Mechanics of Structures (CE130N)
Lab 1

1 Objective

The objective of this lab is to expose you to solving differential equations which govern mechanical phenomenon, not through hand calculations, but through numerical methods. As you will see in your studies, as the mechanical system becomes more and more complex, hand solutions will become more and more difficult to obtain; in certain cases analytical closed form solutions may not even be accessible. By using numerical methods, even when these closed form solutions are not available one will still be able to obtain good approximations for the exact solution. Good numerical methods let you control the magnitude of the permissible errors in your approximate solutions.

In this lab you will be using the software MATLAB and its built in functions to solve the mechanical problem associated with bars in tension and compression.

In this lab you will be asked to conduct the following things:

1. Understand the procedure of using MATLAB to solve the tension-compression bar problem.
2. Use it to solve a set of boundary value problems.

2 Tension-compression bar

2.1 Governing differential equation

The governing equations for a tension-compression bar are the following,

- Equilibrium

$$\frac{dR}{dx} + b(x) = 0 \tag{1}$$

The distributed load $b(x)$ can vary along the length of the bar.

- Kinematics

$$\varepsilon = \frac{du}{dx} \tag{2}$$

- Constitutive relation

$$\sigma \;\; = \;\; E(x)\varepsilon \tag{3}$$

The Young's modulus $E(x)$ can vary along the length of the bar.

- Resultant definition

$$R \;\; = \;\; A(x)\sigma \tag{4}$$

Here we have assumed a simple system, where the stress σ is constant across any cross-section. The cross-sectional area $A(x)$ can vary along the length of the bar.

In the lecture these four relations have been combined to obtain a single equation representing equilibrium in terms of the displacement,

$$\frac{d}{dx}\left[EA\frac{du}{dx}\right] + b = 0\ . \tag{5}$$

In order to utilize the solver in MATLAB, one must convert the governing equations into first-order form,

$$\boxed{\frac{d\boldsymbol{y}}{dx} = \boldsymbol{f}(\boldsymbol{y}, x),} \tag{6}$$

where $\boldsymbol{y}$ is a vector of unknown variables, and $\boldsymbol{f}$ is a vector of known functions depending on $\boldsymbol{y}$ and the position x. For the tension-compression bar, it is convenient to choose the variables u and R as the unknown variables (since the boundary conditions are typically given in terms of u and R). Two expand from first order form to second order form we can re-introduce R into (5). This yields a system of two coupled equations,

$$\begin{aligned} \frac{dR}{dx} + b \;\; &= \;\; 0, \\ R \;\; &= \;\; EA\frac{du}{dx}\ . \end{aligned}$$

The two equations can be rewritten as,

$$\frac{d}{dx}\begin{bmatrix} u \\ \\ R \end{bmatrix} = \begin{bmatrix} \dfrac{R}{E(x)A(x)} \\ \\ -b(x) \end{bmatrix}.$$

By defining,

$$\boxed{\begin{aligned} \boldsymbol{y} &:= \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} &&= \begin{bmatrix} u \\ R \end{bmatrix}, \\ \boldsymbol{f}(\boldsymbol{y}, x) &:= \begin{bmatrix} f_1(\boldsymbol{y}, x) \\ \\ f_2(\boldsymbol{y}, x) \end{bmatrix} &&= \begin{bmatrix} \dfrac{y_2}{E(x)A(x)} \\ \\ -b(x) \end{bmatrix}, \end{aligned}} \tag{7}$$

one obtains the desired first-order form,

$$\frac{d}{dx}\begin{bmatrix} y_1 \\ \\ y_2 \end{bmatrix} = \begin{bmatrix} \dfrac{y_2}{E(x)A(x)} \\ \\ -b(x) \end{bmatrix}.$$

2.2 Boundary condition

To solve the differential equation, one must apply boundary conditions. For the second-order differential equation (5), one requires 2 boundary conditions.

In order to apply boundary conditions in the solver in MATLAB, one must define a function which returns a residual of how much the boundary conditions are not satisfied; a residual of zero implies that the boundary conditions are satisfied exactly. The function has the form,

$$\boxed{\boldsymbol{g}(\boldsymbol{y}(a), \boldsymbol{y}(b))}$$

where $\boldsymbol{g}$ is vector of functions depending on the value of $\boldsymbol{y}$ evaluated at the boundary points $x = a$ and $x = b$ (Here we assume the problem is defined on the interval (a, b)).

