# Differential equations - Boundary Value Problems

Last time we studied initial value problems of ordinary differential equations (ODEs). In the first order case, the value of the unknown function, in the second and higher order case, the value of the unknown function and derivatives up to the order of the equation were given at the start of the solution interval.

In the case of boundary value problems, at least one of these given values are for the end of the solution interval. As an example, let's look at the following second order ODE on the interval $[a, b]$.

$$\frac{d^2y}{dt^2} = f(t, y, \frac{dy}{dt})$$

As this is a second order DE, we need two values to find a unique solution. If it is an initial value problem, one of these is the value of the unknown function, the other is the value of the first derivative of the function and both of these values are given at $a$ (the start of the interval).

However, in the case of the boundary value problem, there can be more than one option. If the value of the function is given at the start and at the end of the interval as well, it is called a Dirichlet problem.

$$y(a) = Y_a \qquad y(b) = Y_b$$

Another possibility is to specify the values of the first derivative on the boundaries, this is called the Neumann problem:

$$\frac{dy}{dt}(a) = D_a \qquad \frac{dy}{dt}(b) = D_b$$

Conditions specifying the value of the function and the derivative can be mixed as well, e.g. the function value is given at the beginning of the interval and the derivative is given at the end of the interval.

We will deal with the general cases, when the problem can be transformed into an explicit system of ODEs. In such a case, we have to solve the following system of ODEs:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y})$$

On the two boundaries ($t = a$ and $t = b$), the following values are given:

$$\begin{aligned} \mathbf{y}_i &= \mathbf{A}_i \qquad i = 1, \cdots, k \\ \mathbf{y}_j &= \mathbf{B}_j \qquad i = k+1, \cdots, n \end{aligned}$$

One method of solution is that we reduce the problem into solving an initial value problem multiple times. This is called the shooting method.

### Shooting method

The idea behind this method is that we solve the initial value problem for an arbitrarily specified $\mathbf{y}_j(a) = \mathbf{u}_j$ value and check whether the $\mathbf{y}_j(b) = \mathbf{B}_j$ condition is satisfied (by solving the system of ODEs numerically). If the condition is not met, we modify the value of $u$. Let's suppose that the connection between the unknown $\mathbf{u}_j$ values and the $\mathbf{y}_j(b)$ values are given by a function $\mathbf{g}(u)$, that is:

$$\mathbf{y}_j(b) = \mathbf{g}_j(\mathbf{u})$$

Given this function, the unknown initial values are given by finding the solutions of the following system containing $(n-k)$ number of equations:

$$\mathbf{h}_j(\mathbf{u}) = \mathbf{g}_j(\mathbf{u}) - \mathbf{y}_j(b) = 0$$

### Example for the shooting method

Let's look at the following example. We shoot a fireworks pellet into the air that explodes after 5 seconds. The vertical movement of the pellet (omitting drag and other effects) is given by the following second order ODE:

$$\frac{d^2y}{dt^2} = -g$$

What is the needed initial velocity if we want the pellet to explode at exactly 40 meters from the ground?

First, we have to reduce the problem into solving a system of two first order ODEs, as seen in the previous practical. In order to do this, we introduce two new variables ($\mathbf{w} = [w_1, w_2]$). $w_1 = y$ denotes the vertical position of the pellet and $w_2 = \frac{dy}{dt}$ denotes the vertical velocity of the pellet. Now, the system of first order ODEs is given by the derivatives of the new variables:

$$\begin{aligned} f_1 &= \frac{dw_1}{dt} = \frac{dy}{dt} = w_2 \\ f_2 &= \frac{dw_2}{dt} = \frac{d^2y}{dt^2} = -g \end{aligned}$$

The boundary conditions:

$$y(0) = 0 \qquad y(5) = 40$$

That is, the vertical position at the time of launch is 0 m, after 5 seconds, when the pellet explodes, it is 40 m. The question is that what value should the first derivative have at the time of launch in order for the $y(5) = 40$ to be satisfied.

Let's first solve the system using the Runge-Kutta method and by defining different initial values for the velocity. Let $w_2(0) = 20, 30, 40, 50\text{m/s}$. Let the vector $\mathbf{w} = [w_1, w_2], where$ is the vertical position and $w_2$ is the vertical velocity (the first derivative of the position).

The system can be given in a separate file (diff_pellet.m):

```
function F = diff_pellet(t, w)
    g = 9.81;
    f1 = w(2);
    f2 = -g;
    F = [f1; f2];
end
```

Or we can also define it as an anonymous function, given that the system is not too complex:

```
g = 9.81;
dwdt = @(t, w) [w(2); -g];
```

