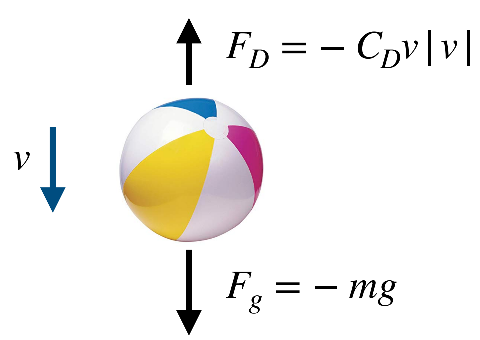
Lecture 13: Using Events to Measure and Control ODE Simulations

Example: Falling Motion with Air Drag

A. Single Solution
B. Family of Solutions with a Varying Parameter
C. Using Events to End the Simulation
D. Combining Events with a Family of Solutions
E. Use of Multiple Events During the Simulation

Free Fall Motion of a Ball with Turbulent Air Drag



Equation of Motion:

$$\boxed{\frac{d^2x}{dt^2} = -g - \frac{C_D}{m} v|v|}$$

A. Single Numerical Solution

Review of Steps:

1. Define an m-file function **`ode04_derivs.m`** that returns two derivatives: dx/dt and dv/dt

In a separate Matlab program **`ode04.m`**, do the following:

2. Initialize all parameters, initial conditions, etc.
3. Call the Matlab function **`ode45()`** to solve the ODE.
4. Separate out x(t) and v(t) solutions
5. Plot the results

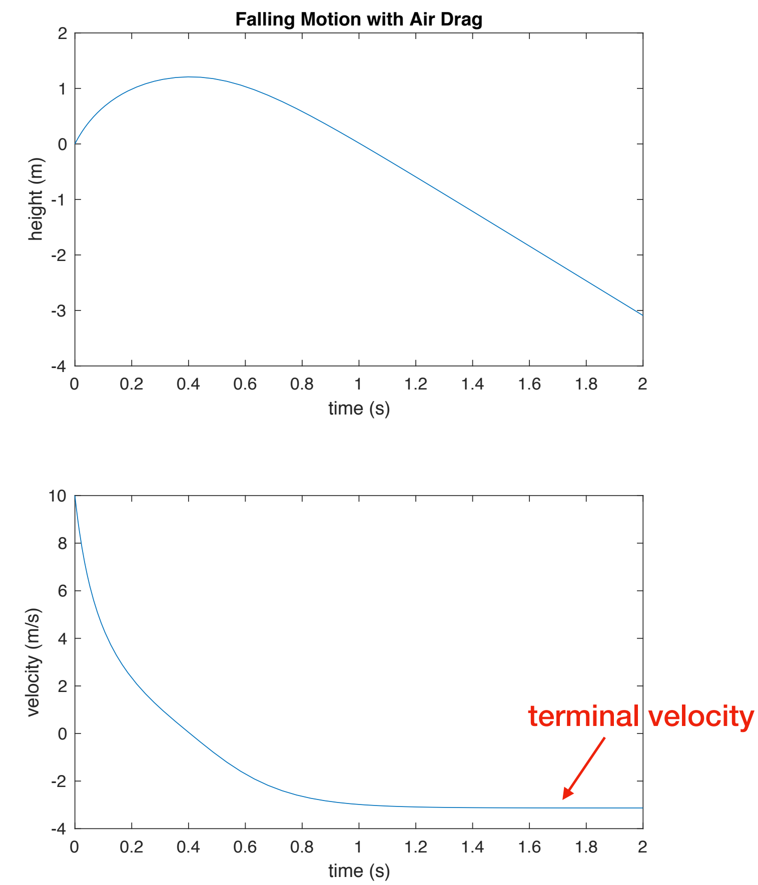
ode04A.m

```
% Initialize Parameters
tBegin = 0;      % time begin
tEnd   = 2;      % time end
x0 = 0;          % initial position (m)
v0 = 10;         % initial velocity (m/s)

% global variables
global m;        m   = 1;     % air drag
global C_d;      C_d = 1;     % mass

% Integrate ODE
[t,w] = ode45(@ode04_derivs, ...
          [tBegin tEnd], [x0 v0]);
y = w(:,1);      % extract x(t)
v = w(:,2);      % extract v(t)

% top plot - x(t)
subplot(2,1,1)
plot(t,y);
ylabel('height (m)');
xlabel('time (s)');

% bottom plot - v(t)
subplot(2,1,2)
plot(t,v);
ylabel('velocity (m/s)');
xlabel('time (s)');
```

