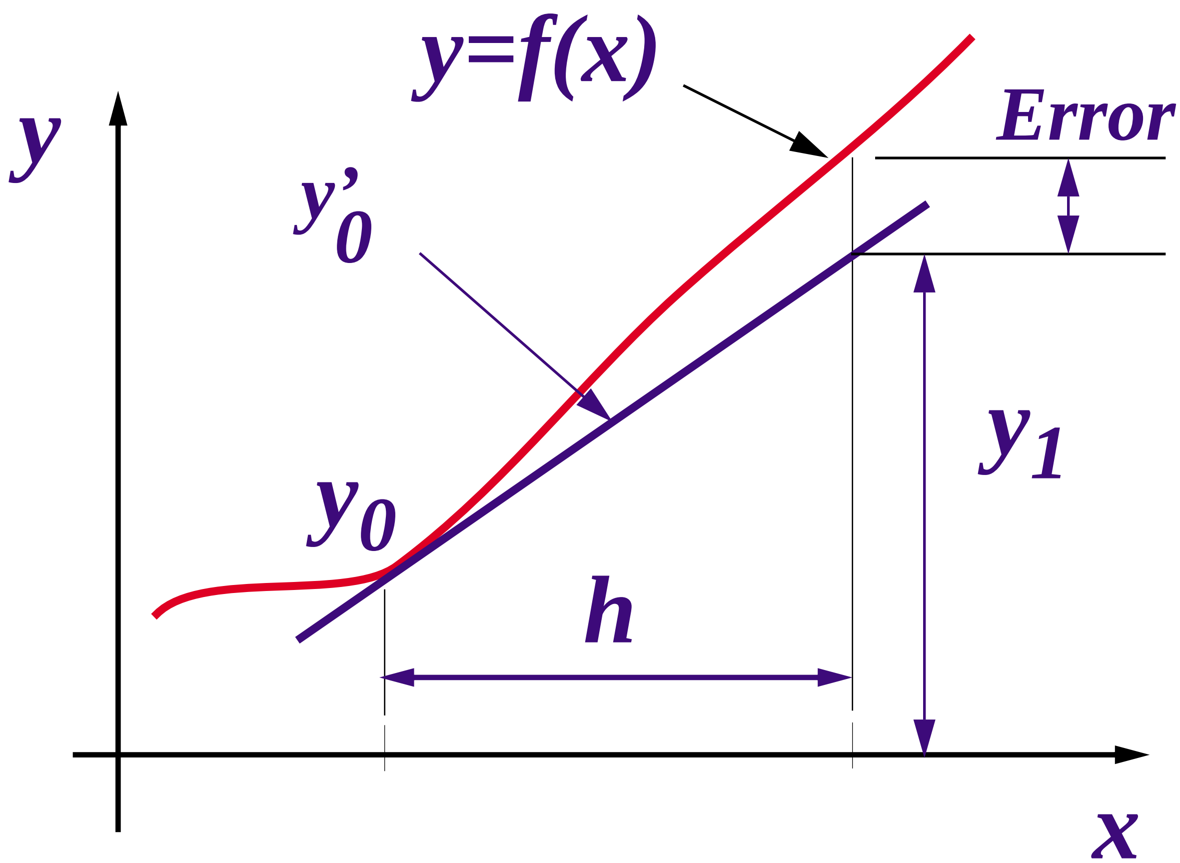
Most innacurate of all

$$y(t_0 + h) = y(t_0) + y'(t_0)h$$

In general for any n interval of solution,

$$y_{n+1} = y_n + hy'_n \text{ for } n = 0, 1, 2, \ldots \text{ error } \propto h^2$$

Geometric Interpretation



Matlab Functions

Runge Kutta Methods

Define various intermediate functions:

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf(t_n + h/2, y_n + k_1/2)$$

$$k_3 = hf(t_n + h/2, y_n + k_2/2)$$

$$k_4 = hf(t_n + h, y_n + k_3)$$

$$y_{n+1} = y_n + (k_1 + 2k_2 + 2k_3 + k_4)/6 \text{ error } \infty h^4$$

Matlab Function ‘ode’

```
[t,y] = ode23('func',tspan,y0);     % low order method

[t,y] = ode45('func',tspan,y0);    % med. order method

[t,y] = ode113('func',tspan,y0);   % var. order method

% ‘func’ is the function to be integrated
% tspan is a vector with lower and upper limits of
                    integration
% y0 is the initial value of the state variables
```

