CS101

INTRODUCTION TO COMPUTING FOR ENGINEERS AND PHYSICAL SCIENTISTS

Lectures 6&7

What will I learn in this lecture?

Dynamics, two examples of the use of ode45.

Numeric Integration using trapz and quad/quadl functions.

Readings: Matlab by Pratap Chapter 5.4,5.5

Dynamics: time evolution of physical processes

1. Problem Definition

Use Matlab to plot the velocity of a free-falling object. Assume that the object is near the earth's surface, so that the force due to gravity is given by mass * g where 9.8 m/s^2. Further assume that air friction is present and that the force due to air friction satisfies $F_{air\ friction} = b * v^2$, where b is constant and v is the velocity of the falling object (downward is negative velocity).

2. Refine, Generalize, Decompose the problem definition (i.e., identify sub-problems, I/O, etc.)

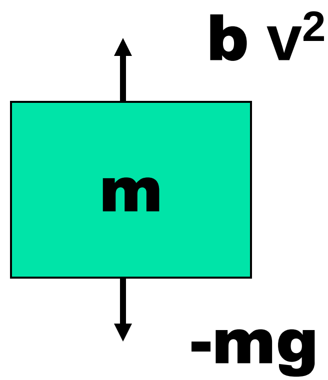
Mass of object is 70Kg , acceleration due to gravity is -9.8 m/s^2 and the coefficient of air friction b = .25 Kg/s. At time = 0 assume v=0, that is,

$$v(0) = 0 \qquad \text{(initial condition)}$$

Falling object

2. (continued)

From Newton's 2^{nd} law, the sum of the forces on the falling object equal it's mass times acceleration:



$$m*\frac{dv(t)}{dt} = -m*g + b*v^2(t)$$

$$\frac{dv(t)}{dt} = -g + (b/m)*v^2(t)$$

Falling object

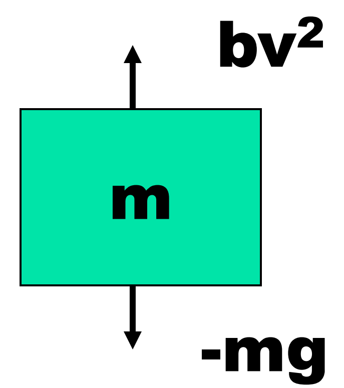
2. (continued)

Using calculus we can rewrite the differential equation in the form:

$$dv = \left(-g + \frac{b}{m} * v^2(t) \right) dt$$

and integration of both sides gives,

$$v(t) = -\sqrt{\frac{mg}{b}} \tanh\left(\frac{gt}{\sqrt{\frac{mg}{b}}} \right)$$



Falling object

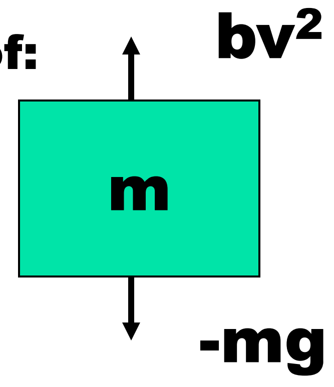
2. (continued)

Since we can write tanh(x) as:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

when x -> infinity tanh(x) -> 1 and so the falling body has a terminal velocity of:

$$v(\infty) = -\sqrt{\frac{mg}{b}}$$



Matlab ode45 function

3. Develop Algorithm (processing steps to solve problem)

Rather than using calculus to solve this problem we can use a built-in Matlab function... ode45. The format of this function is:

```
[t, v] = ode45(@aux_function, time, initial_condition);
```

aux_function: user created function that computes the derivate

time: a vector [start, stop] for the range of time of the solution

initial_condition: initial value v(start)

t: solution column vector of time values on range [start, stop]

v: solution column vector velocity values at times t

Falling object

4. Write the "Function" (Code)

(instruction sequence to be carried out by the computer)

Use the Matlab editor to create a file vprime.m .

```
function vp = vprime(t,v)

%  function vp = vprime(t,v)
% compute dv/dt
m = 70;

g = 9.8;

b = .25;
vp = -g +(b/m)*v.^2;
```

$$\frac{dv(t)}{dt} = -g + (b/m) * v^2(t)$$

Falling object

5. Test and Debug the Code

To test our program in Matlab, we will plot velocity versus time for t=0 seconds to t=10 seconds,

```
plot(t,v)
```