ode04_derivs.m

```
function derivs = ode04_derivs(t, w)

global C_d;       % air drag
global m;         % particle mass

g = 9.8;          % g

y = w(1);         % w(1) stores x
v = w(2);         % w(2) stores v

% calculate dx/dt and dv/dt
dydt = v;
dvdt = -g - v * abs(v) * C_d / m ;

derivs = [dydt; dvdt];
```

Here's the result for ode04A:



Try the following:

1. Change the value of the drag coefficient, mass, and initial velocity to see how each affects the terminal velocity and maximum height.
2. Change the plot style from a solid line **`'-'`** to a line with circular markers **`'-o'`**. Where are the time steps largest? Where are they smallest?

B. Family of Solutions with a Varying Parameter (Example: Vary the Drag Coef.)

Often, it is helpful to plot a series of solutions as some parameter is varied. The “family” of solutions can provide insight into how the system behaves. In this example, we vary the drag coefficient to see how increasing air resistance affects the dynamics:

$$C_D = 0,\ 0.1,\ 0.5,\ 2$$

1. Define the C_D values in an array
2. Create a loop to solve the ODE and plot the result for each C_D value
3. After the loop, label the axes, create a legend, set plot limits, etc.

Code Snippets:

1. Define the C_D values in an array

```
C_d_list = [0 0.1 0.5 2];
```

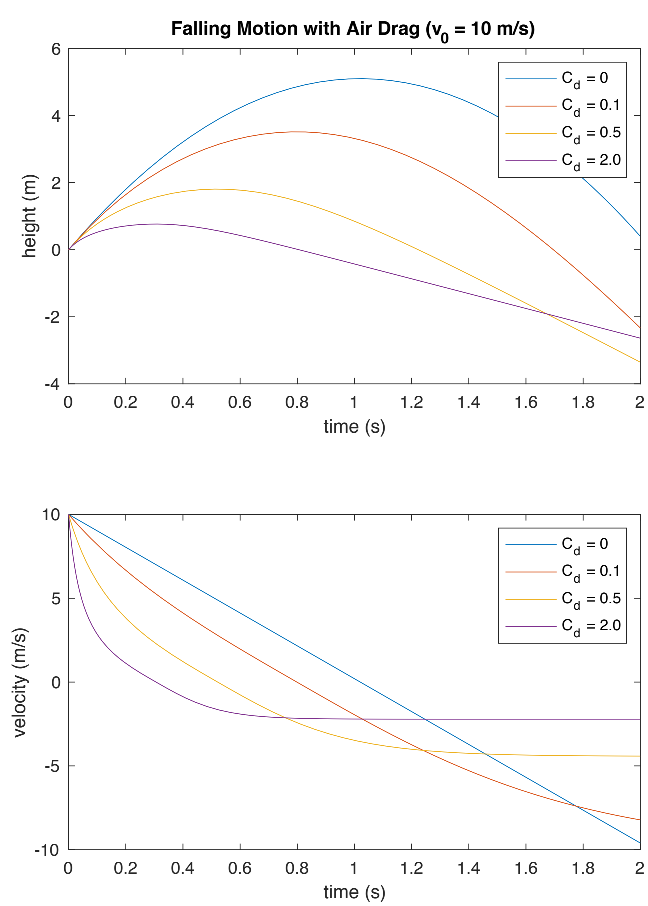
2. Create a loop to solve the ODE and plot the result for each C_D value

```
for k = 1:length(C_d_list)  % loop through all elements
                            % in C_d_list
    C_d = C_d_list(k);

    [t,w] = ode45(@ode04_derivs, [tBegin tEnd], [y0 v0]);
    y = w(:,1);
    plot(t,y, '-');
    hold on          % prevent next plot from overwriting
                     % current plot
 end
```