To clarify the form of the function, consider the boundary condition,

$$\begin{aligned} u(0) &= u_0, \\ R(L) &= R_L, \end{aligned}$$

where the displacement is known as u_0 at the end point $x = 0$, and the force is known as R_L at the end point $x = L$. The vector defining the residual of how much the boundary condition is not satisfied is,

$$\begin{bmatrix} u(0) - u_0 \\ \\ R(L) - R_L \end{bmatrix}.$$

Using the correspondence between u, R and $\boldsymbol{y}$ defined in (7), one defines $\boldsymbol{g}$ as,

$$\boxed{\boldsymbol{g}(\boldsymbol{y}(0),\boldsymbol{y}(L)) \;:=\; \begin{bmatrix} g_1(\boldsymbol{y}(0),\boldsymbol{y}(L)) \\ \\ g_2(\boldsymbol{y}(0),\boldsymbol{y}(L)) \end{bmatrix} \;=\; \begin{bmatrix} y_1(0)-u_0 \\ \\ y_2(L)-R_L \end{bmatrix}.} \quad (8)$$

2.3 Numerical method

To solve the differential equation in first-order form (6), we will use the function built-in MATLAB function `BVP4C`. The following point is important to keep in mind when using numerical methods to solve differential equations.

A NUMERICAL METHOD ALWAYS GIVES APPROXIMATE SOLUTIONS!

Only in special cases does one obtain an exact solution when using a numerical method. There are cases when it seems like one has the exact solution, but this is only because the solution has been obtained to a high-degree of accuracy. Thus when invoking a numerical method, one must specify the degree of accuracy one desires in the approximate solution. The accuracy of the approximate solution $\boldsymbol{y}_{\text{approx}}$ can be measured by the relative accuracy, defined as,

$$\frac{||\boldsymbol{y}_{\text{approx}} - \boldsymbol{y}_{\text{exact}}||}{||\boldsymbol{y}_{\text{exact}}||},$$

where $\boldsymbol{y}_{\text{exact}}$ is the exact solution.

In the case of using `BVP4C` to solve the differential equation (6), there are three parameters one can adjust to determine the attained accuracy in the approximate solution. Let us assume we would like to solve the differential equation on the interval (a, b). The three parameters are,

1. x_i $(i = 1, \ldots, N)$: The points at which you (definitely) want to satisfy the differential equation, where N is the total number of points and $x_1 = a$ and $x_N = b$.

2. $RELTOL$: `BVP4C` will return an approximate solution $\boldsymbol{y}_{\text{approx}}$ which satisfies the relation,

$$\left\| \frac{d\boldsymbol{y}_{\text{approx}}(x_i)}{dt} - \boldsymbol{f}(\boldsymbol{y}_{\text{approx}}(x_i), x_i) \right\| \leq \left\| \boldsymbol{f}(\boldsymbol{y}_{\text{approx}}, x_i) \right\| RELTOL.$$

   for $i = 1, \ldots, N$.

3. $ABSTOL$: `BVP4C` will return an approximate solution $\boldsymbol{y}_{\text{approx}}$ which satisfies the relation,

$$\left\| \frac{d\boldsymbol{y}_{\text{approx}}(x_i)}{dt} - \boldsymbol{f}(\boldsymbol{y}_{\text{approx}}(x_i), x_i) \right\| \leq ABSTOL,$$

   for $i = 1, \ldots, N$.

The number and location of the points x_i must be placed so that one will be able to sufficiently represent the behavior of the solution on (a, b). A smaller value of $ABSTOL$ and $RELTOL$ will lead to a more accurate solution but will in general require more time to obtain the solution. Typically, $ABSTOL$ and $RELTOL$ are chosen between the values of 1×10^{-1} and 1×10^{-16}. (The lower bound in accuracy is due to finite precision of numbers representable on a computer.)

3 MATLAB solution method with `BVP4C`

To solve boundary value problems in MATLAB, will use the built-in ODE solver `bvp4c`. `bvp4c` take 4 arguements and returns a structure that contains the solutions.

```
SOL = BVP4C(ODEFUN,BCFUN,SOLINIT,OPTIONS)
```

Type `help bvp4c` in MATLAB to see more information on the function. In short, the first arguement is a pointer to a function which computes the right-hand side of the first order form of the ODE, the second arguement is a pointer to a function which computes the boundary condition residual, the third arguement is a structure with information to start the computation, and the fourth argument is structure with solution options.

