eDifferential equations - boundary value problem

Till this point we only worked with initial value problems of the ordinary differential equations, when the value of the function and its derivatives are given at the beginning of the relevant interval. The functions value is given at the initial position in a first order case; the value of the functions and its first derivative is given at the initial position in a second order case; the value of the functions, its first and second derivative is given at the initial position in a third order case, etc. In case of a boundary value problem the value of the function and its derivatives is not known at the initial position (or at least one of them is not known), but instead at the other endpoint, therefor the task is to find the function that satisfies the conditions.

Let’s see a second order ODE in the range of [a,b]:

\[
\frac{d^2y}{dt^2} = f\left(t, y, \frac{dy}{dt}\right)
\]

To solve a second order differential equation we need two conditions. In case of an initial value problem, these are the values of the function \(y\) and its first derivative \(\frac{dy}{dt}\) at the initial location. In case of a boundary value problem, there could be multiple possibilities:

The functions value could be given at the start and the end of the interval, this is called the Dirichlet boundary condition:

\[
y(a) = y_a, \quad y(b) = y_b
\]

The values of the derivative functions could be given at the start and the end of the interval, this is called the Neumann boundary condition:

\[
\left.\frac{dy}{dt}\right|_{t=a} = D_a; \quad \left.\frac{dy}{dt}\right|_{t=b} = D_b
\]

Or they could be given in a mixed version, e.g. the functions value is given at the start, and the derivative functions value is given at the end of the interval.

Let’s examine this in general, when the problem could be derived into an explicit, first order differential equation system. In this case the system is:

\[
\frac{dy}{dt} = f\left(t, y\right)
\]

The following values are known on the two ends of the interval \((t = a \text{ and } t = b)\):

\[
y_i(a) = A_i; \quad i = 1, \ldots, k
\]

\[
y_j(b) = B_j; \quad j = k + 1, \ldots, n
\]

One possible solution is to trace back the task into a repetition of initial value problems, this is called the shooting method.

**Shooting method**

The main idea of this method is to solve the problem as an initial value problem by using a trial-and-error method with an initial value of \(y_j(a) = u_j\), then we check the function value on the other end of the interval, whether the given condition \(y_j(b) = B_j\) is satisfied or not (by solving the differential equation system in a numerical sense). If there is a difference, then we change the initial value of \(u\).

Let’s assume, that the connection between the unknown \(u_j\) and the value of \(y_j(b)\) could be described with a function \(g(u)\):

\[
y_j(b) = g_j(u)
\]

Based on this the unknown initial value could be determined by finding the root of the following equations:
Applying the shooting method

A rocket was launched vertically in the sky, and it explodes after 5 seconds. The vertical movement of the rocket could be described with the following second order differential equation (if air resistance is excluded):

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

How fast do we need to fire the rocket to let it explode exactly at 40 meters high?

We could derive the second order differential equation into a system of two first order differential equations if we introduce a new variable \( y_2 \) for the first derivative (the velocity): \( \frac{dy}{dt} = y_2 \)

In this case the first order system of equations:

\[
f_1 = \frac{dy}{dt} = y_2
\]

\[
f_2 = \frac{dy_2}{dt} = -g
\]

The boundary conditions:

\[
y(0) = 0; \quad y(5) = 40;
\]

This means the vertical location \( y \) at the launch is 0 m, and after 5 seconds it is 40 m. The question is the value of the first derivative, the velocity, at the launch.

Let's solve this system at first via Runge-Kutta method, with an initial guess for the velocity. Let it be \( y_2(0) = 20,30,40,50 \text{ m/s} \) ! Let variable \( y \) be a vector: \( y = [y_1, y_2] \), where \( y_1 \) is the value of the vertical displacement, and \( y_2 \) is the first derivative, so the velocity!

