Mauricio Deguchi
UID: 405582562
CEE/MAE 20
November 16, 2021

Homework 7

1 Euler-Bernoulli Beam Bending

1.1 Introduction

This problem will simulate a common static mechanics problem: Euler-Bernoulli Beam bending. Through this simulation, we will be able to demonstrate the displacement of the beam as a function of distance in a graphical manner. We will also want to investigate different positions of the load and where the beam will experience its maximum displacement.

1.2 Models and Methods

The simulation uses a matrix (A) multiplication with a vector (Y) to obtain a vector set of moment values B. Our objective is to find the vector Y with the given information about the moment and initial matrix A. We start the script by establishing all the necessary constants. We then initialize our matrix A, and using the same discretization method from discussion, we obtain a patterned diagonal matrix with the values [1 -2 1] repeating from corner to corner. We then need to determine our set of moment values. This is done through evenly distributed x values called nodes. Using the formulas from Equation 2 in the problem statement, we can calculate the B vector. We are now dealing with the form A*Y = B, where A is our transformation matrix, Y is our displacement and B is our moment vector. Thus, to solve for Y, we use the following function: Y = A\B. This creates a vector Y with all the displacements at the corresponding node values for X. From there, we are able to create our plot of x positions versus y displacements:

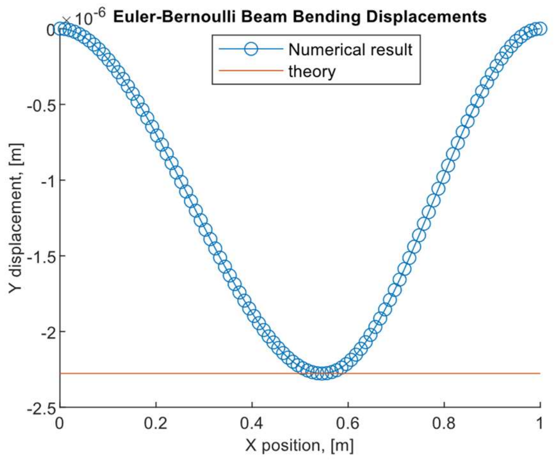

Figure 1. Euler-Bernoulli Beam Bending Vertical Displacement versus position plot when load applied at d = 0.6m

1.3 Calculations and Results

Figure 1 provides various information, specifically regarding the maximum displacement and its position. While figure 1 provides these values visually, we turn to the following formulas to find the theoretical maximum values:

$$X_{theoretica} = \frac{2dL}{3d + L - d} \qquad \left\{d > \frac{L}{2}\right\}$$

Equation 1.1 Theoretical value of X at point of greatest displacement when the load is applied a distance $d > \frac{L}{2}$.

$$Y_{theoretica} = \frac{2Pd^3(L-d)^2}{3EI(2d+L)^2} \qquad \left\{d > \frac{L}{2}\right\}$$

Equation 1.2 Theoretical greatest value of displacement Y, when the load is applied a distance $d > \frac{L}{2}$.

$$X_{theoretica} = L - \frac{2(L-d)L}{3(L-d) + L - (L-d)} \qquad \left\{d < \frac{L}{2}\right\}$$

Equation 1.3 Theoretical value of X at point of greatest displacement when the load is applied a distance $d < \frac{L}{2}$.

$$Y_{theoretica} = \frac{2P(L-d)^3(L-(L-d))^2}{3EI(2(L-d)+L)^2} \qquad \left\{d < \frac{L}{2}\right\}$$

Equation 1.4 Theoretical greatest value of displacement Y, when the load is applied a distance $d < \frac{L}{2}$.

$$error = \frac{|Y_{theoretica} - Y_{calculated}|}{Y_{theoretica}}$$

Equation 1.5 The error in our simulation's estimated value when compared to the theoretical value.

