|  |
| --- |
| CEE 280 Advanced Structural Analysis |
| Programming Project |
| Fall 2015 |

|  |
| --- |
| Amanda Dunekacke and Siddharth Awasthi  Group 12 |

PROGRAM SUMMARY

This programming assignment is a compilation of code that interacts with the Mastan2 interface using the user-defined analysis functionality provided by the creators of Mastan2. It has the capability to analyze three-dimensional, elastic structures using the stiffness method of structural analysis. For the most part, the program can handle everything Mastan2 can handle with the exception of torques, thermal loads, member-end releases and spring stiffness.

To interact with Mastan2, the program accepts the standard arguments that Mastan2 provides, and returns matrices of deflections, reactions, and element forces formatted to match Mastan2’s expectations. This allows users to use the Mastan2 graphical user interface to define a problem and view the results, reports, and visuals.

The user-defined analysis code, ud\_3d1el.m, contains very little active code. It simply constructs an analysis object and runs the analysis. When the Analysis object is created, the constructor stores the necessary Mastan2 input as properties and also calls three private methods: CreateNodes(), CreateElements(), and CreateStiffnessSubMatrices().

CreateNodes() loops through all the nodes and constructs Node objects for each node and appends each object to form a vector of Node objects. The node properties are the node’s global coordinates, its name/number, and the degrees of freedom assigned to the displacements and rotations along the three axes at that node. Degrees of freedom are assigned when the Node constructor calls AssignDOF().

When the Element objects are created, the element constructor calls functions to create seven additional properties beyond those passed into the constructor. ComputeLength() gets the node coordinates for each end of the element and uses the Pythagorean Theorem to compute the element’s length. ComputeTransformationMatrix() uses the node coordinates and the web direction to compile a transformation matrix using vector products. With the length calculated, ComputeElasticStiffness() can use the provided formulas for the stiffness terms to build an element stiffness matrix. Then, it uses the transformation matrix to transform the local element stiffness matrix into global coordinates. RetrieveDOF() calls the public GetNodeDOF() method to store a vector of degrees of freedom associated with the ends of the element. Finally, ComputeFixedEndForces() uses length and distributed loads in fixed end force equations to store the fixed end forces in local coordinates and also in global coordinates.

The Analysis constructor concludes by calling the private method, CreateStiffnessSubMatrices(). This method calls GetGlobalStiffness() and GetElementDOF() to sum the stiffness terms at each pair of degrees of freedom into one large stiffness matrix that is stored as a sparse matrix for efficiency. The program loops over each element and gets its vector of degrees of freedom and global stiffness matrix. Then, it loops through all twelve rows and columns of the element stiffness matrix. If the value in the element stiffness matrix is non-zero, that value and its associated degrees of freedom are stored in vectors of rows, columns, and stiffnesses. Finally, one sparse structural stiffness matrix is constructed using these vectors. This large matrix is never used in its complete form, so it is not stored as a property. Instead, it is immediately dissected into the smaller sub-matrices based on the classification of the degrees of freedom which are then stored as properties for later use.

Once the Analysis object is created, the analysis can be run. In the script, ud\_3d1el.m, AFLAG, the logical indicator that indicates to Mastan2 and subsequently to the user whether the analysis has been successful, is initialized to be infinity. Infinity would indicate to Mastan2 that the analysis didn’t run as no analysis code was available. However, when the RunAnalysis() method is called, it first calls the CheckKffMatrix() method. This method estimates the condition number of the stiffness matrix for the free-free degrees of freedom. The difference between the number of reliable significant digits in the input and the return is estimated by the logarithm of the condition number. To ensure that the results have at least three significant digits, the number of lost figures is limited to thirteen (as MATLAB uses sixteen significant digits). The analysis will be successful if thirteen or fewer digits are lost; otherwise, Mastan2 will display a message that the structure is unstable, and all Mastan2 returns are set to zeros.

If the program has determined that the analysis will be successful, the ComputeDisplacementReactions() method is called in RunAnalysis(). This method conducts nearly all the rest of the stiffness method computations. It first classifies the degrees of freedom at each node based on the specified fixity within ClassifyDOF(). Degrees of freedom are assigned at each node when the node objects are created in such a way that the numbering system for degrees of freedom will match the linear indexing of the transposed fixity matrix, which greatly simplifies classification. It then sends the classified degrees of freedom to CreateLoadVectors() which creates a vector of concentrated loads and fixed end forces for each type of degree of freedom: free, fixed, and known. Only the vector of fixed end forces at the free degrees of freedom is saved as a property because it will be used again in computing error. The concentrated loads are pulled directly from Mastan2 input, but the method must call two ‘getter’ functions to get each element’s degrees of freedom and fixed end forces. ComputeDisplacementReactions() concludes by performing the matrix calculations of deflections and support reactions and formatting them into two matrices that Mastan2 can interpret.