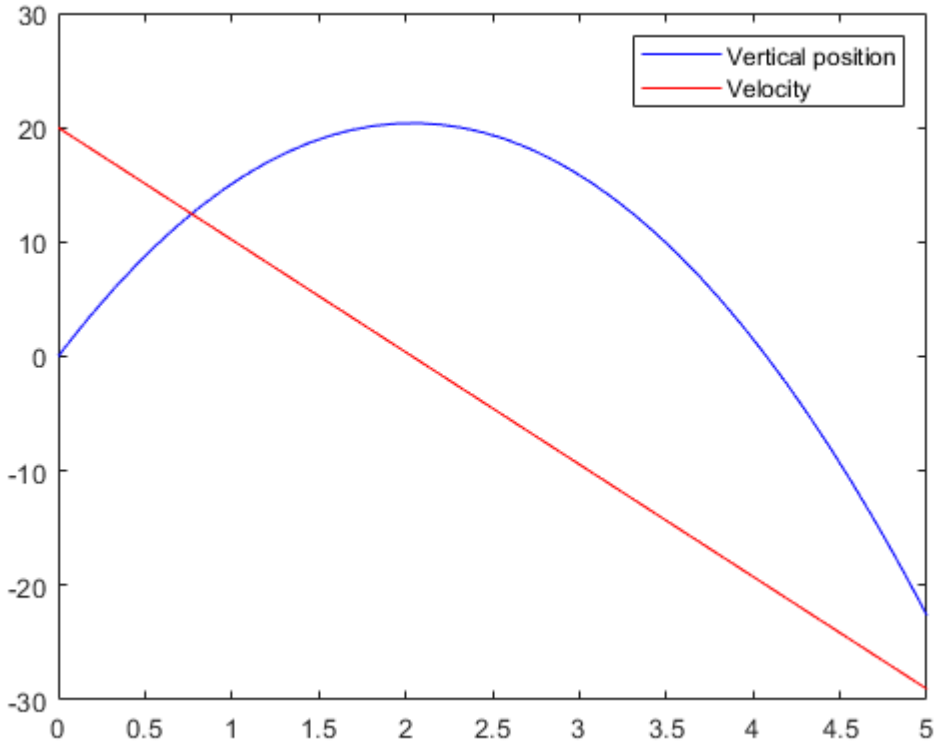
To solve the system with the Runge-Kutta method, use the **ode45** command in MATLAB (if the system is defined in a separate file, the "@" symbol has to be used before the name of the file, when calling **ode45**). Let's specify 20 m/s as the initial value of the velocity and $0 \leq t \leq 5$ as our solution interval:

```
ta = 0;
tb = 5;
y0 = 0;
v0 = 20;
% if the system is defined in a separate file
% [T, W] = ode45(@diff_pellet, [ta; tb], [y0; v0]);

% if the system is defined as an anonymous function
[T, W] = ode45(dwdt, [ta; tb], [y0; v0]);
```

The vector $\mathbf{T}$ contains the steps of the independent variable on the interval $[0, 5]$. The matrix $\mathbf{W}$ contains two columns and as many rows as there are elements in $\mathbf{T}$. The first column of $\mathbf{W}$ contains the vertical position values $(y)$, the second column contains the vertical velocities $\left(\frac{dy}{dt}\right)$. Let's visualize the results on a plot:

```
figure(1);
Y = W(:, 1);
V = W(:, 2);
plot(T, Y, 'b', T, V, 'r');
legend('Vertical position', 'Velocity');
```

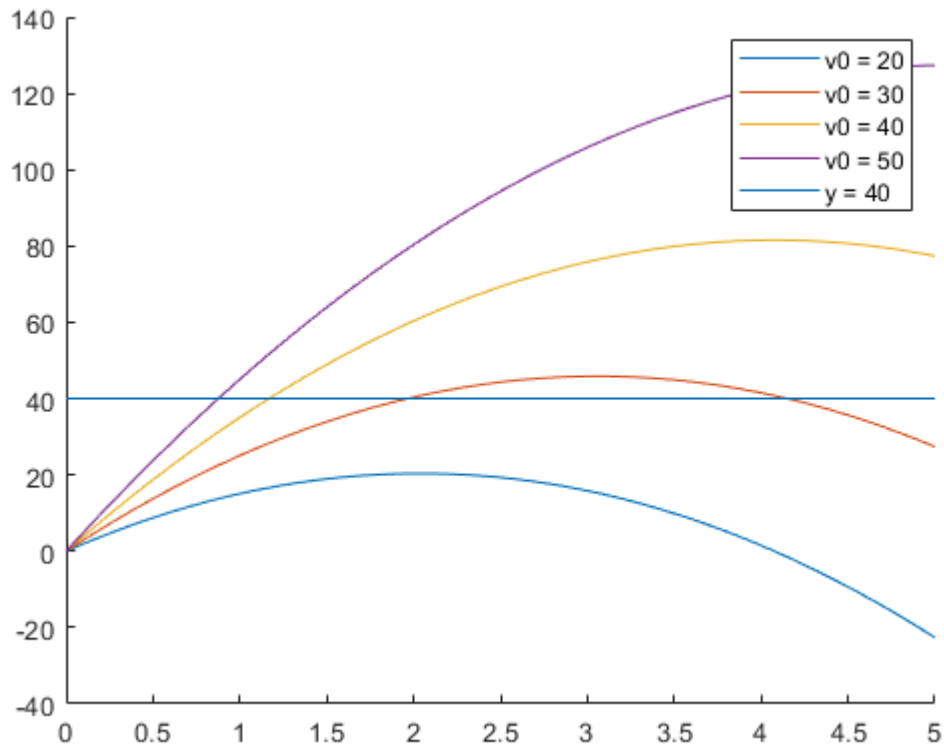


From the figure, we can see, that using 20 m/s as the initial velocity takes us very far from reaching the desired height of 40 m after 5 seconds. Let's try to increase the initial velocity (to 30, 40 and 50 m/s) and plot the vertical positions. Print the values of the position after 5 seconds for each initial velocity:

```
figure(2);
hold on;
for vi = 20:10:50
    [T, W] = ode45(dwdt, [ta; tb], [y0; vi]);
    fprintf('v0 = %d m/s -- y(5) = %.3f m\n', vi, W(end, 1));
    plot(T, W(:, 1));
end
```

```
v0 = 20 m/s -- y(5) = -22.625 m
v0 = 30 m/s -- y(5) = 27.375 m
v0 = 40 m/s -- y(5) = 77.375 m
v0 = 50 m/s -- y(5) = 127.375 m
```

```
refline(0, 40);
legend('v0 = 20', 'v0 = 30', 'v0 = 40', 'v0 = 50', 'y = 40');
```




From the figure and the printed values, we can see if the initial velocity is 30 m/s, the pellet fall under 40 m when it explodes and when the initial velocity is 40 m/s, the explosion happens above 40 m. The solution is somewhere between the two values.