Using these equations, we can calculate any of the theoretical maximum displacements and their location, given the position of the load. However, we still need to compute the simulation maximum to check the error in our estimate. To do this, we will use the max function shown at the footnotes of page 2 of the problem statement. Applying this function to the Y array will provide us our Y_{sim} max value and its index which will allow us to find the corresponding X value. After running various trials using theses equations, we can find the tradeoff of more nodes and the cost of computation through Table 1.

Number of Nodes	Error in Max Displacement	Computation Time
5	0.115994	0.0069
10	0.027134	0.0097
100	0.000219	0.0123

Table 1. Displays value of number of nodes when compared to computation time and error.

1.4 Discussion

From Table 1, we can see that the minimal increase in computation time greatly increases the precision of our results. Thus, we can confidently conclude that, at least with a maximum of 100, more nodes was worth the computational cost. One caveat to this is that computation time will undoubtedly vary from trial to trial as there are a lot of external factors consuming RAM which leads to some variation. Lastly, let’s discuss how I arrived at Equations 1.3 and 1.4. After some experimentation in varying the d value, I found that there was symmetry about the center point of the beam at $d = \frac{L}{2}$. This symmetry allowed me to create a shifted version of Equations 1.1 and 1.2 by substituting $d = L - d$. This shift would create that symmetry about the center point and then for Equation 1.4, I needed to subtract the final value from the original length to get the same value as its symmetric counterpart.

2 Langton's Ant

2.1 Introduction

This problem takes on the challenge to model Langton's ant's movement within a grid. Its movement is dependent on the color of the tile it is standing on and the direction it is facing. For each change in position, the ant will change the color of the tile and the direction it is facing. The goal is to create a video of the ant's movement through 2000 generations.

2.2 Models and Methods

For starters, we need to construct the grid the ant will operate on. This is done by creating a matrix of zeros with the length and width of the grid given in the problem statement as 50. We then need to generate a randomized starting position for the ant; this is satisfied by generating two random numbers, multiplying them by 50 and taking the ceil of them. Such an action will generate a random integer between 1 and 50 which can be interpreted within our grid. These two random values will be stored as our x-position and y-position which will be updated for each movement. My color convention is as follows: color A = 0, color B = 1, color C = 2, & color D = 3. We also set our starting facing direction as 0. My facing convention is as follows: 0 = north, 1 = east, 2 = south, & 3 = west. Having numerical values will make turning and color changes easier in the subsequent steps. The final step prior to starting the loop will be to preallocate memory to store our quantity of each colored tile per generation. This is done by initiating a generations x 4 size matrix which will store our color quantities for each color (note: the problem statement only required tracking of colors B, C, and D but I was curious to track A and now I understand the reasoning behind this decision).

Now, we can start the loop. Within the loop, there are two major components and a few other storage pieces towards the end. The two major components are switch statements; these will determine our movement. The first switch is based on color; depending on what time we are standing on, we will change our direction accordingly as outlined in the bulleted problem statement. The color tile we are standing on will change the direction we are facing, thus within each case, we vary our facing direction. For turns 90 degrees clockwise, we can simply add 1 to our current facing value (except if the current value is 3, we will reset manually to 0). Likewise for 90 degree counterclockwise turns, we will subtract 1 from our facing direction (except if the value is 0, we will reset manually to 3). Also within each of these cases, we will store the tile color to the next colored tile and then reassign this tile color to the matrix at the end of the switch statement. Then, we can move on to the next switch statement. Here, we deal with the movement of the ant. The switch statement deals with the facing direction of the ant. If the ant is facing north, it will move up 1 on the grid (which oddly means we must subtract one from our current column position). Similar conditions apply for the other 3 facing directions, we will add 1 to our x position if we are facing east, add 1 to our y position if we are facing south, and subtract one from our x position if we are facing west. Within each of these scenarios, we must also deal with boundary conditions. So, inside each of these cases, I have a check which will ask 'if I were to do the next move, would I be off the grid?'. If the answer is yes, the x or y position will then be manually set to the opposite end of the grid (1 to 50 or 50 to 1). This ensures that the ant can walk from one of the grid to the other in a pac-man like manner. Once that switch statement is closed out, we finish that generation by storing the amount of each color through the following lines:

```
colors(i,1) = sum(sum(A == 0));
```

A similar approach is done for the rest of the colors.

