

Introduction to MATLAB

MATLAB Functions

Dr. Trani

**Civil and Environmental Engineering
Virginia Polytechnic Institute and State University**

Fall 2013

Purpose of this Section

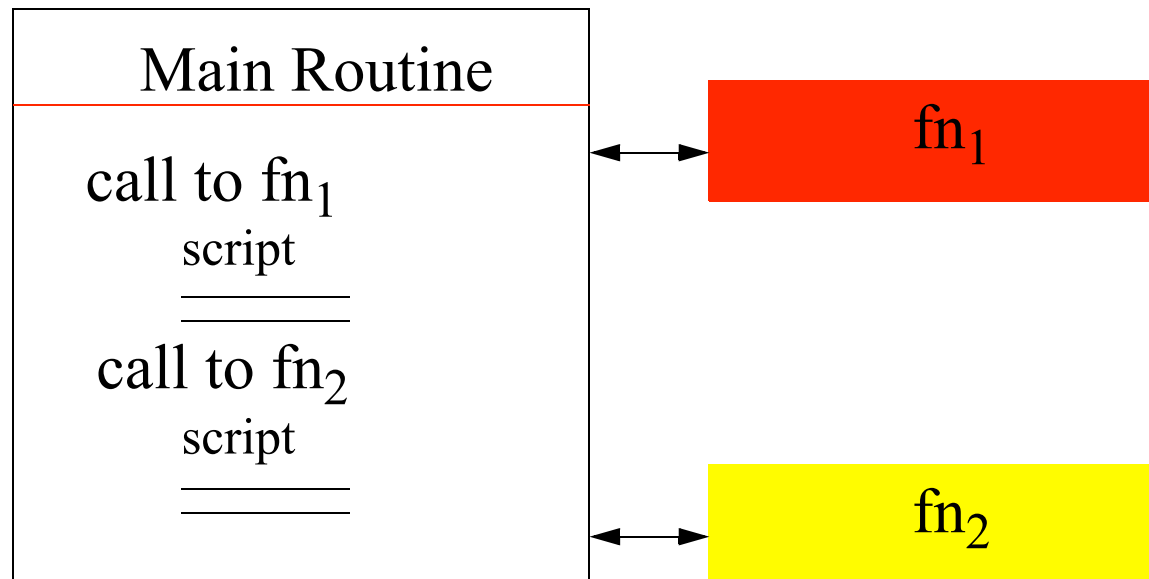
- To illustrate how MATLAB can be extended using functions
- To understand some of the data structures in MATLAB
- To understand some of the language specific features of the language

MATLAB Functions

- Provide the highest degree of functionality in the language
- Function files in MATLAB are equivalent to subroutines in FORTRAN, functions in C, and procedures in Pascal
- Function files constitute the basis for complex programs and model prototyping
- Functions can be **profiled** (statistics can be obtained in their execution times line by line)

Functions in MATLAB

- Typical framework for functions
- Good to avoid tedious code repetitions



General Syntax for Functions in MATLAB

function [output var.] = function_name (input var.)

- The word function should always be present and typed in lowercase
- The output variable list is optional
- The function_name should be the same as the file name (except for the .m termination) containing the function script
 - A function called **atmosphere** should reside inside an M-file called **atmosphere.m**

Local vs. Global Variables

- Function files can incorporate two types of variables:
 - Local
 - Global
- Local variables exist inside the function that uses them. All variables defined inside a function are local unless otherwise defined
- Global variables are shared among various functions and are defined as such in all function files where they are expected to be used
 - `global x y z`
 - This statement defines 3 global variables x,y, and z

A Simple Function in MATLAB

An empirical formula to estimate the pavement thickness is known to be:

$$t = \sqrt{\frac{P}{8.1(CBR)} + \frac{A}{\pi}}$$

where:

t is the design **pavement thickness** (in inches)

P is the **equivalent single wheel load** of the aircraft tires on the pavement (in pounds)

A is the **single tire contact area** (in²)

CBR is the **California Bearing Ratio** (dimensionless)
which measures the shearing strength of the soil
(compared with the characteristics of crushed rock)

and π is 3.141592..

MATLAB Implementation (thickness.m)

```
% Function to estimate the pavement thickness
% of a flexible pavement
% Input arguments:
% load = single wheel equivalent load (pounds)
% area = tire contact area (in-in)
% cbr = California bearing ratio (dimensionless)
%
% Output arguments
% t = pavement thickness

function t = thickness(load,area,cbr)

t = sqrt(load ./ (8.1 .* cbr) + area/pi);
```

A Few Explanations

- The function thickness should be defined in a separate **M-file** file called **thickness.m**
- The function thickness has three input arguments:
 - load
 - area
 - cbr
- The function thickness has one output argument: **t**
- The dots before the division and multiplication signs allow vector operations (more on this later)
- The function is quite simple but illustrates the point on how to create a MATLAB function

A Numerical Example

Suppose we have a critical aircraft (such as the Boeing 727-200) to be operated at this airport. The aircraft has an equivalent single load wheel value of 60,000 lb. The contact area of each tire is 240 in².

The following command executes the function:

```
>> thickness(60000,240,10)
```

Note: this tells MATLAB to evaluate a function called **thickness** with arguments 60000, 240 and 10, respectively.

```
>> t =  
    28.6634
```

Further Use of thickness.m

- The value of functions is to extend the capabilities of MATLAB
- For example MATLAB could be instructed to estimate numerous pavement thicknesses based on several values of CBR
- The following code estimates thicknesses (for CBR values ranging from 10 to 30 at steps of 1)
- The same MATLAB script plots the relationship between *CBR* and *t*
- The new script file is called **pavement_cbr.m**

Using thickness.m in Script **pavement_cbr.m**

```
% This M file uses the function thickness to estimate  
% various pavement thicknesses for CBR values  
% ranging from an initial value (ini_cbr_value) to a  
% final CBR value (fin_cbr_value)
```

```
P = 60000; % load (pounds)  
A = 254; % tire contact area
```

```
in_cbr_value = 10;  
fin_cbr_value = 30;  
npoints = 100;
```

```
cbr_vector = linspace(in_cbr_value, fin_cbr_value,...  
                      npoints);
```

```
t_vector = thickness(P,A,cbr_vector);
```

```
plot(cbr_vector,t_vector)
```

```
xlabel('CBR')
```

```
ylabel('Pavement Thickness (in)')
```

