Torrence Gue
Bilal Sharqi
Report Starting: 06-08-2020
Last Report: TBD

Weekly Reports Summer 2020

Synopsis of Week of June 29th, 2020

- Improved grid sort function using the following:
    - Added initial class definition to class compGrid to include storage of 'x' and 'y' tolerances. When sorting through the x and y grids, the code can now sort based upon a given tolerance
    - Revised X sorting process:
        - Once correct Y range of values are inputted into variable temp_list, algorithem uses nested if else statements to separate points into three categories
            1. Scenario where the experimental data x-coordinate is close to zero and numerical data x-coordinate is close to zero
                - Scenario 1 uses a essentially a small angle approximation to estimate percent difference. This equation is as follows:

                  Abs(Numerical X – Experimental X) * 10e-2

                  The values in this equation will be quite small, and will be always be stored as the smallest of possible x percent differences
            2. Scenario where the experimental data x-coordinate is close to zero, but the numerical data point is not
                - Scenario 2 just prevents a divide by zero error by making the denominator and experimental data point in the percent difference equation very very small (~10e-3)
            3. Scenario where the experimental data point is neither 1 or 2
                - Follows the typical percent difference formula relative to the experimental data point:

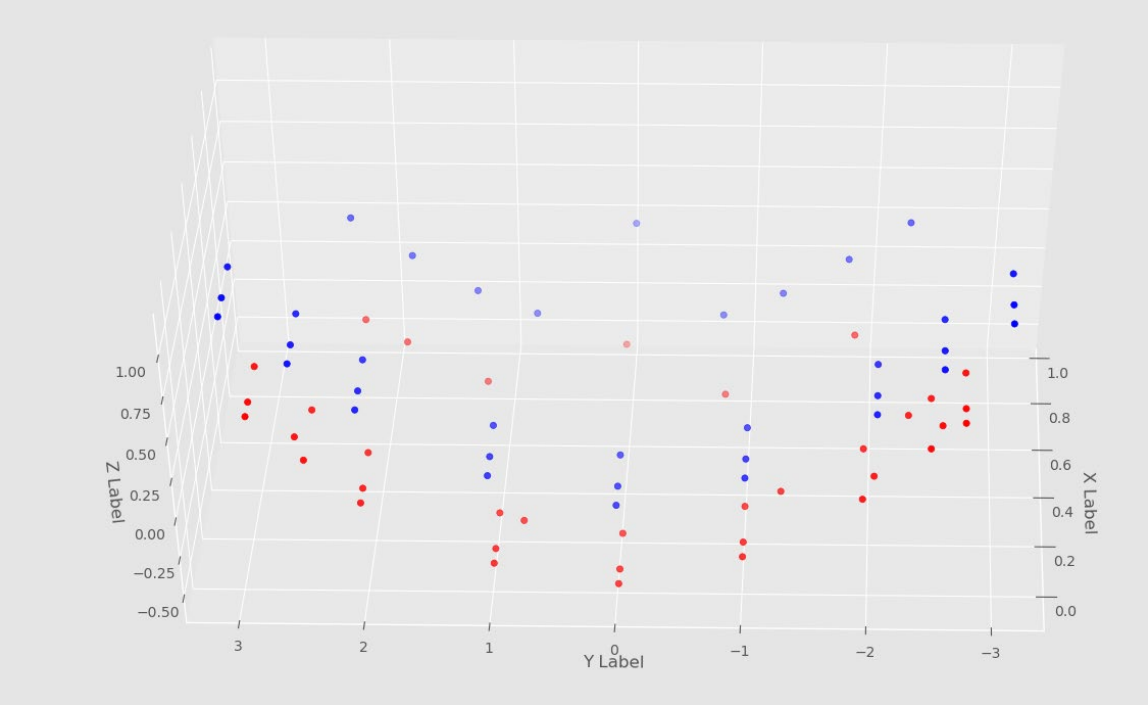
                  $$\frac{Num.x - Exp.x}{Exp.x}$$

                  This creates a percentage and dimensionless number to implement tolerances
        - After this sorting is complete. All points are filtered relative to being within 90% or 110% of the SMALLEST x tolerance. This new feature was created to get the smallest possible tolerances in x from a point. These grid points are added to a second list, sorted in order from highest to lowest tolerance, and the first index (the smallest x-tolerance) is used as the 'matched' grid point

