Pishawikar, Pranav

University of Connecticut | ME 5895

Final project report DFEM

Design of bar element to minimize compliance

**Problem Statement**

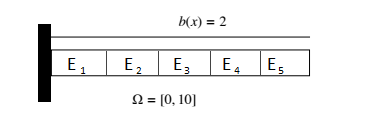


Figure 1

We want to find an optimum design for a 5 element bar of length 10 (Fig 1). The bar has a constant body load across each element, defined as b(x) = 2 for Ω = [0, 10] for our problem. We want to minimize the compliance of this bar, defined as:

Where,

U is the displacement, which is a function of position, x and the modulus E(x)

b(x) is the body load (defined = 2 for our problem)

and is the Prescribed stress (considered to be zero in our problem)

Substituting this zero prescribed stress condition in the above equation, we get the compliance function:

To prevent our optimizer from giving us the trivial solution here by increasing all the moduli of elasticity to infinity, we implement a constraint function. We want the sum of all moduli of elasticity to be no more than 10 for our problem. That is, our constraint function is:

**Approach**

We will be using MATLAB to solve the above design problem. We will be modifying the spring system code provided in Chapter 3 of Prof. Tortorelli’s book to fit our bar system.

To calculate the sensitivities, we will use the Adjoint method. This method would best suit our needs because there are a total of just 2 functions that we need to solve for (The cost function and constraint function introduced in the problem statement and formally restated below), while we have 5 design variables, one for the modulus of each of the elements in our problem.

We shall first use a gradient based optimization method (fmincon in MATLAB) to find a solution and then try to reach the same solution using a Genetic algorithm. We will also attempt to use fmincon without supplying the gradients for the cost and constraint functions.

Our goal is to compare these optimization methods in terms of amount of time spent to get to the solution using each of these methods, number of iterations required in each case and the number of function evaluations made using each method.

Our cost function is formally defined as:

Subject to the single constraint function:

Our design variables here are the moduli of the elements themselves, ie,

Due to the way our constraint function is defined, and by the fact that modulus E can never fall below zero, we need not really define any upper or lower bounds for our optimization. However, as we shall see in the following sections, while using a GA, it becomes necessary to define very strict upper and lower bounds. This becomes necessary in order to limit the search space for the design variables and ensure that our algorithm starts to converge within a reasonable amount of time. For this purpose, during our GA run, we will define the upper and lower bounds as:

**Code usage and modifications**

In order to apply the sensitivity analysis code to the bar, certain changes had to be made to the spring analysis code discussed in chapter 3.

1. First off, the input file itself needed to be modified (Please see file 1.txt attached in the google drive link in references section to see exactly what each number in the input.txt file represents) :
2. To accommodate the Node coordinates, . This is defined in the N\_NODE lines following the first line of the input file.
3. The Element body load, data needs to be added to the N\_ELEM data lines that define the Elements themselves
4. The initial design values for the N\_PARAM lines need to be added so that the optimization module has an initial design to start from.
5. The input module needed to be modified to accommodate the above mentioned changes.
6. The Element module needed to be changed in order to take the new Element load data and the nodal coordinates and use these to calculate and output the element force array PEL(i). The computation of the element stiffness matrix, KEL(i) also needed to be modified slightly.
7. The Element stiffness assembly module needed to take the new PEL(i) input and add prescribed and free element forces , PP and PF as output after appropriately computing them.
8. The Force assembly (assemble all) module needed to be modified in order to take the coordinate and element load data and use this in the computation that uses the aforementioned element module.
9. The evaluate module had to be rewritten according to the problem definition.
10. Minor changes needed to be made to the Adjoint module to properly interpret and make use of the new data.
11. As I was unable to create the optimization module for the spring analysis, I had to start afresh here and make sure that the module worked properly.

The modifications made to the code were thoroughly tested by using different solved examples from chapters 4 and 5 of Prof. Tortorelli’s book and checking the solution and sensitivity computations. Even though our problem is one with no prescribed stress at , problems with prescribed stress were also tested to make sure that the code is fairly robust. The weak problem breakdown and finite element transformation of this problem is handled according to the Galerkin Finite Element Method discussed in section 5.2 of prof. Tortorelli’s book.

**Results**

1. Fmincon with gradients:

This method converges onto the correct result with the least number of iterations.