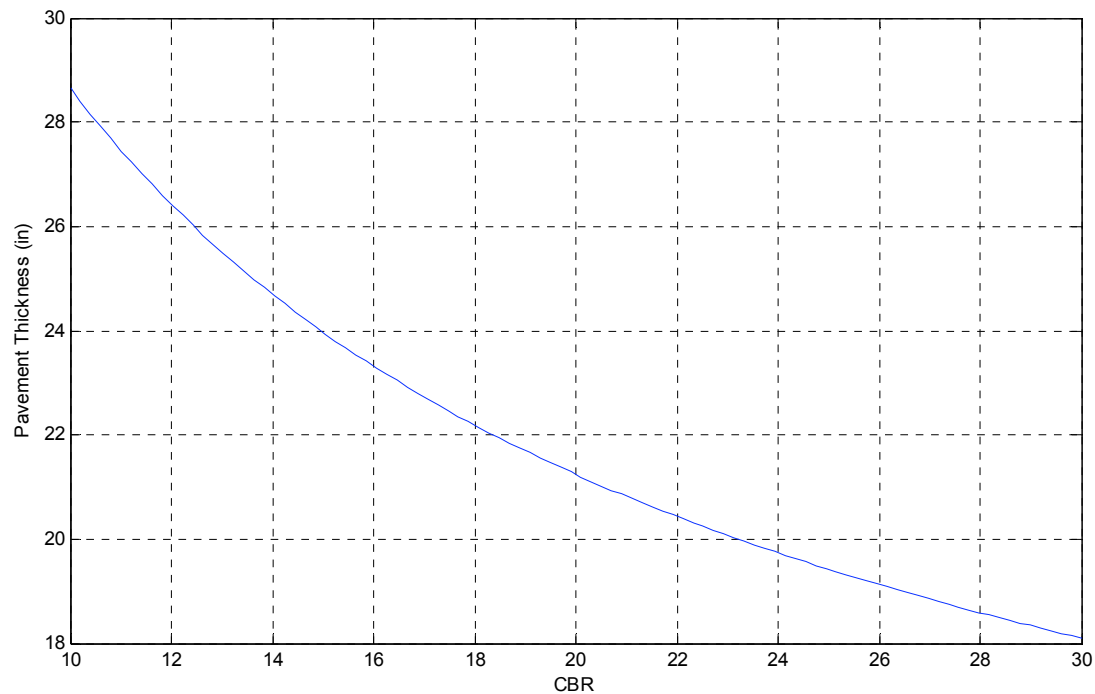
```
grid
```

Notes:

- A vector called `cbr_vector` is defined as a linearly spaced vector with values of *CBR* ranging from 30 to 10
- The function **thickness** is called 100 times to estimate values of *t* in the line evaluating `t_vector`
- The plot in the last few statements shows graphically the relationship between *CBR* and *t*

Sample Output (pavement_cbr.m)

The following plot illustrates the use of function `thickness.m` inside the script `pavement_cbr.m`



Using the feval Function in MATLAB

- MATLAB has a function called feval to evaluate functions of any type
- feval allows users to specify the file name dynamically
- This is useful when complex calls to a function are needed and perhaps various algorithms have been programmed in your code

Using **feval** with **thickness** (**feval_cbr.m**)

```
% This M file uses function thickness to estimate  
% various pavement thicknesses for CBR values  
% ranging from and initial value (ini_cbr_value) to a  
% final CBR value (fin_cbr_value)
```

```
P = 60000; % load (pounds)  
A = 254;  % tire contact area
```

```
in_cbr_value    = 10;  
fin_cbr_value   = 30;  
npoints         = 100;
```

```
cbr_vector = linspace(in_cbr_value, fin_cbr_value,  
                      npoints);
```

```
t_vector = feval('thickness',P,A,cbr_vector);
```

```
plot(cbr_vector,t_vector)
```

```
xlabel('CBR')
```

```
ylabel('Pavement Thickness (in)')
```

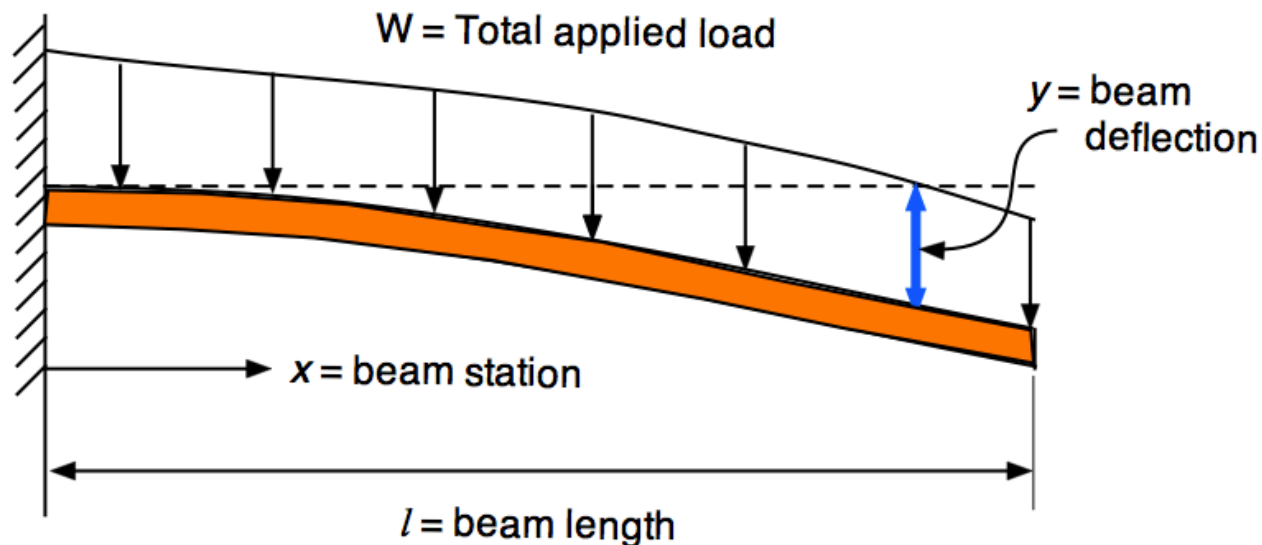
```
grid
```

Notes:

- The **feval** function invokes **thickness** (first line of this page)
- The invocation statement is done using single quotes and thus allows various function files to be called from within **feval_cbr.m**

A Beam Problem Using a Matlab Function

- Consider the cantilever beam problem



y = deflection (in)
 s = stress (lb/sq. in.)
 Z = section modulus
 E = Modulus of elasticity (psi)
 I = Moment of inertia (in^4)
 W = Applied load (lb)
 l = beam length (in)

$$y = \frac{Wx^2}{24EI} (2l^2 + (2l - x)^2)$$

$$s = \frac{W}{2ZI} (l - x)^2$$

$$Z = I / d$$

Matlab Function (cantiliverBeam.m)

```

1 % Function to calculate the deflection (y) and stress (s) of a cantilever beam
2
3 function [y,s] = cantiliverBeam(W,E,I,x,beamLength,distNeutralAxis)
4
5 % Date:
6 % Programmer: T. Trani
7
8 % Inputs to function cantiliverBeam
9
10 % W = load (lb)
11 % E = Modulus of elasticity (lb/sq-in)
12 % I = Moment of inertia (in-in-in-in)
13 % x = distance from datum point (in)
14 % l = beam length (in)
15 % distNeutralAxis = distance from edge of beam to neutral axis (in)
16
17 % Outputs of function cantiliverBeam
18
19 % y = deflection (in)
20 % s = stress at the cross-section being evaluated (lb/in-in)
21 % Z = section modulus of the cross section of the beam
22 % Z is calculated as I / distance from neutral axis to edge of the beam
23
24 % Calculations
25
26 y = -W * x.^2 ./ (24 * E * I * beamLength) .* (2 * beamLength^2 + (2 * beamLength - x).^2);
27 Z = I / distNeutralAxis;
28 s = W / (2 * Z * beamLength) .* (beamLength - x) .^2;

