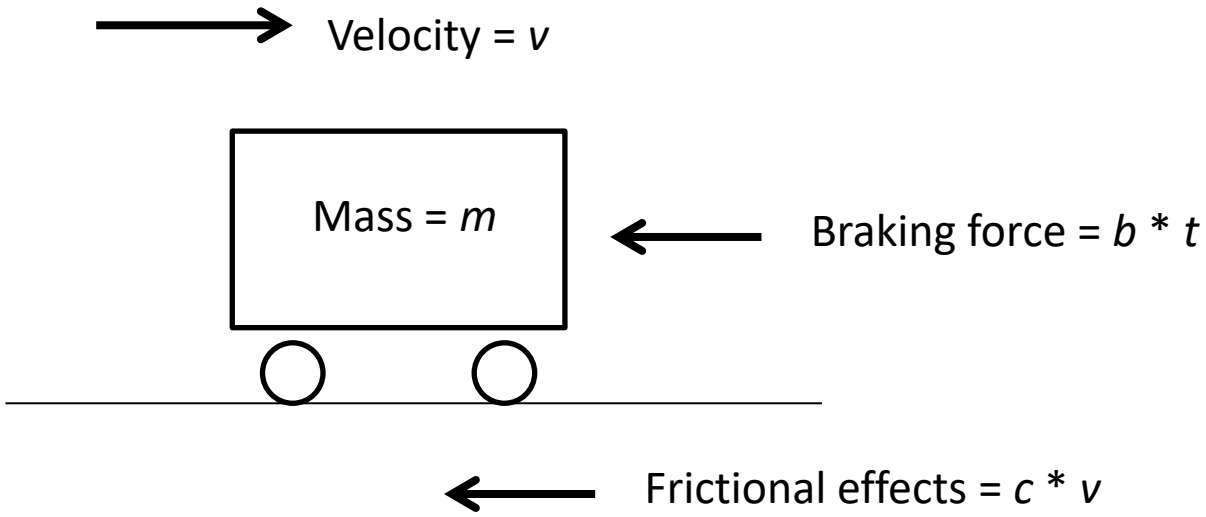
Problem:



Assume: $b = 10$ N/s
$c = 1.45$ N/(m/s)
$m = 200$ kg
$v(0)$ = initial velocity = 20 m/s

How will the velocity vary over time? What is $v(t)$?

Analysis:

$F = ma \quad => \quad a = F/m$

F = braking force + frictional force $= -((b * t) + (c * v))$

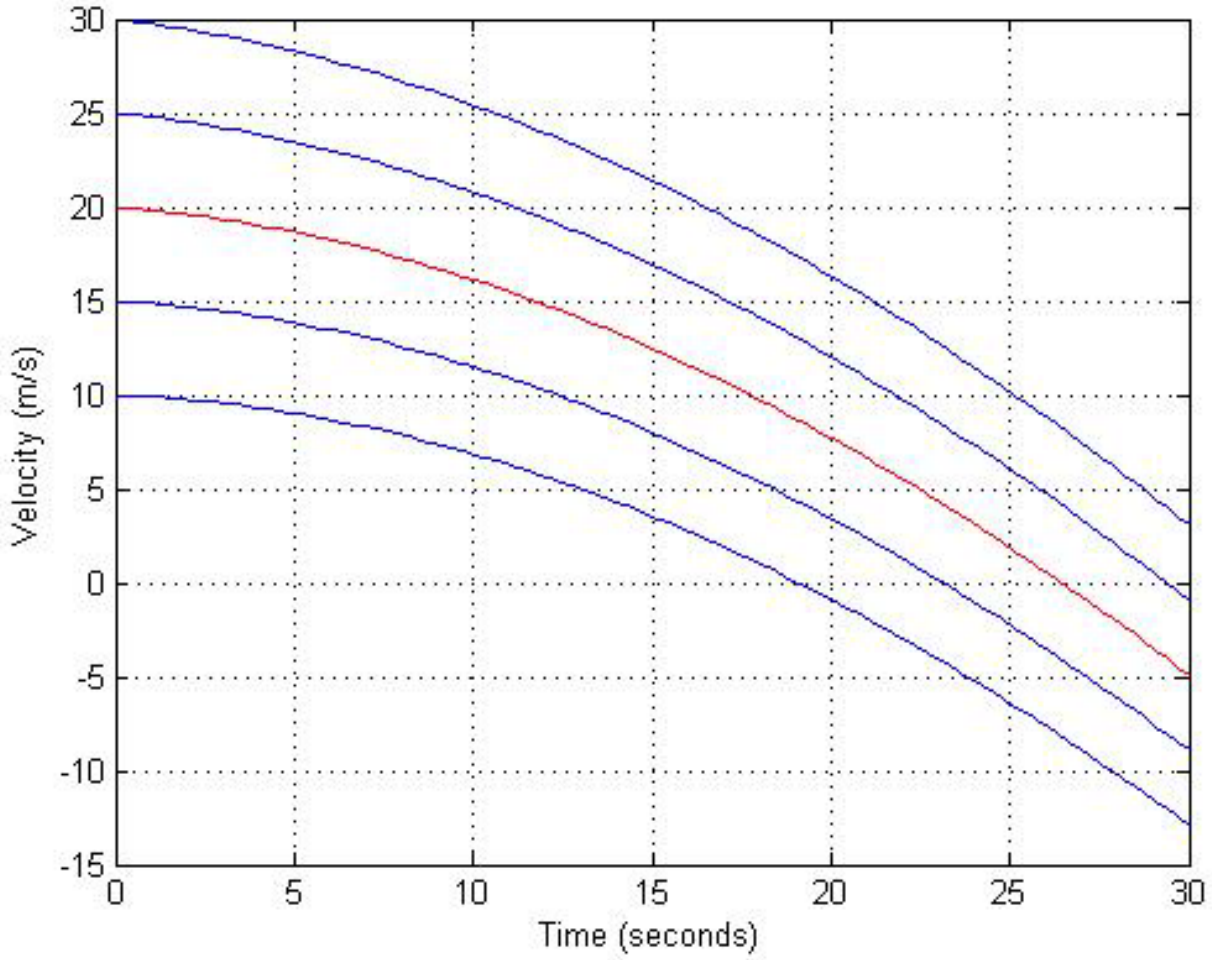
(the minus sign is because the force acts to the left)

Therefore $a = [-((b * t) + (c * v))] \ / \ m$

Replacing a with dv/dt gives $$\frac{dv}{dt} = \frac{-(bt + cv)}{m}$$

This is a first order (1st derivative only) Ordinary Differential Equation (ODE)

There is one independent variable (time t) and one dependent variable (velocity v).



There are an infinite number of curves that satisfy the differential equation. All have the correct slope (dv/dt) at every possible combination of t and v.

Only one of these curves satisfies the given initial condition ($v(0) = 20$ m/s). This curve is the desired $v(t)$.

Initial Value Problem:

Differential equation (slope function) + Initial Condition

$$\frac{dv}{dt} = \frac{-(bt + cv)}{m} \qquad v(0) = 20$$

Solution

$$v(t) = ...$$

The slope function gives dv/dt for all values of t and v. It defines a family of curves.

The initial condition requires that the solution passes through a particular point.

Analytical Solution:

As always, analytical solutions are best when they are possible. In this case

$$v(t) = v(0)e^{-t/\tau} + (-b/c)(\tau e^{-t/\tau} - \tau - t)$$

$$\text{where } \tau = \text{time constant of system} = m/c$$

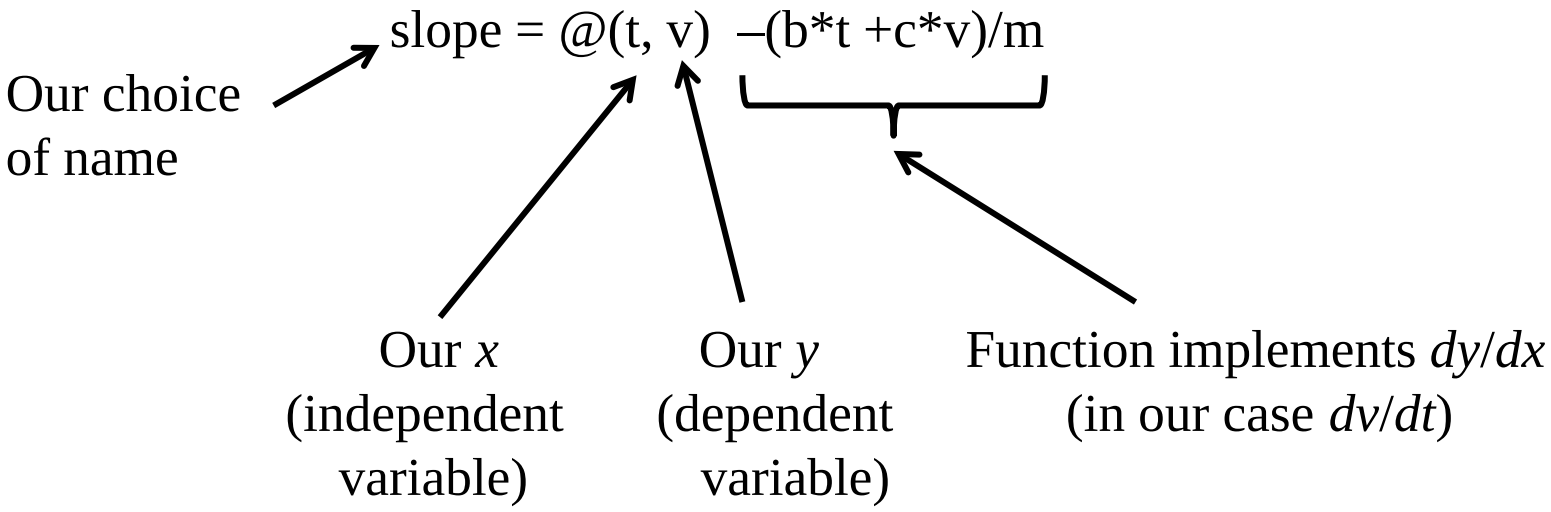
Aside: The response is the sum of the system's response to the initial conditions and its response to a ramp input. SYSC 3600 deals with the details.

In cases in which an analytical solution cannot be obtained numerical methods provide an alternative means of obtaining a solution.

Matlab Solution:

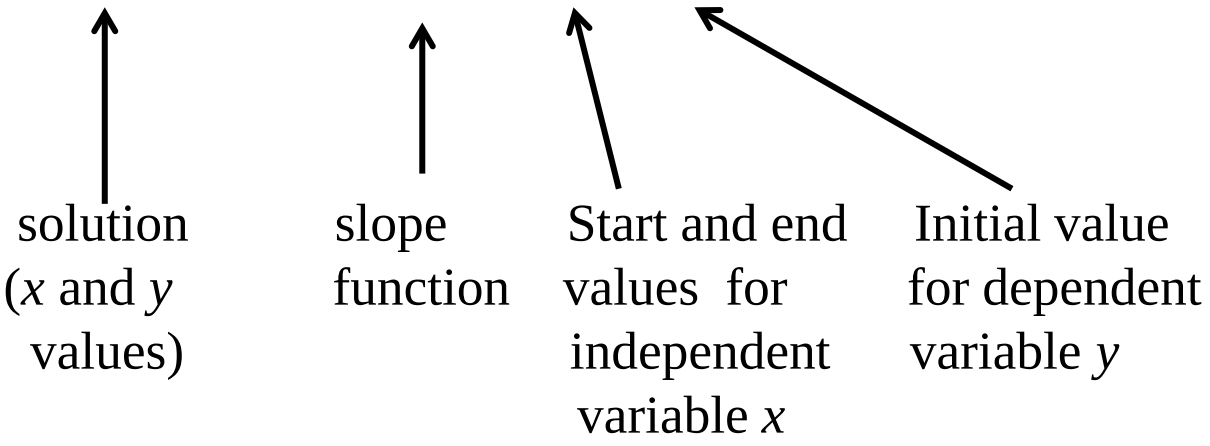
Step 1: Define a function that implements the differential equation. If the independent variable is *x* and the dependent variable is *y*, then this function must accept *x* and *y* and return d*y*/d*x*.

In our case the independent variable happens to be called *t* and the dependent variable happens to be called *v*. The actual names used don't matter.



Step 2: Use ode45 to solve the problem.

```
[t, v] = ode45 (slope, [0 30], v0);
```



The solution consists of two vectors:

i) A vector of values of the independent variable *x*
ii) A vector of corresponding values of the dependent variable *y*

In our case vector *t* will contain time values and vector *v* will contain corresponding velocity values.

If only initial and final *x* values are specified *ode45* chooses the spacing of the output points.

The number and spacing of the output points can be controlled by providing a list of values for the independent variable instead of providing just initial and final values.