Matlab Function ‘odexxs’

```
[t,y] = ode23s('func',tspan,y0);     % stiff low order

[t,y] = ode45s('func',tspan,y0);    % stiff med. order

[t,y] = ode113s('func',tspan,y0);   % stiff var. order

% ‘func’ is the function to be integrated
% tspan is a vector with lower and upper limits of
                    integration
% y0 is the initial value of the state variables
```

What is a Stiff ODE?

Those whose rate variables display very rapid changes over time

Many systems of differential equations display this behavior

- A fast rate vs a slow varying one
- A very fast rate of change

In most systems modeling and analysis stiff system do not pose a problem.

Solution of Differential Equations in MATLAB

There are few steps needed to solve ODE in MATLAB:

1) Write the differential equation(s) as a set of first order ODEs

2) Perform necessary variable substitutions and write a MATLAB function to compute the derivatives of the state variables

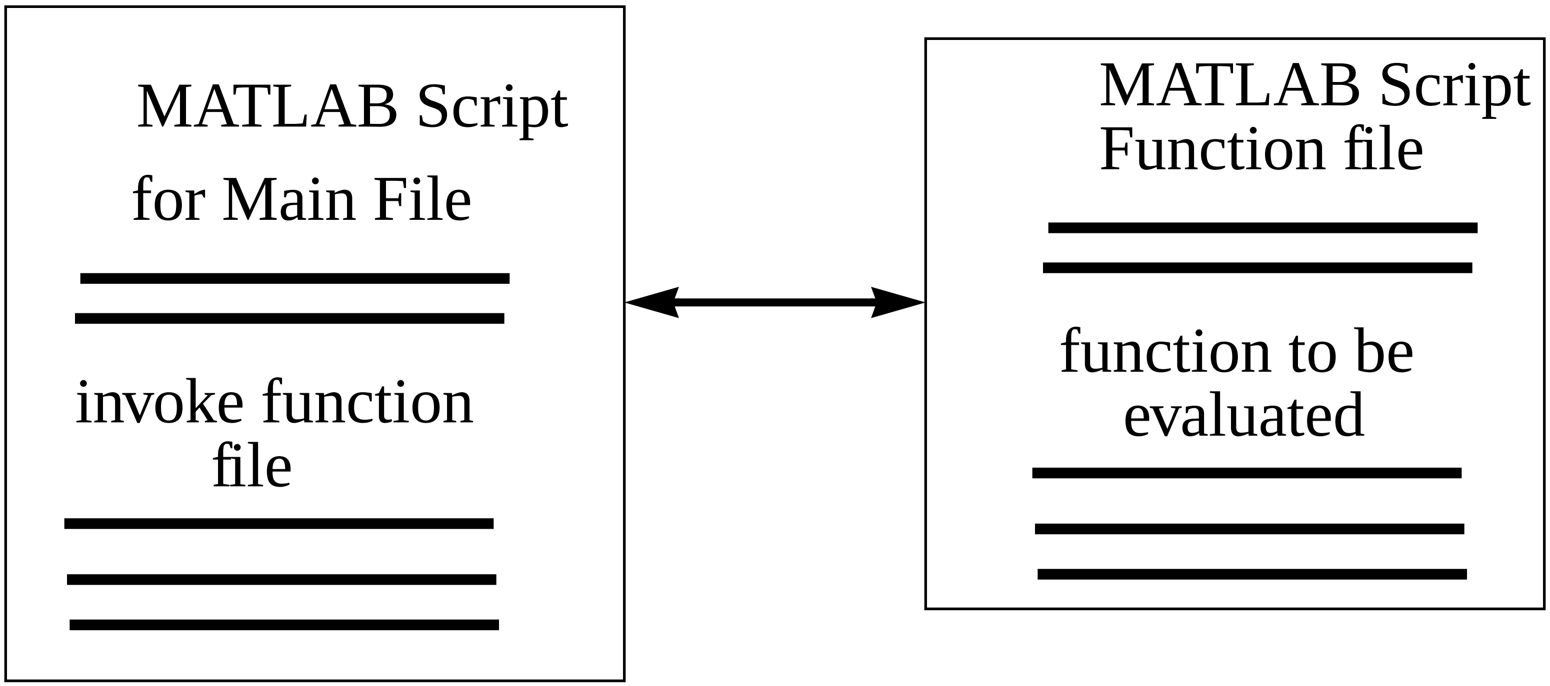
This function returns the derivatives of every state of the system