6. Run Code

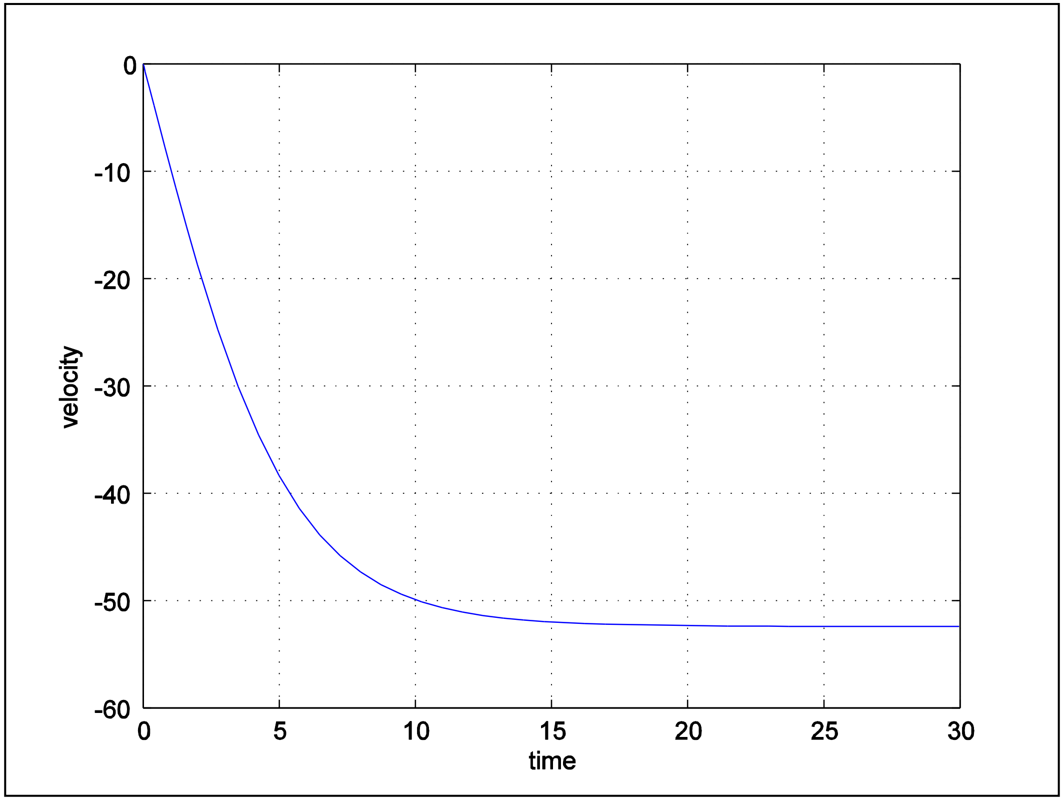
To run our program we will use ‘ode45’ as follows:

```
[t, v] = ode45(@aux_function, time, initial_condition);
```

```
>> [t, v] = ode45(@vprime, [0,30], 0);
>> plot(t,v)
```

(See plot on next slide.)

Falling object



As time increases the velocity levels out around -52.3818 m/s. This velocity is called the terminal velocity of the object. This agrees with the solution since v(infinity) = -sqrt(m*g/b) = -52.3832

Use Matlab to solve 2nd order ODE's.

1. Problem Definition

Use Matlab to plot both the velocity and position(height) of a free-falling object.

2. Refine, Generalize, Decompose the problem definition

(i.e., identify sub-problems, I/O, etc.)

Mass of object is 70Kg , acceleration due to gravity is 9.8 m/s^2 and the coefficient of air friction b = .25 Kg/s.

Initial conditions: at time = 0 assume v=0 and y = 2000m.

Plot the height y and the velocity v versus time for t = 0 to t = 30 seconds.

