Adrian Loekman
UID: 105785820
CEE/MAE M20
November 16, 2021

HOMEWORK 7

1. Euler–Bernoulli Beam Bending

1.1. Introduction

In this problem, we are introduced into a beam problem. The goal in this problem is to obtain y displacement of the beam due to the force, P, for every length of the beam, x. The problem has provided the equations for the bending moments for a particular location on the beam and boundary conditions. A theoretical value of the maximum displacement will be compared to the simulation value using different nodes and output different CPU processing time.

1.2. Model and Methods

The first step in this problem is to discretize the bending moment equation, $\mathrm{EI}\frac{d^2y}{dx^2} = M(x)$. Using Taylor's Expansion, we obtain $\frac{d^2y}{dx^2} \approx \frac{y(k+1)-2y(k)+y(k-1)}{\Delta x^2}$. Thus, our equation would look like: $y(k+1) - 2y(k) + y(k-1) = \Delta x^2 * M(x)/EI$. We will need to do this for the number of nodes desired so we need to build a matrix. But before that, the parameters for this problem could be initiated first. The parameters include the magnitude of the force, the length of the bar, the point where the force is applied, cross sectional radius of the bar, and the modulus of elasticity of the bar. Using those parameters, we can build an A matrix, which will be a nodes x nodes matrix containing coefficients 1 -2 1 (except for the top left and bottom right corner which will only be 1). We can easily update the value of the corners, but the value of the others can be gotten using a for loop

```
for i = 2:nodes-1 % set a 1 -2 1 matrix for every other elements
    A(i,i-1:i+1) = [1, -2, 1];
End
```

This allows us to create a 1 -2 1 for every row that skips one matrix element every iteration so that we will have something like a diagonal -2 with 1 on its right and left.

The RHS matrix, B, can also be calculated using a for loop. Matrix B has 1 column with number-of-nodes rows. The boundary conditions are both 0. The array for X will be from 0 to L with "nodes" equally spaced elements. This way we can calculate dx. The other elements in B can be calculated using a for loop:

```
for i = 2:nodes2-1
    x = X(i);
    B(i) = dx^2 * M(x,d,P,L)/EI;
end
```

x is the point on the beam which will be evaluated. M is a function that is used to calculate the bending moment given the different variation of equation. We can rewrite the equation for M0, M1, and M2 in MATLAB form and use an if else statement to calculate M(x). I used four inputs: x,d,P,L, so that this function works for any force applied on the bar (has to be single point force) and any beam length.

After the matrix A and B is finished, we can finally calculate the y using left division (or matrix division) to obtain values for the displacement. The variable y should have elements equal to the nodes.

The next step is to find the maximum displacement of the beam due to the force. Note that the positive y axis is vertical down, thus the maximum displacement will be a positive value.

```
[ymax, idx] = max(abs(y)); % find the maximum displacement
x_sim = X(idx);
```

We can print the value of ymax and idx using fprintf. The value of ymax will be used to calculate the error with respect to the theoretical y value. There are 2 theoretical y value equations, for d > L/2 and another for d < L/2 so an if statements will be needed to evaluate for different values of d. The formula for theoretical y is given for d > L/2. For d < L/2 there are some modifications needed to be made to the formula for theoretical y when d > L/2. More of this will be discussed in section 1.4 below.

The last part of this problem would be to plot the data. The data will be the displacement in the y axis and x values from 0 to 1 with 'nodes' elements on the x axis. In addition to that, also plot the theoretical y value onto the same plot. This way, it will be easier to analyze the error between the simulation and theoretical value for the maximum displacement.

We will need to calculate this for different nodes. It can be done by easily changing the value of nodes in the parameter but doing it this way won't let us see how different nodes impact the simulation maximum displacement. Instead, I created a similar code in the script but with different nodes. I also added a tic toc command right before the nodes is initialized and after the maximum displacement value is printed in the command window.

1.3. Calculations and Results

When the main script is run, we obtain 2 outputs, statements on the command window and a figure.

```
The maximum displacement of the beam is 2.277032e-06 and occurs at
0.545455
The relative error using 100 nodes from the calculation is: 0.000219
Elapsed time is 0.077018 seconds.
The relative error using 5 nodes from the calculation is: 0.115994
Elapsed time is 0.003073 seconds.
```

```
The relative error using 10 nodes from the calculation is: 0.027134
Elapsed time is 0.005389 seconds.
```