The differential equation could be given in a separate file or in the following form:

```matlab
g = 9.81;
dydt = @(t,y) [y(2); -g]
dydt = function_handle with value:
@t,y)(y(2);-g]
```

```matlab
% function dydt = diff_rocket(t,y)
% g = 9.81;
% f1 = y(2);
% f2 = -g;
% dydt = [f1; f2];
% end
```

We can use the built-in ode45 function in Matlab to apply the Runge-Kutta method! If you want to use the a stand-alone function for the differential equation, then an @ sign is necessary before the functions name for the input. Let's use 20 m/s as an initial guess for the velocity, and use the range of \( 0 \leq t \leq 5 \text{ s} \)!

```matlab
% [T Y] = ode45(@diff_rocket,[ta; tb],[y0; v0]); % in a separate file
% or
[T Y] = ode45(dydt,[ta; tb],[y0; v0]); % as a single line function
```

In the results the T vector contains the time steps in the range of \([0, 5]\), and the Y vector contains the values in the same time stamps for the two variables: the vertical displacement and the velocity. Let's plot the vertical displacement and the velocities in function of time!
We can see on the figure, that if the initial velocity was 20 m/s the rocket can't reach the 40 m height. Let's rerun the process with different initial velocities (20, 30, 40, 50 m/s) and plot only the vertical displacement! The height of the rocket after 5 seconds is the last element of the result vector (last element of the first column in vector Y).

```matlab
figure(2); hold on;
for vi=20:10:50
    [T Y] = ode45(dydt,[ta; tb],[y0; vi]);
    fprintf('%vd m/s; y(5)=%.1f m\n',vi,Y(end,1))
    plot(T,Y(:,1));
end

v0=20 m/s; y(5)=-22.625000 m
v0=30 m/s; y(5)=27.375000 m
v0=40 m/s; y(5)=77.375000 m
v0=50 m/s; y(5)=127.375000 m

ezplot('40', [0 5])
legend('v0=20', 'v0=30', 'v0=40', 'v0=50', 'y=40')
```
We can see from the figures, where the rocket will be after the 5 seconds, and hence the initial velocity should be between 30 m/s and 40 m/s.

Let's define the unknown boundary condition (the initial velocity) as a free parameter \( u \), and write a function \( g \) based on the initial speed, that gives us the functions value at the end of the 5 seconds. Let's determine when would this value be 40!

\[
y_i = g(u) = 40
\]

So we are searching the root of the following function: \( h = g(u) - 40 \)

Let's write the function \( g \) in a separate file, which gives us the height after 5 seconds according to the initial velocity! For this we will use the separate file describing the differential equation system 'diff_rocket.m'.

```matlab
function y = rocket_height(u)
 ta = 0; tb = 5; y0 = 0;
 [T Y] = ode45(@diff_petarda,[ta; tb],[y0; u]);
 y = Y(end,1); % the last element of the first column
 end
```

Let's plot the height after 5 seconds based on the initial velocity!

```matlab
figure(3)
 ezplot(@rocket_height,[20, 50]);

Warning: Function failed to evaluate on array inputs; vectorizing the function may speed up its evaluation and avoid the need to loop over array elements.

hold on; ezplot('40',[20,50])
```
Based on the figure the initial velocity should be somewhere between 30 and 35 m/s. Let the initial value be \( u(0) = 32 \)!

```matlab
% Determining the missing boundary condition
h = @(u) rocket_height(u) - 40

h = function_handle with value:
@u rocket_height(u) - 40

v0 = fzero(h, 32) % 32.5250

v0 = 32.5250

Lets plot the solution with dashed lines beside the previous initial tries!

```
There are several other methods to solve boundary value problems, e.g. using the method of finite difference, or using trial functions (minimizing the global residuals, eliminating local residuals).

**Solving boundary value problems with the built-in methods of Matlab**

The built-in function in Matlab to solve boundary value problems is the **bvp4c** (bvp = boundary value problem). Let's check this method on another example!

Below you see a 4 meter length \((L=4\ m)\) simply supported beam, which is loaded with an evenly distributed load \((q)\). The following expression describes the beam’s deflection \((y)\), i.e. a second order differential equation.

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

\(y(0)=0;\ y(L)=0;\) - boundary conditions

\(EI = 1.4 \times 10^7 N m^2;\) - bending stiffness

\(q = 10 \times 10^3 N/m;\) - load

Let's solve this task with the built-in Matlab function! Determine the deflection values in function with the horizontal location \((x)\) and plot it! Plot the first derivatives too! What will be the deflection at 1.35 m? What is the maximum deflection? At which locations will be the deflection exactly 1 mm?