3) Use anyone of the MATLAB ODE solvers and invoke the function

MATLAB Scripting Approach

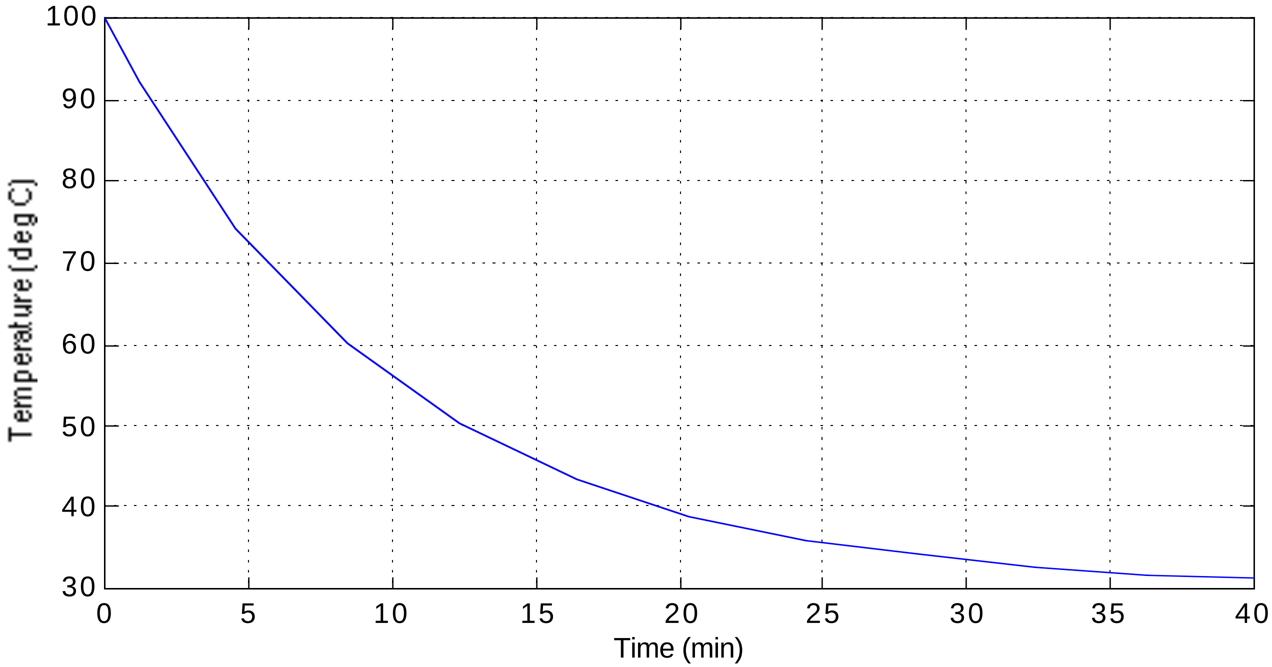
The system is represented by ODE

Create two M files: a) a main file and b) a function file



Sample Experiment

Suppose that we would like to decsribe the process of cooling of water from near boiling point to room temperature. The figure shows our observations.