Let the unknown initial velocity be $u$ and define the position at the end of the 5 seconds as a function $g$ of this $u$ unknown velocity. The value of this function has to be equal to 40:

$$w_1 = g(u) = 40$$

In other words, we are looking for the root of the $h = g(u) - 40$ equation.

Let's first define this function $g$ in a separate file, that is a function of the unknown initial velocity and name it **`pellet_height.m`**. This will use the **`diff_pellet.m`** file inside:

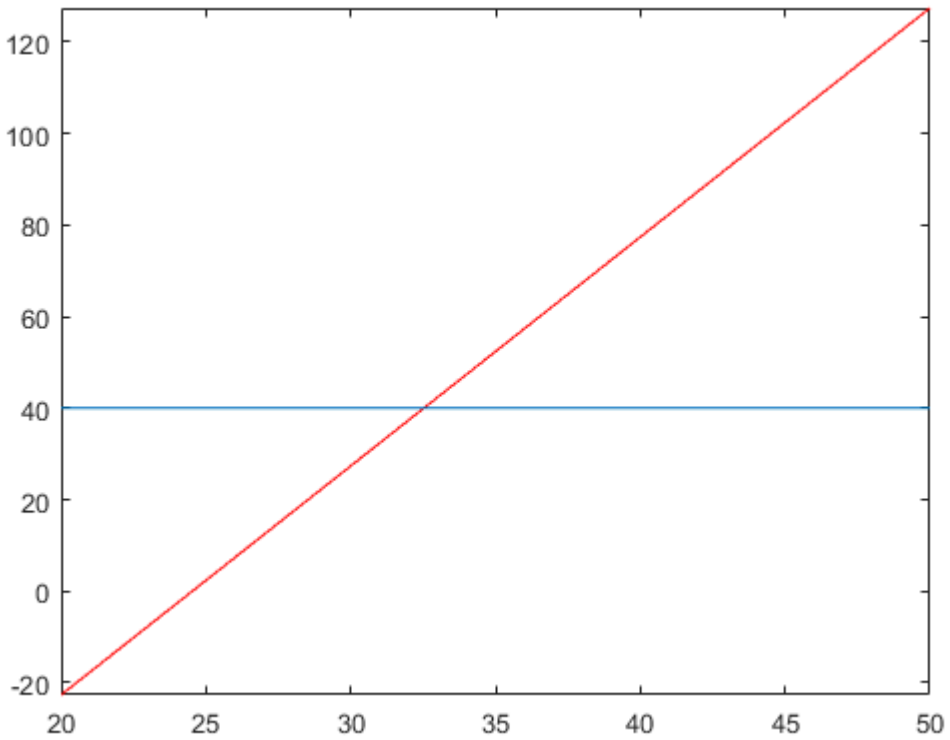
```
function y = pellet_height(u)
    ta = 0;
    tb = 5;
    y0 = 0;
    [T, W] = ode45(@diff_pellet, [ta; tb], [y0; u];
    y = W(end, 1); % the final position is given in the
                   % last element of the first column of W
end
```

Plot the final heights as a function of the initial velocity between 20 and 50 m/s:

```
figure(3);
fplot(@pellet_height, [20, 50], 'r');
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize your
function to return an output with the same size and shape as the input arguments.
```

```
hold on;
refline(0, 40);
```



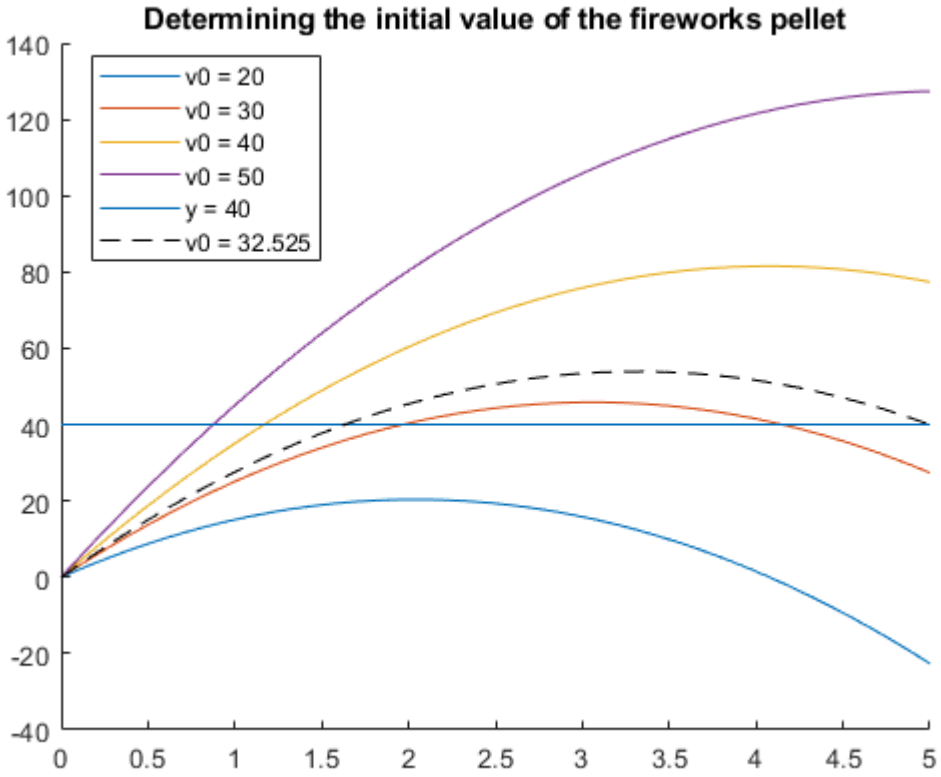
According to the figure, the solution is somewhere between 30 and 35 m/s. Let our initial guess be $u_0 = 32$.

```
h = @(u) pellet_height(u) - 40;
v0 = fzero(h, 32)
```

```
v0 = 32.5250
```