The design history is shown below:

|  |  |
| --- | --- |
| Iteration number |  |
| 1 | 2.0000 2.0000 2.0000 2.0000 2.0000 |
| 2 | 3.9038 3.4714 1.5451 1.0598 0.0199 |
| 3 | 2.9519 2.7357 1.7725 1.5299 1.0100 |
| 4 | 3.1397 2.6440 2.8748 1.3265 0.0150 |
| 5 | 3.0458 2.6898 2.3237 1.4282 0.5125 |
| 6 | 3.4308 2.9291 2.0684 1.4401 0.1316 |
| 7 | 3.2383 2.8095 2.1960 1.4342 0.3220 |
| 8 | 4.2120 3.0901 2.1507 0.0171 0.5300 |
| 9 | 3.7252 2.9498 2.1733 0.7256 0.4260 |
| 10 | 3.4817 2.8796 2.1847 1.0799 0.3740 |
| 11 | 4.4949 1.6258 1.8544 1.5253 0.4996 |
| 12 | 3.9883 2.2527 2.0195 1.3026 0.4368 |
| 13 | 3.7350 2.5662 2.1021 1.1912 0.4054 |
| 14 | 3.4490 3.0622 1.5628 1.4397 0.4863 |
| 15 | 3.5920 2.8142 1.8325 1.3155 0.4459 |
| 16 | 3.5850 2.8068 2.0129 1.1969 0.3984 |
| 17 | 3.6010 2.7995 2.0002 1.1992 0.4001 |
| 18 | 3.5999 2.7999 2.0000 1.2001 0.4001 |
| 19 | 3.6000 2.8000 2.0000 1.2000 0.4000 |
| 20 | 3.6000 2.8000 2.0000 1.2000 0.4000 |

Figure 2.1

As seen in figure 2.1, the design converges within 20 iterations. The plot for the design history is given in figure 2.2

As seen in figure 2.3, the optimzation completed in less than 2 seconds. 20 calls were made by the costfunction and 20 calls were made by the constraint function on the analysis module, making the objective function converge within 40 calls to the analysis module. This means on an average, each call cost just about .05 seconds

