Project 3 Write-Up

In the first phase, we were given a linear solver method via Gaussian elimination and upper triangular solving (gauss\_elim.m, upper\_solve.m), and we had to use the solution vector from these functions to test out an LU decomposition method that we created. I created 3 scripts in phase 1: lu\_decomp.m, lower\_solve.m, and Phase1.m. lu\_decomp.m takes in a matrix A that must be square, and it outputs L and U, which are respective lower and upper triangular matrices that when multiplied together, they equal A. L is composed of 1’s on the diagonal, m values underneath them, and 0’s everywhere else. I found the m values by dividing the current location of A(i,k) by the value on the diagonal of the same column A(k,k) (k = column number, I = row number). U is the matrix A after it has gone through Gaussian elimination, and is therefore an upper triangular matrix. Lower\_solve.m works well with LU decomposition because it takes in a lower triangular matrix L and a right hand side b and solves for a solution vector c via forward substitution. The Phase 1 testing script Phase1.m is a pure comparison between Gaussian elimination/upper solve and LU decomposition/lower solve/upper solve, and shows that they produce the same solution when they are executed with the same A matrix and b vector. First it calls gauss\_elim.m and inputs A and b, and produces another A, now an upper triangular matrix, and b after the method is complete. Then it calls upper\_solve.m with the new A and b and outputs the solution vector x, which it finds via backwards substitution (x = [9.12; 10.4; 7.96; 9.96]). The second part of Phase1.m is dedicated to proving that the LU decomposition method produces the exact same solution vector. First, the lu\_decomp.m function is called with an input of a square matrix A, and it outputs the lower and upper triangular matrices L and U. Then it models the equation LUx = b, which must be broken into two smaller equations: Lc = b and Ux = c. After lu\_decomp.m is called, lower\_solve.m is called with inputs of lower triangular matrix L and right hand side vector b, and it outputs a vector c which is the solution vector of L and b which is found via forward substitution (c = [20; 66; 82.1429; 111.4925]). Then directly after this function call, it calls upper\_solve.m with inputs of upper triangular matrix U and right hand side c. upper\_solve.m. upper\_solve.m uses backwards substitution to return a vector x which is the final solution vector of the LU decomposition method (x = [9.12; 10.4; 7.96; 9.96]). Finally, the function Phase1.m outputs the solution vector of the LU decomposition method and subtracts the Gaussian elimination/upper solve and LU decomposition/lower solve/upper solve solution vectors, which is a 0 vector of length n.

For Phase 2, I only made changes to one script: bridge.m, but I used the functions that I created in Phase 1 (aside from the testing script Phase1.m). Initially, I added 4 more forces to bridge.m, which are titled force2, force3, force4, and force5. With these forces, I added different weights to all of them by making sure that none of them were equal to each other, and then I created 4 more right-hand-side vectors via bridge\_rhs.m which takes in a vector and an integer representing the bridge type. I synchronized the 5 forces with the 5 rhs vectors, and titled the rhs vectors b, b2, b3, b4, and b5. I then created B, B2, B3, B4, and B5 and set them equal to the 5 rhs vectors, just in case I changed them during runtime. The main point of this script was to time the difference between Gaussian elimination/upper solve and LU decomposition/lower solve/upper solve, and to see how much they differentiated from each other. I began with starting a timer t and began to find the solution vectors via Gaussian elimination/upper solve for the 5 different systems (Ax = b, Ax = b2 … Ax = b5). Once all 5 were solved, the timer ended and printed the total elapsed time to the screen, which was 3.42247 seconds. Then I started a different timer and began finding the LU decomposition of the bridge matrix, and this created the matrices L and U. I then used the L and U matrices to solve the equation LUx = b (which I computed as Lc = b and Ux = c) 5 times for the 5 different rhs vectors. The total time of this was .81194 seconds, and most of this time came from solving the Gaussian elimination one time within the initial LU decomposition method. I then solved via guess-and-check and a partial human binary method that the minimum load that can be placed in a single location on the bridge to cause a beam to break is 7.8909 lbs. I placed this weight in the very center of the bridge because that is where a weight will have the most impact. I then found again via guess-and-check that the maximum overall weight that can be supported by the bridge, assuming a uniform load across the bridge, was 159.549 lbs, or 4.091 lbs per node.

The reason that the Gaussian elimination/upper solve was much slower than LU decomposition/lower solve/upper solve is because gauss\_elim.m and upper\_solve together have a complexity of O(n)^3, and this is repeated 5 times in the second phase. lu\_decomp.m has a complexity of O(n)^3 as well, but it is only computed once, and then only lower\_solve.m and upper\_solve.m are used to find out the 5 solution vectors, and these 2 solver methods have a complexity of O(n)^2. This overall decreased complexity is why LU decomposition is preferable to Gaussian elimination if you are keeping a standard matrix A but changing the rhs vector b, especially with increased sizes of A and/or b.