Let's plot the solution using dashed line on figure 2:

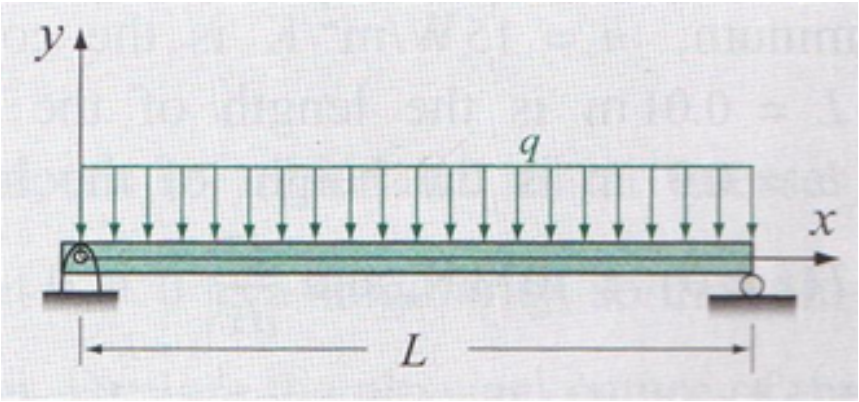
```
[T, W] = ode45(@diff_pellet, [ta; tb], [y0; v0]);
figure(2);
plot(T, W(:, 1), 'k--');
legend('v0 = 20', 'v0 = 30', 'v0 = 40', 'v0 = 50', 'y = 40', ...
    sprintf('v0 = %.3f', v0), 'Location', 'best');
title('Determining the initial value of the fireworks pellet');
```




There are many other methods for solving boundary value problems (minimizing global/local residuals, homotopy method, finite difference method etc). Due to time constraints, we do not deal with these now.

### Using the built-in solver in MATLAB

MATLAB contains a built-in method for solving ODEs with boundary value problems, the **bvp4c** command (bvp = boundary value problem). In the following, we will look at an example using this method:




A 4m long beam is given with supports at both ends and a constant load $(q)$ on top. For large deflections, the value of the deflection $(y)$ along the beam can be computed using the following ODE:

$$EI\frac{d^2y}{dx^2} = \left[1+\left(\frac{dy}{dx}\right)^2\right]^{\frac{3}{2}} \cdot \frac{1}{2} \cdot q \cdot (L \cdot x - x^2)$$

with the boundary values $y(0) = 0$ and $y(L) = 0$. The constants in the equation:

- $EI$ - flexural rigidity: $1.4 \cdot 10^7\,\mathrm{N}/m^2$
- $q$ - constant load: $10 \cdot 10^3\,\mathrm{N}/m$

Determine the function of the deflection with respect to $x$ using the built-in solver in MATLAB. Plot the results and the first derivative as well. How much is the deflection at 1.35 m? What is the maximum deflection? At which $x$ value is the deflection exactly 1 mm?

The built-in **bvp4c** command uses the following syntax:

```
SOL = bvp4c(ODEFUN, BCFUN, SOLINIT)
```

The command has one output, which is a structure containing the independent variable (`sol.x`), the function values and the derivatives as well (`sol.y`). The command takes three inputs:

- `ODEFUN`: the system of first order ODEs (as a function handle),
- `BCFUN`: the boundary values given as a function,
- `SOLINIT`: the interval of the solution and the approximated average values of the function and its derivatives (can be given using the **bvpinit** command).

### Definition of the system of first order ODEs (ODEFUN)

Similarly to when using the **ode45** command, we first have to define the system of first order ODEs the second order equation is reduced to. To do this, we have to express the second derivative from the equation first:

$$\frac{d^2y}{dx^2} = \left(\left[1+\left(\frac{dy}{dx}\right)^2\right]^{\frac{3}{2}} \cdot \frac{1}{2} \cdot q \cdot (L \cdot x - x^2)\right) \cdot \frac{1}{EI}$$

Then, we can introduce our new variables and reduce the equation into a system of first order ODEs.

