CEE 3804

Advanced MATLAB Functions

Antonio A. Trani
Virginia Tech

Spring 2023

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:

$$f = x^2 + 3x + 1 \text{ or } f = [1 \quad 3 \quad 1]$$

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,

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

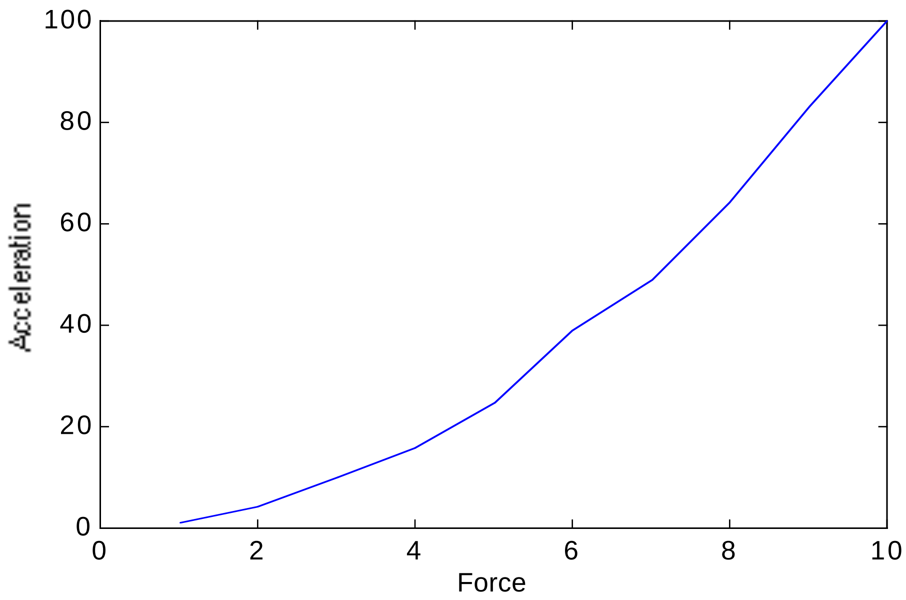
by polynomial $f = x^2 + 3x + 1$ (both have been defined)

```
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

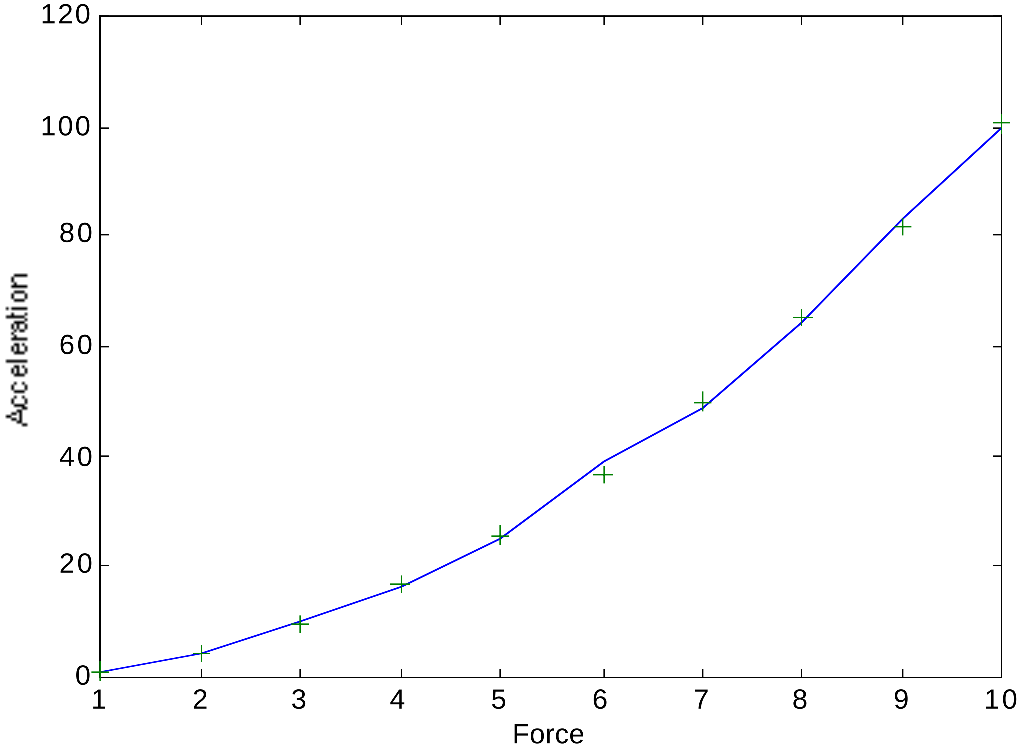
Create a new vector (xnew) with values to be evaluated

```
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

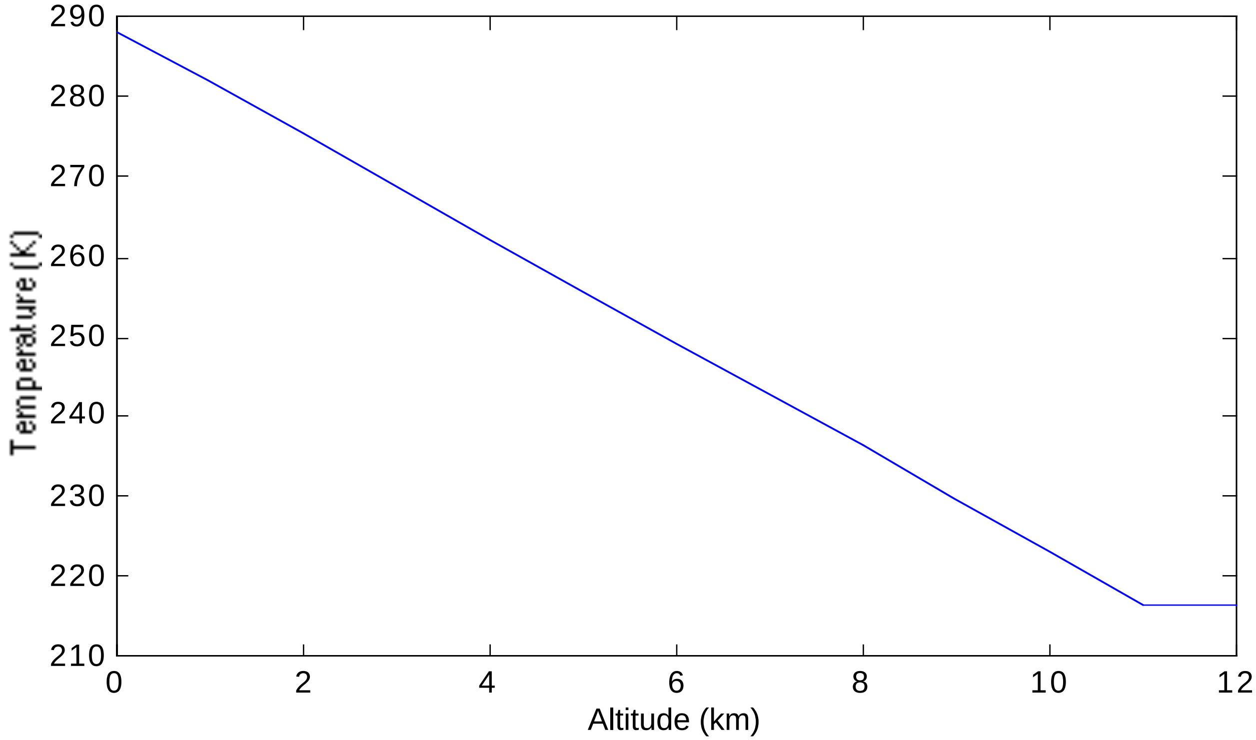
Several interpolation functions exist to facilitate data handling.

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

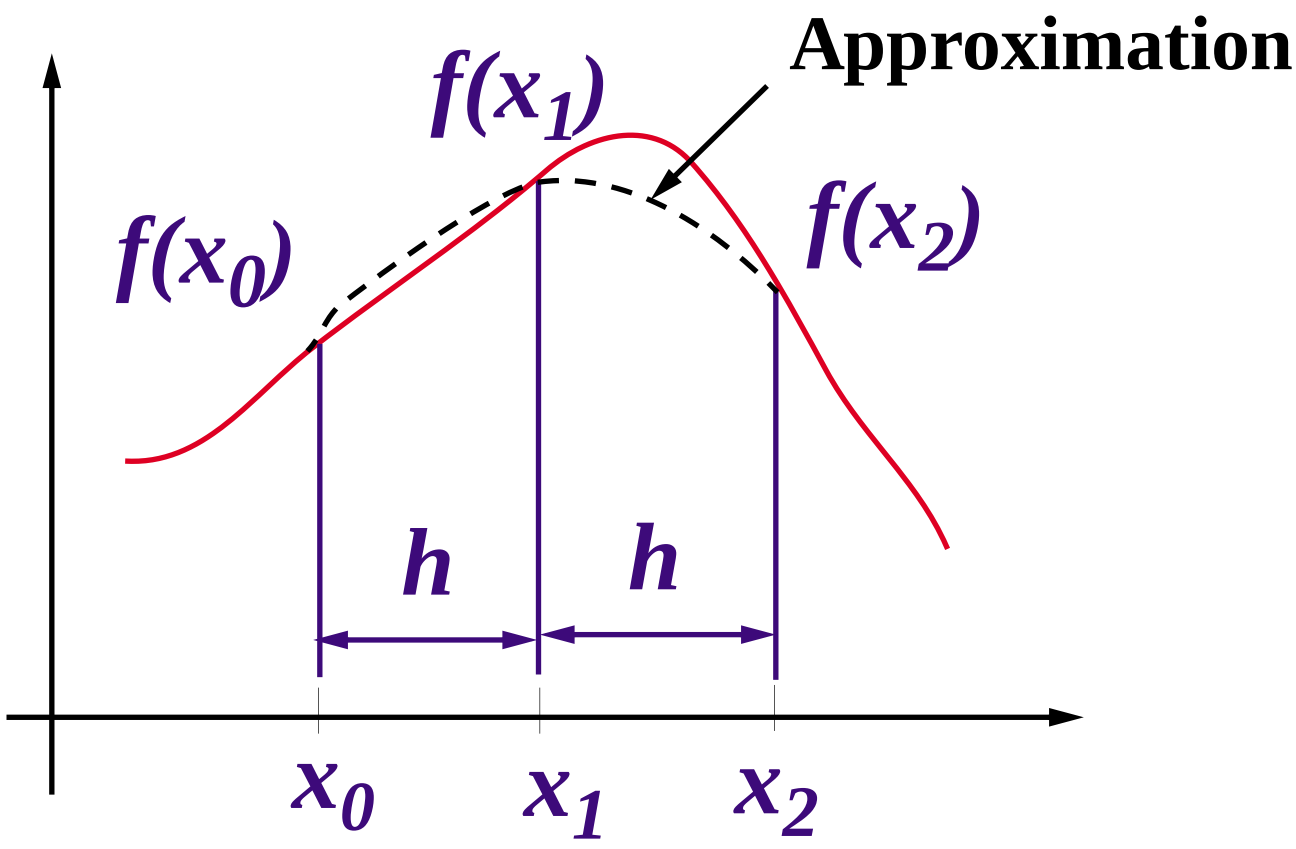
Goal is to evaluate definite integrals of the form:

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

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, $a \leq t \leq b$

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
```

Sample Numerical Integration

Runoff in Civil Engineering Applications

Problem 1

A civil engineer is designing a rainstorm water management system for a shopping mall. During a severe thunderstorm, the water runoff generated by the large parking lot at the shopping mall is given by the function:

$$runoff = k_2 + k_1 \sin(t / k_3) e^{(-t/k_4)}$$

Where *runoff* is the runoff volume (cubic meters per second) generated by the parking lot, *t* is the time (in seconds) after the thunderstorm starts and k_1 through k_4 are parameters of the runoff function.

Task 1

Create a Matlab function to calculate the runoff for a given value of time t. As part of the input variables to the function runoff, include the four input parameters k1 through k4 for a 100 year storm with numerical values as follows:

k1 = 50;
k2 = 2;
k3 = 1500;
k4 = 800;

All infrastructure generates runoff (examples: parking lots, runways at airports, large structures)

Function to Calculate Runoff

```
________________________ function_Runoff_rev.m function _________

% Function to estimate runoff volume at a shopping mall

function [runoff] = function_Runoff_rev(t)

global k1 k2 k3 k4

% runoff is the runoff volume (cubic meters per second)
% generated by the shopping mall  into the ponding area
% t is the time (in seconds) after the thunderstorm starts

runoff = k1 * sin(t/k3) .* exp(-t/k4) + k2;
```

Runoff Function

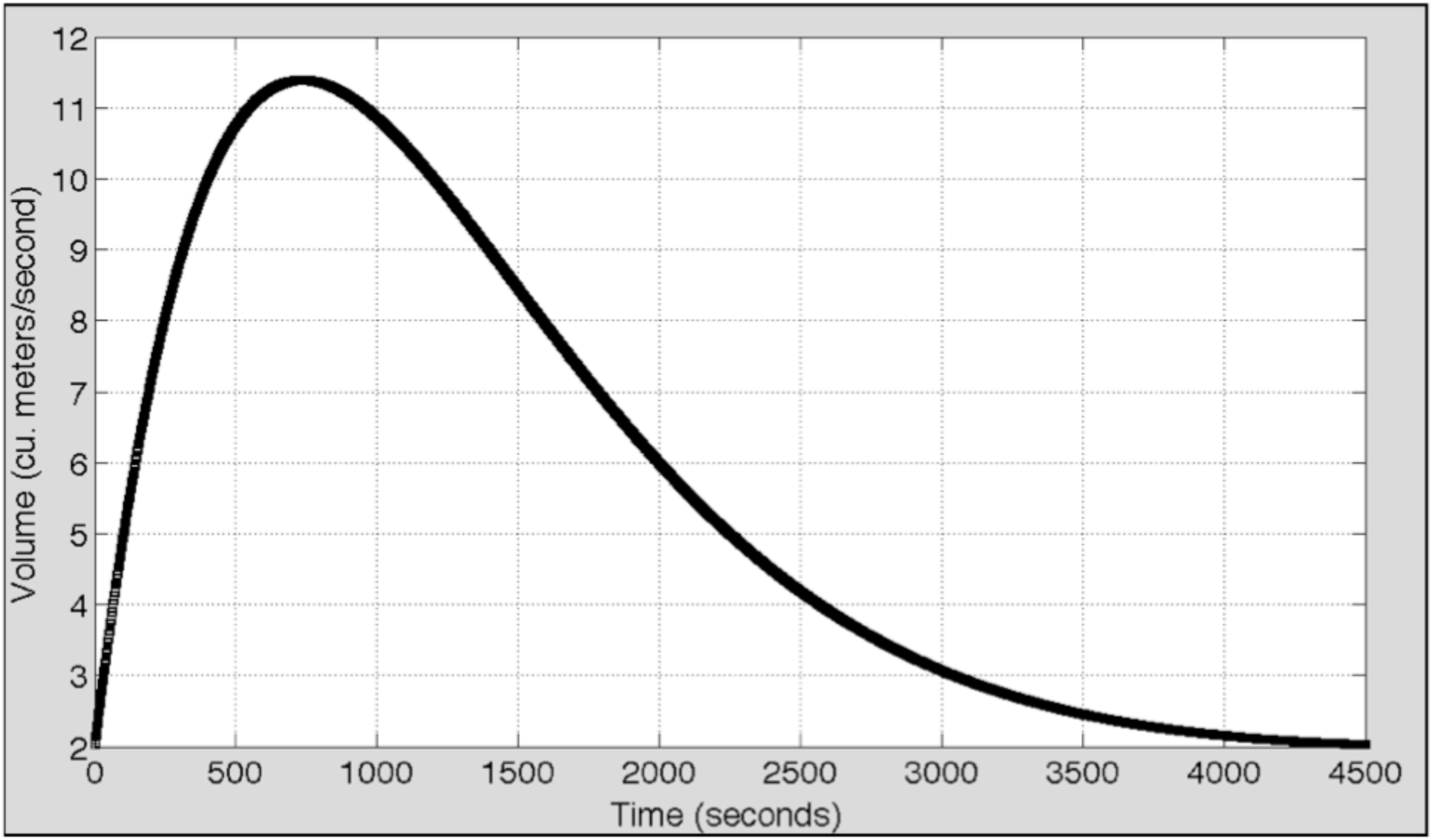


Figure 1. Runoff Function.

Script to Calculate the Area Under the Curve

```
% Script to calculate the area under the runoff function
% Programmer: T. Trani

global k1 k2 k3 k4

% Calls function function_Runoff_rev

% areaUnderRunoff is the runoff volume (cubic meters per second)
% generated by the shopping mall  into the ponding area
% t is the time (in seconds) after the thunderstorm starts

k1 = 50;         % multiplicative parameter of the function
k2 = 2;            % additive parameter of the function
k3 = 1500;         % parameter of sinusoidal term
k4 = 800;          % parameter of exponential term

tLast = 4500;      % final time to do the calculation (seconds)

% Estimate the area under runoff function

areaUnderRunoff = quad('function_Runoff_rev',0,tLast);