Excerpts of code **ode04B.m** (initialization not shown)

```
C_d_list = [0 0.1 0.5 2];          % drag coefs for four different runs

%%%%%%%%%%  Loop through each model run  %%%%%%%%%
for k = 1:length(C_d_list)

    C_d = C_d_list(k);  % assign the global variable C_d to the kth value in the C_d_array

    % Use the Runge-Kutta 45 solver to solve the ODE
    [t,w] = ode45(@ode04_derivs, [tBegin tEnd], [y0 v0]);
    y = w(:,1);     % extract positions from first column of w matrix
    v = w(:,2);     % extract velocities from second column of w matrix

    % top subplot graphs x vs t
    subplot(2,1,1)
    plot(t,y, ‘-‘);
    hold on

    % lower subplot graphs v vs t
    subplot(2,1,2)
    plot(t,v, ‘-‘);
    hold on
end

% title, labels, legend for top subplot
subplot(2,1,1)
str = sprintf('Falling Motion with Air Drag (v_0 = %.0f m/s)',v0);
title(str);
ylabel('height (m)');
xlabel('time (s)');
legend('C_d = 0', 'C_d = 0.1', 'C_d = 0.5', 'C_d = 2.0')

% title, labels, legend for lower subplot
subplot(2,1,2)
ylabel('velocity (m/s)');
xlabel('time (s)');
legend('C_d = 0', 'C_d = 0.1', 'C_d = 0.5', 'C_d = 2.0')
```

Here's the result for ode04B:



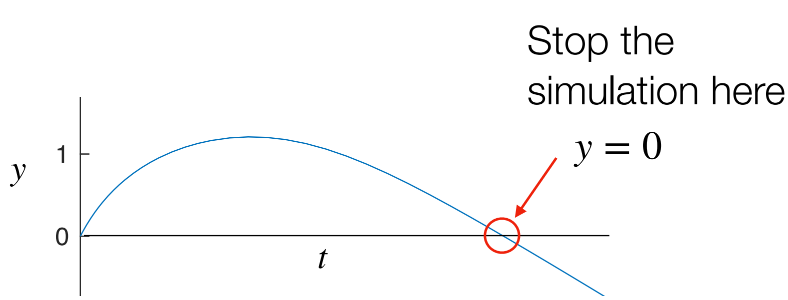
Try the following:

1. Add an additional curve to both the position and velocity plots, with $C_d = 1.0$. Include the new curve in the legend.

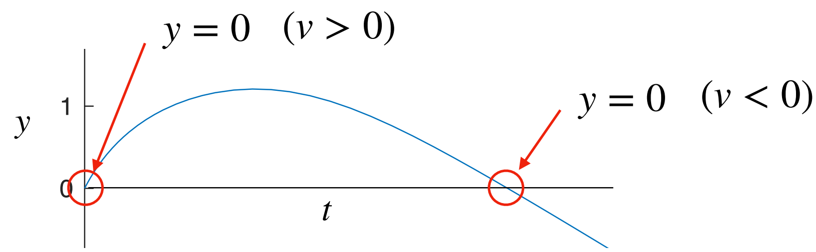
2. Draw the $C_d = 1.0$ curves as dashed lines to distinguish them from the others.

C. Using Events to End the Simulation (Example: Stop the simulation when the ball hits the ground)

Aim: Modify program ode04A.m so that the numerical integration stops when the ball returns back to ground level at y = 0.



We need a precise condition for stopping the simulation. The problem is that there are two solutions for the condition y = 0: the desired one and one at t = 0 when the ball is thrown upward.



We can distinguish these two events by noticing that the first event occurs as y is increasing in value (v > 0) at launch, while the second event occurs when y is decreasing in value (v < 0) when the ball falls and hits the ground.