```

Note: Six inputs required

Note: Two outputs produced
y = deflection and
s = stress at cross section

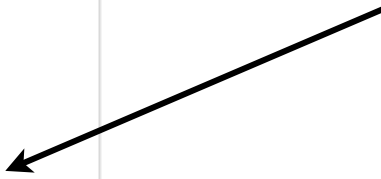
Matlab Script Calling CantileverBeam.m

```

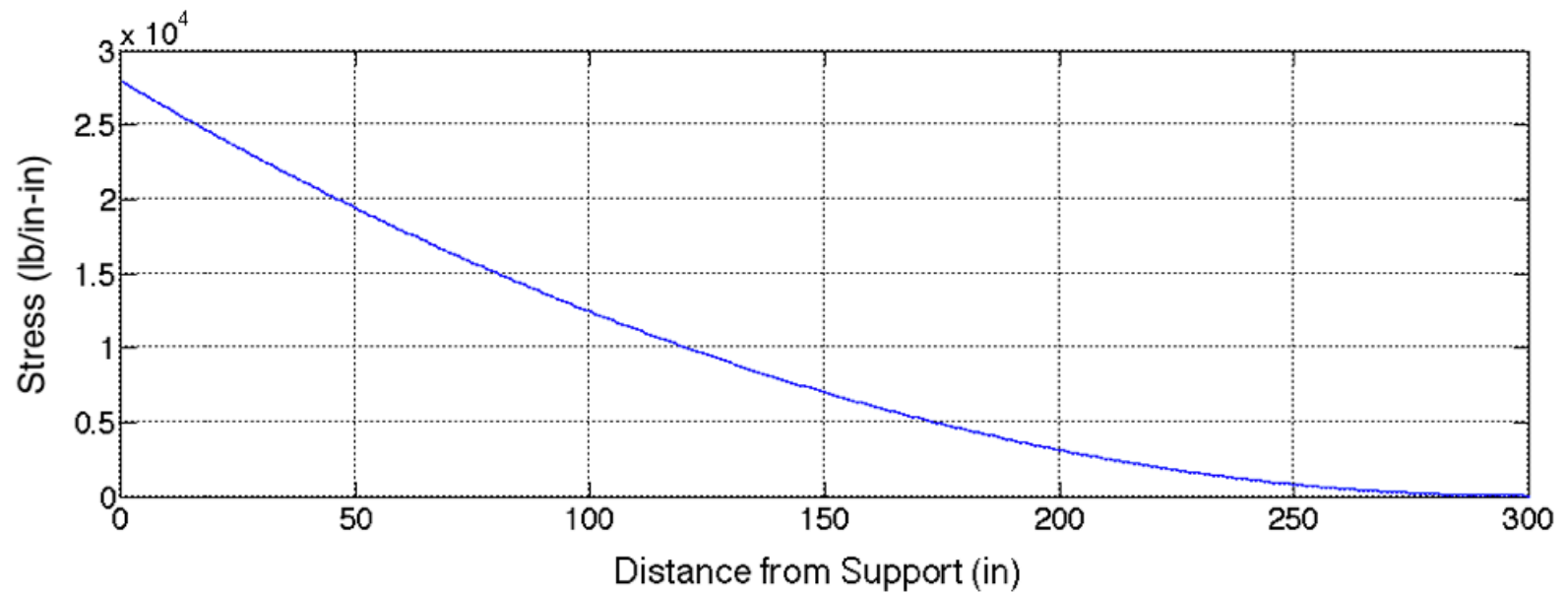
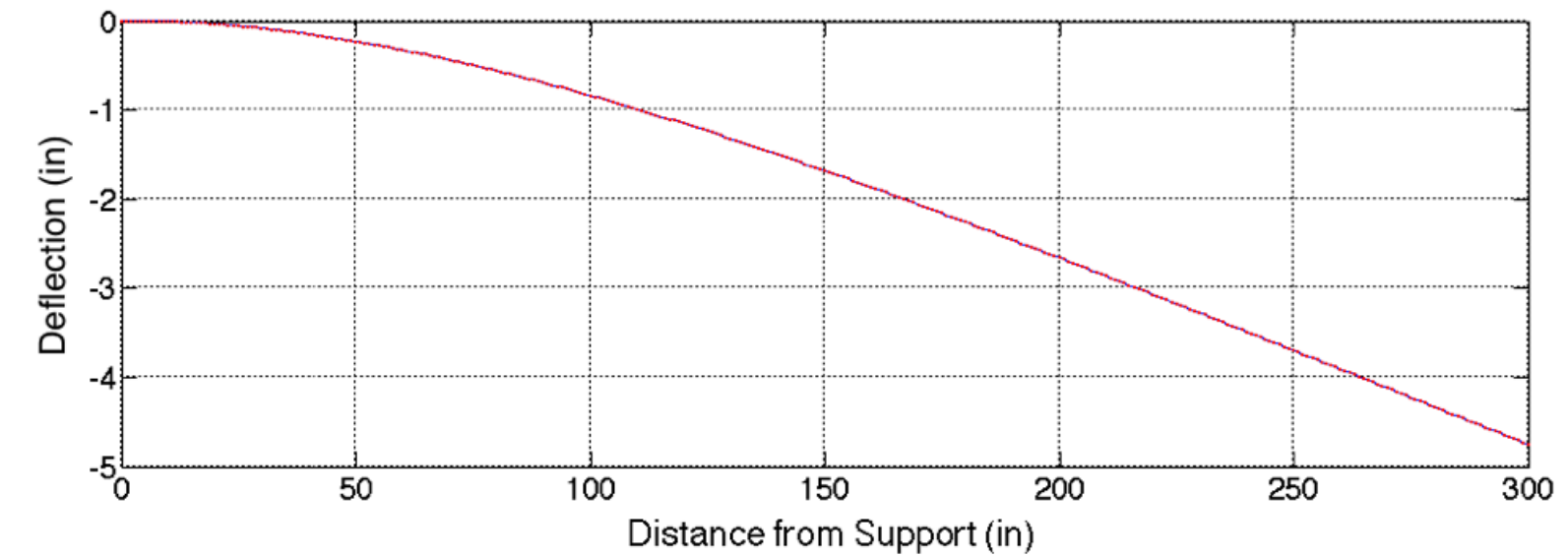
1  % M File to calculate the deflection of a beam
2  % This file calls the cantiliverBeam function (contained in cantiliverBeam.m file)
3
4  % Date:
5  % Programmer : T. Trani
6
7  % Inputs to function cantiliverBeam
8  % W = load (lb)
9  % E = Modulus of elasticity (lb/sq-in)
10 % I = Moment of inertia (in-in-in-in)
11 % x = distance from datum point (in)
12 % l = beam length (in)
13 % distNeutralAxis = distance from edge of beam to neutral axis (in)
14
15 % Outputs of function cantiliverBeam
16 % y = deflection (in)
17 % s = stress at the cross-section being evaluated (lb/in-in)
18 % Z = section modulus of the cross section of the beam
19 % Z is calculated as I / distance from neutral axis to edge of the beam
20
21 % Enter the properties of a beam to be calculated
22 W = 4000;
23 E = 11e6;
24 I = 258;
25 x = 0:1:300;
26 beamLength = 300;
27 distNeutralAxis = 12;
28
29 % Call the function cantilever beam
30
31 [y,s] = cantiliverBeam(W,E,I,x,beamLength,distNeutralAxis);