Example:

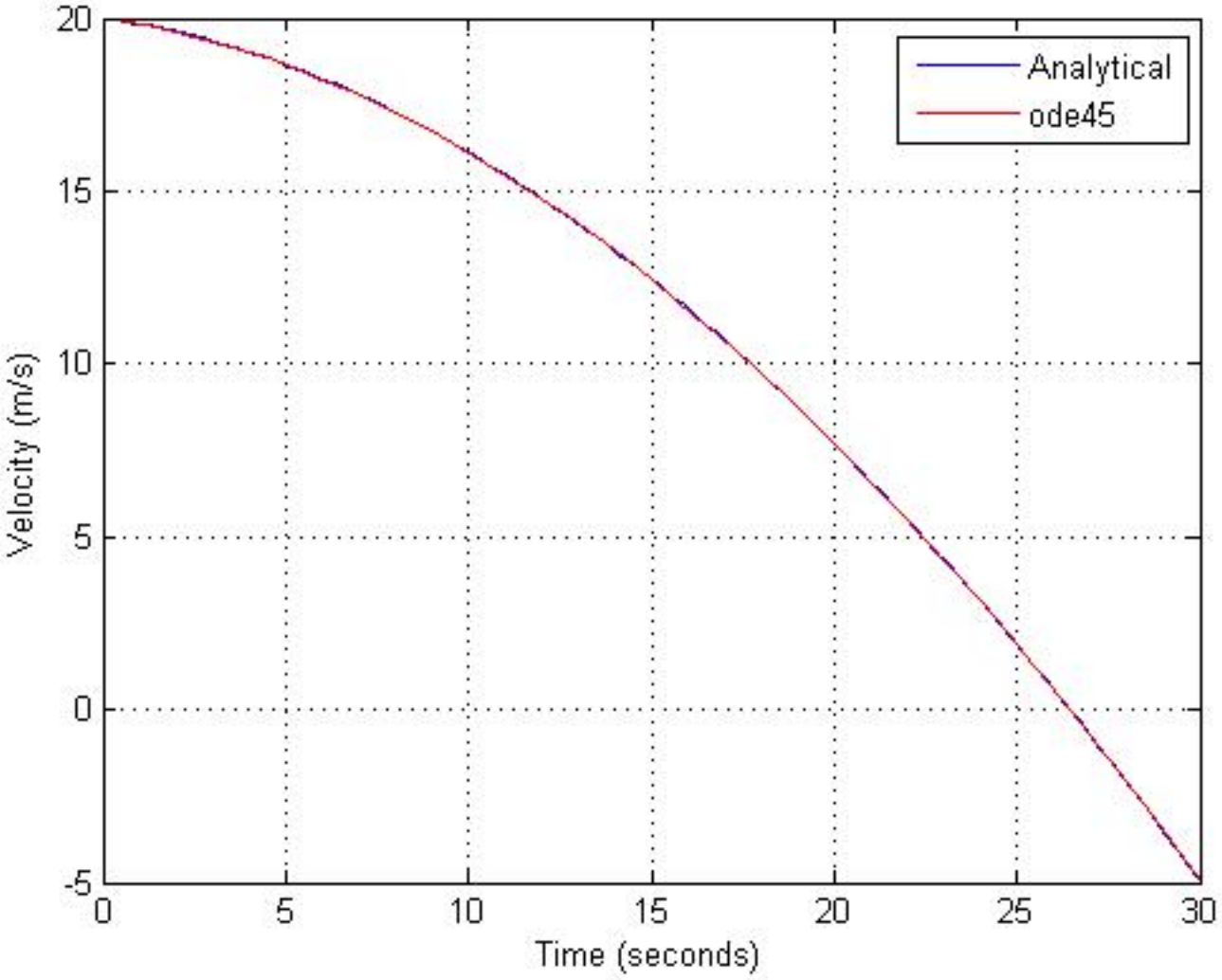
```
ta = linspace (0, 30, 101);
[t,v] = ode45 (slope, ta, v0);
```

Vector *t* will contain exactly 101 evenly spaced values starting with 0 and ending with 30.

Vector *v* will contain the corresponding velocity values

The graph on the next slide compares the analytical and *ode45* solutions.

Unless the slide is viewed in colour no differences can be seen and if the output times are aligned (as described above) absolutely no difference can be seen even in colour (the two solutions are effectively identical).



See basicCartDemo.m

Matlab offers a whole family of ODE functions (ode23, ode113, ode23s, etc.)

All do the same job and have the same inputs and outputs.