Matlab Event Handler

The Matlab event handler works as follows. The user defines an event with a specified function $g_{event}(x, v, t)$:

1. An event is triggered when the function passes through zero, i.e. when $g_{event}(x, v, t) = 0$. This is called a "zero crossing"
2. The user can optionally define events based on the direction of the zero crossing, i.e. if it occurs from above or below.
3. The user can also specify whether the event should terminate the integration, or whether it should just record the integration variables (i.e. x, v, t, etc.) and continue integrating.

Event function `ode04C_events.m`

Define the function `ode04C_events.m`:

- Use the same passed parameters (`t, w`) as `ode04_derivs()`.
- Returned variables: `trigger, is_terminal, direction`

```
function [trigger,is_terminal,direction] = ode04C_events(t,w)
```

Extract position y and velocity v from the w matrix

```
y = w(1);            % w(1) stores y
v = w(2);            % w(2) stores v
```

Set `trigger` equal to the function that triggers the event. Events are detected when value of `trigger` passes through zero. In this example, an event will trigger when y = 0.

```
trigger = y;            % Event is triggered when y = 0
```

Event function `ode04C_events.m`

The variable `is_terminal` is a flag that tells Matlab if the simulation should stop once the event is detected:
- if `is_terminal == 1`, the simulation will stop when an event is detected
- if `is_terminal == 0`, the simulation will keep going.
In this example, we want to stop the integration when the event it detected:

```
is_terminal = 1;            % Stop the integration
```

The variable `direction` is a flag that tells Matlab if event detection depends on the direction of the zero crossing.

- if `direction == 1`, trigger function must increase from negative to positive
- if `direction == -1`, trigger function must decrease from positive to negative
- if `direction == 0`, sign of zero crossing doesn't matter

In this example, we want to trigger when the ball drops (downward) below zero:

```
direction = -1;            % trigger on downward zero crossing
```

Event function `ode04C_events.m`

Bring it all together now. Here's the full event function:

```
function [trigger,is_terminal,direction] = ode04C_events(t,w)

y = w(1);              % w(1) stores y
v = w(2);              % w(2) stores v

trigger = y;           % Event is triggered when y = 0
is_terminal = 1;       % Stop the integration
direction = -1;        % Negative direction only (crosses y = 0
                       % from above)
```

Back in the main program `ode04C.m`

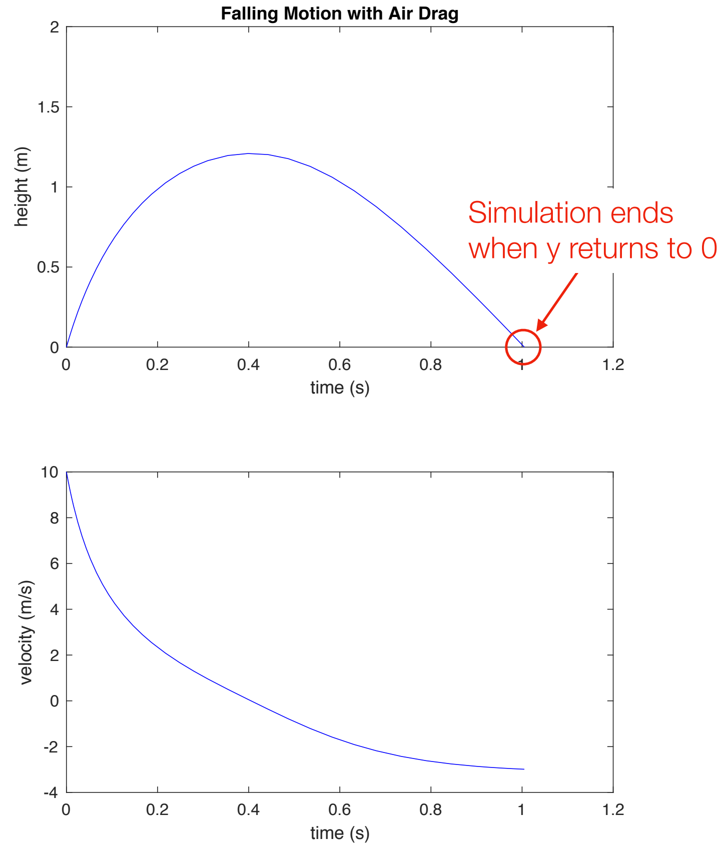
To implement the event handling, we need to replace this line of code in ode04A.m:

```
[t,w] = ode45(@ode04_derivs, [tBegin tEnd], [y0 v0]);
```

with the following lines of code in ode04C.m:

```
options = odeset('Events',@ode04C_events);

% Use the Runge-Kutta 45 solver to solve the ODE
[t,w] = ode45(@ode04_derivs, [tBegin tEnd], [y0 v0], options);
```