Next, RunAnalysis() calls RecoverElementForces(). This method loops through all the elements and calls the public method in the element class, ComputeForces(). ComputeForces() receives the deflections at the element’s nodes as an argument and transforms the global displacements into the element’s local coordinates using the gamma matrix. The local deflections are multiplied by the local stiffness and added to fixed end forces to return the element internal forces in local coordinates. RecoverElementForces() compiles all of the local internal forces into a matrix for return to Mastan2.

The program performs one final function to check its own limitations; ComputeError() checks the error in the analysis by computing the difference between the input loads and the loads back-calculated from the deformations. To calculate the loads, ComputeError() performs the degrees of freedom classification again and splits the deflections into those classifications. The loads are back-calculated using those deflections multiplied by their respective stiffness sub-matrices. Then, it creates the real concentrated load vector from the input and subtracts the fixed end force vector from the concentrated loads. This creates a comprehensive load vector that is comparable to the one that was back-calculated. The difference between these two load vectors results in a vector summarizing the error in the analysis caused by loss of significant digits in the intermediate computations.

Once the error has been computed, it prints to the command window. The RunAnalysis() method is complete and returns AFLAG, DEFL, REACT, and ELE\_FOR to the ud\_3d1el.m script where Mastan2 reads the data. From that point, operations return to the Mastan2 graphical user interface where users can view the results, make changes, and run more analysis. If the same results are needed again, the GetMastan2Returns() method can access the stored properties and return the desired values.

**CEE280 Advanced Structural Analysis Fall 2014**

**Programming Project Verification Problems**

**Verification Problem 1**

Deflections at points b and c

**point b [WRITE UNITS]**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| x | 0.745 mm | 0.745 mm |
| y | -0.098 mm | -0.098 mm |
| z | -0.0006226 rad | -0.0006226 rad |

**point c [WRITE UNITS]**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| x | 0.7131 mm | 0.7131 mm |
| y | -0.1529 mm | -0.1529 mm |
| z | -0.0005621 rad | -0.0005621 rad |

Reactions at point a

**point a [WRITE UNITS]**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| Fx | 28.13 N | 28.13 N |
| Fy | 98.17 N | 98.17 N |
| Mz | -9572 N-mm | -9572 N-mm |

**point d [WRITE UNITS]**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| Fx | -28.13 N | -28.13 N |
| Fy | 87.37 N | 87.37 N |
| Mz | 74370 N-mm | 74370 N-mm |

Log10 of condition number of Kff: 8.4



Sketch of bending moment diagram

Our results perfectly match Mastan results for the digits displayed.

**Verification Problem 2**

Deflections at point b

**point b [WRITE UNITS]**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| x | 0 mm | 0 mm |
| y | 0 mm | 0 mm |
| z | -35.5 mm | -35.5 mm |
| x | -0.001078 rad | -0.001078 rad |
| y | 0.01048 rad | 0.01048 rad |
| z | 0 rad | 0 rad |

Reactions at point a

**point a [WRITE UNITS]**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| Fx | 0 | 0 |
| Fy | 18920 N | 18920 N |
| Fz | 0 | 0 |
| Mx | 877.7 N\*m | 877.7 N\*m |
| My | 0 | 0 |
| Mz | 91740 N\*m | 91740 N\*m |

Value of torsion Mx’

|  |  |  |
| --- | --- | --- |
| Member: | MASTAN results | your results |
| a-b | -877.7 N-m | -877.7 N-m |

Log10 of condition number of Kff: 7.8

Sketch of bending moment diagram



Our results perfectly match Mastan results for the digits displayed.

**Verification Problem 3**

Deflections at point d

**point d**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| x | 7.744 mm | 7.744 mm |
| y | -6.439\*10-17 mm | -6.653\*10-18 mm |
| z | 1.574\*10-15 mm | 3.672\*10-16 mm |
| x | -1.298\*10-19 rad | -3.295\*10-20 rad |
| y | -2.111\*10-19 rad | -3.255\*10-19 rad |
| z | -.00002648 rad | -.00002648 rad |

Reactions at points a and b

**point a**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| Fx | -1125 N | -1125 N |
| Fy | -2813 N | -2813 N |
| Fz | 266.2 N | 266.2 N |
| Mx | 0 | 0 |
| My | 0 | 0 |
| Mz | 0 | 0 |

**point b**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| Fx | -1125 N | -1125 N |
| Fy | 2813 N | 2813 N |
| Fz | -266.2 N | -266.2 N |
| Mx | 0 | 0 |
| My | 0 | 0 |
| Mz | 0 | 0 |

Log10 of condition number of Kff: 7.9

Axial forces

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| a-c | 3029 N | 3029 N |
| c-b | -3029 N | -3029 N |
| c-d | 6.322\*10-14 N | -1.391\*10-14 N |

Due to the slightly different operations performed in our code versus Mastan, the very small numbers have lost some accuracy. In reality, the values that are in the order of magnitude of ten to the negative tenth or lower should probably be zero. Because Matlab only stores 16 significant figures, these numbers are inaccurate and have straggling digits that are just along for the ride through the calculations.