First Law of Cooling ODE

Observations:

- The temperature drops very quickly initially
- The temperature decay (rate of change) tapers as the water and room temperatures get closer
- The temperature approaches to the room temperature as time goes to infinity

Write down possible solutions or forms of the solution

Proposed Model

Suppose the model is of the form,

$$\frac{dT}{dt} = -H(T - T_a)$$

where:

H is a constant of proportionality in the experiment

T is the temperature of the water (deg C)

T_a is the room temperature (deg C)

Step 1 in ODE Solution

1) Write the differential equation(s) as a set of first order ODEs

This is already in place since the system has only one ODE to start

$$\frac{dT}{dt} = -H(T - T_a)$$

Step 2 in ODE Solution

2) Perform necessary variable substitutions and write a MATLAB function to compute the derivatives of the state variables

This function returns the derivatives of every state of the system

In this case we write two M-files:

1) one initializes the problem (state variable definition at time zero)

2) one function to computer the derivative of T (temperature)

MATLAB Equations (Main Routine)

```
% Define Initial Conditions of the Problem
global Ta H          % define global variables

To = 100;   % To is the initial temperature of the water
to = 0.0;   % to is the initial time to solve this equation
tf = 40;    % tf is the final time (min)
Tspan = [to  tf];% Spanning time for the ODE solution

% Define T ambient (Ta) and cooling constant (H)
Ta = 30;    % ambient temperature (deg C)
H = 0.10;   % Cooling constant (1/min)
```

Step 3 in ODE Solution

3) Invoke the ODE solver in MATLAB

```
% Use Runge-Kutta 3rd order solver
[t,T] = ode23('ftem',Tspan,To);

% Plot the results of the numerical integration procedure

plot(t,T)
xlabel('Time (min)')
ylabel('Temperature (deg C)')
grid
```

MATLAB Function 'ftem.m'

This function estimates the value of the rate of change of the ODE.

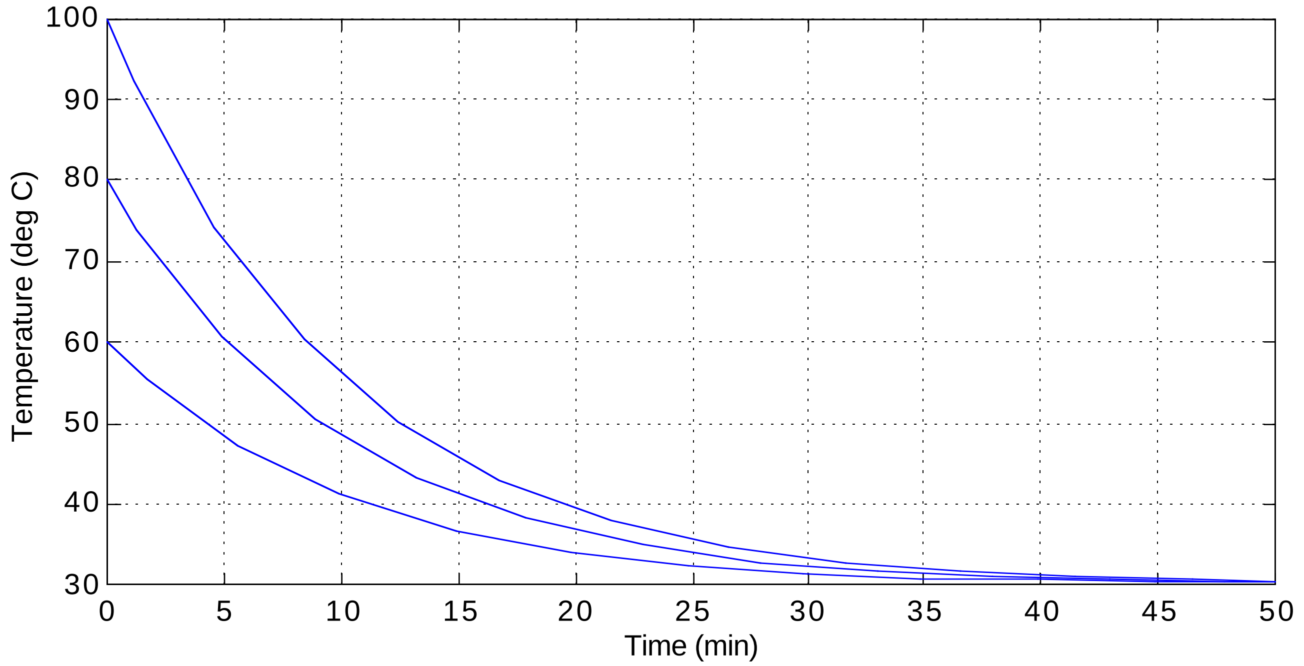
```
% First Order Differential Equation Function

function tprime = ftem(t,T)
global Ta H
tprime = - H * (T - Ta);
```