Here's the result for ode04C:



One last modification for `ode04C.m`

We often use events to measure some aspect of the ODE solution. For example, we can time how long the ball stays aloft. To do this we make one last modification. We "capture" the event data returned by ode45:

Modify this line of code:

```
[t,w] = ode45(@ode04_derivs, [tBegin tEnd], [y0 v0], options);
```

to look like this:

```
[t,w,te,we,ie] = ode45(@ode04_derivs, [tBegin tEnd], [y0 v0], options);
```

where:

- `te` is the time of the event
- `we` is the `w` matrix of the event. In this example `we` is a 1x2 matrix containing the position and velocity of the ball
- `ie` is the event index, which only matters when multiple event triggers are used

One last modification for **ode04C.m**

We can now display information on the ball's motion as follows:

```
ye = we(1);      % height of the ball when it lands (should be 0!!)
ve = we(2);      % velocity of the ball when it lands

fprintf('The ball is aloft for %f s\n',te);
fprintf('The ball lands with a velocity = %f m/s\n',ve);
```

D. Combining Events with a Family of Solutions

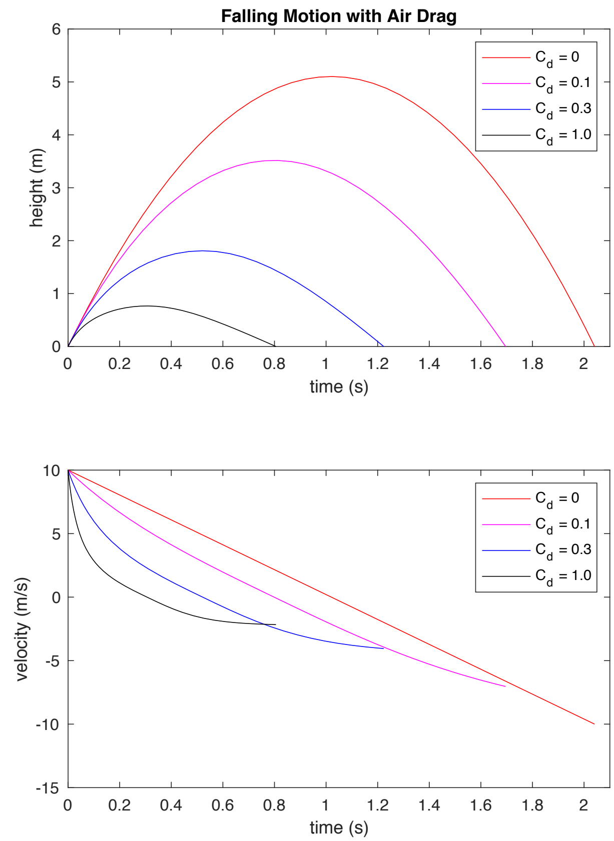
This example doesn't introduce anything new. It just combines programs ode04B.m with ode04C.m.

Events are used to terminate the program when the ball returns to y = 0. Information about the dynamics are printed to the command window for each value of the drag coefficient.