Falling object2

3. Develop Algorithm (processing steps to solve problem)

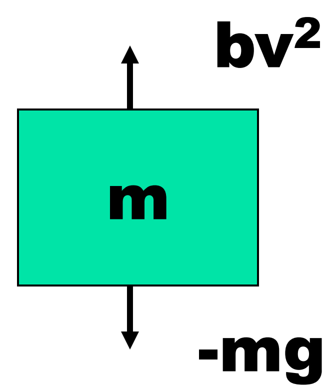
The ODE describing the motion of a falling object is:

$$m\frac{d^2y}{dt^2} = -mg + b\left(\frac{dy}{dt}\right)^2$$

which is equivalent to the following system of ODEs:

$$\frac{dy}{dt} = v$$

$$\frac{dv}{dt} = -g + \frac{b}{m}v^2$$



Falling object2

4. Write the "Function" (Code)

(instruction sequence to be carried out by the computer)

Use the Matlab editor to create a file yvprime.m .
Function yvprime has two inputs:
- **t: a scalar value for time**
- **yv: a vector containing [y , v] values at time = t**

Function yvprime has one output:
- **yvp: a vector containing [dy/dt ; dv/dt]**

```
function yvp = yvprime(t, yv)
%  function yvp = yvprime(t, yv)
m = 70;

g = 9.8;

b = .25;

y = yv(1);

v = yv(2);
yvp = [v ;  (-g +(b/m)*v.^2)];
```

$$\frac{dy}{dt} = v$$

$$\frac{dv}{dt} = -g + \frac{b}{m} v^2$$

Falling object2

6. Run Code

To run our program we will use 'ode45' in a slightly different way as follows:

```
[t, yv] = ode45(@yvprime, [0,30], [2000;0]);
```

```
[t, yv] = ode45(@aux_function, time, initial_conditions);
```

aux_function: user created function that computes the derivates

time: a vector [start stop] for the range of time of the solution.

initial_condition: initial value(s) [y(start) ; v(start)]

t: solution column vector of time values from [start, stop]

yv: solution matrix with two columns [y , v] representing values y in the first column and v in the second at time t

Falling object2

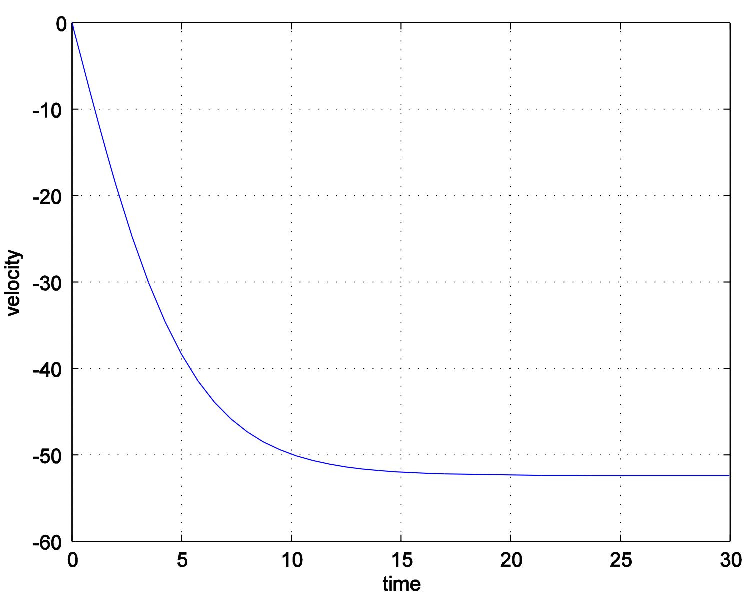
5. Test and Debug the Code

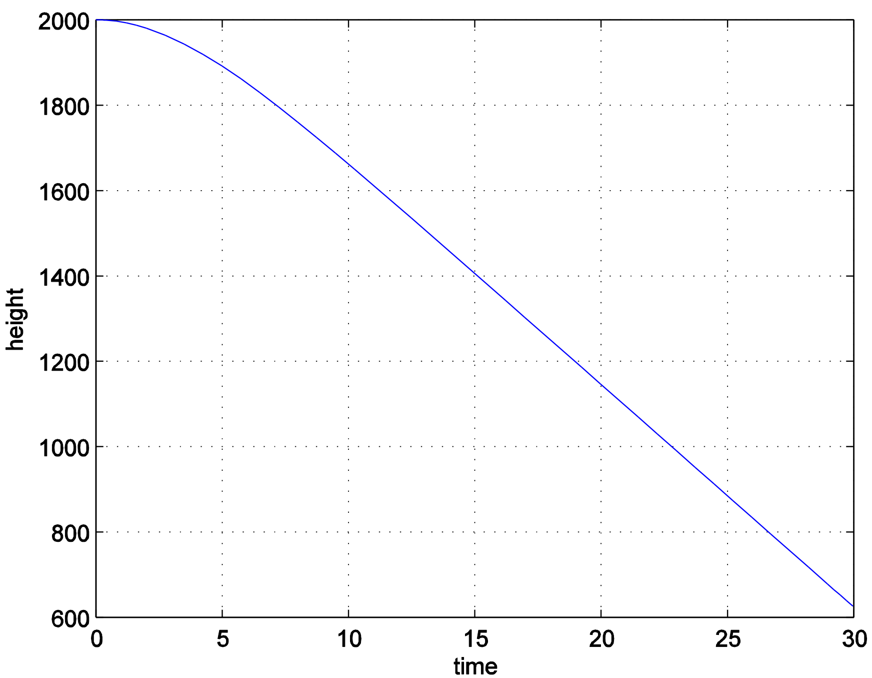
To test our program in Matlab, we will plot velocity versus time

```
>> plot(t,yv( : , 2 ))
```