disp(['Area under the Curve is   ',num2str(areaUnderRunoff)])
```

Calculations in Matlab

- Find the volume of water generated in the design thunderstorm (say 100 year event)

Area under the curve is the volume of water going into a pond

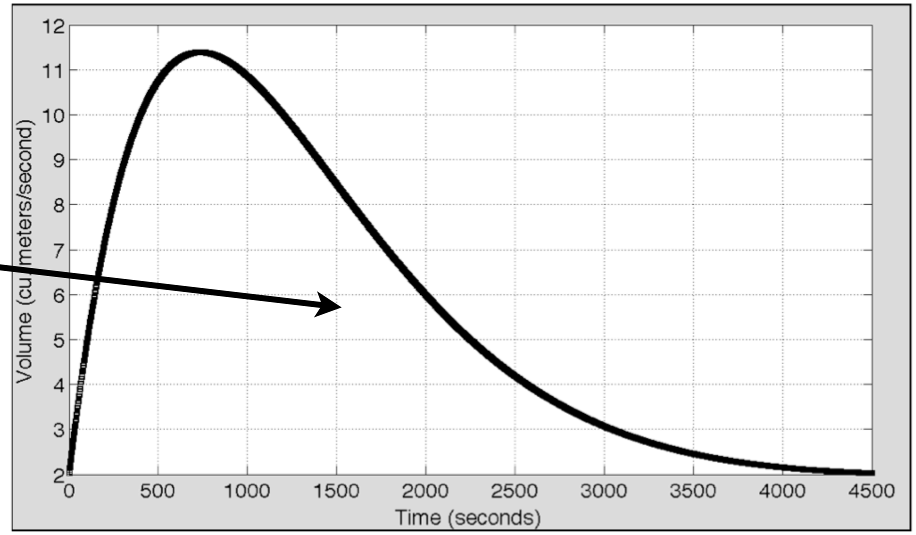


Figure 1. Runoff Function.

Area under the Curve is 25652.45 cubic meters

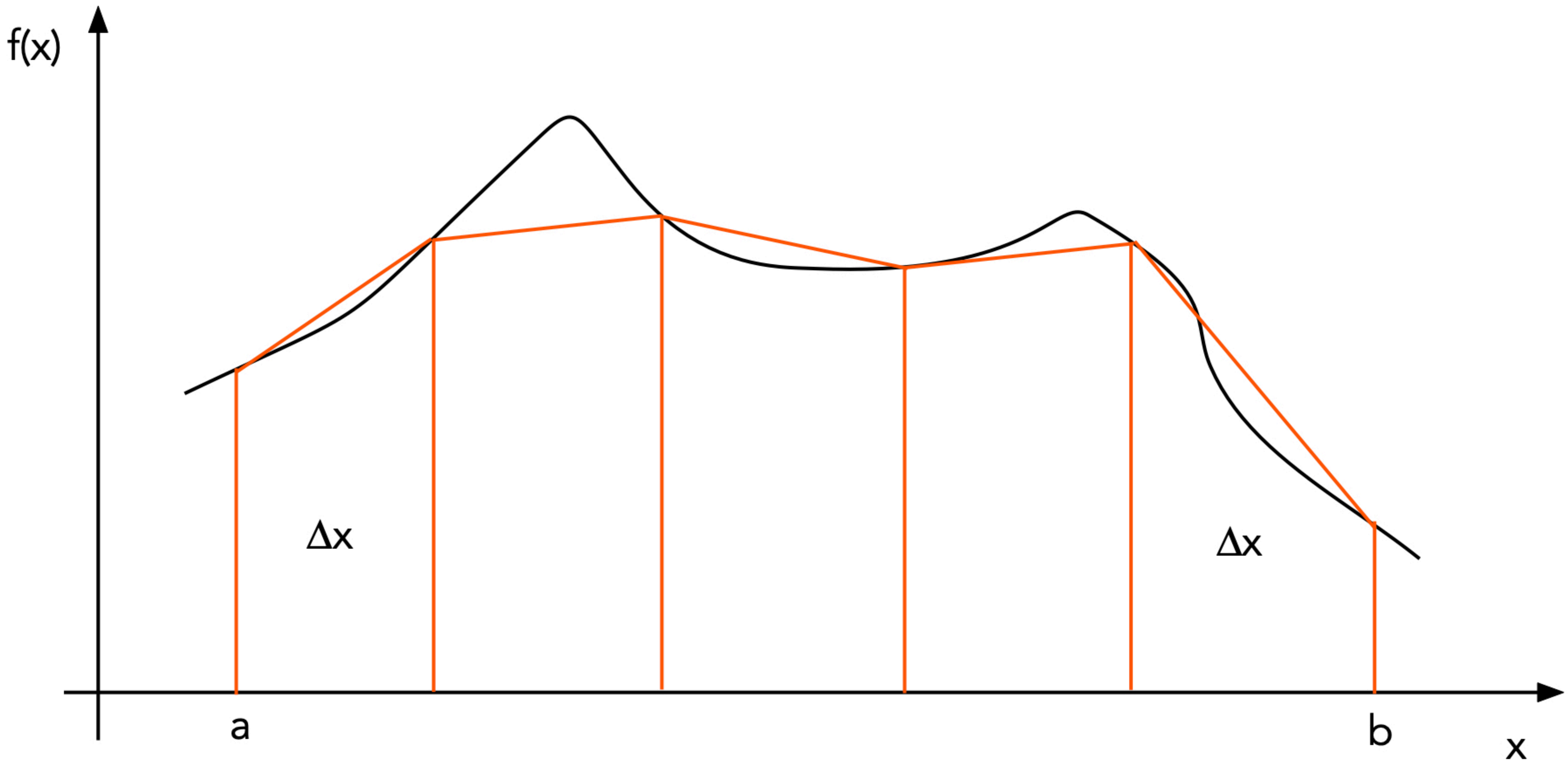
Task 4

Estimate the dimensions of a ponding volume needed to store all the runoff volume generated by the 100 storm event. State dimensions of the ponding volume (base x width x height).

One estimate of the ponding volume would be 100 x 100 x 2.57 meters. This is equivalent to two Football stadiums with a depth of 2.27 meters (8.5 feet).

Trapezoidal Rule

- Approximates the function f(x) using small trapezoids spanning the range between a and b
- The accuracy improves when the interval size (Δx) is small



Example: Runoff Calculation Using the Trapezoidal Rule

```
% Script to generate values for runoff calculations
% A. Trani

% Given a vector of time values (t) in seconds
% Given four coefficients k1, k2, k3 and k4

% Calculate runoff (cu. meters/second) using the following equation:
% runoff =k2 * k1*sin(t/k3) .* exp(-t/k4) ;

clear all
close all
clc

t=0:1:4500;                          % time vector (seconds)

% Define constants for the problem
k1 = 50;
```

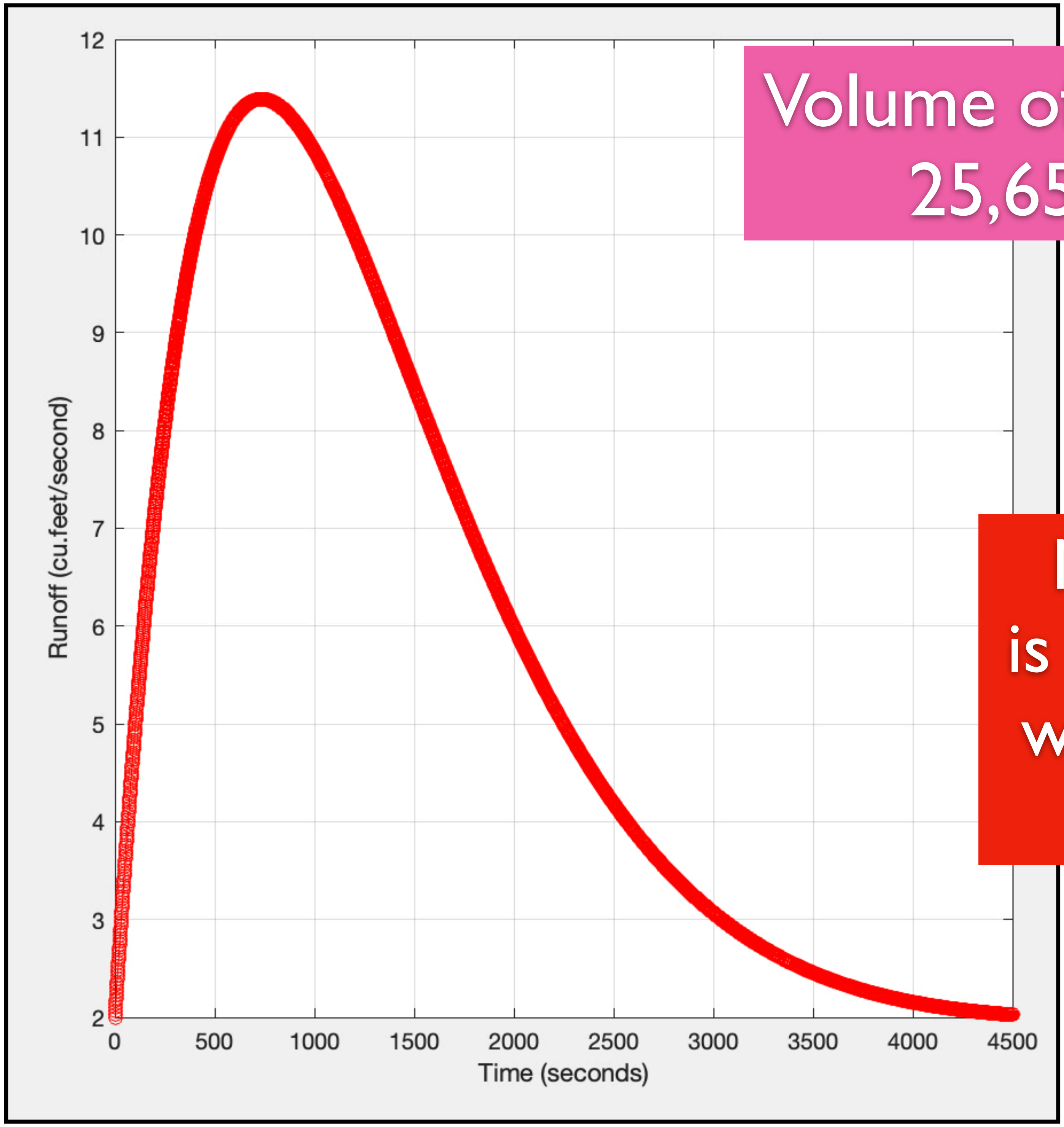
Generates values of parameter t (time) from t=o to t=4500 seconds

Example: Runoff Calculation Using the Trapezoidal Rule (2)

```
k2 = 2;
k3 = 1500;
k4 = 800;

runoff =k1 *sin(t/k3) .* exp(-t/k4) + k2;   % runoff (cu.meters/second)

plot(t,runoff,'o-r')
xlabel('Time (seconds)')
ylabel('Runoff (cu.feet/second)')
grid