```

call function
cantileverBeam.m



Output of Matlab Script



Function to Estimate Speed and Distance Profiles for a Linear Acceleration Model

- The analysis of many transportation systems requires an understanding of kinematic equations of motion
- Such formulas can be used in the design of the following systems:
 - highway design
 - runway length design at airports
 - high-speed rail tracks

Equations of Motion with Linearly Decreasing Acceleration Function

$$\frac{dV_t}{dt} = k_1 - k_2 V_t$$

$$\frac{dV_t}{dt} = \text{rate of change of velocity with time} = \text{acceleration (m/s}^2\text{)}$$

k_1 = maximum instantaneous acceleration (m/s²) - static conditions

k_2 = lapse rate of acceleration with velocity (1/s)

V_t = vehicle velocity at time t (m/s)

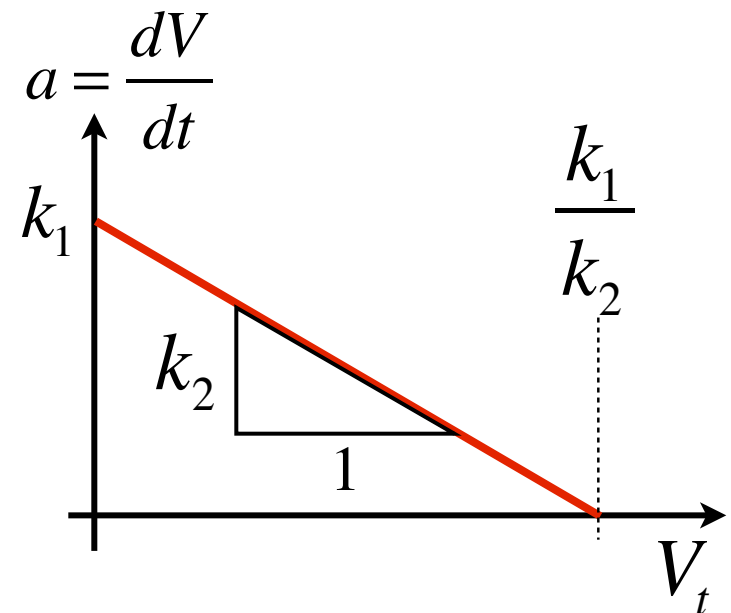
$$a_t = (k_1 - k_2 V_0) e^{-k_2 t}$$

$$V_t = V_0 e^{-k_2 t} + \frac{k_1}{k_2} (1 - e^{-k_2 t})$$

$$S_t = S_0 + \frac{k_1}{k_2} t + \frac{V_0}{k_2} (1 - e^{-k_2 t}) + \frac{k_1}{k_2^2} (1 - e^{-k_2 t})$$

a_t = acceleration (m/s²)

S_t = distance traveled (m)



Example: Aircraft Takeoff

- Aircraft starts at zero speed at the start of the runway
- Aircraft reaches lift-off speed (V_{lof})



Example: Aircraft Takeoff Calculations

- Boeing 737-800 (aircraft shown in the picture)
- Sea level conditions

$$k_1 = 3.0 \text{ (m/s}^2\text{)} \text{ and } k_2 = 0.02 \text{ (1/s)}$$

$$V_0 = 0 \text{ (m/s)} \text{ and } V_{lof} = 75 \text{ m/s}$$

$$S_0 = 0 \text{ (m)}$$



- Calculate the time to reach lift-off speed
- Calculate the lift-off distance (runway length needed to reach lift-off speed)
- Estimate the acceleration at the lift-off point

Matlab Function to Calculate Speed and Distance (speedDistanceCalculator.m)

```

1  function [speed, distance, acceleration]= speedDistanceCalculator(k1,k2,t,Vo,So)
2
3  % Function to estimate the velocity and distance traveled by a vehicle with
4  % linearly decreasing acceleration function
5  % This problems evaluates the analytic solution to the first order model
6  % presented below (a first order differential equation)
7
8  % dV/dt = k1 - k2 * V
9  %
10 % where:
11 %
12 % dV/dt = acceleration (m/s-s)
13 % V = speed = vehicle speed (m/s)
14 % S = distance = distance traveled (m)
15 % k1 = maximum acceleration (m/s-s)
16 % k2 = slope of dV/dt vs V profile (1/s)
17
18 % Programmer: T. Trani
19 % Date: 10/15/2013
20
21     speed = k1/k2 * (1 - exp(-k2 .* t)) + Vo .* exp(-k2 .* t);
22
23     % Calculate distance as a function of time
24     distance = So + k1/k2 * t - k1/(k2^2) * (1 - exp(-k2 .* t)) ...
25         + Vo/k2 .* (1-exp(-k2 .* t));
26
27     % calculate teh acceleration as a function of time
28     acceleration = (k1 - k2 * Vo) * exp(-k2 .* t);
29
30 end

```

Five input arguments

Note: three outputs are produced
V = speed (m/s)
S = distance traveled (m)
a = acceleration (m/s²)