and we will plot y versus time,

```
>> plot(t,yv( : , 1))
```





Falling object2

5. Test and Debug the Code

Matlab produces a discretized solution

>> yv

>> t

```
>> t

t =

                   0
   0.000005126298840
   0.000010252597680
   0.000015378896519
   0.000020505195359
   0.000046136689558
   0.000071768183757
   0.000097399677956
   0.000123031172156
   0.000251188643151
   0.000379346114146
   0.000507503585142
   0.000635661056137
```

```
>> yv

yv =

  1.0e+003 *

   2.000000000000000                   0
   1.999999999999871  -0.000000050237729
   1.999999999999485  -0.000000100475457
   1.999999999998841  -0.000000150713186
   1.999999999997940  -0.000000200950915
   1.999999999989570  -0.000000452139558
   1.999999999974762  -0.000000703328201
   1.999999999953515  -0.000000954516844
   1.999999999925830  -0.000001205705487
   1.999999999690831  -0.000002461648701
   1.999999999294873  -0.000003717591912
```

Matlab ODE solvers

Matlab offers a family of ODE solvers that use different methods to solve ODEs:

ode45, ode15i, ode15s, ode23, ode23s, ode23t, ode23tb, ode113

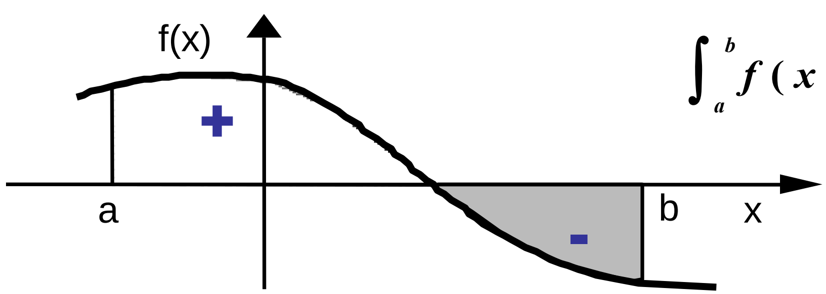
Matlab does NOT guarantee that the solution any of the solvers produces is accurate.

Knowledge that a particular solver will produce an accurate solution for a given ODE is beyond the scope of this course. CS 357 / CS 457 are courses in numerical analysis that cover methods of solving ODEs.

Numerical Integration

Most integrals arising from solutions of problems in engineering and the physical sciences cannot be represented in "closed form"- they must be evaluated numerically.

- **For a function of a single variable, we seek an approximation to the area "under" the curve:**