In Matlab the **bvp4c** function which can solve boundary value problems should be used in the following form:

\[
\text{SOL} = \text{bvp4c}([\text{ODEFUN,BCFUN,SOLINIT}])
\]

This function (**bvp4c**) only has one output, which is a structure type variable containing the independent variable
(sol.x) and the calculated functions and its derivatives values (sol.y). It has three inputs: The first two are functions
(odefun, bcfun), from this the first is a first order differential equation system, the second is the boundary conditions
defined as a function. The third input contains the range of evaluation, including an initial estimation for the average
values of the function and its derivatives; these could be initialized by the function bvpinit.

For this function the higher order differential equations should be reformed into a first order system of equations (just
like in case of the ode45 function). Lets express at first the second derivative from the second order differential
equation!

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

Lets convert this into two first order equations by introducing a new variable: \( \frac{dy}{dx} = y_2 \).

In this case the first order system of differential equations:

\[
f_1 = \frac{dy}{dx} = y_2
\]

\[
f_2 = \frac{dy_2}{dx} = \left(1 + \left(\frac{dy}{dx}\right)^2\right)^{\frac{3}{2}} \cdot \frac{1}{EI} \cdot \frac{q^*}{\left(L^* - x^2\right)}
\]

Lets define the system of differential equations in a separate function. You can do this in a separate file, or at the end
of your script (from Matlab version R2016b):

```matlab
% function dydx = diff_deflection(x,y)
%     EI = 1.4e7; q = 10e3; L = 4;
%     f1 = y(2);
%     f2 = ((1+(y(2))^2)^3/2) * (1/2) * q * (L*x-x^2) / EI;
%     dydx = [f1; f2];
% end
```

The boundary conditions (deflection at the supports):

\[ y(0) = 0; \quad y(L) = 0; \]

For the function bvp4c the boundary conditions should be defines as a function. In the range of \([a,b]\] the conditions
should be defines in vectors of \(ya\) and \(yb\), in which \(ya\) is the condition at the start, and \(yb\) is the condition at the end of
the interval. The first element of the vectors \((ya(1), yb(1))\) is the value of the first variable (so the function we wish to
find) in the equation system at the two ends, the second element \((ya(2), yb(2))\) is the value of the variable introduced
instead of the first derivative at the two ends, etc (in case of higher order differentials). In this case the deflection
values are given at the two ends:

\[ ya(1)= 0, \quad yb(1)= 0 \]

Let the function of boundary conditions be \(g\). The definition is similar to the root finding, it should be in a form
rearranged to zero:

\[ g_1 = ya(1) - 0 = 0 \]
\[ g_2 = yb(1) - 0 = 0 \]

This means the residuals, if they have zero values, the function satisfies the conditions. In this case the boundary
conditions are zeros at the two ends, therefor the definition is simple: \(g_1 = ya(1), \quad g_2 = yb(1)\). If at the starting point
the first derivative would be given (e.g. 2), and at the endpoint the functions value would be 3 the definition would be:

\[ g_1 = ya(2) - 2, \quad g_2 = yb(1) - 3. \]

Let the boundary conditions be defined in a separate function: boundary_deflection (or at the end of the script).
Because we have conditions for the deflection values on both ends, the first elements of \(ya\) and \(yb\) are given.

```matlab
% function g=boundary_deflection(ya,yb)
```
The third input argument for the function `bvp4c` is the range of evaluation, and the estimated average value for the function and its derivatives (as a constant), these should be initialized using the function `bvpinit`. The range of evaluation is now the total length: \(0 \leq x \leq 4\). Lets determine the deflection at every 10 cm. First we need to estimate the average values for the function and its derivative. Now we only have information for the function values at the two end of the interval, and they are both 0, therefor this will be the estimated value for the functions and its derivative. (In other cases we would choose the average of the two value as a constant.)

```matlab
clc; clear all; close all;
% Estimated initial values - bvpinit
x0 = 0:0.1:4; % range of evaluation
y10 = 0; y20 = 0; % estimated constant values for the function and the first derivative
init_conditions = bvpinit(x0,[y10; y20])
```