Function is designed to accept vectors to t but other quantities as scalars

Matlab Script to Call Speed and Distance Calculator Function

```

1 % Script to calculate the velocity and distance profiles of a vehicle
2 %The vehicle has a non-uniform
3 % acceleration model of the form:
4
5 %  $dV/dt = k1 - k2 * V$ 
6 %
7 % where:
8 %
9 %  $dV/dt = \text{acceleration} = \text{acceleration (m/s-s)}$ 
10 %  $V = \text{speed} = \text{vehicle speed (m/s)}$ 
11 %  $S = \text{distance} = \text{distance traveled (m)}$ 
12 %  $k1 = \text{maximum acceleration (m/s-s)}$ 
13 %  $k2 = \text{slope of } dV/dt \text{ vs } V \text{ profile (1/s)}$ 
14
15 % Programmer: T. Trani
16 % Date: 10/15/2013
17
18 % Enter constants of the problem
19
20 k1 = 3.0; % maximum acceleration (m/s-s)
21 k2 = 0.02; % slope of dV/dt vs. V curve ((1/s)
22 Vo = 0; % initial speed (m/s)
23 time = 0:0.01:50; % time vector to estimate velocity vs time profile (s)
24 So = 0; % initial distance traveled (m)
25
26 % Solve equations by calling the function speedDistanceCalculator.m
27
28 [speed, distance, acceleration]= speedDistanceCalculator(k1,k2,time,Vo,So);
29
30 % Plot the results

```

Create a time vector to study the movement of the vehicle at various points in times