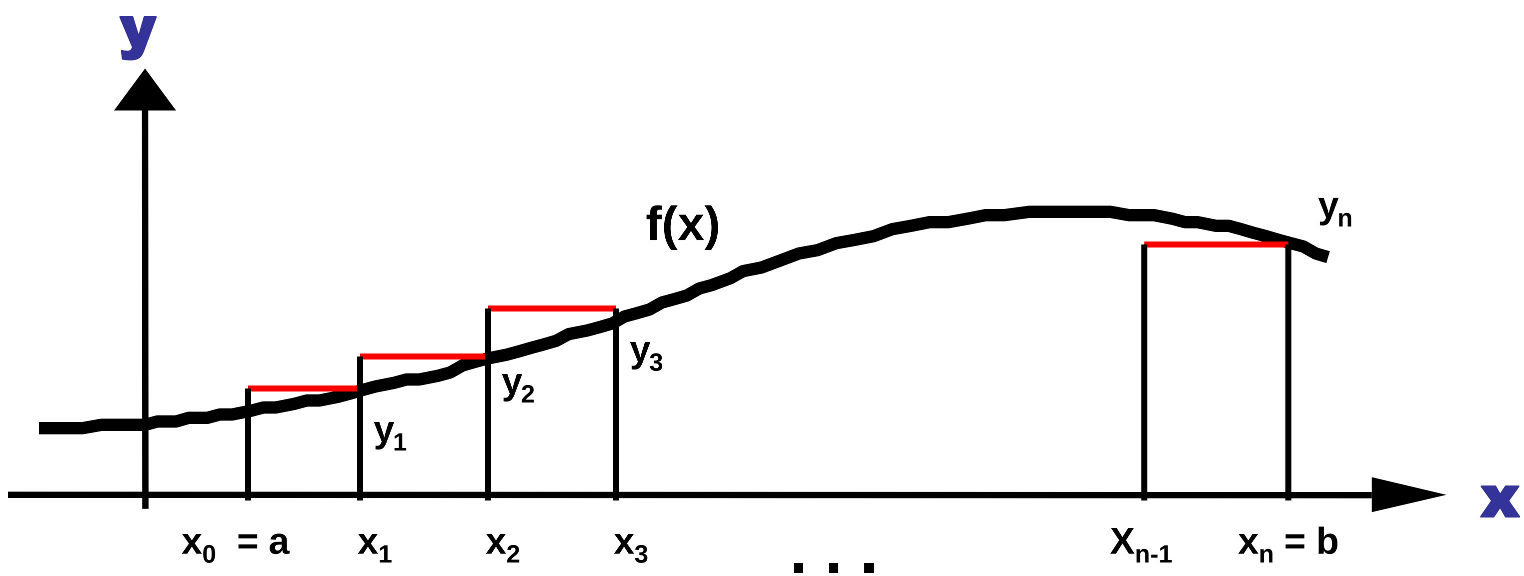
Definite Integral

$$\int_a^b f(x)\,dx = \text{ Sum of the “signed” areas}$$

The Matlab functions: quad (quad8) and trapz only apply to continuous functions f(x) of one variable - x (a scalar).

Integration - Using Rectangles

Approximation of f(x) by using constant functions on the intervals $[x_{k-1}, x_k]$ (not a good approximation).

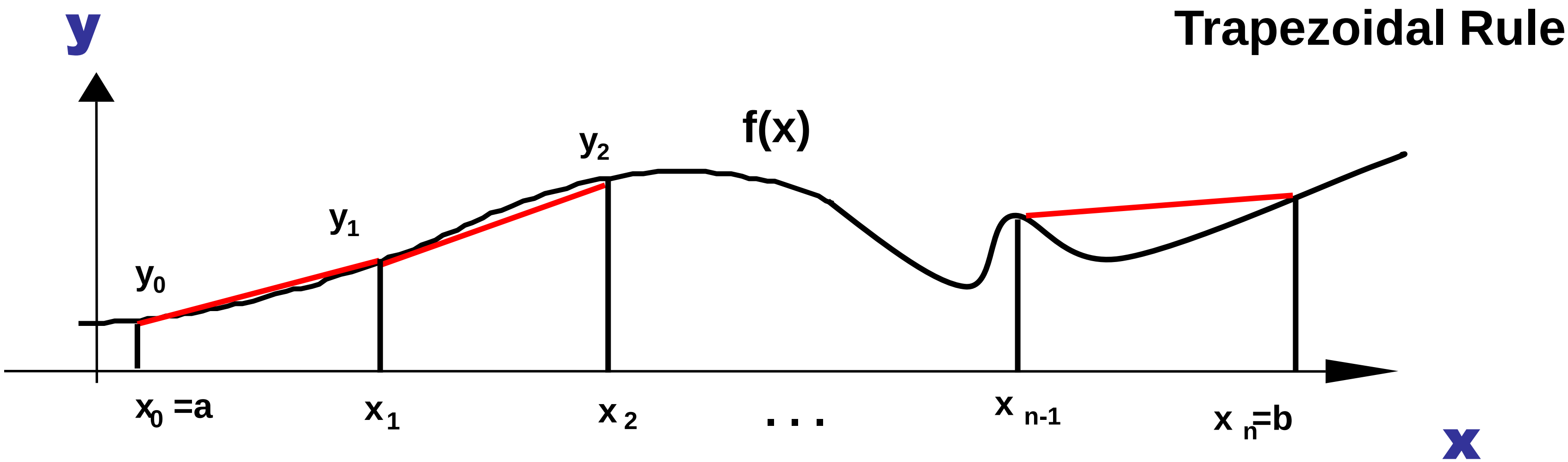


In this case, we approximate the integral by adding the areas of a set of approximating rectangles.