The differences lie in the details of how a solution is obtained.

ode45 – medium accuracy, best for non-stiff systems, usual first option

ode15s – low to medium accuracy, best for stiff systems

etc. (use HELP *odexxx* for more information)

A “stiff” system is one that combines rapidly changing components with slowly changing ones (e.g. involves both periods of rapid change and periods of slow change).

Euler's Method

This is the simplest possible method of solving ODE's numerically.

The following transformation is of course not mathematically legitimate. The second equation is an approximation that is only strictly valid as $x_{i+1} - x_i$ approaches zero.

$$\frac{dy}{dx} = f(x, y) \quad \Rightarrow \quad \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = f(x_i, y_i)$$

Rearranging the second equation gives an iterative formula:

$$\frac{y_{i+1} - y_i}{x_{i+1} - x_i} = f(x_i, y_i) \quad \Rightarrow \quad y_{i+1} - y_i = f(x_i, y_i)(x_{i+1} - x_i) \quad \Rightarrow \quad y_{i+1} = y_i + f(x_i, y_i)\Delta x$$

Starting with the known initial value of y (y_0) the formula is applied repeatedly to generate subsequent values of y.

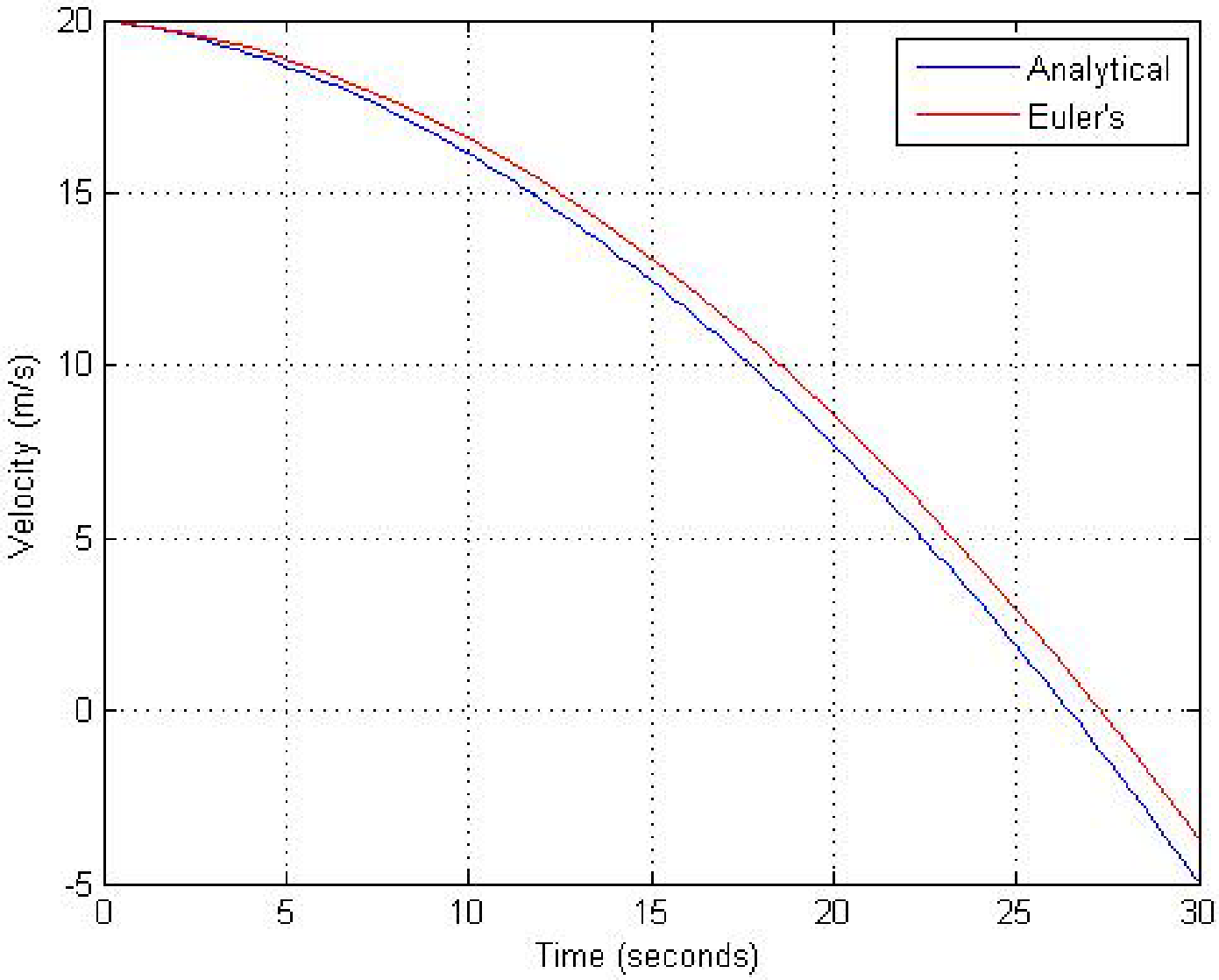
Euler's Method Example

Cart problem with c = 1.5 m/s, b = 10 N/s, m = 200 kg, v_0 = 20 m/s

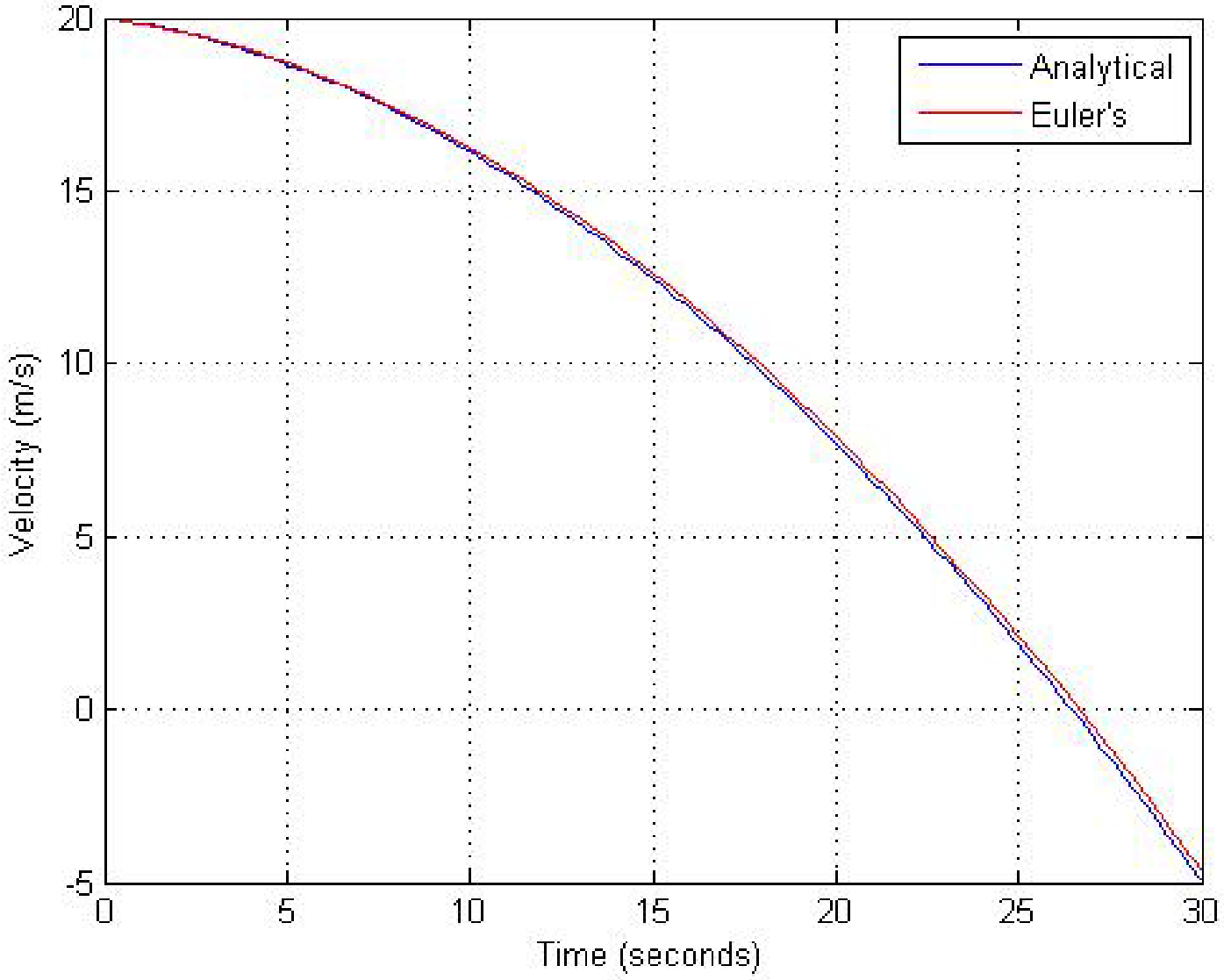
$$f(t,v) = -(b*t + c*v)/m = -(10*t + 1.5*v)/200$$