Let $w_1 = y$ and $w_2 = \frac{dy}{dx}$. Our system of first order ODEs look like the following:

$$f_1 = \frac{dw_1}{dx} = \frac{dy}{dx} = w_2$$
$$f_2 = \frac{dw_2}{dx} = \frac{d^2y}{dx^2} = \left( \left[1 + w_2^2\right]^{\frac{3}{2}} \cdot \frac{1}{2} \cdot q \cdot (L \cdot x - x^2) \right) \cdot \frac{1}{EI}$$

We can define the system of equations in a separate file called **`diff_beam.m`**:

```
function F = diff_beam(x, w)
   EI = 1.4e7;
   q = 10e3;
   L = 4;
   f1 = w(2);
   f2 = ((1 + w(2)^2)^(3/2)*(1/2)*q*(L*x - x^2))/EI;
   F = [f1; f2];
end
```

### Definition of the boundary values as a function (BCFUN)

The boundary values for the problem are the following:

$$y(0) = 0 \qquad y(L) = 0$$

In order to use **bvp4c**, these boundary values have to be given as a function as well (BCFUN). The boundary conditions on the interval $[a, b]$ have to be given as a vector $w_a$ and vector $w_b$. The vector $w_a$ contains the conditions for the start of the interval, while $w_b$ contains the conditions for the end of the interval. The first elements of these vectors ($w_a(1), w_b(1)$) are the values of the unknown function ($w_1 = y$) at the start/end of the interval, the second elements are the values of the first derivatives $\left(w_2 = \frac{dy}{\mathrm{dx}}\right)$ at the start/end of the interval and so on.

In the case of our problem, only the function values are given, therefore $w_a$ and $w_b$ only contain one value:

$$w_a(1) = 0 \qquad w_b(1) = 0$$

Let the function of the boundary value be denoted by $g$. The function of the boundary value can be defined similarly to when searching for the root of an equation, its expression has to have zero on one of the sides:

$$g_1 = w_a(1) - 0 = 0$$
$$g_2 = w_b(1) - 0 = 0$$

These functions in essence define the residuals. If these are 0, the solution functions satisfies the given criteria. In our case, as the boundary values are 0 for both boundaries the above equations simplify to:

$$g_1 = w_a(1)$$
$$g_2 = w_b(1)$$

For the sake of example, if the boundary value at the beginning of the interval for the first derivative is 2 and the boundary value at the end of the interval for the function value is 3, then our functions would look like the following:

$$g_1 = w_a(2) - 2$$
$$g_2 = w_b(1) - 3$$

We can define the boundary conditions in a separate file called **beam_bv.m**:

```
function G = beam_bv(wa, wb)
   g1 = wa(1);
   g2 = wb(1);
   G = [g1; g2];
end
```

### Definition of the solution interval and the approximating the average values (SOLINIT)

The third input to the **bvp4cm** command is the interval of the solution and the approximated average values of the function and its derivatives (as constants, specified using the **bvpinit** command). Let our solution interval be $0 \leq x \leq 4$ in this case and the step size be 10 cm.

```
x0 = 0:0.1:4;
```

Now, we have to approximate the average value of the function. As we only have values on the boundaries, that are both zero, let our approximated average be zero as well for both the function and its first derivative. (If we had different boundary values, the average could be the mean of those two values.)

```
% Approximated averages
w10 = 0;
w20 = 0;

% Solution interval and approx. averages using the bvpinit command
solinit = bvpinit(x0, [w10; w20]);
```

### Solution using the bvp4c command

Before we solve the ODE, specify a relative tolerance of $10^{-4}$. Now, this is specified by the **bvpset** command (and not the **odeset**):

```
opts = bvpset('RelTol', 1e-4);
```

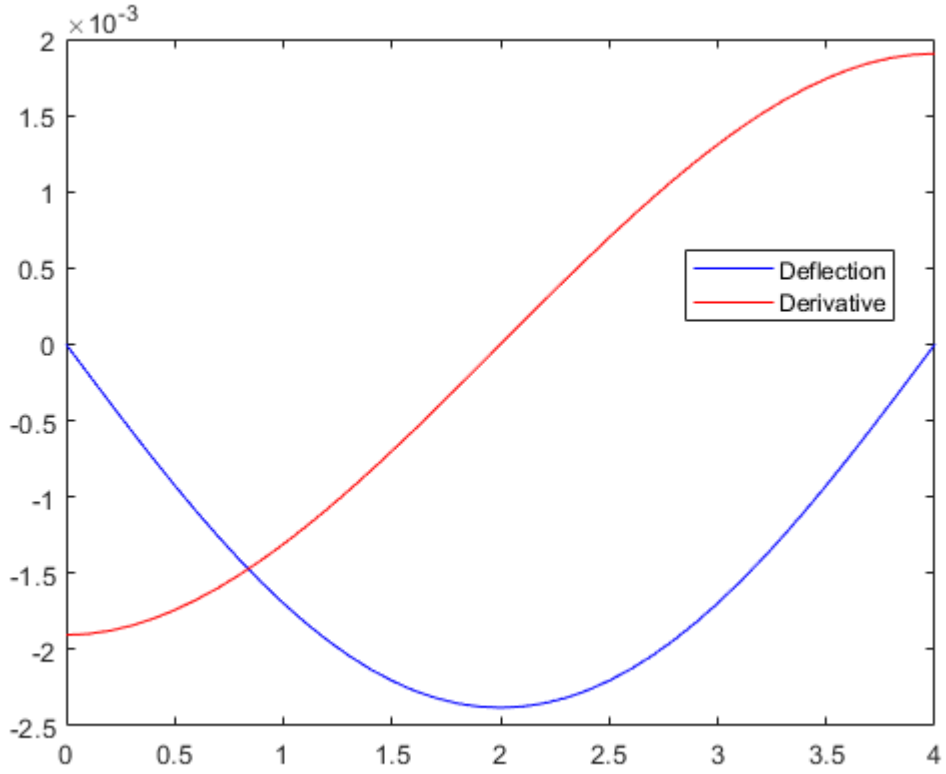
The solution of the ODE:

```
sol = bvp4c(@diff_beam, @beam_bv, solinit, opts);
X = sol.x;
W = sol.y;
```

Plot of the solution:

```
figure(4)
```

```
plot(X, W(1, :), 'b', X, W(2, :), 'r');
legend('Deflection', 'Derivative', 'Location', 'best');
```




One of the question asked about the value of the deflection at $x = 1.35\,\mathrm{m}$. Another question was that at which value of $x$ is the deflection exactly 1 mm?