% Use the trapezoidal function to evaluate the area under the curve

volumeOfWater = trapz(t,runoff);

disp(['Volume of water accumulated is  ', num2str(volumeOfWater)]);
```

Generates values of runoff from t=o to t=4500 seconds

Example: Runoff Calculation Using the Trapezoidal Rule (3)



Volume of water accumulated is 25,652.45 cubic meters

Note: This calculation is very accurate because we defined a very small interval (Δx)

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

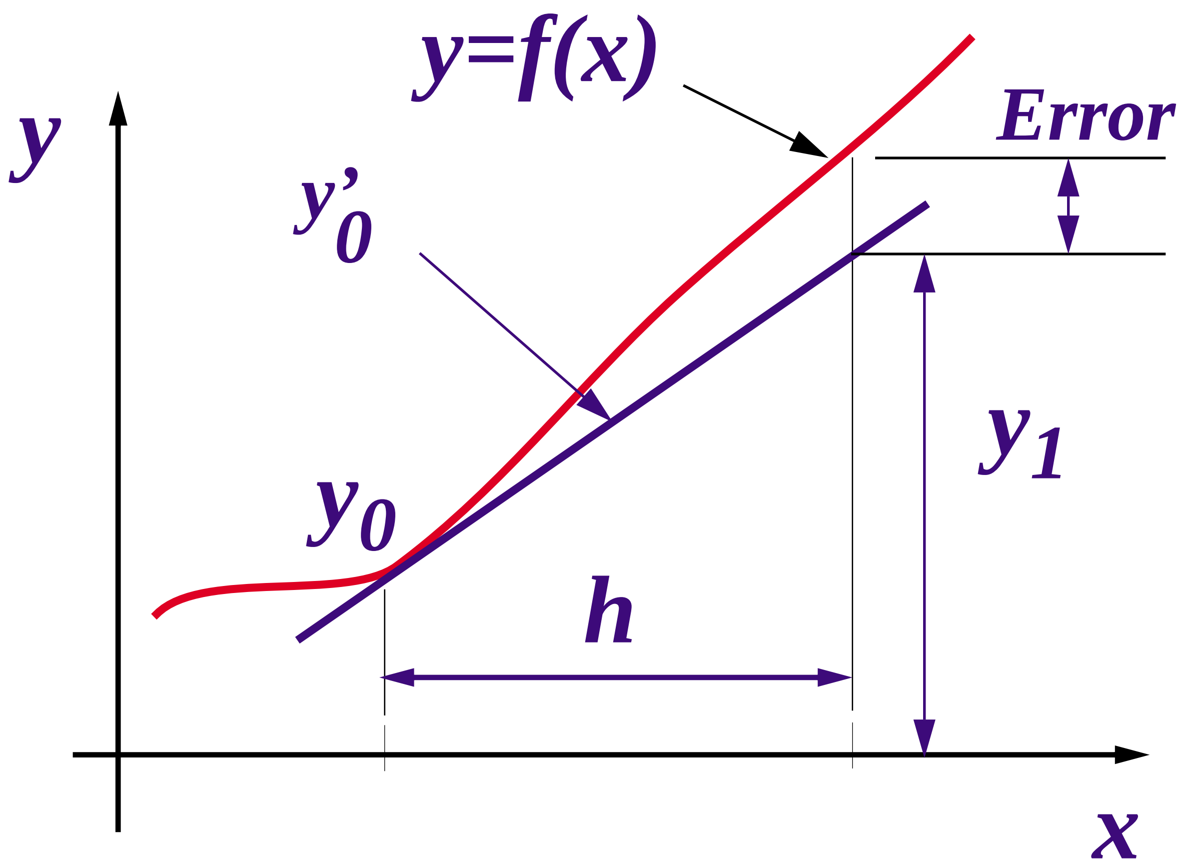
Most innacurate of all

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

In general for any n interval of solution,

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

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

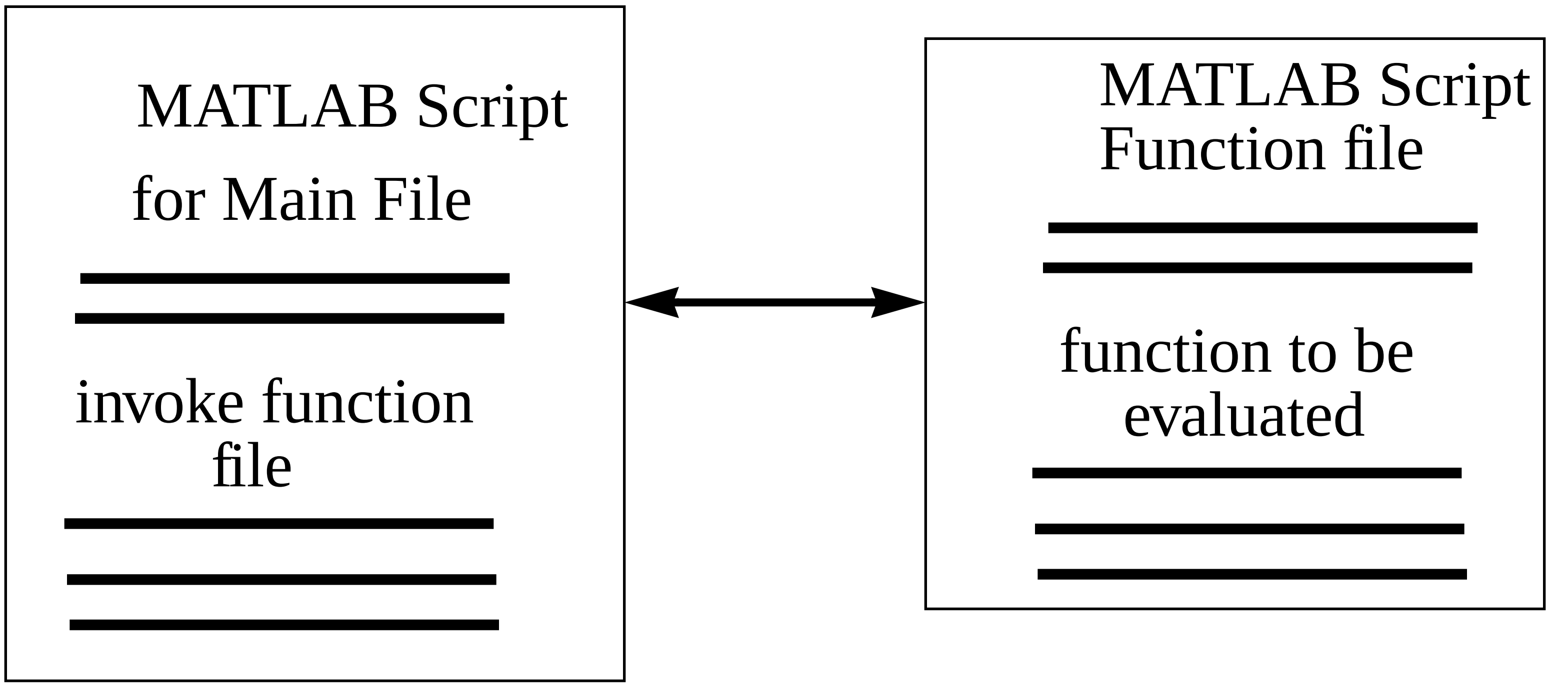
This function returns the derivatives of every state of the system

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

MATLAB Scripting Approach

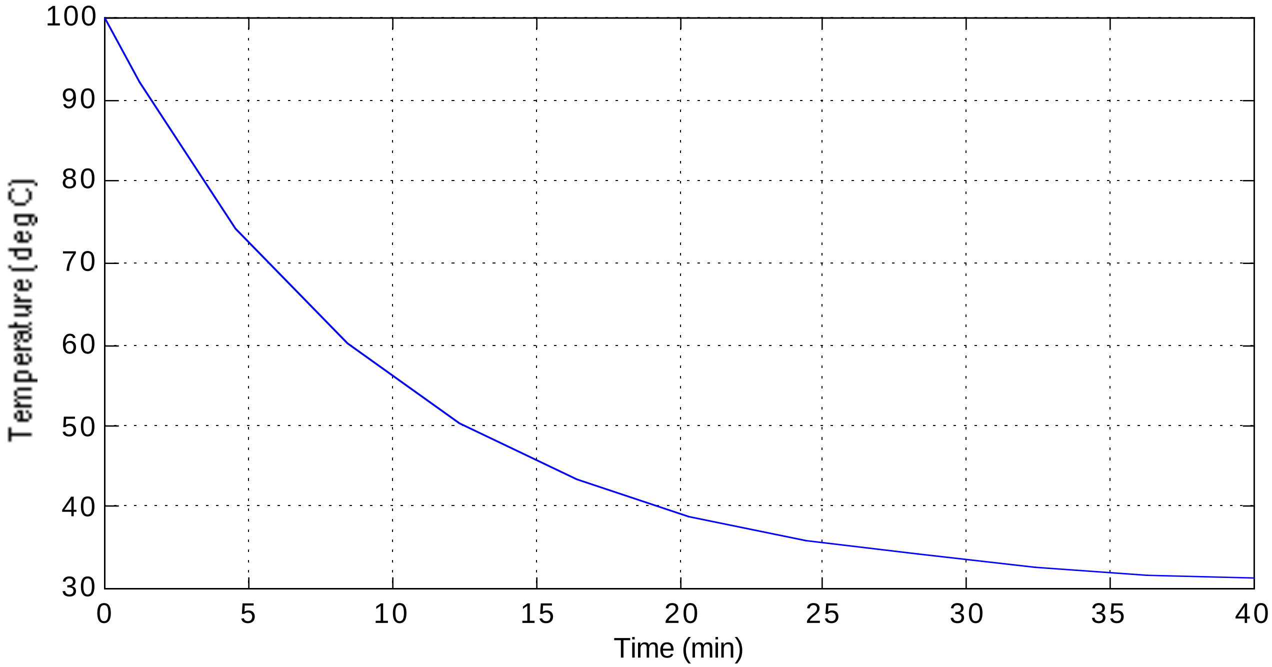
The system is represented by ODE

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’

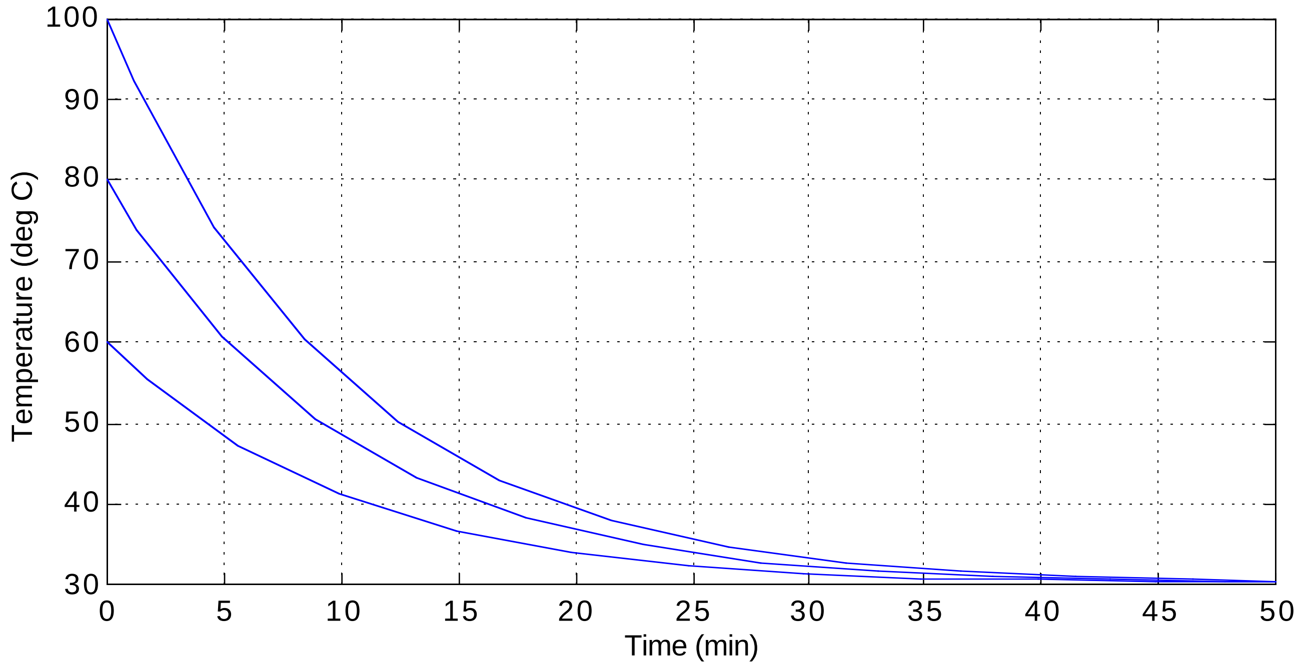
This function estimates the value of the rate of change of the ODE.