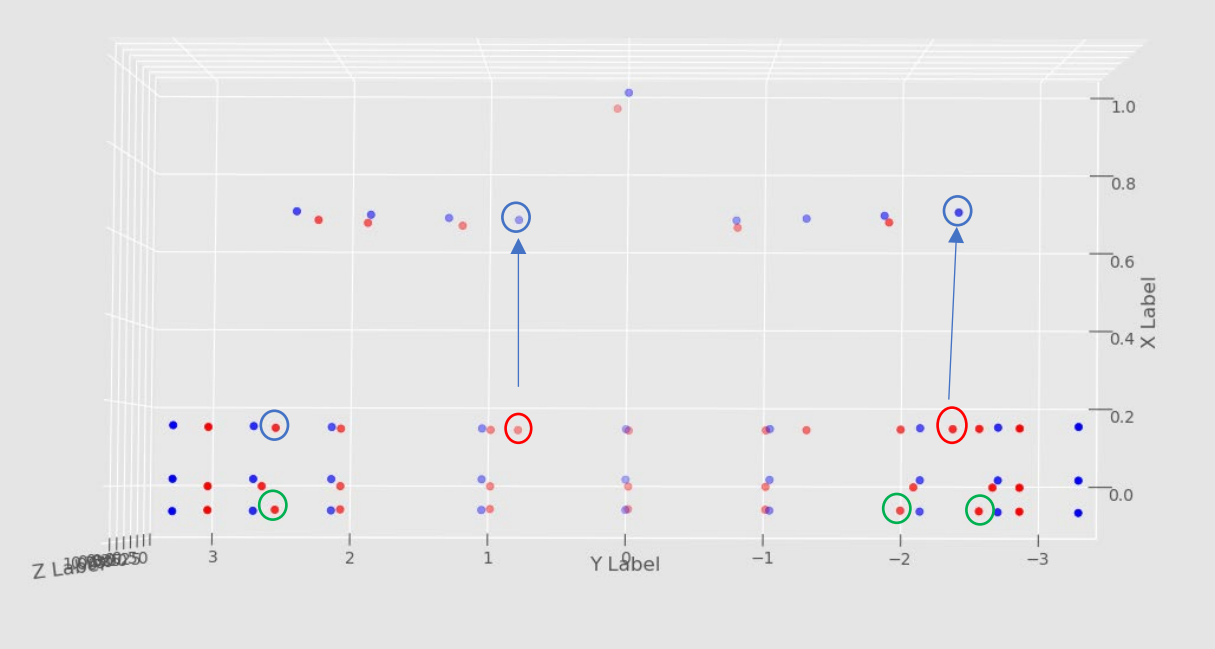
      *Note that this sorting method has also been implemented for the y functions as well
- Current problems
    - I have two grid point cases that correspond to a single outlier case in which a properly matched grid point lies outside of the initial sweep 5% tolerance (it has a tolerance of about 8% in this data set in the y coordinate, but has an extremely close x tolerance). It is excluded from the second more precise x coordinate sweep. Initially you could suggest increasing the y tolerance, but that would create a ripple effect down the line in which

less precise points are selected. I have also tried adding additional inequality statements in which the first sweep could only pass in points that are within a certain percentage of percent difference, but this filter system was too stringent and grid points could not be matched.

- This week:
    - Find a way to match the last two grid points to their place in the tail.
        - Additionally find a way to tighten grid point selection to avoid anomalies in sorting (shown in green). They are relatively close to the point, but not quite the best match. See Appendix A-2
    - Test algorithm with beam data by Wednesday at the latest
    - Fix code to match with beam data in a more general form.



Appendix A-1: Front oblique view of currently matched grid points.



Correct positions (blue)

Appendix A-2: Top view of matched grid points
(Matching grid point with two tail outliers, circled in red. Additional anomalies in green)