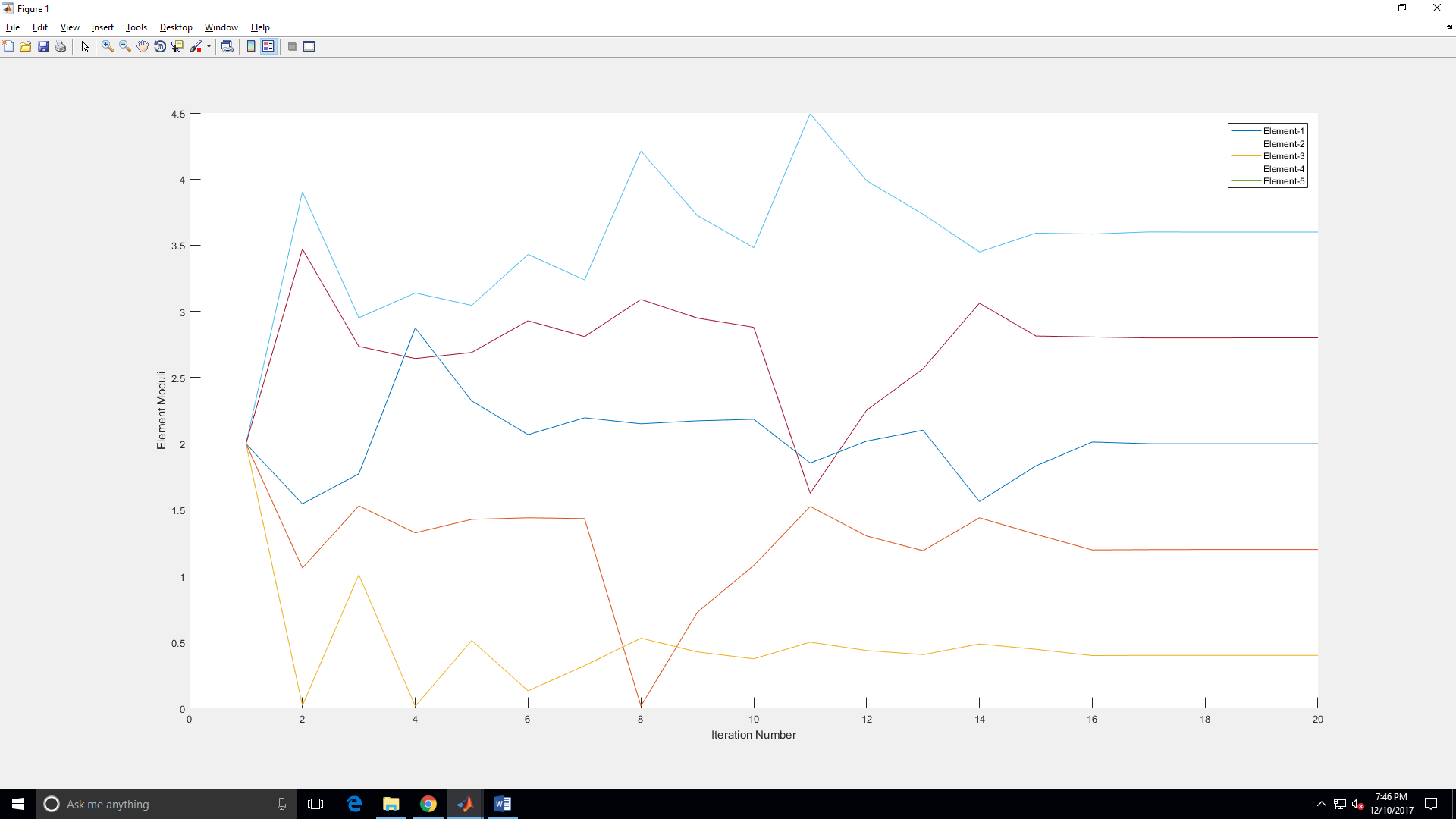


Figure 2.2

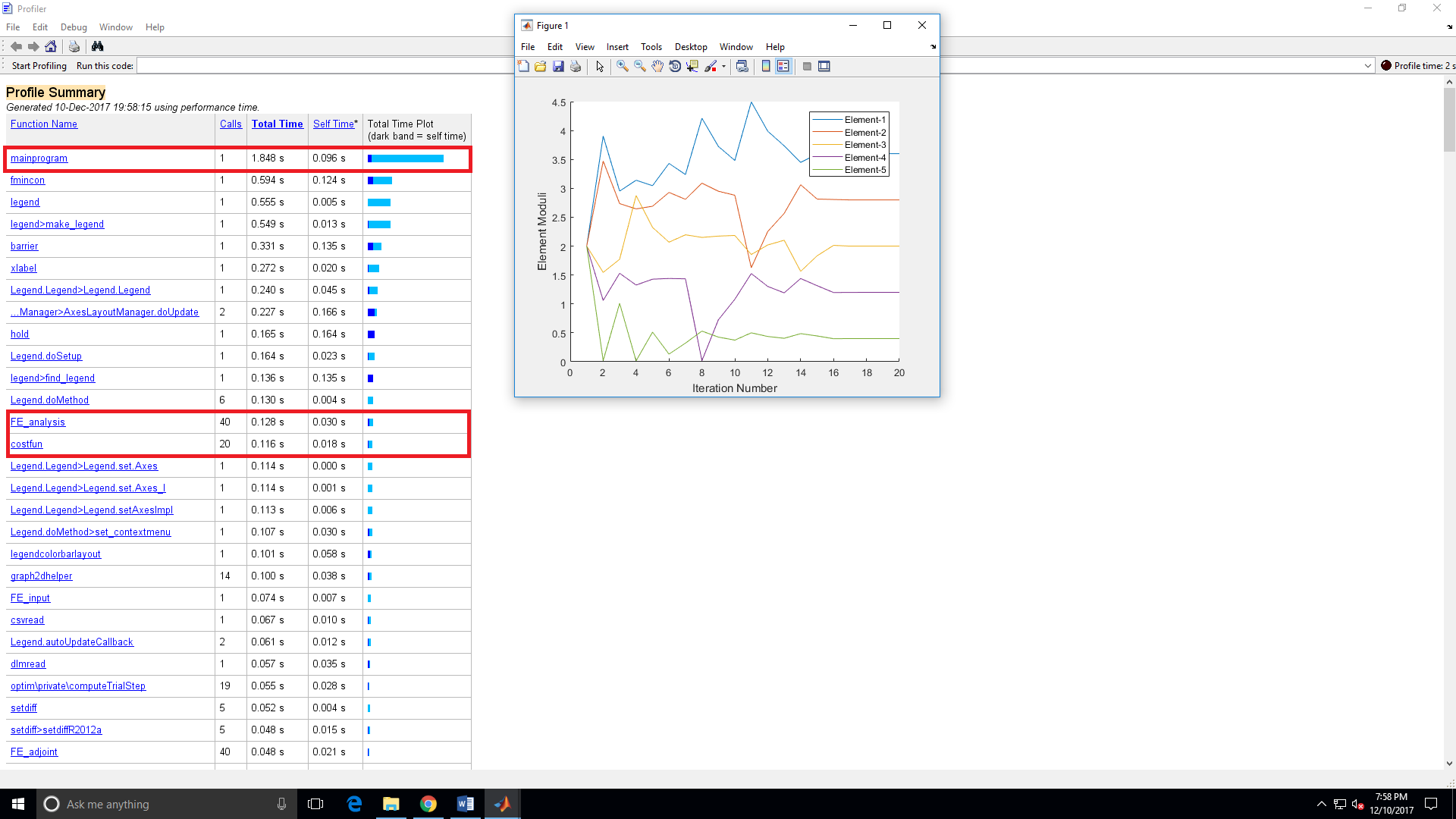


Figure 2.3

1. Fmincon without gradients:

This method converges relatively slowly. The results are aslso a little less accurate as it is not able to satisfy the constraintsaccurately. However, this gmethod gives a fair enough result and also gets more accurate when the initial design is changed. However, for the purpose of this exercise, no such considerations will be made, and starting at the same initial design as fmincon with grad, we see that this method fails.

The design history is shown below (for conciseness of space, intermediate itterations have been deleted from the table):

|  |  |
| --- | --- |
| Iteration number |  |
| 1 | 2.0000 2.0000 2.0000 2.0000 2.0000 |
| 2 | 2.0000 2.0000 2.0000 2.0000 2.0000 |
| 3 | 2.0000 2.0000 2.0000 2.0000 2.0000 |
| 4 | 2.0000 2.0000 2.0000 2.0000 2.0000 |
| 5 | 2.0000 2.0000 2.0000 2.0000 2.0000 |
| 6 | 2.0000 2.0000 2.0000 2.0000 2.0000 |
| 7 | 4.1187 3.4046 1.5080 0.9488 0.0199 |
| 8 | 3.0593 2.7023 1.7540 1.4744 1.0100 |
| 9 | 3.0593 2.7023 1.7540 1.4744 1.0100 |
| 10 | 3.0593 2.7023 1.7540 1.4744 1.0100 |
| … |  |
| 81 | 3.7289 3.6798 2.4784 0.8703 0.2341 |
| 82 | 3.7289 3.6798 2.4784 0.8703 0.2341 |
| 83 | 3.7289 3.6798 2.4784 0.8703 0.2341 |
| 84 | 3.7289 3.6798 2.4784 0.8703 0.2341 |

Figure 3.1

As seen in figure 3.1, the design converges within 84 iterations. The plot for the design history is given in figure 3.2

As seen in figure 3.3, the optimzation completed in less than 1 second. 84 calls were made by the costfunction and 84 calls were made by the constraint function on the analysis module, making the objective function converge within 168 calls to the analysis module. This means on an average, each call cost just about .01 seconds! This seems counter intuitive that fmincon should be faster to converge albeit with more iterations, when the gradient is not supplied. However, this should actually be expected as we shall discuss in the conclusions section.