```
% 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



Example: Use of ODE Solvers
Train Kinematics
2nd Order Dynamic System

Vehicle Kinematics

- An engineer collects data during the certification of the new high-speed train to be introduced in the Northeast Corridor in the United States
- The data collected records train acceleration (a) vs. velocity (V)
- The data is presented in the table

Train Velocity (m/s)	Maximum Train Acceleration (m/s^2)
0.00	2.1
20	1.56
30	1.30
40	1.06
50	0.76
60	0.51
80	0.00

Vehicle Kinematics (2)

- Use a Matlab script to find the best first-order polynomial that fits the acceleration vs. train speed data (i.e., use the "polyfit" command)
- The resulting polynomial will be of the form:

$$\frac{dV}{dt} = A + BV \qquad \text{Equation (1)}$$

- where A, B and C are the polynomial coefficients found and V is the train speed.

Train Velocity (m/s)	Maximum Train Acceleration (m/s^2)
0.00	2.1
20	1.56
30	1.30
40	1.06
50	0.76
60	0.51
80	0.00

VirginiaTech Invent the Future

Vehicle Kinematics (3)
Matlab Script to Find Best Polynomial

```
% Script to estimate best curve fit for two vectors
clear
clc

% T. Trani

% Task 1

% Define two vectors for velocity and acceleration

velocity       = [0 20 30 40 50 60 80] ;                    % velocity in m/s
acceleration  = [2.1 1.56 1.30 1.06 0.76 0.51 0.00];  % accceleration (m/s-s)

% Do a  basic polynomial fit

coefficients = polyfit(velocity,acceleration, 1);              % Fits a first-order polynomial

% Evaluate the polynomial found for the range of velocities of the train in
% the table

velNew = min(velocity):1:max(velocity);          % define a new velocity vector to evaluate the polynomial
accelerationFromPolyFit = polyval(coefficients,velNew);   % evaluate the polynomial using the coefficients found

% Make a plot and compare

% Create a label for the plot with the values of coefficients found

labelPlot = horzcat('Acceleration = ', num2str(coefficients(1)), ' * Velocity^2 + ', ...
    num2str(coefficients(2)), ' * Velocity + ', num2str(coefficients(3)));

figure
plot(velocity,acceleration,'or',velNew,accelerationFromPolyFit,'b--')
xlabel('Train Velocity (m/s)','fontsize',20)
ylabel('Train Acceleration (m/s-s)','fontsize',20)
title(labelPlot)
grid
```

$$\frac{dV}{dt} = A + BV$$

Train Velocity (m/s)	Maximum Train Acceleration (m/s^2)
0.00	2.1
20	1.56
30	1.30
40	1.06
50	0.76
60	0.51
80	0.00

Calculates coefficients

Regression Coefficients for Acceleration Function

$$\frac{dV}{dt} = A + BV$$

- B = -0.0268; % coefficient of acceleration function (1st power)
- A = 2.0997; % coefficient of acceleration function (constant)

Other Tasks

- Using the Matlab Ordinary Differential Equation solver ODE45, to solve numerically the differential equation (1) as a function of time
- This problem is similar to the Water Cooling problem discussed in class except that the differential equation is a little more complex
- Use as initial conditions zero for the train speed and solve numerically the speed of the train for 200 seconds
- Plot the velocity profile of the high-speed train as a function of time. How fast is the train going after 200 seconds?

More Tasks (2)

- Add code to the script and function containing the differential equation created in Task 2 to calculate the distance traveled by the train. Recall that distance (S) can be obtained from the first order differential equation:

$$\frac{dS}{dt} = V \qquad \text{Equation (2)}$$

- The solution to this problem requires solving two first order equations (1-2). Refer to the mass-spring damper system discussed in class to help you setup these equations. You can see how these two equations are coupled as follows:

$$\dot{x}_1 = \frac{dV}{dt} = A + Bx_1$$

$$\dot{x}_2 = x_1 = \frac{dS}{dt}$$

- where: x_1 be the speed of the train, x_2 be the position of the train and and be the derivatives of speed and position

Matlab Main File to Solve Problem

```
% Main file to solve two differential equations of motion
% Solution to a set of dynamic equations of the form:
%
% define the following state equations
%
% x(1) = speed (m/s)
% x(2) = position (m)

% xdot(1) = A + B *x(1) + C * x(1) .^2;   % acceleration of train
% xdot(2) = x(1);                          % velocity of train
%
% subject to initial conditions:
%
% x (t=0) = xo
%
% where:

global A B C

% Define Initial Conditions of the Problem

xo = [0 0];              % xo are the initial velocity and distance traveled
to = 0.0;                % to is the initial time to solve this equation
tf = 200;                % tf is the final time

% define the coefficients of the acceleration function (A, B, and C)

% Previously obtained as: coefficients = 0.0000   -0.0268    2.0997
% from highest order to lowest (A associated with x(1) ^2, B with x(1) and
% C if the constant term)

A = 0.0;        % coefficient of acceleration function (2nd power)
B = -0.0268;    % coefficient of acceleration function (1st power)
C = 2.0997;     % coefficient of acceleration function (constant)

tspan =[to tf];

[t,x] = ode45('trainDynamics',tspan,xo);    % call ODE solver
```

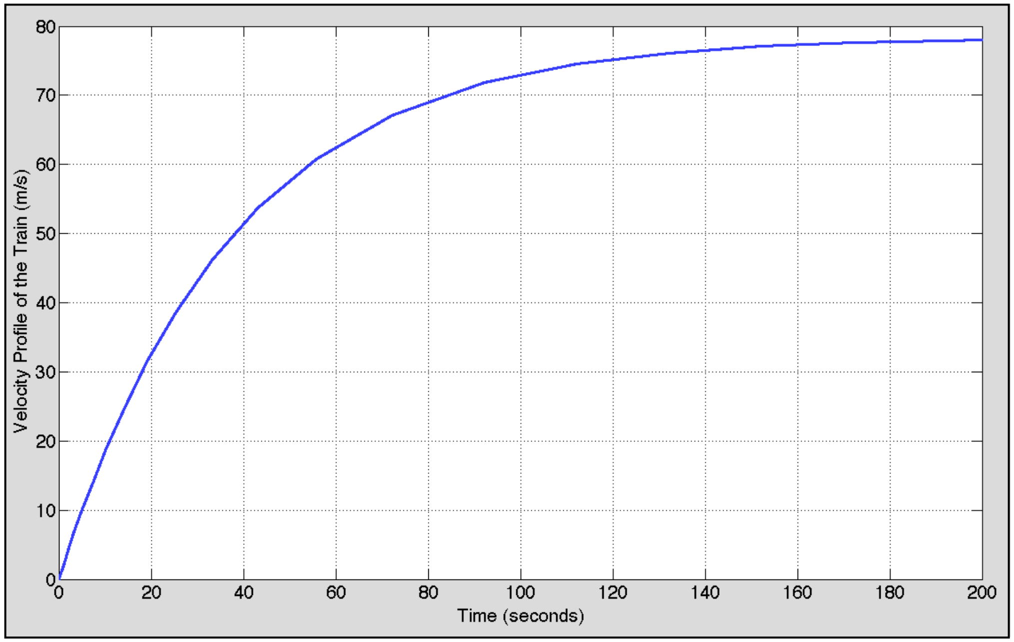
```
% Plot the results of the numerical integration procedure

figure
plot(t,x(:,1))
xlabel('Time (seconds)','fontsize',20)
ylabel('Velocity Profile of the Train (m/s)','fontsize',20)
grid


figure
plot(t,x(:,2))
xlabel('Time (seconds)','fontsize',20)
ylabel('Distance Traveled (m)','fontsize',20)
grid
```

Matlab Function File (trainDynamics.m)