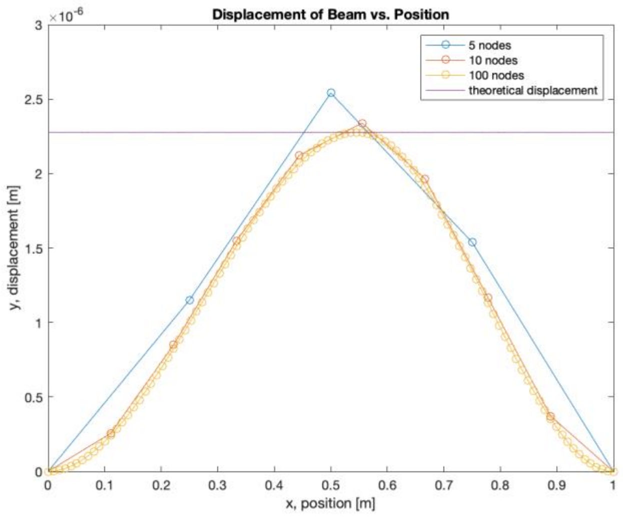

Figure 1 Displacement of Beam Plot

The statement shows the maximum y displacement from the simulation, the position where the maximum displacement is at, the relative error from the theoretical value, and the time elapsed. As an additional reference, the theoretical y value is 2.276532879572566e-06 m.

1.4. Discussion

Parts a and b are quite straightforward because it provides us with steps for initializing the problem. This includes matrix A, matrix B, and how to get the y displacements from the given matrices. From this part, a few important aspects to pay attention to are creating the matrices and making sure you have the right values for each element. The easiest way for me is to check with the theoretical value. This way, we at least know the maximum displacement and check whether this matches the theoretical value.

From the given plot, the maximum displacement of the beam due to the force is 2.277032e-06 at x = 0.545455. The theoretical maximum displacement is 2.276532879572566e-06 m. The relative error is 0.000219, 2*10^4. This can be considered as a good approximation because it has a relatively small enough error with respect to the theoretical value.

The error in approximating the node rises when the node is changed to 5 and 10. Less nodes will result in more error because now each displacement will be evaluated in the node and if the distance between the nodes is far, then the accuracy would decrease. We can see from the 5 nodes that we do not have a smooth curve. This is because each point provides a sample that corresponds to the curve. Note that deltaX is used in the calculation for matrix B. The calculation would be different for different deltaX values. However, this process uses iterations, thus using less nodes would take less time than using larger nodes.

So, the takeaway from using 100 nodes is that although it gives us a good approximation, it takes a long time for the iteration to complete. From this we can assume that for even more nodes (e.g. 1000 or 1000000) , we would have a very low error but computational time would take long because there are a lot of iterations. It's important to know a cutoff when an approximation is good enough approximation for a calculation.

When the location of d is changed to somewhere less than L/2, the y theoretical formula does not work anymore (i.e. the error becomes to large, thus not a good approximation). In order to modify the equation, we need to think backwards. We want to use a similar equation to the formula for d > L/2. So, one way we can think about it is to change our point of view. Initially, we set our point of view from the left, but now for d < L/2 we can look at it from the left. the distance d is now L-d. So in conclusion, change every d on the theoretical y formula to L-d. This should result the correct simulation value with respect to the theoretical value.

2. Langton's Ant

2.1. Introduction

The main goal of this problem is to track the path taken by an "ant" going through grids with 4 colors. The initial color is 0 and, as designed in the game rules, will change every grid color to 1, 2, and 3 according to the color the "ant" is at. The ant also turns either clockwise or counterclockwise depending on the color. The product of this code is a video consisting of the "ant's" history of steps taken in different colors and the proportion of the colors 1, 2, and 3, in a plot with respect to the timestep (2000).

2.2. Model and Methods

In this problem, we want to create a grid, so we would create a matrix of n rows and n columns where n is 50. Then, the next step would be to put the ant on a random location on the plane and we need need the ant to be facing north.

```
ant_x = randi(50,1); % random x element
ant_y = randi(50,1); % random y element
face = "North";
```

The code above puts the ant on a random x and y position on the grid and make it face north. The 3rd line actually doesn't virtually set an ant facing north but it's just a place holder so that it can change to the other directions based on the game rule.

The next step would be initializing the generations and frames from the problem statement and use linspace to create an equally spaced vector of time from 0 to the generations with the number of frames (equally spaced elements). The matrix for the color proportions could also be initiated around this line.

The next line would be initiating the video writer. This will enable MATLAB to record and create a video file that could be played showing the ant for each iteration.

```
    v = VideoWriter('LangtonAnt.avi'); % create a video with a
    particular filename
    open(v);
```

This code will create the video with the file name written and open the file later.