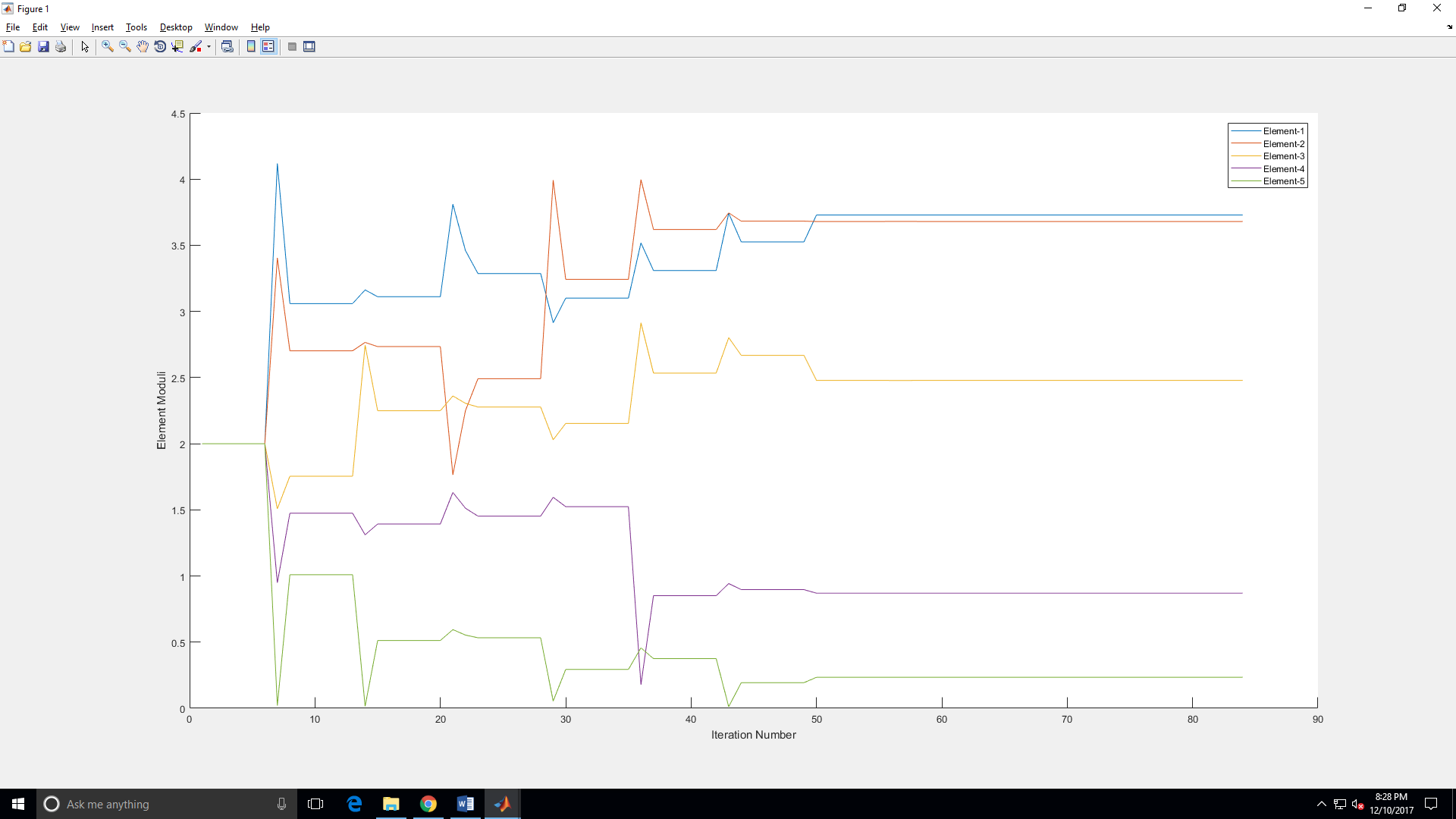


Figure 3.2

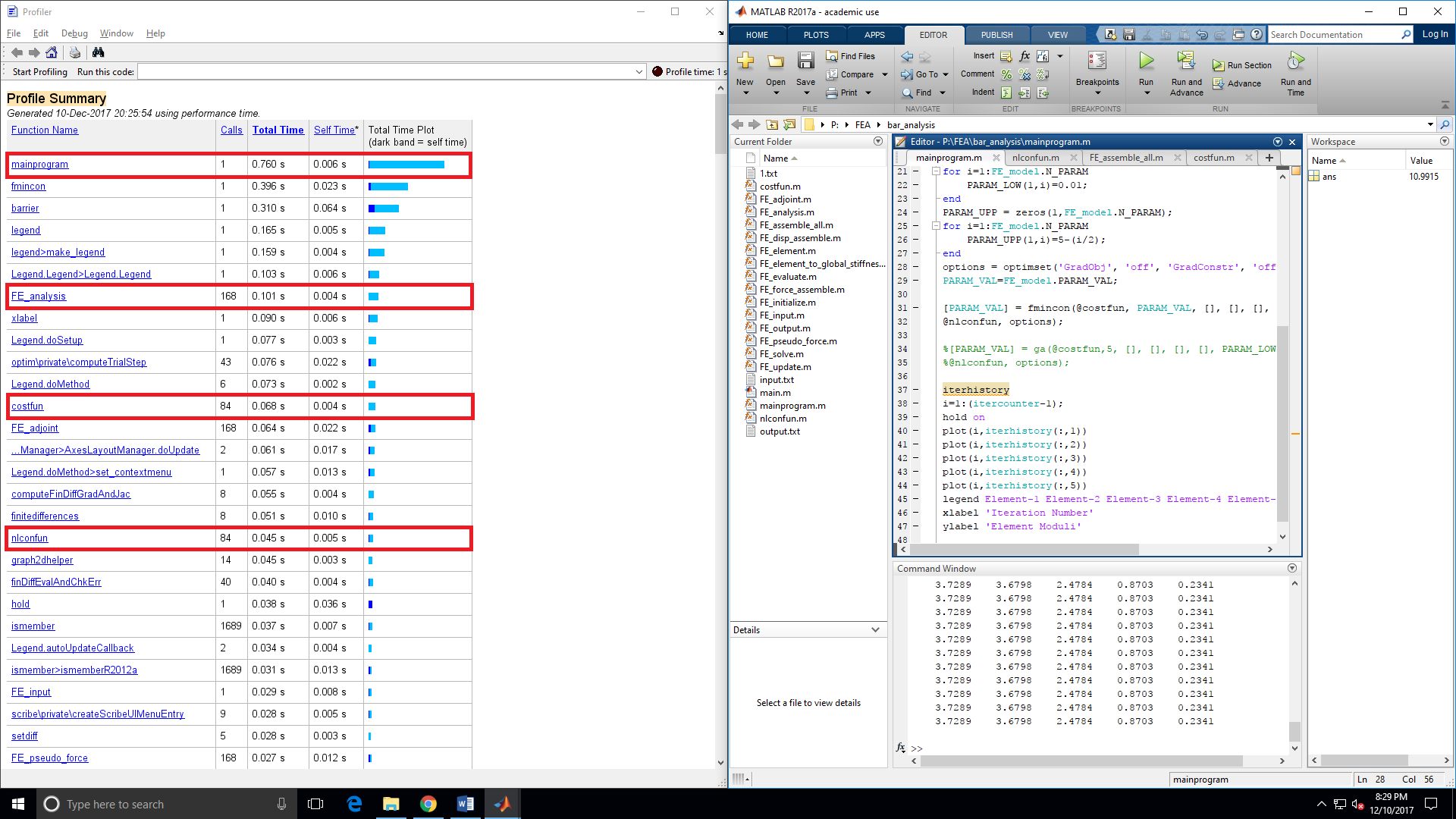


Figure 3.3

1. Genetic Algorithms:

I had a lot of difficulty while trying to use this method. After trying to run multiple analyses, I saw that the algorithm would not terminate (even after running for well over 3 hours in one particular case). After some further research, I figured out I was having problems because my design space was too large. This was evident when tested the code without semicolons ‘;’ at the end of the lines where the parameter values are added to the design history.

I then changed my upper and lower bounds to as mentioned in the approach section. I could set these bounds knowing they were safe as a result of my analyses using fmincon.

However, even with this, the code was taking too long to run. During a long run of this refreshed code, I lost patience after observing that the algorithm had gone through well past 600,000 iterations without even coming close to the expected result.

I finally decided to help the GA further by reducing the design space and imposing the inequality constraint condition, as an equality constraint, knowing that logically, the ideal solution should lie in the design space where

With this final push, the GA was finally able to converge to a solution. However, the solution still infeasible even with multiple runs of the algorithm.

The plot for one such run is given in figure 4.1. As seen from the figure, during this particular run, the solution offered by the GA is nowhere close to the actual expected result. The anwer provided by the analysis during this particular run was:

Although the analysis did manage to get fairly close to the expected results during one particular run (while still violating the constraints), I had not started to record all my results at that point and I never managed to get that close again during my future runs.

The GA took by far the longest time to run, even after shaping the constraints and the bounds to help it out. The run times can be clearly seen in figure 4.2. The algorithm took well over 4 seconds to terminate, having made 6821 calls to the analysis module!