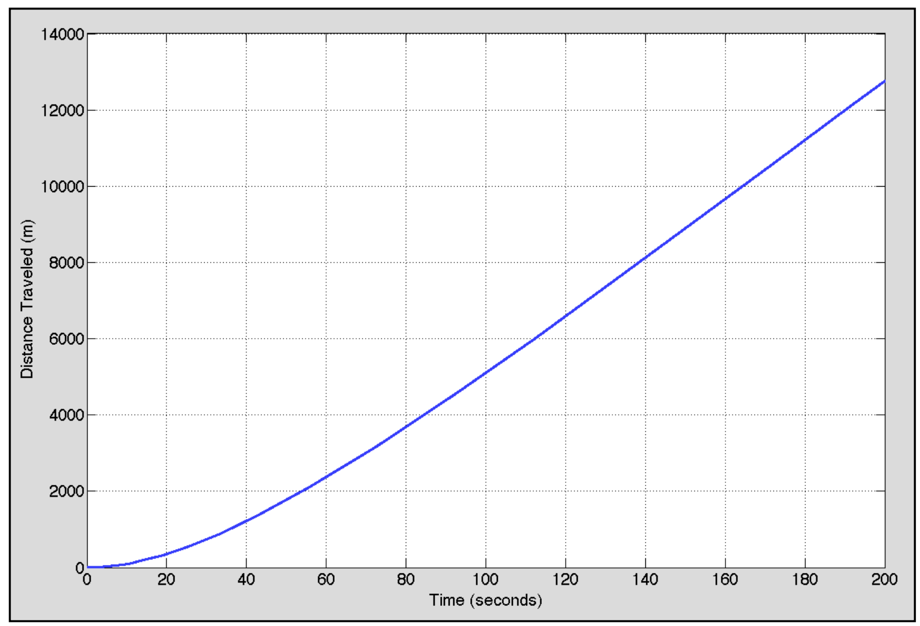
```
% Two first-order DEQ to solve two equations of motion for the train

function xdot = trainDynamics(t,x)

global A B C

% define the rate equations
%
% x(1) = speed (m/s)
% x(2) = position (m)

xdot(1) = A *  x(1) .^2 + B *x(1) + C ;
xdot(2) = x(1);

xdot = xdot';
```

Note: I setup the problem for a quadratic model but the coefficient of x^2 is zero. The acceleration is linear with speed.

Matlab Main File Output



The train travels at 78 m/s after 200 seconds

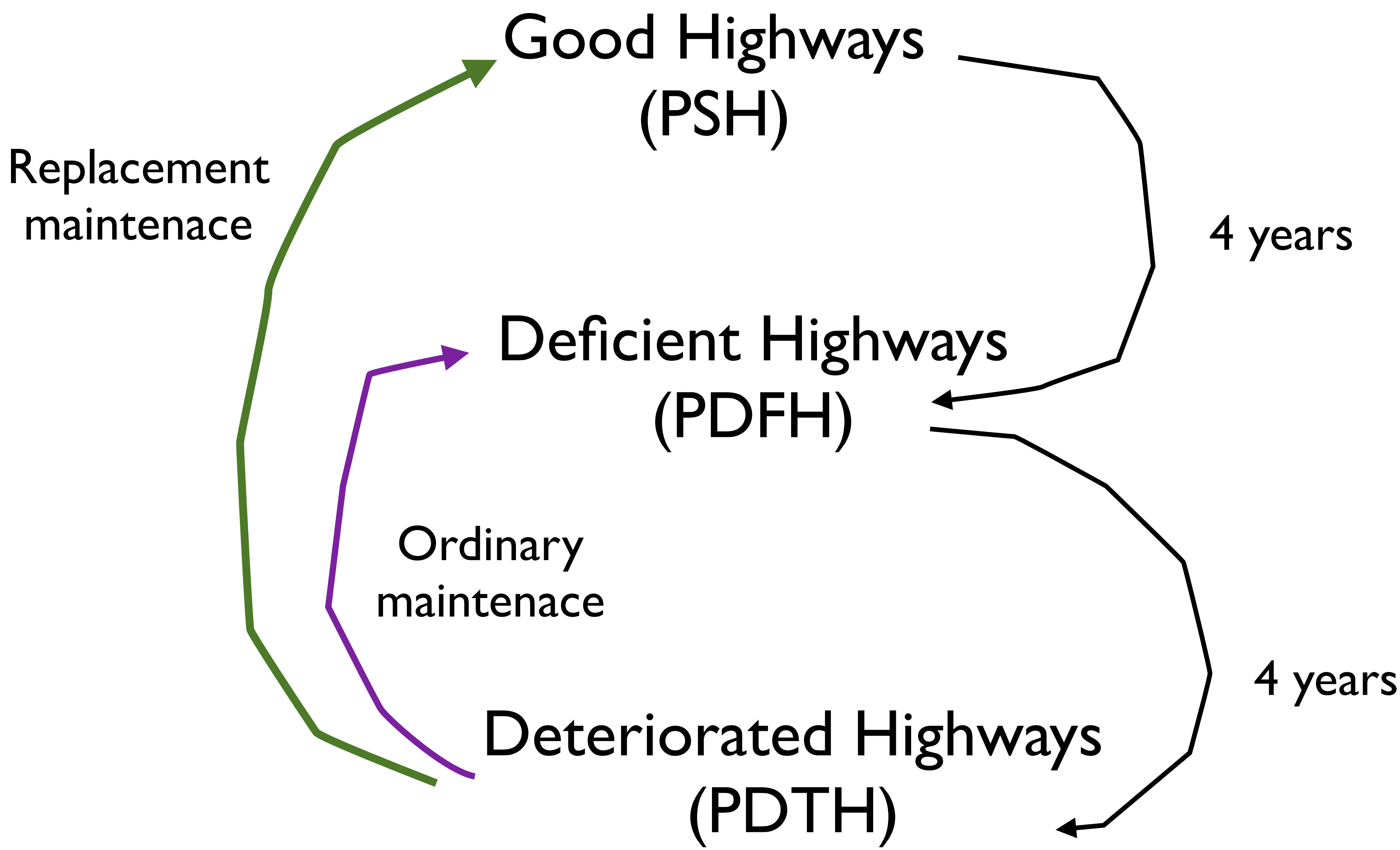


Highway Maintenance Model
Example of Higher-Order ODE
Conserved System

Highway Maintenance

- Departments of Transportation are responsible for keeping up many of the roads and highways that we use every day
- Maintenance requires substantial amounts of State money
- Money can be invested in two types of maintenance actions:
  - Ordinary - fix cracks, rutting
  - Replacement - repaving operations

Highways According to their Condition



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

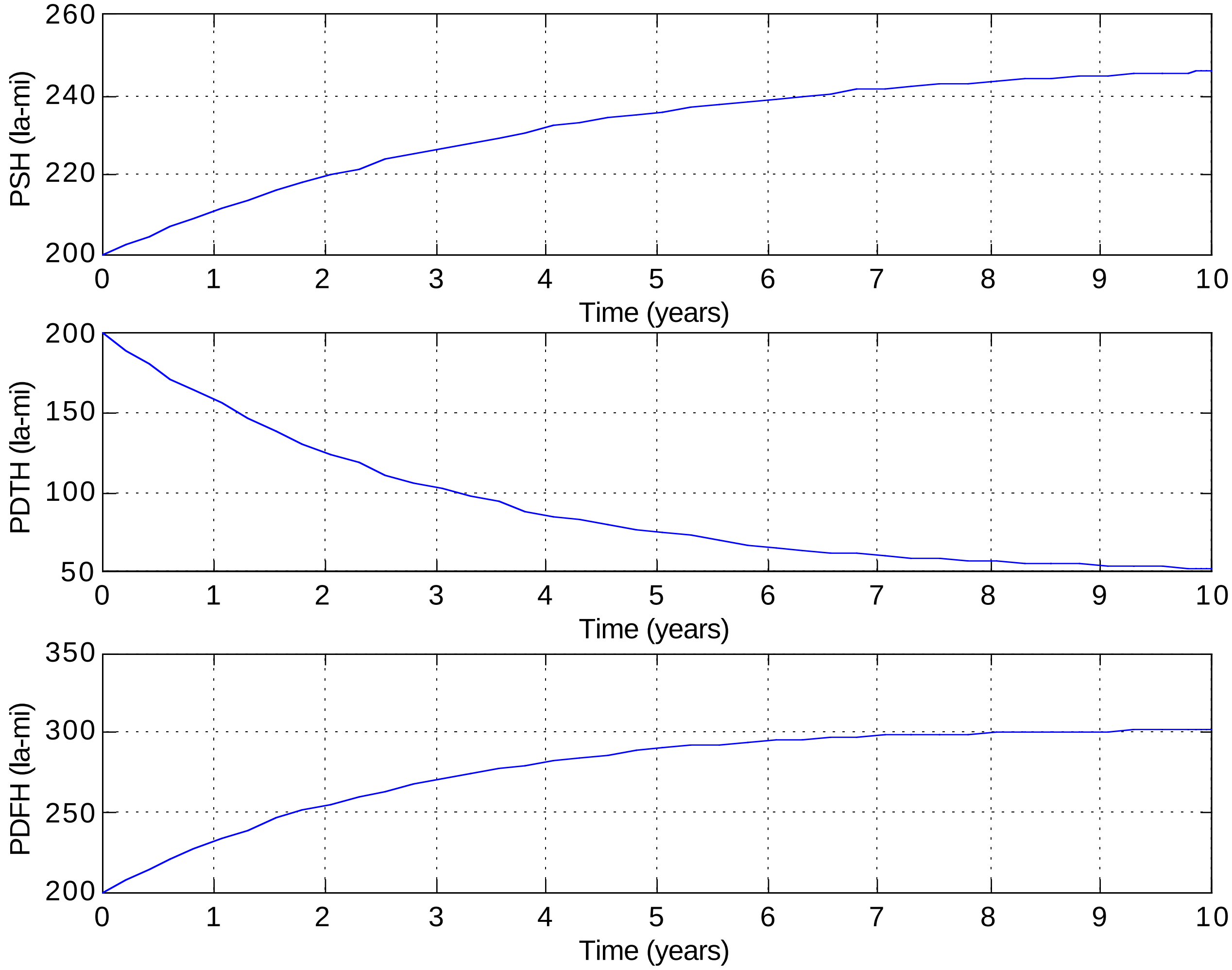
% y2 = PDFH

% y3 = PDTH

Function File (fhwy3_rev)