To solve utilize this solver in MATLAB using the, one must go through the following steps.

1. Convert the differential equation into first-order form (6) and choose the unknown variables in the vector $\boldsymbol{y}$. Using this representation, construct the function `ODEFUN`.
2. Construct a function $\boldsymbol{g}$ which returns a residual of how much the boundary conditions are not satisfied as in (8). Using this representation, construct the function `BCFUN`.
3. Use the function `BVPINIT` (built-in in MATLAB) to construct an initial solution structure `SOLINIT`.
4. Use the function `BVPSET` to construct an options structure `OPTIONS`, which determine the degree of accuracy to which the solution is obtained.
5. Pass the arguments `ODEFUN` and `BCFUN` as function handles and the structures `SOLINIT` and `OPTIONS` into the function `BVP4C`.

In the following function and structure explanations, $nvar$ defined the number of unknown variables in the vector $\boldsymbol{y}$ defined in (6).

ODEFUN

```
[F] = ODEFUN(X,Y)
```

- This function serves the purpose of computing $\boldsymbol{f}(\boldsymbol{y}, x)$ mentioned in (6).
- *INPUT:*

`X` :Scalar value defining the position x
`Y` :Vector ($nvar \times 1$) representing $\boldsymbol{y}$ evaluated at x.

- *OUTPUT:*

  `F` :Vector ($nvar \times 1$) representing $\boldsymbol{f}$ evaluated at x using $\boldsymbol{y}$, i.e., $\boldsymbol{f}(\boldsymbol{y}(x), x)$.

- *EXAMPLE:* For the tension-compression bar in 1D with $E = 1$, $A = 1$ and $b(x) = \sin\left(\frac{\pi}{2}x\right)$, the function is defined as,

```
function [fxy] = bar1d_ode(x,y)

% -- Define material property and geometry
E = 1;
A = 1;
L = 1;

% -- Define distributed load
b = sin(x*pi/(2*L));

% -- Define function
fxy = [y(2)/(E*A);
                -b];
end
```

BCFUN

```
[RES] = BCFUN(YA,YB)
```

- To impose boundary conditions in MATLAB one must define a function $\boldsymbol{g}$ of the form,

$$\boldsymbol{g}(\boldsymbol{y}(a), \boldsymbol{y}(b)), \tag{9}$$

  where $\boldsymbol{y}(a)$ denotes the value of the function $\boldsymbol{y}$ at $x = a$, and $\boldsymbol{y}(b)$ denotes the value of the function $\boldsymbol{y}$ at $x = b$. This function returns a vector of residuals which defines how much the boundary conditions are not satisfied; the function $\boldsymbol{g}$ takes the value $\boldsymbol{g} = \boldsymbol{0}$ if the value of $\boldsymbol{y}$ at $x = a, b$ exactly satisfy the boundary conditions.

- *INPUT:*

  `YA` :Vector ($nvar \times 1$) defining the value of $\boldsymbol{y}$ at position $x = a$
  `YB` :Vector ($nvar \times 1$) defining the value of $\boldsymbol{y}$ at position $x = b$

- *OUTPUT:*

RES :Vector ($nvar \times 1$) representing $\boldsymbol{g}$.

- *EXAMPLE:* For the tension-compression bar in 1D with boundary conditions,

$$u(0) = u_0,$$
$$R(L) = R_L\ .$$

One defines the function $\boldsymbol{g}$ in the following way,

$$\boldsymbol{g}(\boldsymbol{y}(0), \boldsymbol{y}(L)) := \begin{bmatrix} y_1(0) - u_0 \\ y_2(L) - R_L \end{bmatrix}\ .$$

```
function [res] = bar1d_bc(ya,yb)

% -- Boundary Conditions (BC)
%    u: displacement
%    f: force

ua = 0;
fb = 1;

res= [ya(1)-ua;
      yb(2)-fb];

end
```

BVPSET

```
OPTIONS = BVPSET('RelTol',RELTOL,'AbsTol',ABSTOL)
```

- To set the degree of desired accuracy in the approximate solution, one must construct an options structure OPTIONS to pass to BVP4C.
- *INPUT:*

  RELTOL : See Section 2.3 for definition.
  ABSTOL : See Section 2.3 for definition.