**Verification Problem 4**

**a) include shear deformation**

Lateral deflections at each floor level

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| x3 (Roof) | .113 in | .113 in |
| x2 (3F) | 0.09067 in | 0.09067 in |
| x1 (2F) | 0.04813 in | 0.04813 in |

The maximum moments

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| column a-b | 324.7 k-in | 324.7 k-in |
| beam b-d | 429.7 k-in | 429.7 k-in |

Log10 of condition number of Kff: 7.5

**b) exclude shear deformation**

Lateral deflections at each floor level

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| x3 (Roof) | .09695 in | .09695 in |
| x2 (3F) | 0.07769 in | 0.07769 in |
| x1 (2F) | 0.04091 in | 0.04091 in |

The maximum moments

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| column a-b | 316.2 k-in | 316.2 k-in |
| beam b-d | 422.4 k-in | 422.4 k-in |

Log10 of condition number of Kff: 7.5

The percentage change in lateral deflection change

[ = 100 \* (x\_include - x\_exclude)/x\_include ]

|  |  |
| --- | --- |
|  | your results |
| x3 (Roof) | 14.2% |
| x2 (3F) | 14.3% |
| x1 (2F) | 15.0% |

The percentage change in the maximum moments

[ = 100 \* (\_include - \_exclude)/\_include ]

|  |  |
| --- | --- |
|  | your results |
| column a-b | 2.6% |
| beam b-d | 1.7% |

Neglecting shear has a much more significant impact on deflection than internal moments. In addition, deflections were taken at the left-most nodes. **Problem 5**

Base reactions at points a b c and d

**point a**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| Fx | 29.55 k | 29.55 k |
| Fy | 131.5 k | 131.5 k |
| Mz | -1466 k-in | -1466 k-in |

**point b**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| Fx | -3.961 k | -3.961 k |
| Fy | -286.8 k | -286.8 k |
| Mz | 312.4 k-in | 312.4 k-in |

**point c**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| Fx | -26.43 k | -26.43 k |
| Fy | 180.2 k | 180.2 k |
| Mz | 1512 k-in | 1512 k-in |

**point d**

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| Fx | 0.8401 k | 0.8401 k |
| Fy | -24.95 k | -24.95 k |
| Mz | 91.03 k-in | 91.03 k-in |

Shear and moments in beam e-f

|  |  |  |
| --- | --- | --- |
|  | MASTAN results | your results |
| V | 46.62 k | 46.62 k |
| Me | -7162 k-in | -7162 k-in |
| Mf | 9623 k-in | 9623 k-in |

Our results perfectly match Mastan results for the figures displayed. This is very good news that our program can handle neglecting shear as well as including it.

# 

CONCLUDING COMMENTS

Through completing this project, we learned a lot about working with a partner and programming, and we also have some feedback on the assignment. Collaborating wasn’t always easy, but we developed as a team and improved significantly in our Matlab skills and understanding of object oriented programming.

Working as partners, we had to do our best to divide tasks evenly. Over the course of development, the weight of the work and the leadership shifted at some points, but we did our best to share these. Things went a lot smoother once we began using GitHub to share and merge our code. With this tool, we could work on the same documents simultaneously without worrying about having to figure out how to combine things. It really simplified the process, especially since we did most of the work in each other’s presence. We seemed to have trouble making progress on our own, so we would work simultaneously, having the other available for questions and planning along the way. We would begin by going over what we had done recently and what we thought the next steps should be. Then, we’d each claim a task, work on it, and claim another when we finished. Siddharth took on the design document, while Amanda wrote the program summary because she likes writing and is somewhat particular about it. The coding was divided fairly evenly. It was a little difficult because we both had different ideas about how things should be done, but it resulted in a finished product where it’s hard to tell where one person’s code starts and finishes. We’ve both had our hands on every method and come to an agreement on the processes. A number of the methods were even written more than once by both of us.

Producing this project has been a very educational experience. In addition to improving our collaboration, we’ve heightened our technical skills with Matlab and program development. Before, we were both beginners with object oriented programming, and now we have become very familiar with it and (hopefully) executed good programming practice. We discovered some new functions of Matlab, namely the handling of sparse matrices. Furthermore, we explored code efficiency more thoroughly than ever before and weighed whether it would be better to store properties or call the function to create them more than once, whether to store the elements of a stiffness matrix and then stick them in the matrix or compute them in the matrix, whether to perform a calculation in one long line of code or clean it up with a for loop, etc.

In regards to the assignment, we wished for a little more freedom with our methods. When we made our first draft of the flow chart, it looked completely different from our code now because we had failed to look ahead in the assignment. It would have been interesting to see where we would have ended up with that program structure. We realize that we could have deviated from the specified methods, but we thought that would make it difficult for the instructors to grade. In addition, we would suggest the due date not be the morning after break. It makes it difficult to coordinate a polished finished product when one partner will be out of town over Thanksgiving weekend. Obviously, things came together alright, but it would be nice even just to have Monday afternoon to look over the final report, make sure everything is how we both want it, and print.