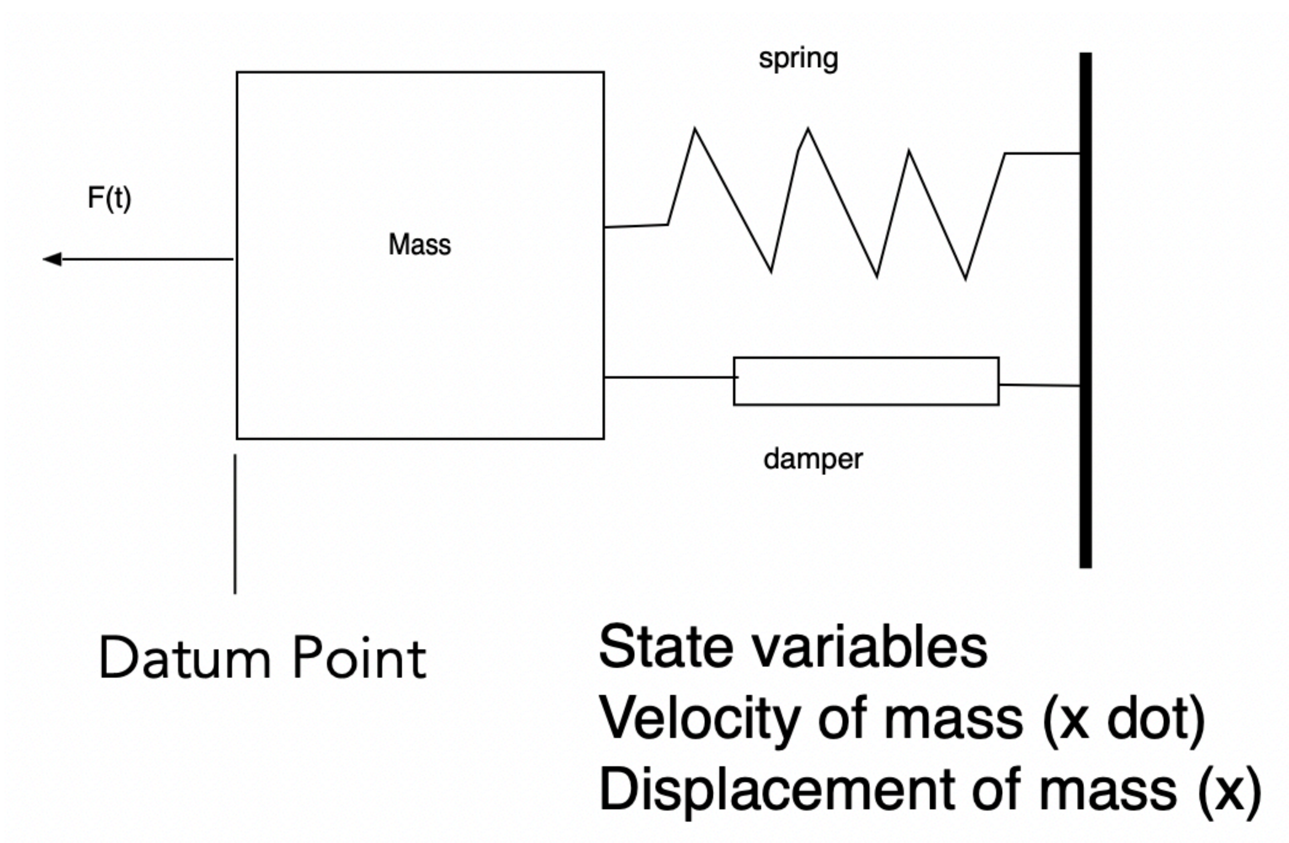
```
yprime(1) = HOM + HMR - HA;
% Rate of change of PSH (la-mi/yr)

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
```

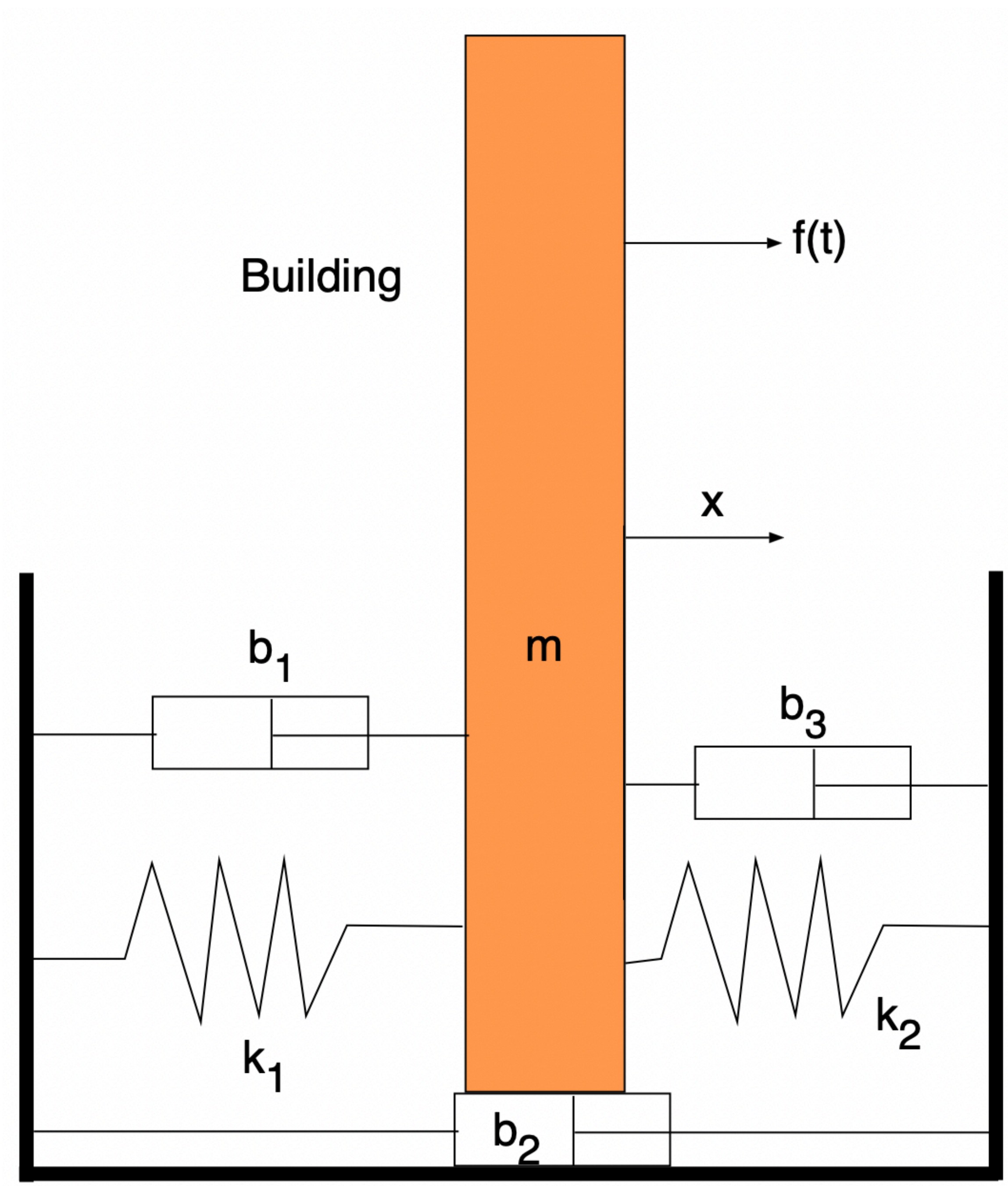
Sample Output of the Highway Maintenance Model



Spring-Mass-Damper (SMD) System

- Spring-mass-damper systems have many applications in mechanical and civil engineering systems

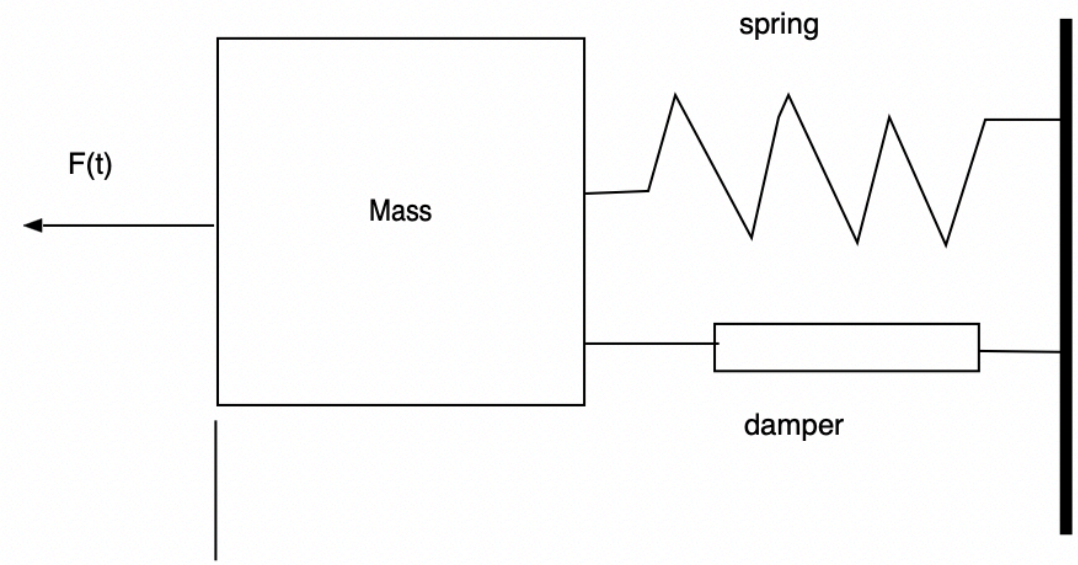




Equations to Describe the SMD System

$$y_1' = y_2$$

$$y_2' = \frac{F(t)}{m} - \frac{k}{m}y_1 - \frac{b}{m}y_2$$



Datum Point

State variables
Velocity of mass (x dot)
Displacement of mass (x)