Here we call the Matlab function speedDistanceCalculator.m

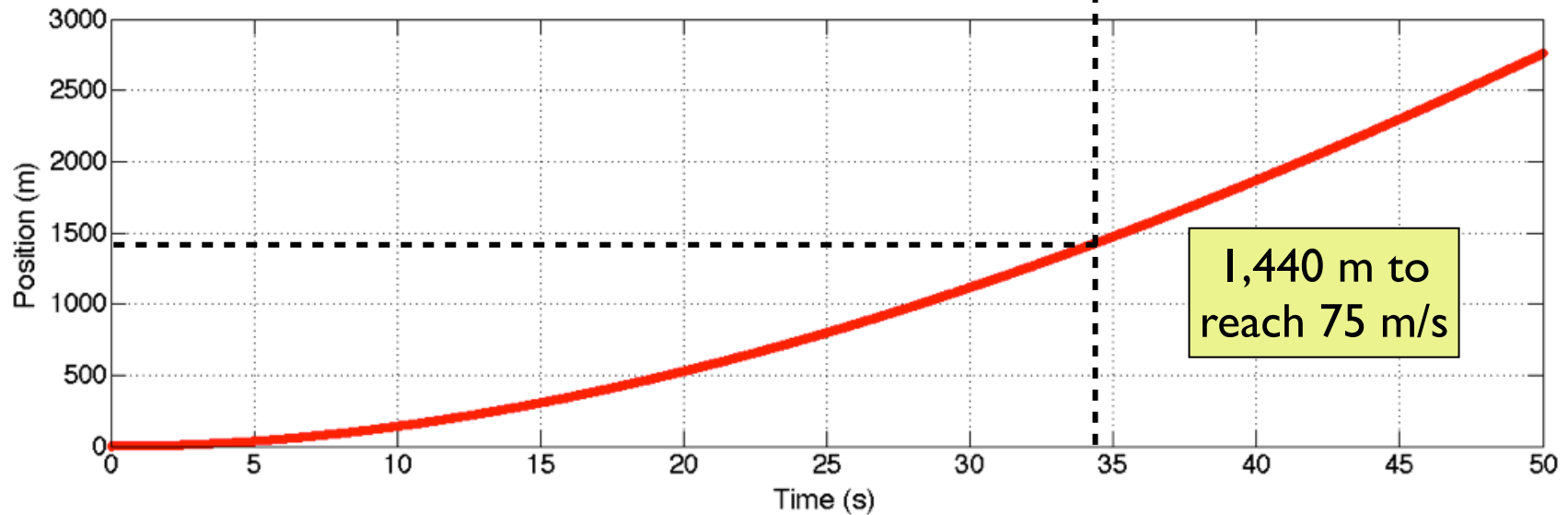
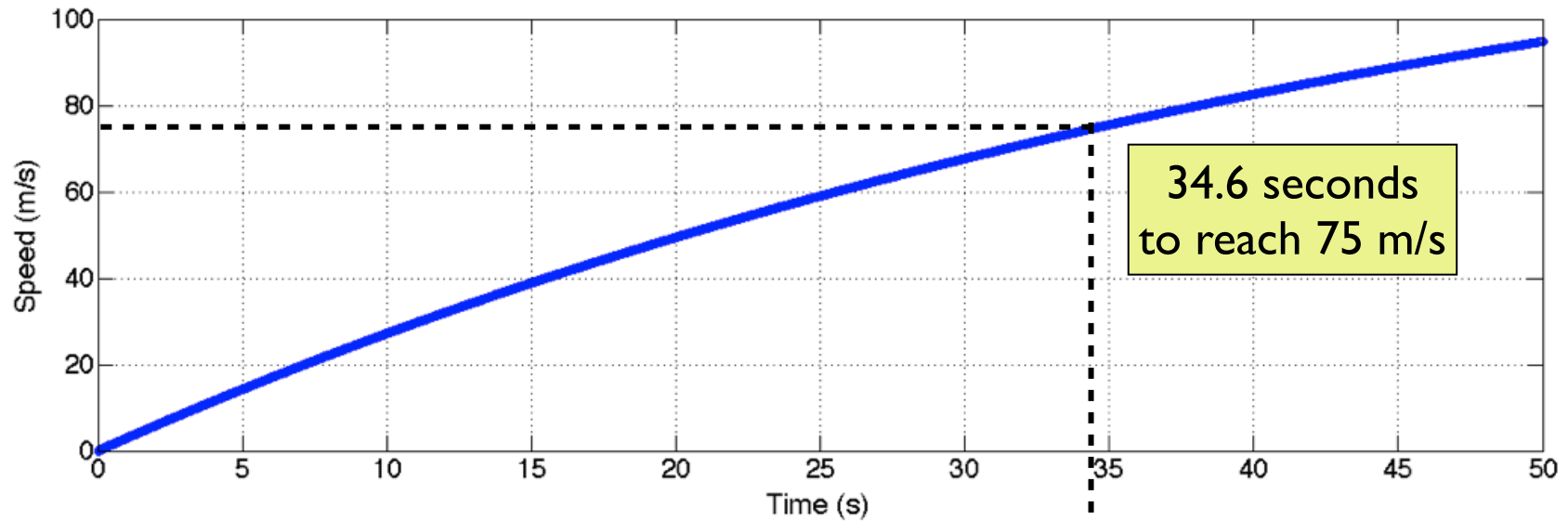
Matlab Script to Call Speed and Distance Calculator Function (cont.)

```

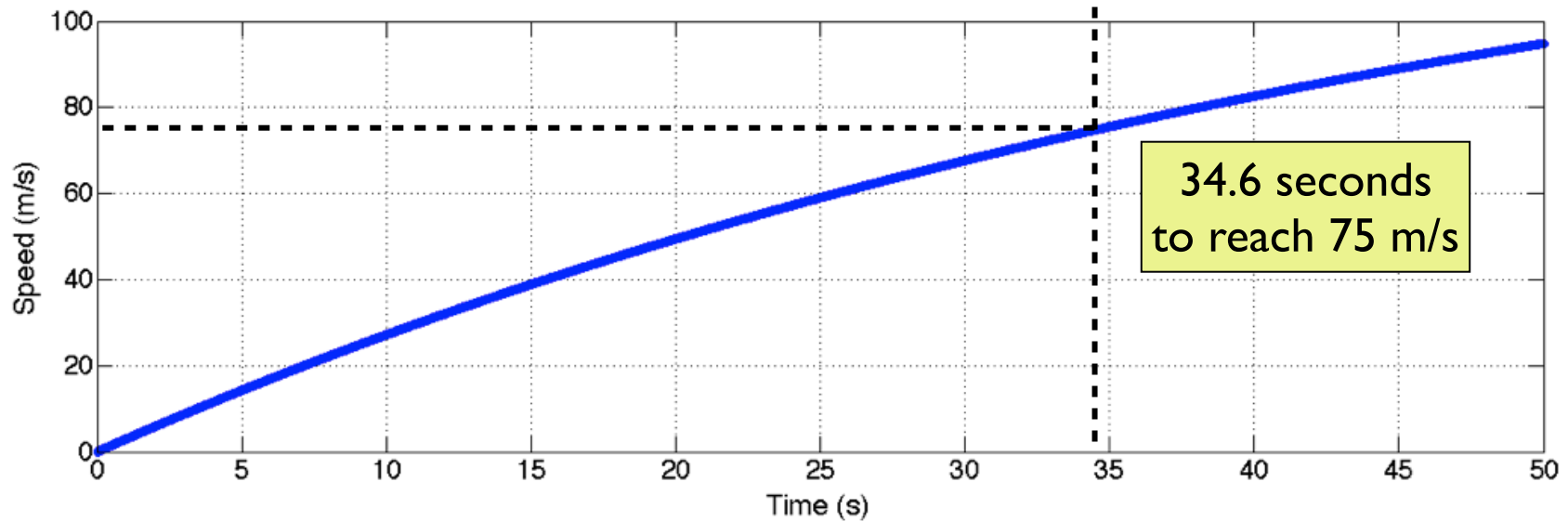
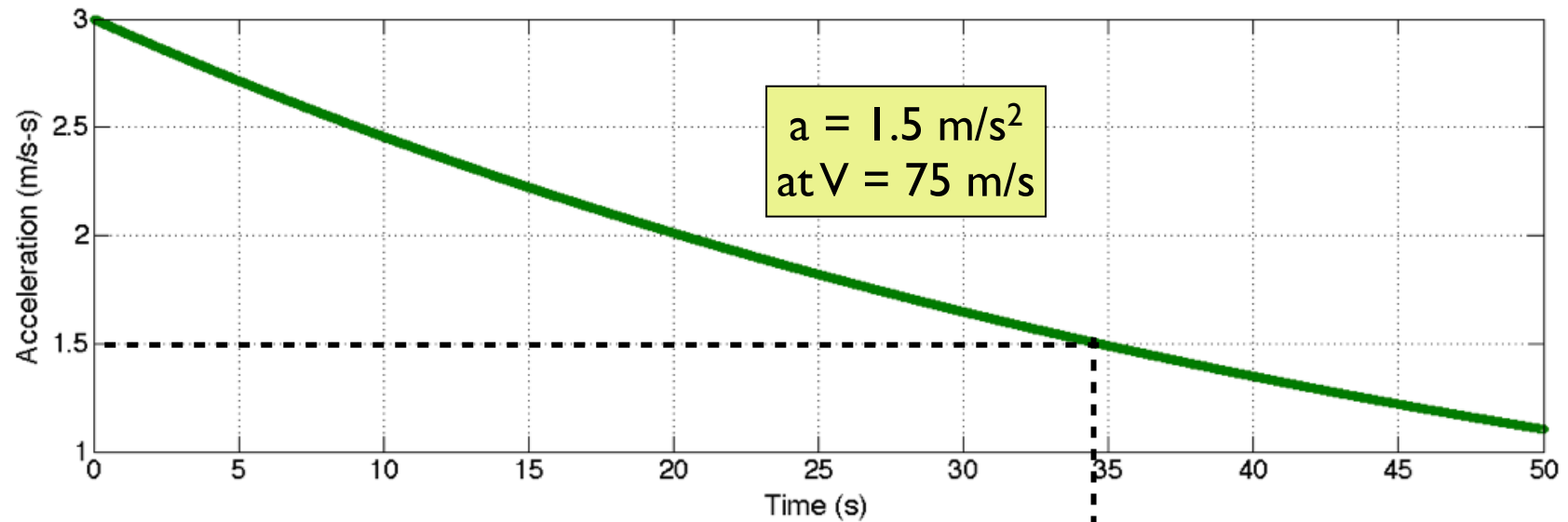
28 [speed, distance, acceleration]= speedDistanceCalculator(k1,k2,time,Vo,So);
29
30 % Plot the results
31
32 % Plot vehicle acceleration (a), speed (V) and position (x)
33 figure
34 subplot(2,1,1)
35 plot(time,acceleration,'o-')
36 xlabel ('Time (s)','fontsize',22)
37 ylabel ('Acceleration (m/s-s)','fontsize',22)
38 grid
39
40 subplot(2,1,2)
41 plot(time,speed,'o-')
42 xlabel ('Time (s)','fontsize',22)
43 ylabel ('Speed (m/s)','fontsize',22)
44 grid
45
46 figure
47 subplot(2,1,1)
48 plot(time,speed,'o-')
49 xlabel ('Time (s)','fontsize',22)
50 ylabel ('Speed (m/s)','fontsize',22)
51 grid
52
53 subplot(2,1,2)
54 plot(time,distance,'o-')
55 xlabel ('Time (s)','fontsize',22)
56 ylabel ('Position (m)','fontsize',22)
57 grid

```

Output of Matlab Script



Output of Matlab Script (cont.)