Since we know the rules of the game form the problem statement, we can implement it in MATLAB. But before that, note that color A = 0, color B = 1, color C = 2, and color D = 3. The color of the grid will initially be A because it's a matrix of zeros. Initiate a while loop from 1 to the number of frames. Just like the game rules, if the color of the grid is A or B (0 or 1), turn 90º clockwise, add 1 to the color and move forward 1 step. Else if the color is C or D (2 or 3), turn 90º counterclockwise, add 1 color if C or 0 if D and move forward one step. The color change is quite simple because an if else statement is sufficient. For every 90º turn, we need to know where the direction is now at. So, for clockwise: North -> East, East -> South, South -> West, West -> North and counterclockwise is basically going from the right to the left. This step is continued by the step forward where if North means that the row (x) element is subtracted by one and South is added by one meanwhile East adds the column (y) element by 1 and West is subtracting by one.

```
    clims = [0 3]; % use 4 indicator as colors
    imagesc(A, clims); % draw image on grid
    title(sprintf('Frames = %d',i)); % update for every timestep
    drawnow();
```

To draw each time step, the function imagesc will be used to generate the colors for the matrix. It will output the matrix A for each step. Use clims = [0 3] to limit that we will limit to 4 colors. The function drawnow is used to draw the update for every timestep immediately. The title can be used to track the frame because it will update each iteration.

Next, we can initiate to find the color proportion. The basic idea is to find the total of each color B, C, and D for each time step. In order to do this, we can use find.

```
colorFind = find(A == j); % find elements that correspond to each color
  colors(i,j) = sum(colorFind(:))/(num_cols * num_rows);
```

This will allow us to find the elements of matrix A equal to j, where j is equal to 1,2,3 in a for loop. The colors matrix will sum the colorFind (the total elements for each iteration) and divide by the total matrix. This will result in the proportion of each color with respect to the whole plane.

We can close the video writing because we only need it for the first part. The second part of this will be the output for the color proportion in a plot. This is gotten using plots. The plot will be filled with descriptions (xlabel, ylabel, title, legend)

2.3. Calculations and Results

When the code is run, it will output a moving image of color changes throughout time that takes around 1 minute to finish. This will be saved using videoWriter with filename LangtonAnt.avi. However, the video file is 30mb so I uploaded it to YouTube and that video could be accessed using this link:

https://youtu.be/7WwFDNsb5oE

After the video, MATLAB outputs a plot of the proportion of the colors with respect to the grid.

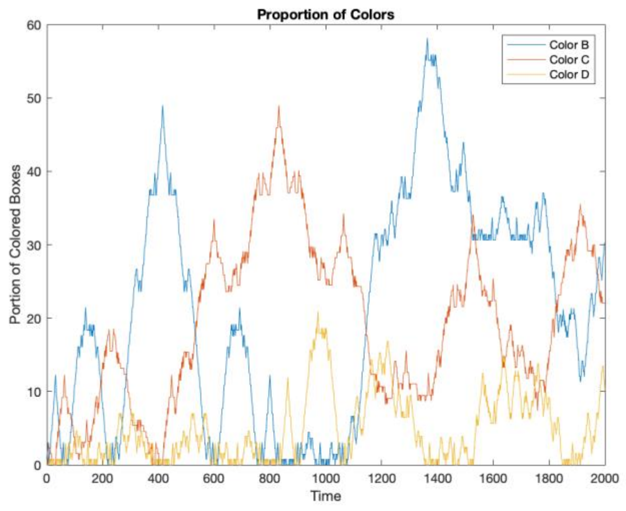

Figure 2 Proportion of Colors Plot

From the plot above we can see how many colors are spread throughout the plot and see how it increases or decreases with respect to time.

2.4. Discussion

In this Langton's Ant problem, we are introduced into a produce a video file using MATLAB and a dynamic image progressing with respect to time instead of a static plot/image of how the data changes with time. This problem is interesting because we can see how the colors are changed for every number using imagesc. This is an application of MATLAB that I did not know before so I'm glad I learned about this.

The second plot (Figure 2) shows the proportion of the color B, C, and D for each time step. As expected, the color B looks to dominate the proportion with a maximum of around 58. This makes sense remembering that most of the grids are color A. This means that for every time step, there's more chance to land on a color A element, thus changes to color B. Color C comes second and color D comes third. The colors seem to be decreasing and increasing and this looks proportional. As the proportion of the color B decreases at T = 600 to 1000, we see color C and D rise. This is because the color B is turned into color C after the majority of color A in the area are changed into color B. This also similar to color C to D and D to A.

In conclusion, we can identify that the color A will change to color B. Then as most color A are changed to B, B starts to change to C. But since C turns counterclockwise, we see a little more of B instead of D and A.