$F(t)$	External force (N)
y_1	SMD displacement (m)
y_2	SMD speed (m/s)
m	Mass (kilograms)
b	Damper constant (N / (m/s))
k	Spring constant (N / m)

Matlab Implementation of the SMD System

```
% Main file to estimate the speed and distance profiles of
% a mass-spring-damper system

clear
close all
clc

% Solution to a set of two ODE equations of the form:
%
% ydot(1) = y(2);
% ydot(2) = F(t)/m - k/m y(1) - b/m y(2)
%
% subject to initial conditions:
%
% y (t=0) = yo
%
```

Main File

```
% Two first-order ODEs to solve the mass-spring-damper
function yprime = fy1(t,y)

global m b k

% Define the forcing function, f(t), here
F = 10000;              % Units are Newtons

% Define the equations of state
% y(1) = position (m)
% y(2) = speed (m/s)

yprime(1) = y(2);
yprime(2) = F/m - k/m * y(1) - b/m * y(2);

yprime = yprime';
```

Function File
With ODE equations

Numerical Solution of the SMD System

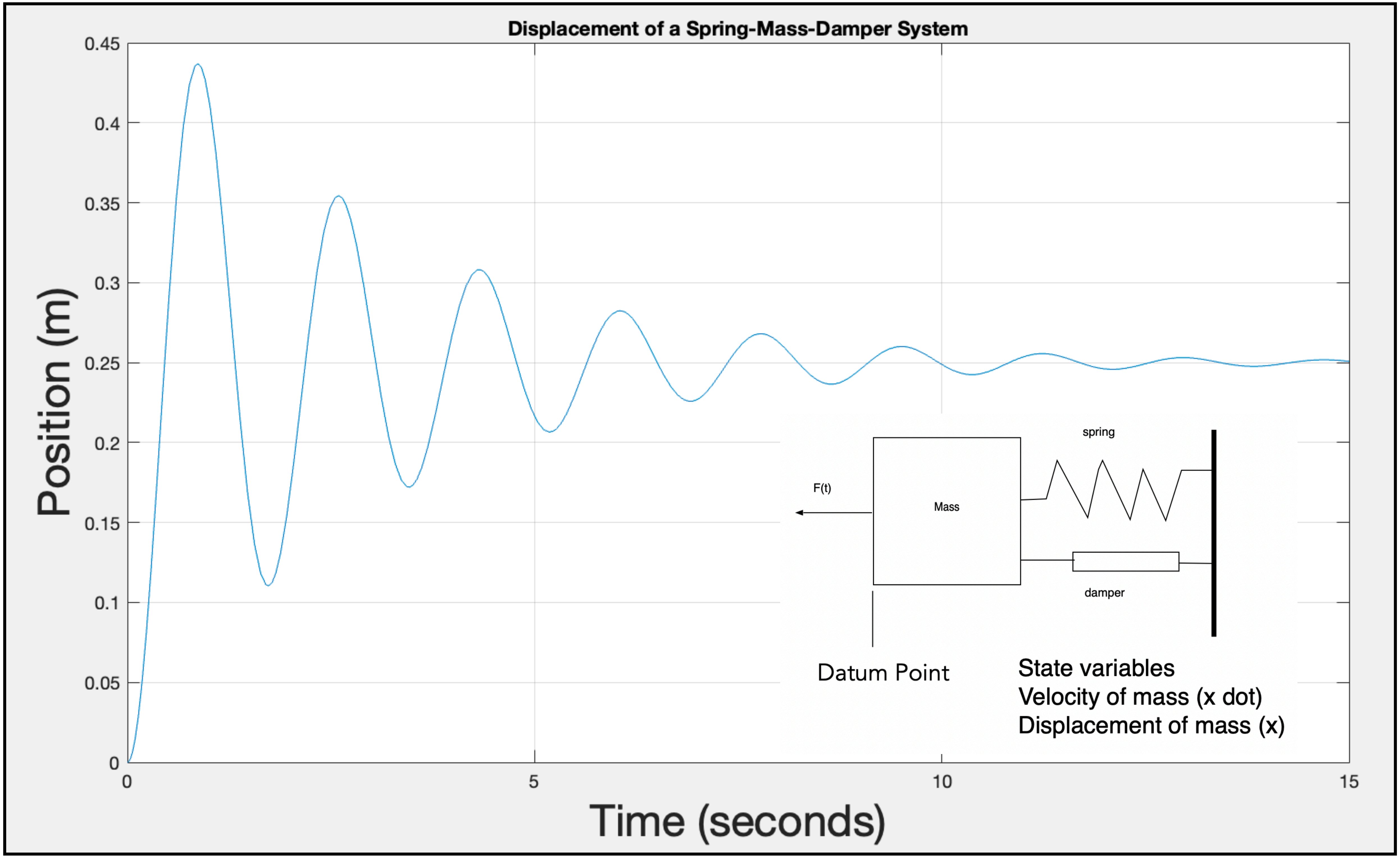
```
% Define Initial Conditions of the Problem

yo = [0 0];                % yo are the initial displacement and velocity
to = 0.0;                  % to is the initial time to solve this equation
tf = 15.0;                 % tf is the final time

% define m and b

m = 3000;          % mass (kg)
b = 2000;          % damper constant (N / (m/s))
k = 40000;         % spring constant (N/m)
tspan =[to tf];

[t,y] = ode23('fy1',tspan,yo);    % call ODE solver
```

Main File

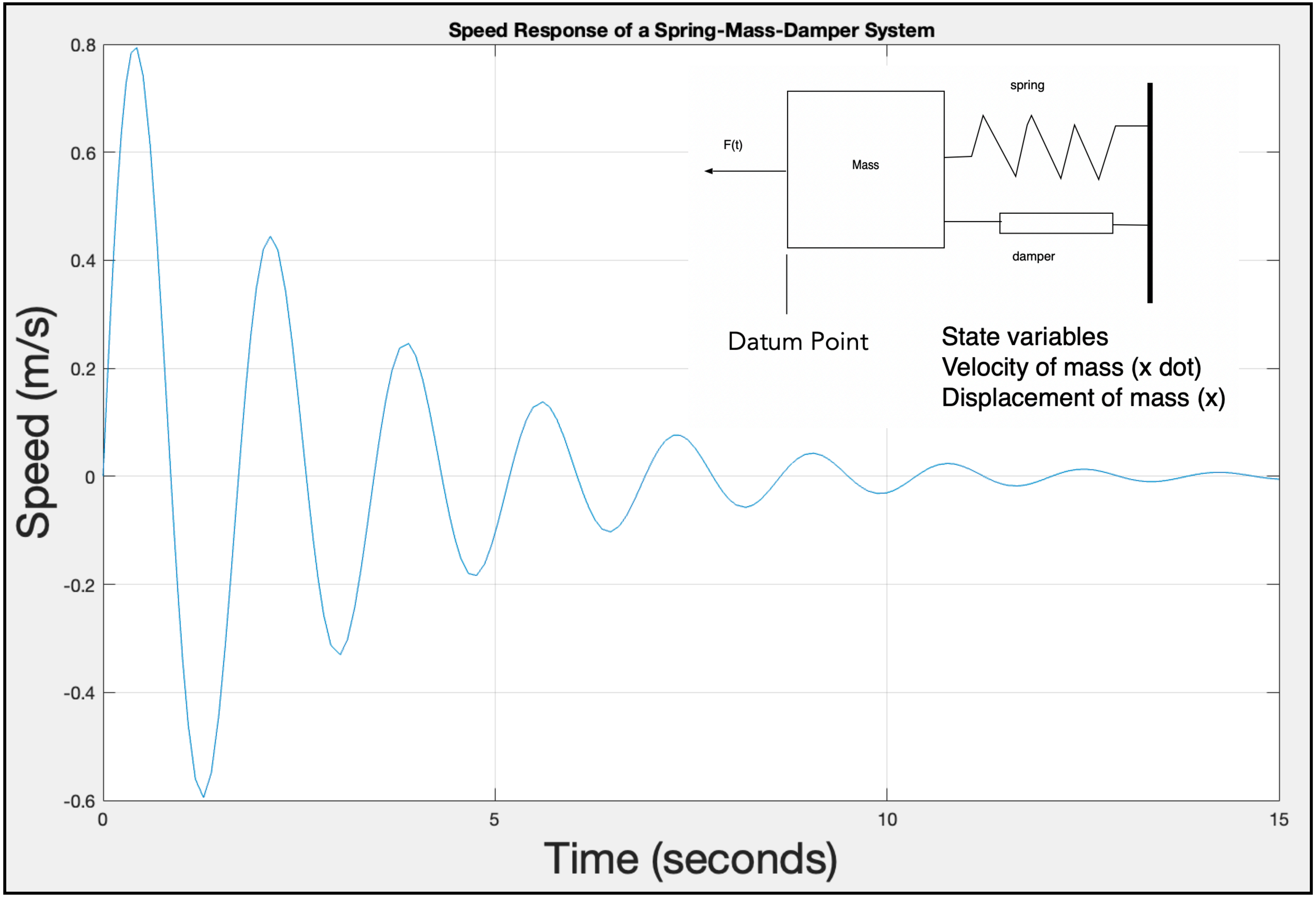
```
% Two first-order ODEs to solve the mass-spring-damper system
function yprime = fy1(t,y)

global m b k

% Define the forcing function, f(t), here
F = 10000;              % Units are Newtons
```

Function File
With ODE equations

Numerical Solution of the SMD System



Numerical Solution of the SMD System



Perform Changes to Damper Constant System