Output:

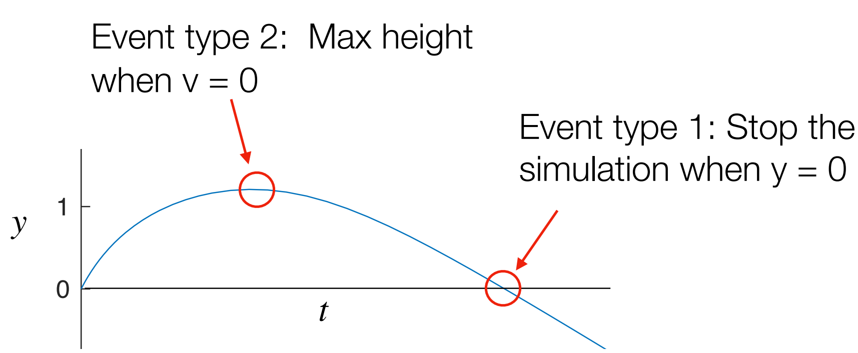
```
Drag coef = 0.00   time aloft = 2.04 s   impact velocity = -10.0 m/s
Drag coef = 0.10   time aloft = 1.70 s   impact velocity = -7.0 m/s
Drag coef = 0.50   time aloft = 1.22 s   impact velocity = -4.0 m/s
Drag coef = 2.00   time aloft = 0.81 s   impact velocity = -2.2 m/s
```

Here's the result for ode04D:



E. Using Multiple Events

In this example, we have two types of events:



Event function `ode04E_events.m`

To define a second type of event, we write a new event function called **`ode04E_events.m`**:

```
function [trigger,is_terminal,direction] = ode04E_events(t,w)

y = w(1);                % w(1) stores y
v = w(2);                % w(2) stores v

trigger = [y v];         % Event 1 is triggered when y = 0
                         % Event 2 is triggered when v = 0

is_terminal = [1 0];     % Event 1 stops the integration
                         % Event 2 doesn't stop the integration

direction = [-1 0];      % Event 1 requires zero crossing to go from
                         %         positive to negative
                         % Event 2 doesn't depend on direction
```

Notice each returned variable is now a 1x2 matrix. The first element defines Event type 1 and the second element defines Event type 2.

Back in the main program `ode04E.m`

Event handling with two events looks similar to that for one event. The **`ode45()`** call is unchanged from the call in **`ode04C.m`**:

```
[t,w,te,we,ie] = ode45(@ode04_derivs, [tBegin tEnd], [y0 v0], options);
```

Now, however, the returned event variables will have multiple rows, one for each event detected during the integration. In this example, only one type 1 event and one type 2 event will be found, giving us two rows for each variable **`te`**, **`we`** and **`ie`**:

	event time		event position	event velocity		event type	
te =	0.359	we =	0.7662	0.000	ie =	2	first event
	0.858		0.0000	-2.1617		1	second event

Back in the main program `ode04E.m`

We can separate out the position and velocity info from the **`we`** matrix just like we do for the **`w`** matrix:

```
ye = we(:,1);    % height of the ball for each event
ve = we(:,2);    % velocity of the ball for each event
```

To search for events of a particular type, we can do the following:

```
event_type1 = find(ie == 1);  % indices of all type 1 events
event_type2 = find(ie == 2);  % indices of all type 2 events
```

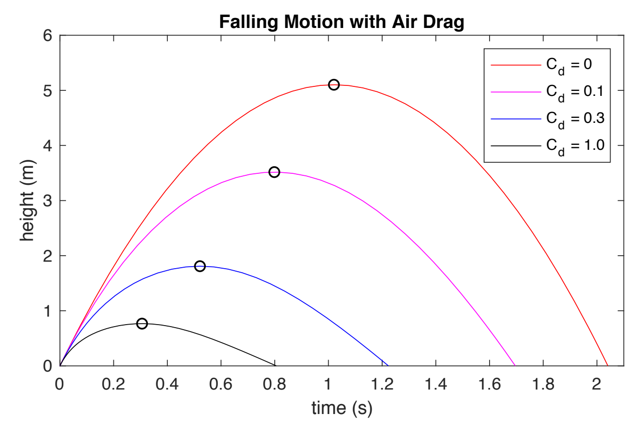
For example, we can now get all the velocities of type 1 events like this:

```
v1 = ve(event_type1);    % velocities of all type 1 events
```

Or we could plot a circle at the max height of the y vs. t graph:

```
plot(te(event_type2),ye(event_type2),'ko')
```

Here's the result for ode04E:



The open circles mark the position and velocity where the ball reaches its maximum height