- *OUTPUT:*

  OPTIONS : A MATLAB structure.

BVPINIT

`SOLINIT = BVPINIT(X,YINIT)`

- One must construct a structure `SOLINIT`, defining initial parameters of the solution, to pass to `BVP4C`.
- *INPUT:*

  `X` : Vector defining the minimum points at which one desires the solution.
  `YINIT` : Vector ($nvar \times 1$) defining the initial guess for solution.

- *OUTPUT:*

  `SOLINIT` : A MATLAB structure.

4 Lab Exercises

4.1 Function handles

If you are unfamiliar with function handles in MATLAB, try the exercise in Section 5 regarding usage of function handles in MATLAB. Make sure you understand the importance of adding the `@` mark.

4.2 Download files

1. Download the files for Lab 1 from bspace.

4.3 Tension-compression bar

1. Execute the file,

   ```
   >> bar1d
   ```

   This should give you a plot showing the displacement $u(x)$ and the internal force $R(x)$. The obtained solution is to a problem where

$$\begin{aligned} A &= 1, \\ E &= 1, \\ L &= 1, \\ b(x) &= \sin\left(\frac{\pi}{2L}x\right), \end{aligned}$$

and boundary condition,

$$\begin{aligned} u(0) &= 0, \\ R(L) &= 0\,. \end{aligned}$$

The exact solution to this problem can be obtained by solving the differential equation by hand to yield:

$$\begin{aligned} u(x) &= \left(\frac{2}{\pi}\right)^2 \sin\left(\frac{\pi}{2L}x\right), \\ R(x) &= \left(\frac{2}{\pi}\right) \cos\left(\frac{\pi}{2L}x\right)\,. \end{aligned}$$

Check the solution at a few points in the interval and verify that it satisfies the input tolerances. Note that you can use the MATLAB function `deval` to evaluate the solution structure at a set of desired points.

2. Change the file so that you obtain the solution for the problem,

$$\begin{aligned} A &= 1, \\ E &= 1, \\ L &= 1, \\ b(x) &= 0, \end{aligned}$$

and boundary condition,

$$\begin{aligned} u(0) &= 0, \\ R(L) &= 1\,. \end{aligned}$$

What should the shape of the displacement look like?

3. Change the file so that you obtain the solution for the problem,

$$\begin{aligned} A &= 1, \\ E &= 1, \\ L &= 1, \\ b(x) &= \sin\left(\frac{2\pi}{L}x\right), \end{aligned}$$

and boundary condition,

$$\begin{aligned} u(0) &= 0, \\ u(L) &= 0\,. \end{aligned}$$

Does the solution make sense?

4. Change the file so that you obtain the solution to the problem,

$$\begin{aligned} A &= 1, \\ E &= 1, \\ L &= 1, \\ b(x) &= \delta(x - L/2), \end{aligned}$$

and boundary condition,

$$\begin{aligned} u(0) &= 0, \\ u(L) &= 0\,. \end{aligned}$$

Does the solution make sense? Note, you will need to "define" the "meaning" of the delta function for MATLAB, since it is not a built-in function.

5 MATLAB tips

To use the MATLAB `BVP4C` solver, one must know how to use function handles. In basic programming, one learns how to pass "numbers" into a function. Analogous to this one can also pass a "function" into a function. This is done through the use of function handles.

The simple exercise below illustrates how they are used. We will combine the two functions:

- A function which takes a number as an argument and returns its 3rd power.
- A function which takes a function as an argument and displays the function value evaluated at 3.

Exercise

1. Write a function called `cubic` which takes a number `x` as an argument and returns its 3rd power,

```
function y = cubic(x)
y = x^3;
```

and save this as the file `cubic.m`.

2. Write a function which takes a function `func` as an argument and prints the value of `func` evaluated at 3,

```
function evalat3(func)
func(3)
```

and save this as the file `eval_print.m`.

3. Execute the function `eval_print` with `cubic` as the input argument in the command window,

```
>> evalat3(cubic)
```

You will see that this gives an error. The proper way of executing this is,

```
>> evalat3(@cubic)
```

The `@` tells MATLAB that the argument `cubic` is a function. Thus whenever MATLAB requires a function handle, one must be careful not to omit the `@`.