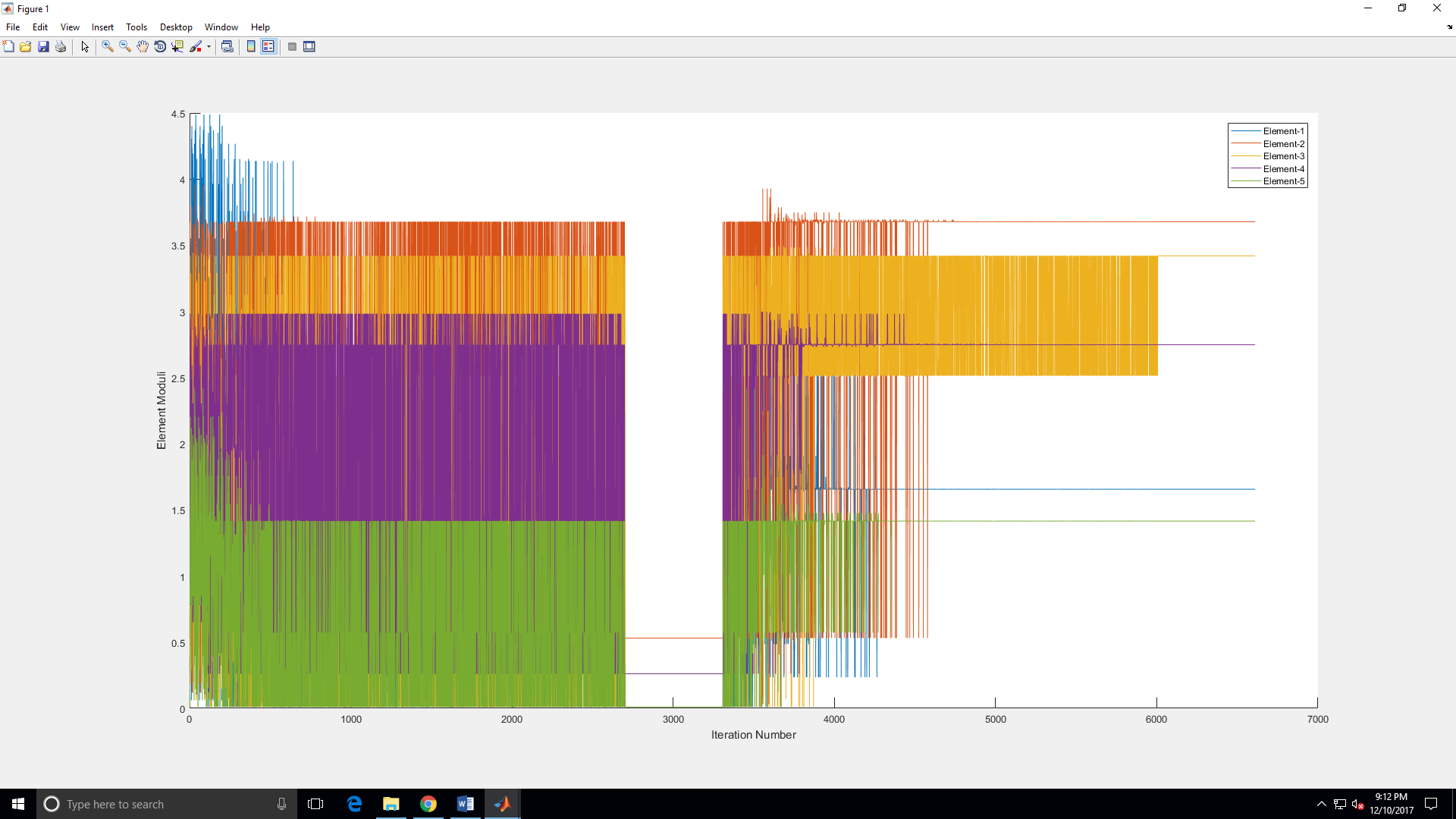


Figure 4.1

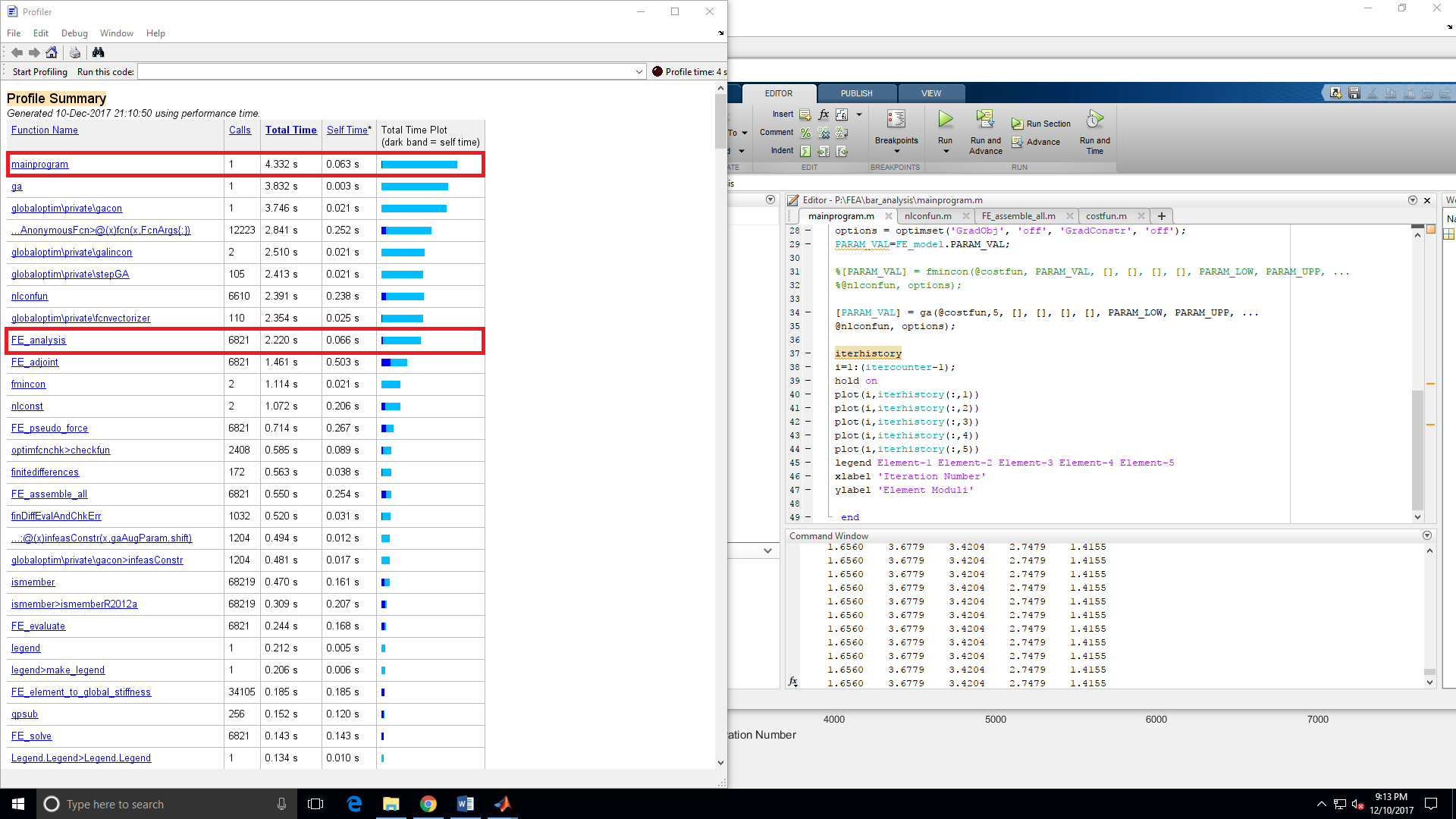


Figure 4.2

**Conclusions**

In my observations, the gradient based optimizations worked far better than the genetic algrithm. With better knowledge and limits defined on the GA, we could probably get a better result from it. However, this would take a lot of involvement from the analyst, having to provide stricter bounds and running the algorithm multiple times to converge onto a good feasible solutions.

For fmincon itself, we note that although providing fmincon with gradients yields more accurate results, fmincon without gradients converges on the solution faster. Although this is slightly counter intuitive, we must note here that due to the nature of the problem defined, our analysis module itself took far less time than the overhead involved in processing fmincon with gradients.

With an analsis that is harder to solve (typically this is the most computationaly hungry calculation in real world engineering problems), we would spend far more time using fmincon without supplying gradients. This is because we would need to call the analysis far more times when we do not supply the gradients. Note that in the case of fmincon without gradients, we ended up calling the analysis module more than 4 times that we needed to call it during fmincon with gradients.

**References**

Figure 1 is a modified version of figure 5.12 in prof. Tortorelli’s book.

Equations for compliance are also taken directly from prof. Tortorelli’s book, Chapter 4 Problem 4.24.

Spring analysis code was also generated using prof. Tortorelli’s book chapter 3.

The domain <https://www.mathworks.com/help/> was used for a lot of MATLAB troubleshooting.

The final version of the bar analysis code can be found and downloaded from google drive here:

<https://drive.google.com/drive/folders/1br98FpyGSDPMp3swih8wZ6RukIfDZrbI?usp=sharing>