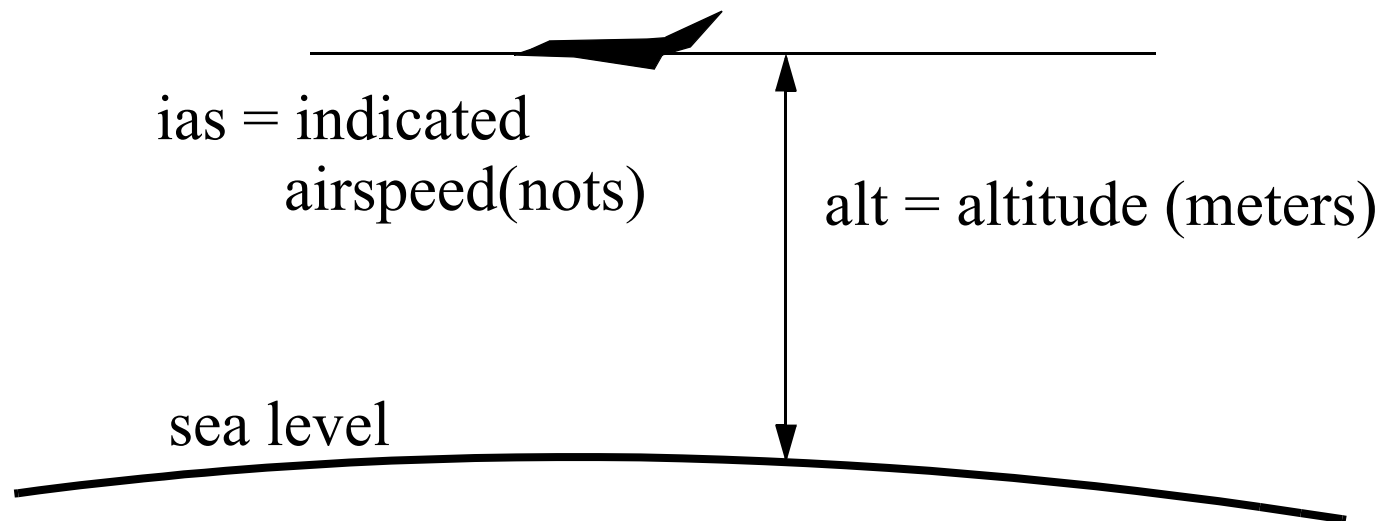


Observations

- The function `speedDistanceCalculator.m` was designed to operate on vectors or arrays with time as the parameter
- The remaining parameters (k_1 , k_2 , V_0 and S_0) are assumed to be constant in the function
- You can improve the function by making the analysis more “general” so that the function takes vector or array values for (k_1 , k_2 , V_0 and S_0)
- The function can be called inside a FOR-loop if needed. However, this will be more expensive computationally (try it out).

A More Complex Function in MATLAB

- This example computes true Mach number of an aircraft flying at altitude (**alt**) and with an indicated airspeed (**ias**)
- The function invokes the 1962 International Standard Atmospheric (ISA) model
- The following table presents the ISA atmospheric model



- Parameters in the international standard atmosphere are:

Geopotential Altitude (m.) h	Temperature (°K) T	Density (kg/m ³) ρ	Speed of Sound (m/s) a
------------------------------------	-----------------------	--	------------------------------

ISA Atmospheric Model

Geopotential Altitude (m.)	Temperature ($^{\circ}$ K) T	Density (Kg/m^3) ρ	Speed of Sound (m/s) a
0	288.2	1.225	340.3
1000	281.7	1.112	336.4
2000	275.2	1.007	332.5
3000	268.7	0.909	328.6
4000	262.2	0.819	324.6
5000	255.7	0.736	320.5
6000	249.2	0.660	316.4
7000	242.7	0.589	312.3
8000	236.2	0.525	308.1
9000	229.7	0.466	303.8
10000	223.2	0.413	299.5
11000	216.7	0.364	295.1
12000	216.7	0.311	295.1

Function to Estimate True Mach Number

A mathematical expression to estimate true airspeed (in terms of true Mach number) from IAS follows:

$$M_{true} = \sqrt{5 \left[\left[\frac{\rho_0}{\rho} \left(\left[1 + 0.2 \left(\frac{V_{IAS}}{661.5} \right)^2 \right]^{3.5} \right) + 1 \right]^{0.286} \right]}$$

where: M_{true} is the true mach number, V_{IAS} is the indicated airspeed in knots (IAS) in our analysis, ρ_0 is the atmospheric density at sea level, ρ is the density at the altitude the aircraft is flying, and the constants 0.2 and 661.5 account for the specific heat of the air and the speed of sound at sea level (in knots), respectively.

Implementation in MATLAB (mach.m)

% Estimates true mach number of an aircraft given its

% alt = altitude (m)

% ias = indicated airspeed (knots)

function mtrue = mach(alt,ias)

rho_zero = 1.225; % density at sea level (kg/m-m-m)

load atmosphere; % loads ISA atmospheric tables

h = atmosphere(:,1);% vector with values of altitude

t = atmosphere(:,2);% vector with values of temperature

r = atmosphere(:,3);% vector with values of density

a = atmosphere(:,4);% vector with values of speed

```
rho = interp1(h,r,alt,'cubic')% interpolates to get density
mtrue = sqrt(5 * ((rho_zero/rho * ((1 + 0.2 * ...
    (ias 661.5)^2)^3.5 - 1) + 1)^0.286 - 1));
```

- This function produces a single output variable called **mtrue**
- The function resides in an M-file called mach.m (the same name as the function)
- Execute from the command line typing:

```
>> mtrue = mach(6000,340)
```

Note: this tells MATLAB to evaluate a function called **mach** with arguments 6000 and 340, respectively.

Remarks About Function Mach

- Uses **interp1** function to interpolate between values of two vectors (using cubic splines)
- Uses ellipsis (...) to state that a line continues (see line where **mtrue** is calculated)
- **rho** is a local variable (not available in the workspace after the function is executed)

Sample Function With Multiple Output Variables

- The following function (called isam) is a variation of mach except that outputs 4 variables

```
function [mtrue,a_alt,rho,temp] = isam(alt,ias)
```

```
rho_zero = 1.225; % density at sea level (kg/m-m-m)
```

```
load atmosphere; % loads ISA atmospheric tables
```

```
h = atmosphere(:,1); t = atmosphere(:,2);
```

```
r = atmosphere(:,3); a = atmosphere(:,4);
```

```
rho = interp1(h,r,alt,'cubic');% interpolates to get density
```

```
mtrue = sqrt(5 * ((rho_zero/rho * ((1 + 0.2 * ...  
                (ias/661.5)^2)^3.5 - 1) + 1)^0.286 - 1));
```

```
a_alt = interp1(h,a,alt,'cubic');% gets speed of sound
```

```
temp = interp1(h,t,alt,'cubic');% gets temperature
```


Profiling a Function in MATLAB

- The profile function allows you to understand the amount of time a function takes to execute
- The profiler will tell you the amount of time each line in the function takes to execute (as a percentage of the total execution time)
- Very useful tool to debug and optimize algorithms within a complex program
- Very useful to prototype CPU resources in other languages (such as C++ or C)

Example Profiling (isam.m function)

Profiling involves invoking a series of commands as shown below for function isam