Note: global variables are "shared" by all functions in the workspace

Use of the Hold Command

Here we use the hold command to plot two solutions to the first order differential equation shown previously



Higher-Order Dynamic Systems

Higher order system can be solved in a simular way using MATLAB recognizing that array variables that contain more than one state variable

- The following highway maintenance example illustrates this (adapted from Drew, 1997)
- The highway maintenance example solves three coupled ODEs to predict the state of the State highways system
- The model assumes investments in ordinary vs.replacement maintenance actions to predict the number of lane-miles of highway in three possible states over time (sufficient,deficient and deteriorating highways)

Highway Maintenance Model (main file)

```
% Highway Maintenance Model
global HME  FEOM OMC FEMR MRC HDETT HAT
% Define constants of the problem
HME = 5E7;  % Hwy maintenance expenditure ($/yr)
FEOM = 0.5;          % Fract. of expenditures to ordinary
                     maint (%)
OMC = 5E5;           % Ordinary maintenance cost($/lane-
                     mile)
MRC = 2E6;           % Maintenance replacement action ($/
                     la-mi)
FEMR = 0.5;          % Frac of expenditures for maint.
                     replacement (%)
HAT = 4;             % Hwy aging time (yr)
HDETT = 8;           % Hwy deterioration time (yr)
```

Highway Maintenance Model (main file)

```
% Define Initial Conditions of the Problem

yN = [200 200 0];% yN defines intial conditions for...
                  state variables
to = 0.0;         % to is the initial time to solve this...
                  equation (yr)
tf = 10.0;        % tf is the final time (yr)
tspan = [to tf]

% Invoke the ordinary differential equation solver
[t,y] = ode23('fhwy3_rev',tspan,yN);
```

Highway Maintenance Model (main file)

```
% Plot the results of the numerical integration procedure
subplot(3,1,1)      % plots PSH in the top half of the...
                    page

plot(t,y(:,1))      % plots all elements of the first...
                    column of y
xlabel('Time (years)')
ylabel('PSH (la-mi)');
grid
```

Highway Maintenance Model

```
subplot(3,1,2)      % plots I in the bottom half of the page
plot(t,y(:,2))      % plots all elements of the second
                    column of y
xlabel('Time (years)')
ylabel('PDTH (la-mi)');
grid

subplot(3,1,3)      % plots PDTH in the bottom third of
                    the page
plot(t,y(:,3))      % plots all elements of the first column
                    of y
xlabel('Time (years)')
ylabel('PDFH (la-mi)')
grid
```

Function File (fhwy3_rev)

```
function yprime = fhwy3_rev(t,y)
global HME  FEOM OMC FEMR MRC HDETT HAT

% define rate equation(s)
HD = y(2) / HDETT;   % Hwy deteriorating (lane-mi/yr)
HA = y(1) / HAT;       % Hwy aging (lane-mi/yr)

HOM = HME * (FEOM / OMC);
% Highway with ordinary maintenance (lane-mi/yr)

HMR = HME * (FEMR / MRC);
% Highway with maint replacement (lane-mi/yr)
```

Function File (fhwy3_rev)

% Define the rate equations (3 rate variables representing PSH, PDFH and PDTH)
%
% PSH - Physically sufficient highways (y1)
% PDFH - Physically defficient highways
% PDTH - Physically deteriorated highways

% Model equivalencies for state variables

% y1 = PSH
% y2 = PDFH
% y3 = PDTH

Function File (fhwy3_rev)

```
yprime(1) = HOM + HMR - HA;
% Rate of change of PSH (la-mi/yr)

yprime(2) = HA - HD - HOM ;
% Rate of change of PDFH (la-mi/yr)

yprime(3) = HD - HMR;
% Rate of change of PDTH (la-mi/yr)

yprime=yprime';   % returns a column vector to main file
```

Sample Output of the Highway Maintenance Model