2.3 Calculations and Results

This problem didn't require much calculations, it was mostly simple addition to change our direction, tile color, or position. As for the results, we are able to produce the following final image:

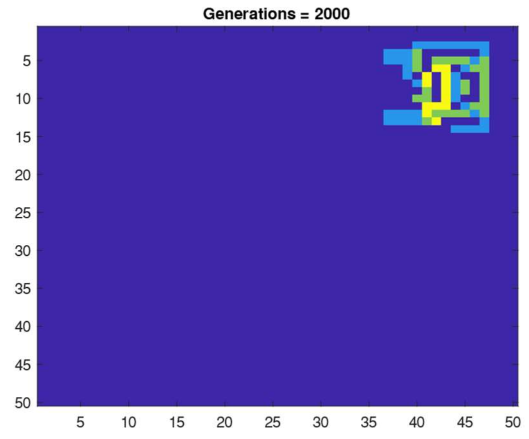

Figure 1.1 Final display of random start location

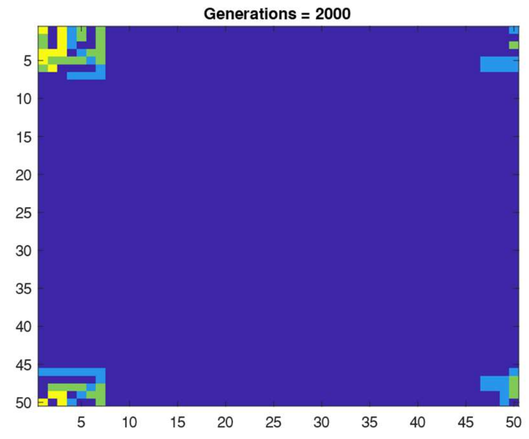

Figure 1.2 Final display from starting in the corner

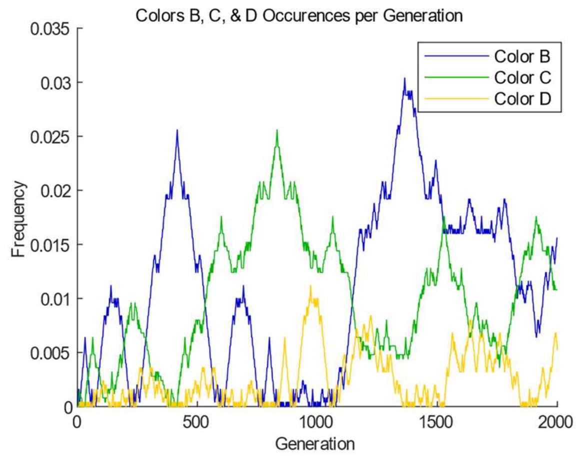

Figure 2. Displays the proportion of each color per generation

2.4 Discussion

Figure 1.1 and 1.2 show two different simulations of the ant. Figure 1.1 shows the final grid after the ant has moved over 2000 generations without crossing a border. Meanwhile Figure 1.2 demonstrates our boundary doesn't prevent movement and our grid displays an identical pattern to that of Figure 1.1 across each of the corners. Also, Figure 2 displays the proportionality of each color as a function of time. One odd observation I made was that as the simulation played out, the grid sometimes flashed different colors, leading me to believe that there was still a bug. If that were the case though, then we would see some sort of instantaneous spike and instantaneous drops in some of the colors which is not shown on the plot. This leads me to believe that the flashing is a display error on my computer or how matlab interprets the colors since no true color value was assigned to each number. Lastly, here is the link to the youtube video: https://youtu.be/n6oFU5e7j0w

This was quite challenging to implement but I found a script online which walked me through how to extract a video. The video contains 2000 frames, one for each generation of the ant's travel and the video has a framerate of 24fps which I assumed would be a reasonable default.