Let's define a relative tolerance as \(10^{-4}\). In this case use the function `bvpset` instead of the `odeset`. Then solve the example using the function `bvp4c` and plot the results!

```matlab
% solution using the built-in function bvp4c
options = bvpset('RelTol',1e-4);
sol = bvp4c(@diff_deflection,@boundary_deflection,init_conditions,options)
```

```matlab
sol = struct with fields:
solver: 'bvp4c'
x: [1x41 double]
y: [2x41 double]
yp: [2x41 double]
stats: [1x1 struct]
```

```matlab
X = sol.x;
Y = sol.y;
plot(X,Y(1,:)); hold on
plot(X,Y(2,:));
legend('deflections','derivatives', 'Location','best')
```
The result is structure type: hence sol.x and sol.y contains the independent (x - vector) and dependent (y - matrix) variables. The y matrix contains the function values for the two variables, therefor the function and its derivative.

The whole solution is the following:

```matlab
% Deflection of the simply supported beam
  clear all; clc; close all;
  % Estimated initial values - bvpinit
  x0 = 0:0:1:4; % range of evaluation
  y0 = 0; y0 = 0; % estimated constant values for the function and the first derivative
  init_conditions = bvpinit(x0,[y0 y0])

init_conditions = struct with fields:
  solver: 'bvpinit'
    x: [1x41 double]
    y: [2x41 double]
    yinit: [2x1 double]

  % solution using the built-in function bvp4c
  options = bvpset('RelTol',1e-4);
  sol = bvp4c(@diff_deflection,@boundary_deflection,init_conditions,options)

sol = struct with fields:
  solver: 'bvp4c'
    x: [1x41 double]
    y: [2x41 double]
    yp: [2x41 double]
  stats: [1x1 struct]

  X = sol.x;
  Y = sol.y;
  plot(X,Y(1,:)); hold on
  plot(X,Y(2,:));
  legend('deflections','derivatives', 'Location','best')
```
What will be the deflection at 1.35 m, and what is the maximum deflection? At which location is the deflection exactly 1 mm?

These could be solved by fitting a spline function on the result points, but we can use the function `deval` too. This evaluates the result from the `bvp4c` or `ode45`, but it can only be used for interpolation (not extrapolation).

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

function g=boundary_deflection(ya,yb)
    g1=ya(1);
    g2=yb(1);
    g = [g1; g2];
end
```

```
% What is the deflection at the location of 1.35 m?
el = deval(sol,1.35) %-0.0021, -0.0009
el =
-0.0021
-0.0009
```

```
el = deval(sol,1.35,1) %-0.0021
```

```
el = -0.0021
```

```
% Therefore the deflection is 2.1 mm
hold on; plot(1.35,el,'ro')
```
The deval function uses the solution structure and the location to calculate the value of the function and its derivative (or if we add an additional input argument, an index, only the needed variable). Now the deflection is 2.1 mm at the relevant location.

What is the maximum deflection? For this lets define the solution as a function! The coordinate system is defined to get negative values for the deflection, this means we are searching for the minimum location, so the built-in fminsearch function can help, only an initial guess is necessary.

```matlab
% What is the maximum deflection?
deflection = @(x) deval(sol,x,1)
deflection = function_handle with value:
@(x)deval(sol,x,1)
xmax = fminsearch(deflection,2) % 2
xmax = 2
lmax = deflection(xmax) % -0.0024
lmax = -0.0024
```

The load was symmetric so it's evident, that the middle point (at 2m) will be the location of the maximum deflection. So we can simply get the answer with the deval function:

```matlab
lmax = deval(sol,2,1) % -0.0024
lmax = -0.0024
plot(xmax,lmax,'r*')
```

At which locations will the deflection exactly 1 mm? This could be two points, which could be determined using the function fzero. Be aware that due to the orientation (and the units) of the coordinate system, currently 1mm deflection means a value of -0.001! The initial guess could be determined based on the figure, let it be 0.5 and 3.5 m:

```matlab
% deflection = -0.001 -> h = deflection + 0.001 = 0
h = @(x) deflection(x) + 0.001
h = function_handle with value:
@(x)deflection(x)+0.001
x01 = 0.5; x02 = 3.5;
x1 = fzero(h,x01) % 0.5437
x1 = 0.5437
x2 = fzero(h,x02) % 3.4563
x2 = 3.4563
plot(x1,deval(x1),'ks',x2,deval(x2),'bd')
```
Vertical movement of the rocket, solving via built-in function bvp4c

Let's check the first example, the launched rocket, and solve it with the built-in Matlab function!

The second order differential equation: \( \frac{d^2y}{dt^2} = -g \)

The boundary conditions: \( y(0) = 0 \); \( y(5) = 40 \);

Solution using the built-in function bvp4c:

```matlab
% Rocket - bvp4c
clear all; close all;
% Estimated initial values - bvpinit
t0 = 0:0.1:5; % range for evaluation
y10 = 20; % estimated average height value (0+40)/2
y20 = 0; % estimated value for the first derivative
init_condition = bvpinit(t0,[y10; y20])

init_condition = struct with fields:
    solver: 'bvpinit'
    x: [1x51 double]
    y: [2x51 double]
    yinit: [2x1 double]

% solution using the built-in function bvp4c
options = bvpset('RelTol',1e-4);
sol = bvp4c(@diffRocket,@boundaryRocket,init_condition,options)
sol = struct with fields:
    solver: 'bvp4c'
    x: [1x51 double]
    y: [2x51 double]
    yp: [2x51 double]
    stats: [1x1 struct]

X = sol.x;
Y = sol.y;
```
plot(X,Y(1,:));
xlabel('Time'); ylabel('Height')

% function dydt = diff_rocket(t,y)
%     g = 9.81;
%     f1 = y(2);
%     f2 = -g;
%     dydt = [f1; f2];
% end
% %----------------------------------------------------------
% function g=boundary_rocket(ya,yb)
%     g1=ya(1);
%     g2=yb(1)-40;
%     g = [g1; g2];
% end
% %----------------------------------------------------------

Practice example for boundary value problem

Let's solve the following boundary value problem in the range of [0,1]:

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

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

The given boundary conditions:

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

Now the function value is given at the start of the interval, and the first derivative on the other end!

The first order system of differential equations if we introduce a new variable $$\frac{dy}{dx} = y_2$$, and $$y = y_1$$:

$$f_1 = \frac{dy}{dx} = y_2$$
\[ f_2 = \frac{dy_2}{dx} = \frac{d^2y}{dx^2} = -y_1 \]

The boundary conditions: \( y(1) = 1, \ y(2) = 3 \)

Rearranging to zero: \( g_1 = y(1) - 1; \ g_2 = y(2) - 3 \)

Solution in Matlab:

```matlab
% Practice, boundary value problem
clc; clear all; close all;
% Estimated initial values - bvpinit
x0 = 0:0.1:1; % range for evaluation
y10 = 1; % estimated average function value
y20 = 2; % estimated value for the first derivative
init_cond = bvpinit(x0,[y10; y20])

init_cond = struct with fields:
solver: 'bvpinit'
x: [0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1]
y: [2x11 double]
yinit: [2x1 double]

% solution using the built-in function bvp4c
sol = bvp4c(@practice_diff,@practice_bound,init_cond)

sol = struct with fields:
solver: 'bvp4c'
x: [0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1]
y: [2x11 double]
yp: [2x11 double]
stats: [1x1 struct]

X = sol.x; Y = sol.y;
plot(X,Y(1,:),X,Y(2,:));
legend('y','dy/dx')
```

```matlab
function dydx = practice_diff(x,y)
```
f1 = y(2);
f2 = -y(1);
dydx = [f1; f2];
end

function g=practice_bound(ya,yb)
g1=ya(1)-1;
g2=yb(2)-3;
g = [g1; g2];
end

function dydt = diff_rocket(t,y)
g = 9.81;
f1 = y(2);
f2 = -g;
dydt = [f1; f2];
end

function y = rocket_height(u)
ta = 0; tb = 5; y0 = 0;
[T Y] = ode45(@(t,y) diff_rocket(t,y), [ta tb], [y0 u]);
y = Y(end,1); % az első oszlop utolsó eleme
end

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

function g=boundary_deflection(ya,yb)
g1=ya(1);
g2=yb(1);
g = [g1; g2];
end

function g=boundary_rocket(ya,yb)
g1=ya(1);
g2=yb(1)-40;
g = [g1; g2];
end