Assume Δt = 2 seconds (rather high)

i	t_i *(secs)*	Formula for v_i	v_i *(m/s)*
0	0	N/A (initial value)	20
1	2	$v_0 + f(t_0, v_0)\,\Delta t$	19.7
2	4	$v_1 + f(t_1, v_1)\,\Delta t$	19.2045



Comparison of analytical solution and solution obtained using Euler's method with Δt = 2 seconds (see EulerDemo.m).



Comparison of analytical solution and solution obtained using Euler's method with Δt = 0.5 seconds. Within reason smaller steps make for a better solution.

Errors in Euler's Method

Truncation Error: The method assumes that the slope remains constant over each interval. This is equivalent to truncating the Taylor series for y_{i+1} after just the first derivative.

The single step truncation error propagates through subsequent steps to give a global truncation error.

one step error: $O(h^2)$

global error: $O(h)$

Round-off Error: Every iteration involves calculations and round-off. These errors also propagate.

Stability of Euler's Method

The method is stable for most functions and small step sizes.

But consider $\frac{dy}{dx} = -ay \quad \Rightarrow \quad y = y_0 e^{-ax}$

For all a >= 0 the system itself is stable.

The Euler's Method iterative formula is $y_{i+1} = y_i + (-ay_i)h = y_i * (1 - ah)$

If $h > 2/a$ the "amplification factor" (1-ah) will have an absolute value greater than 1 and the absolute value of y will increase forever.

Example:

Assume $y_0 = 10$, $a = 4$ and $h = 2$

$1 - ah = -7$

$y_1 = -70$, $y_2 = 490$, $y_3 = -3430$, $y_4 = 24010$, $y_5 = -168070$, $y_6 = 1176490$, etc.

More on the Cart Problem

Suppose we would like to determine the time at which the braking force will bring the cart to a stop (i.e. the time at which $v(t)$ becomes zero).

We could write a function that uses *ode45* to determine the velocity at some given time *T* and then root find using this function (see stoppingTime.m)

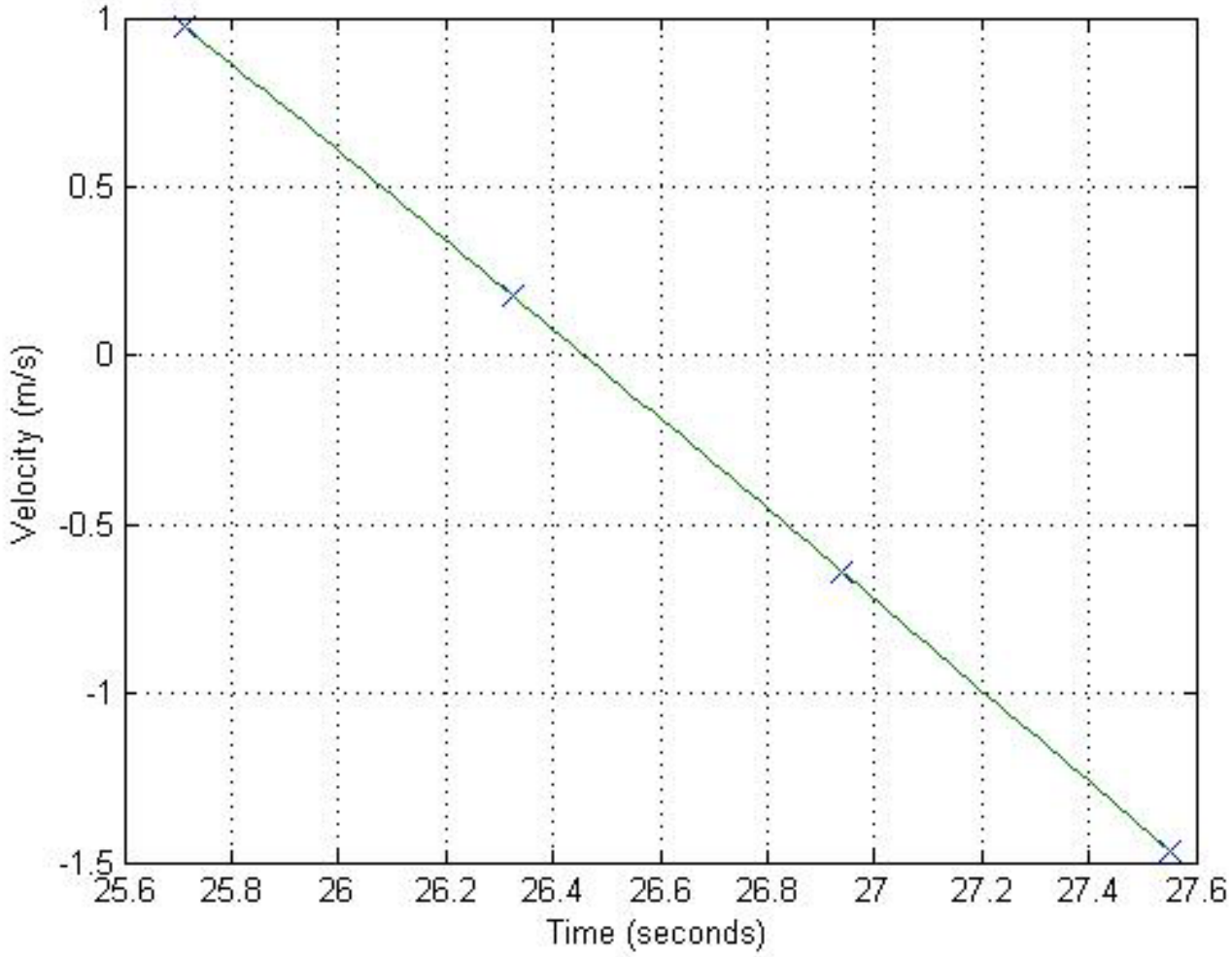
```
function [vAtT] = velocityAtTimeT (T)
   c = 1.5;  b = 10;  m = 200; v0 = 20;
   slope = @(t,v) -(c * v + b * t) / m;
   [t,v] = ode45(slope, [0 T], v0);
   vAtT = v(length(v));  % extract velocity for last time (time T)
end

solution = fzero (@velocityAtTimeT ,  [25 30]);  % limits taken from graph
```