$$f(x) \approx f(x_k) \qquad x_{k-1} \le x \le x_k, \quad k = 1, 2, \ldots, n$$

Integration - Using Trapezoids

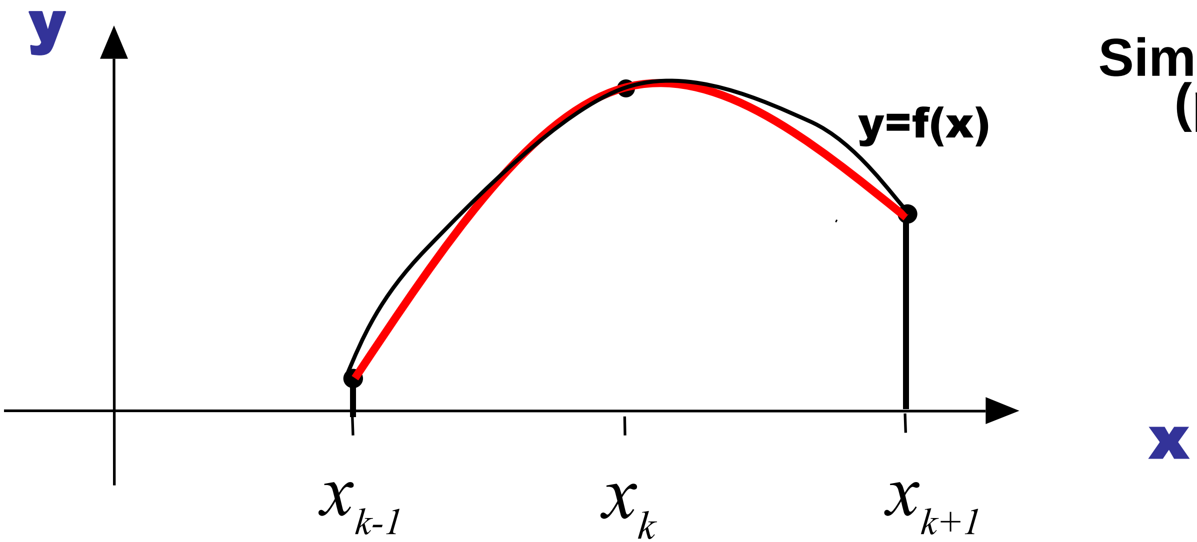
Approximation of f(x) by using a linear function on the intervals $[x_{k-1}, x_k]$ (a better approximation).



In this case, we approximate the integral by adding the areas of a set of approximating trapezoids.

Integration - Using Parabolas

Approximation of f(x) by using a quadratic function on the intervals $[x_{k-1}, x_{k+1}]$ (better approximation).



Simpson's Rule (parabola)

In this case, we approximate the integral by adding the areas under a set of approximating parabolas.

trapz and quadl - Matlab functions

The two built-in Matlab functions (we will use in CS101) for integration are trapz and quad.

- **The function trapz trapezoidal rule (Pratap p. 152) is used when we don't know f(x) explicitly but we have a vector x which is a partition of the interval [a,b] and a vector y = f(x). That is, we know the y values of f(x) only for some discrete set of x values.**
- **We can use quad(Pratap 5.4 adaptive Simpson's Rule) or quadl (Pratap 5.4 adaptive Lobatto) when we know the function f(x). That is, f(x) is a built-in Matlab function or a user defined function.**

Function - trapz

Example:

Use Matlab to compute the integral:

$$\int_a^b f(x)\,dx$$

where f(x) = sin(x) **and a = 0 , b = pi.**

```
>> format long
>> x = linspace(0,pi,1000);
>> y = sin(x);
>> trapz(x,y)
ans =
        1.99999989714020
```

Function - quadl

Example:

Use Matlab to compute the integral:

$$\int_a^b f(x)\,dx$$

where f(x) = sin(x) and a = 0 , b = pi.

```
>> quadl(@sin,0,pi)
ans    =
        1.9999999774713
```

What have I learned in this lecture?

We can use the Matlab function **ode45** to solve a system of ordinary (not partial) differential equations.

For numeric integration we can use **quadl** when we know the function f(x). The function **trapz** is used when we don't know f(x) explicitly.