Synopsis of Week of June 21st, 2020

- MAC function now has automatic 3D bar graph plot. Range is from 0 to 1.
- Plotted new 'captured' grids in 3D scatter plot for accurate representation of GVT node points
- Implemented new tolerance system (still needs to be debugged!!!!)
    - Automatically appends points between -0.1 and 0.1 for further inspection in 'y-axis' (span of the wing) (-0.1 < exp y < 0.1 && -0.1 < num y < 0.1)
        - This was meant to investigate numerical nodes on the line y = 0 line that seemed to have high percent difference between the numerical node and the experimental node (appx: y = 0.01)
        - If outside of the above bounds, it uses a 5% tolerance and then an 11% refined tolerance to look for points in the y span (Proven to work before except for cases in y = 0
        - This combined method does not work because of the new implemented state (first bullet point in series). Currently looking to refine it for further and more general use
    - A similar feature has been added to x search feature matching closest x value in numerical grids to those of the experimental grid points. Needed to capture points at x = 0
        - Unlike the two step refinement
        - Tolerance is between (-0.01 < exp x < 0.01 && -0.01 < num y < 0.01
            - Not capturing any points beyond x = -0.1 (estimate)
            - Needs to debugged for further use
- Tested theory of shifting origin of aircraft from (0,0) to point (4,4)
    - Increasing origin messed with tolerances which were based upon percent difference

        - This change would be too cumbersome and require even smaller tolerances. The larger the physical number is, the less of a dramatic percent difference between two points (separated by a small distance)
- **Work to do this week:**
    - Debug the issue of capturing points at y = 0 and x = 0
    - Create a more general tolerance system to replace the 5% and 11% tolerance.
        - I want to do this tolerance based on a percentage based on the undeformed distance between two nodes in span (y-direction) and a further refinement that is TBD
    - Create an alternative, more general solution to capture points near x = 0 and y = 0 lines
        - Perhaps if I combine the new proposed tolerance method with the origin shift method this could be feasible

Synopsis of Week of June 14th, 2020

- Fixed orderPhi function to include only comparisons of the 'z-component' of the evec-measurements (need to find out if the 'mode shapes' are actual evecs)
- Fixed the remove_Grid_Freq function to read in grid point I.D's rather than grid point I.D. indices
    - Still need to find an accurate method for frequencies to match proper mode shape with proper frequencies. (Numerical and experimental modes and frequencies do not match perfectly)
- Created organizeGrids function to create function. It will sift through undeformed grid points and find the closest comparison with the experimental equivalent undeformed state
    - Progress in function so far:
        - Created class with x component of num coordinates, y component of num coordinates, grid point ID
        - Based on this class idea it should be able to rearrange the y components in order of least to greatest (very difficult to do compared to MATLAB when using the 'sort' function and associated indices on MATLAB)
- **Plans for this week:** I plan to then match closest x and y value using a previous function I created before based upon tolerances. I am still having a few issues creating the class for all of the grid points. I also need to check the validity of the MAC matrix that was calculated. I will be working to plot that in 3D, but the matrix results have been calculated.

Synopsis of Week of June 8th, 2020

- Reviewed data files of sample MAC script in 'sample matlab code for manual comparison' folder.
- Consulted Bilal on how to complete MAC script on XHALE data.
- Mapped the 36 grid points in the experimental data set with 36 of the closest matching grid points in the numerical simulation. The numerical data had to be sorted via MATLAB due to its sheer volume, however the sorting methods used in MATLAB will likely be transferrable to the Python scripts.
- **Plans for this week**
    - Calculate the MAC matrix comparing numerical and experimental modes given the 36 experimental/numerical grid points (highest priority)
    - Set up MAC function to multiply eigenvectors for in the MAC equation in the Python MAC script

- Using the matched grid points as the test case, create an automated function that sorts through numerical data points and matches them with their numerical simulation counterparts
- Find way to match frequencies with their corresponding mode shapes. This presented a difficult problem previously because given an experimental frequency, the numerical result did not match in 4 cases.

Matching Data Points (For reference)

Experimental Data Points were made numbering from starting at most positive 'y' length value and moving to most negative 'y' length value. (Numbering 1 →36)

*Note Experimental Nodes 34,35,36 have large gaps in 'y' length when numerical grid value is compared to experimental value. This may cause issues farther down the road.