The function to be profiled is executed after the profiler is invoked

profile on

```
[mtrue,a_alt,rho,temp] = isam(9000,350)
```

profile viewer

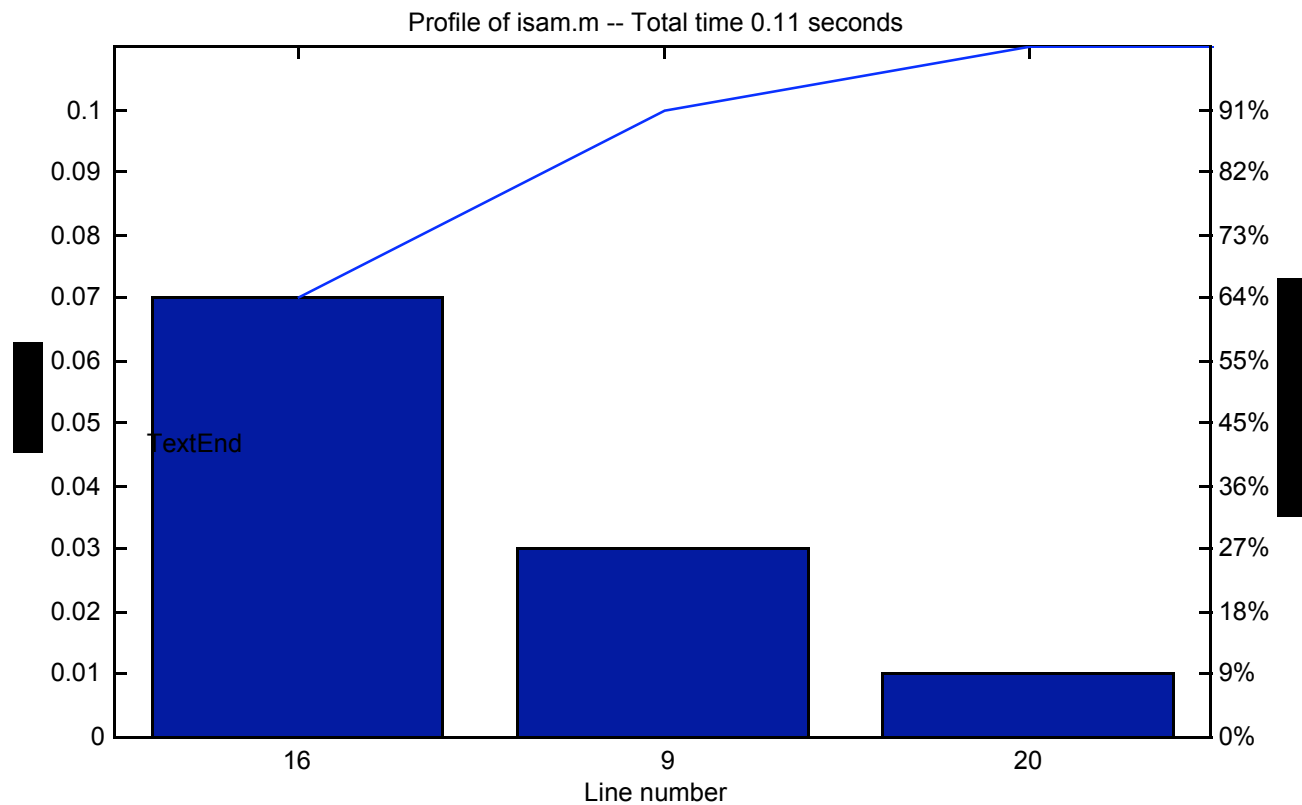
```
p = profile('info');
```

```
profsave(p,'profile_results')
```

- The profiler report provides numerical values on execution CPU time per line

Pareto Plot of Profiler on isam.m

The following Pareto plot illustrates graphically the amount of CPU time spent on each line



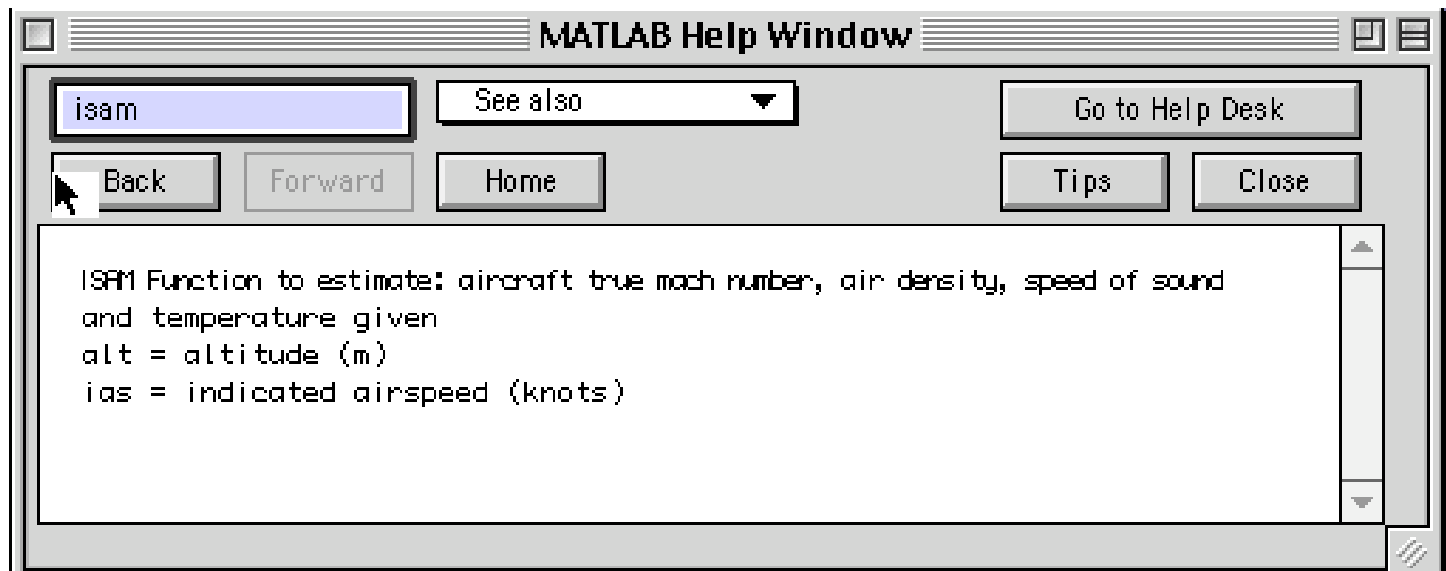
Function Comments as On-line Help

- MATLAB creates a help tag for user-defined functions
- The first comment lines in every function are used as on-line help
- For example, function isam has the following comment lines at the start of the function

```
% ISAM Function to estimate: aircraft true mach number, air  
    density, speed of sound  
% and temperature given  
% alt = altitude (m)  
% ias = indicated airspeed (knots)
```

Help Box for isam Function

Invoking `helpwin` (at the command line) and typing `isam` in the help box yields the following figure.



Note that all functions are catalogued automatically by MATLAB