These could be answered by interpolating the data points of the function using splines for example, but we can also use the **deval** command. The command evaluates the solution from the **bvp4c** solver at any arbitrary point (in the solution interval). This command can also be used in the case of **ode45**. The **deval** command cannot be used for extrapolation!

Let's first answer the first question, that is, the value of the deflection at $x = 1.35\,\mathrm{m}$:

```
e1 = deval(sol, 1.35)    % this will return the value of the function and its derivative as wel
```

```
e1 = 2×1
   -0.0021
   -0.0009
```

```
e2 = deval(sol, 1.35, 1) % this will only return the value of the function
```

```
e2 = -0.0021
```

The deflection is 2.1 mm at $x = 1.35\,\mathrm{m}$.

We can define a new function called `defl` as an anonymous function using the **deval** command to compute the deflection at any point:

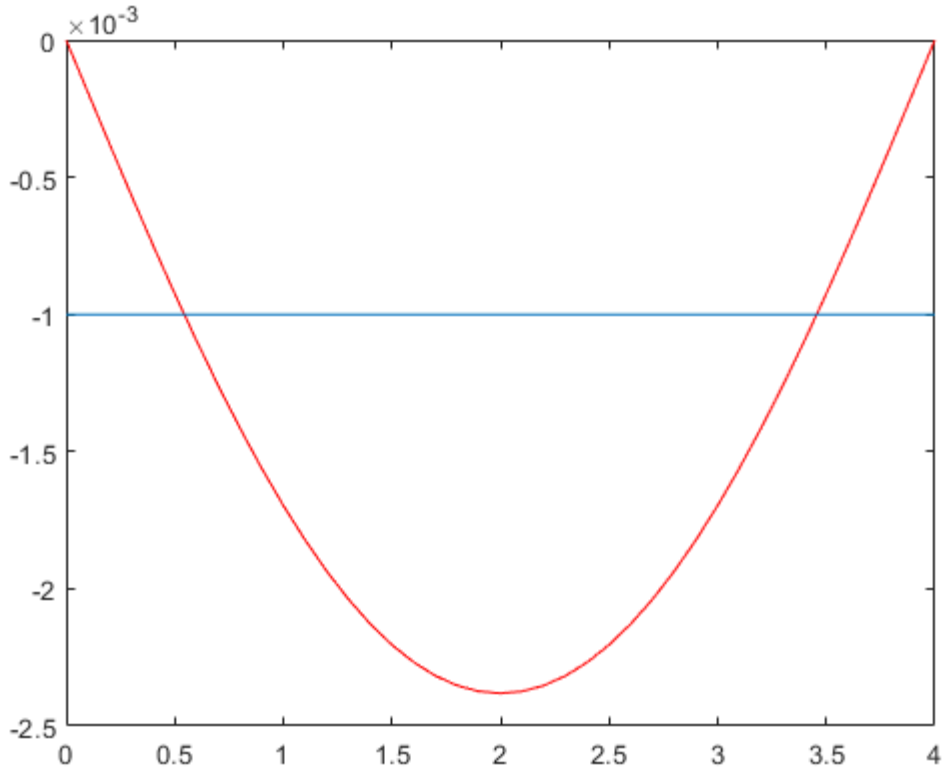
```
defl = @(x) deval(sol, x, 1);
```

The second question can be answered by defining a new function $h(x)$ and solving the following equation (remember, the deflections are negative values!):

$$defl(x) = -0.001 \rightarrow h(x) = defl(x) + 0.001 = 0$$

Before we do that, let's look at the figure to get an idea about the initial guesses:

```
figure(5);
plot(X, W(1, :), 'r')
rl = refline(0, -0.001);
```



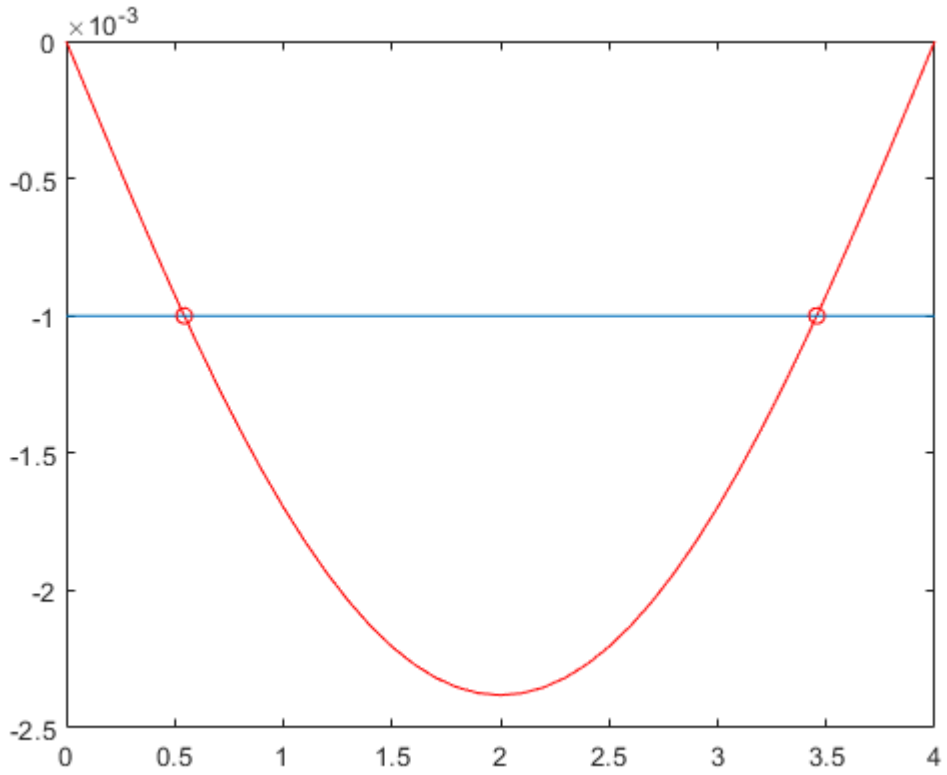

```
% initial guesses
x01 = 0.5;
x02 = 3.5;

% definition of h(x)
h = @(x) defl(x) + 0.001;

% solution
x1 = fzero(h, x01);
x2 = fzero(h, x02);
```