This will work but is very inefficient.

The ODE solution must be recreated from scratch on every iteration of the search for the root.

A better idea: Fit a third order polynomial to the four data points surrounding the point at which the velocity changes sign and then find the point at which this polynomial is zero.



The required code is given below (see stoppingTime.m).

```
% solve ODE once
[t,v] = ode45 (slope, linspace (0, tEnd, 50), v0);  % 50 points

% find i such that v(i) is positive and v(i+1) is negative
for i = 1 : length(v) - 1
   if sign(v(i)) ~= sign(v(i + 1)); break; end;
end

% fit a third order polynomial to the 2 points on either side of the change
p = polyfit(t(i-1:i+2), v(i-1:i+2), 3);

% root find on this polynomial
f = @(t) polyval (p, t);
tzero = fzero(f, [25, 30]);
```

A better approach yet is to have *ode45* stop solving when the velocity becomes zero and to have it tell us the time of this "event".

This requires creating an "events function" that will be invoked repeatedly by ode45 as the solution proceeds.

This function is given the current values of *x* and *y* (*t* and *v* in our case) and must return

1/. A value. Events are triggered by changing the sign of the value returned from one function call to the next.

2/. A flag that indicates whether the occurrence of an event is to terminate the ODE solution.

3/. A flag that indicates whether ode45 is to only watch for +ve to –ve changes in the value returned (flag = -1) , only -ve to +ve changes (flag = +1), or whether the type of change doesn't matter (flag = 0).

Note: The returned quantities are actually vectors – one can check for many possible events "in parallel". For details use "help ode45".

A suitable function for our problem is

```
function [ value, isTerminal, direction ] = stopper (t, v)
   value = v;   % the value returned will change sign when v changes sign
                % (i.e. an event will occur when v changes sign)
   isTerminal = true;   % solution is to stop when the event occurs
   direction = 0;  % direction of change does not matter
end
```

Letting *ode45* know about the function involves setting up an options structure.

```
options = odeset ('Events', @stopper);
[t,v, eventT] = ode45 (slope, [0 30], v0, options);
```

The additional output (eventT) is a vector containing the *x* values (in our case, times) for all events. In our case there will only be one event and hence only one value.

```
fprintf (‘The velocity hits zero at t = %f\n', eventT(1));
```

See stoppingTime.m.