Exp. Node	Num Node	Exp. Coordinate	Num Coordinate
3	15420	(-0.05756,2.985,1.13)	(-0.05754,2,847,0.5326)
1	15020	(0.01762, 2.985,1.13)	(2.223e-5,2.847, 0.5326)
2	15220	(0.1424,2.985,1.13)	(0.1424,2.847,0.5326)
6	15416	(0.05756, 2.492,0.855)	(-0.05755, 2.508,0.3201)
4	15016	(0.01762, 2.492,0.855)	(1.129e-5, 2.508,0.3201)
5	15216	(0.1424, 2.492,0.855)	(0.1425, 2.508, 0.3201)
7	14004	(0.6572,2.24,0.6944)	(0.6586, 2.222, 0.08077)
10	10420	(-0.05756, 2, 0.58)	(-0.05756, 2, 0)
8	10002	(0.01762, 2, 0.58)	(0, 2, 0)
9	10220	(0.1424, 2, 0.58)	(0.1424, 2, 0)
11	13004	(0.6572, 1.76, 0.4656)	(0.6555, 1.76, -0.1025)
12	9003	(0.6572, 1.24,0.25)	(0.6579,1.238,-0.3167)
15	5419	(-0.05756, 1, 0.18)	(-0.05762, 0.9692, -0.3816)
13	5402	(0.01762, 1, 0.18)	(-4.12e-5, 1.046, -0.3601)
14	5220	(1.424, 1, 0.18)	(0.1424, 1.066, -0.3549)
16	8004	(0.6572,0.76,0.11)	(0.6557,0.836,-0.4372)
20	16411	(-0.05756, 0, 0)	(-0.05771, -0.01882, -0.5087)
17	16011	(0.01762, 0,0)	(-0.000152,-0.01883,-0.509)
19	16211	(0.1424, 0, 0)	(0.1423, -0.01887, -0.5098)
18	4005	(0.966, 0, 0.24)	(0.9741, 0.08117, -0.3066)
11	19003	(0.6572, -0.76, 0.11)	(0.6559, -0.7312, -0.4202)

24	21411	(-0.05756, -1, 0.18)	(-0.05759, -0.9984,-0.3241)
22	21011	(0.01762, -1, 0.18)	(-2.739e-5, -0.9985,-0.3243)
23	21211	(0.1424, -1, 0.18)	(0.1424, -0.9986,-0.3247)
21	19002	(0.6572, -0.76, 0.11)	(0.6562, -0.7887, -0.403)
29	26412	(-0.05756, -2, 0.58)	(-0.05756, -2.007, 0.107)
27	26012	(0.01762, -2, 0.58)	(2.213e-6, -2, 0.58)
28	26212	(0.1424, -2, 0.58)	(0.1424, -2.007, 0.107)
26	24001	(0.6572, -1.76, 0.4656)	(0.6566, -1.782, -0.03442)
33	26418	(-0.05756, -2.492, 0.855)	(-0.05754,-2.5148,0.4263)
31	26018	(0.01762, -2.492,0.855)	(1.592e-5, -2.515,0.4263)
32	26218	(0.1424, -2.492, 0.855)	(0.1425, -2.515, 0.4263
30	25004	(0.6572, -2.24,0.6944)	(0.6587,-2.06,0.08008)
36	26420	(-0.05756, -2.985, 1.13)	(-0.05754, -2.684, 0.5327)
34	26020	(0.01762, -2.985, 1.13)	(2.122e-5, -2.684, 0.5327)
35	26220	(0.1424, -2.985, 1.13)	(0.1425, -2.684, 0.5327)

Synopsis of Week of June 1st, 2020

- NASTRAN: Simulated 1D and 2D beam in PATRAN. Was issued corrective measures on how to dimension 2D shell mesh and beam object
- Python: Interpreted original MAC script given to by Bilal. Made several key adjustments as follows
    - Imported numerical and experimental data directly from .MAT (MATLAB) files into MAC script
    - Transposed the 25 x 6 x 348 matrix to workable format of dimension 348 x 25 x 6. This took significant time because MATLAB and Python store and index three dimensional arrays differently. (MATLAB array → row, column, sheet; Python array → sheet, row, column)
    - Added the numerical "deformed state" + "static state" from cell array in MATLAB
    - Given a set number of grid location points (based upon index of imported numerical grid data, not actual node location point), reduced the number of grid points in node matrix from 348 x 25 x 6 to _grid points #_ x 25 x 6,
    - Given the indexes of the numerical frequencies that best match the experimental frequencies, reduced the node's matrix from 348 x 25 x 6 to 348 x __# of experimental frequencies__ x 6.
- **Plans for This Week:**
    - Create function that given experimental frequencies, creates an array with the indices of the numerical frequencies that most accurately match the given experimental frequencies. (This will be based upon a % tolerance that is TBD)
    - Create function that uses location of grid points to match experimental grid locations with their numerical counterparts.