Plotting the solutions:

```
figure(5);
hold on;
plot(x1, defl(x1), 'ro');
plot(x2, defl(x2), 'ro');
```




The last question was the position of maximum deflection. As the load was symmetric, we can already tell that the maximum deflection happened in the middle of the beam. Let's verify this numerically. As the deflection values are negative, we can use the optimization command **`fminsearch`** to find the minimum of the function. For the initial guess, we can use 2:

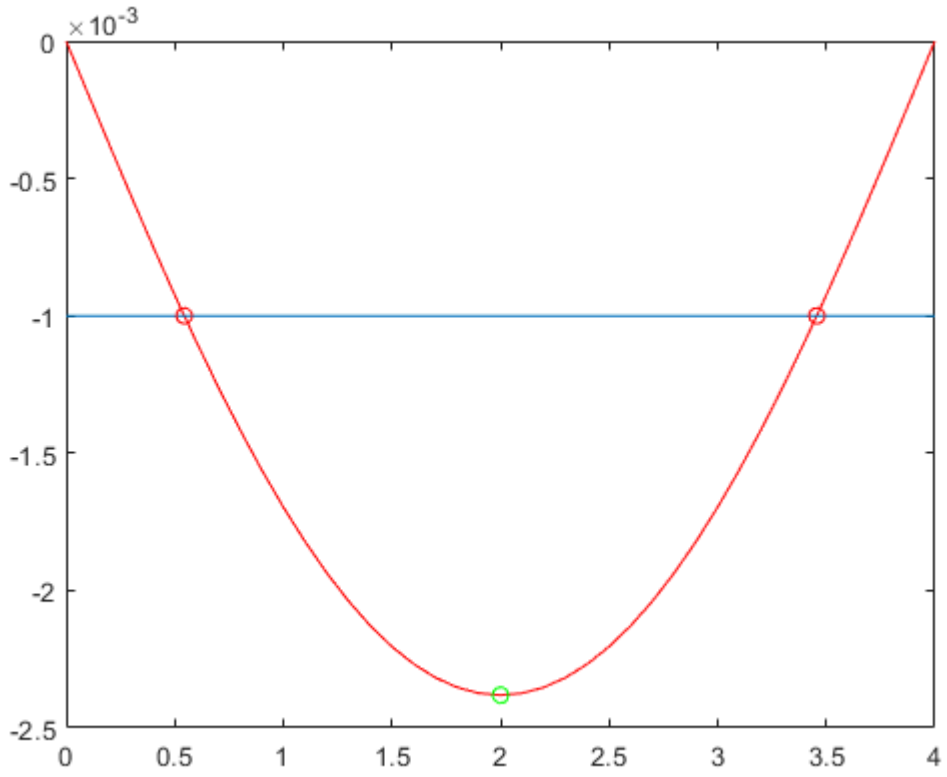
```
xmax = fminsearch(defl, 2)
```

```
xmax = 2
```

```
dmax = defl(xmax)
```

```
dmax = -0.0024
```

```
figure(5);
plot(xmax, dmax, 'go')
```




For the sake of example, let's look at how our first exercise could be solved using the built-in **bvp4c** command:

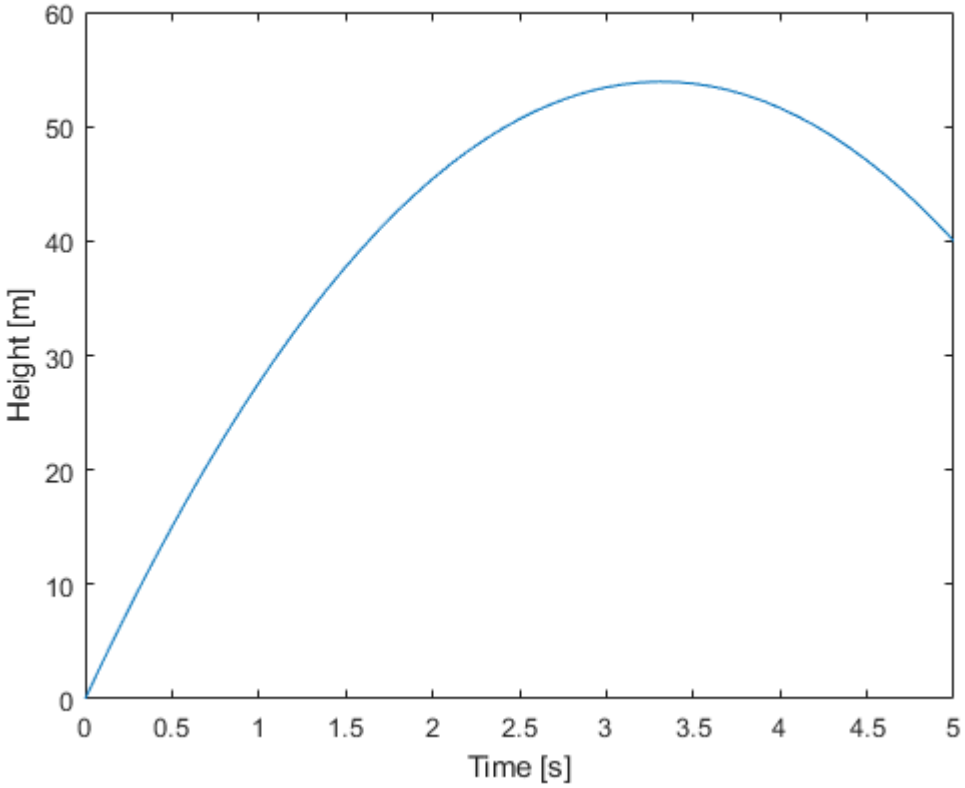
```
clear all; close all;

% Solution interval
t0 = 0:0.1:5;

% Approximated averages for the function value and its derivative
w10 = 20; % average function value (0+40)/2
w20 = 0;  % average value of the derivative
solinit = bvpinit(t0, [w10; w20]);

% Solution using the bvp4c command
opts = bvpset('RelTol', 1e-4); % relative tolerance
sol = bvp4c(@diff_pellet, @pellet_bv, solinit, opts);
X = sol.x;
W = sol.y;

% Plotting the solution
plot(X, W(1, :));
xlabel('Time [s]');
ylabel('Height [m]');
```




### Practice example

Solve the following boundary value problem on the interval $[0, 1]$:

$$\frac{d^2y}{dx^2} + y = 0$$

Rearranging the expression:

$$\frac{d^2y}{dx^2} = -y$$

The given boundary conditions:

$$y(0) = 1 \qquad \frac{dy}{dx}(1) = 3$$

At the beginning of the interval, the value of the function is given, at the end of the interval, the value of the derivative is specified. We can reduce this equation into a system of first order ODEs by introducing two new variables: $w_1 = y$ and $w_2 = \frac{dy}{dx}$:

$$f_1 = \frac{dw_1}{dx} = \frac{dy}{dx} = w_2$$

$$f_2 = \frac{dw_2}{dx} = \frac{d^2y}{dx^2} = -w_1$$

The boundary conditions using the new variables, in the form required by the **bvp4c** solver:

$$w_a(1) = 1 \qquad w_b(2) = 3$$

Rearranging the boundary conditions:

$$g_1 = w_a(1) - 1 \qquad g_2 = w_b(2) - 3$$

The solution in MATLAB:

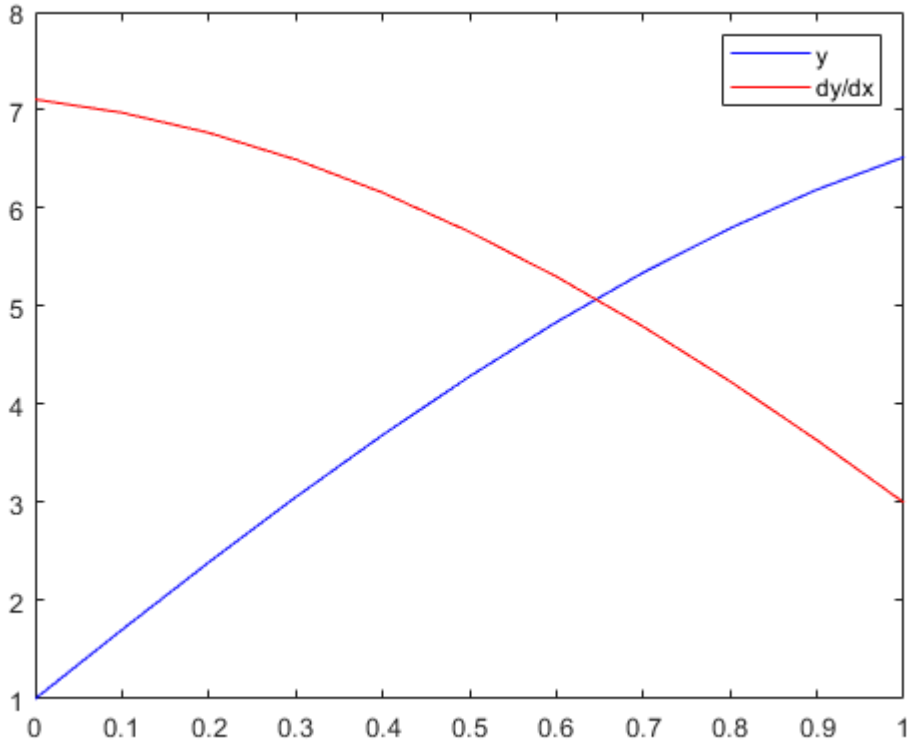
```
clear all; close all;

% Solution interval (the step size is arbitrary)
x0 = 0:0.1:1;

% Approximating the function value and its derivative
w10 = 1;
w20 = 2;
solinit = bvpinit(x0, [w10; w20]);

% Solution
sol = bvp4c(@diff_practice, @practice_bv, solinit);
X = sol.x;
W = sol.y;

% Plotting the solution
plot(X, W(1, :), 'b', X, W(2, :), 'r');
legend('y', 'dy/dx');
```


y
dy/dx
8
7
6
5
4
3
2
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1