Thomas Satterly

AAE 550

HW 1

**Part I**

Given:

*See attached for problem statement definition*

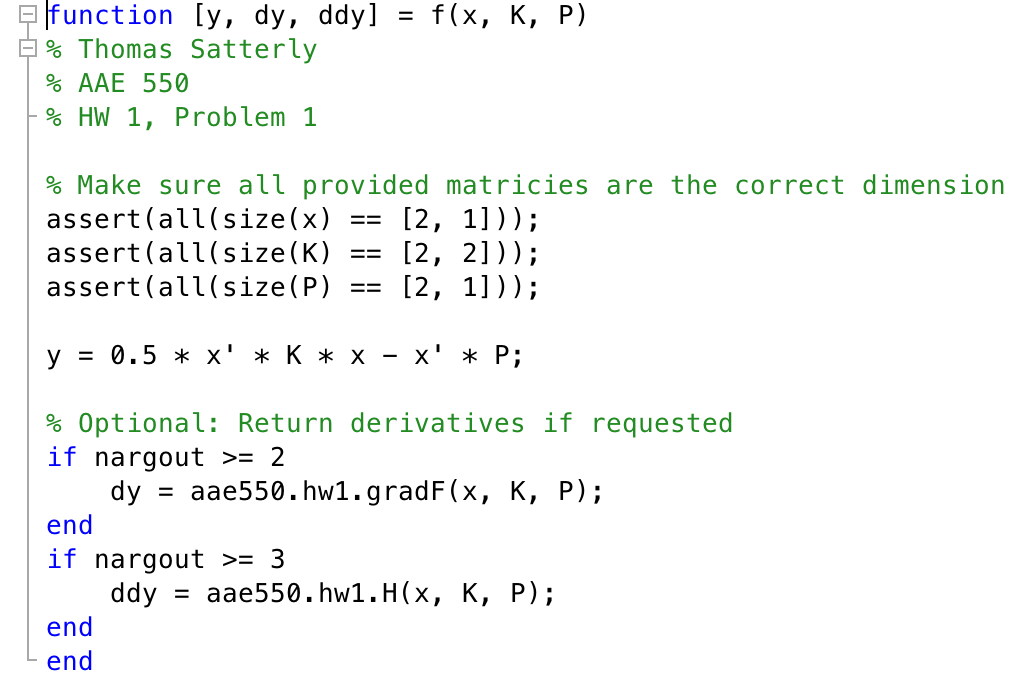
Find:

1. *See attached for definition of Part I: Problem 1*
2. Provide the Matlab snippet for f(x) and the gradient and hessian of f(x)
3. Use “fminunc” in Matlab’s Optimization Toolbox to solve this problem for the equilibrium positions of the carts. Pay attention to the “exitflag” and “message” information to determine if the algorithm has converged. The default algorithm uses the BFGS update
   1. Solve the problem using finite difference gradients. Record x0, x\_star, f(x\_star), grad\_f(x\_star), the number of iterations needed, the number of function evaluations needed, and the value of “exitflag”.
   2. Solve the problem using analytic gradients. Record x0, x\_star, f(x\_star), grad\_f(x\_star), the number of iterations needed, the number of function evaluations needed, and the value of “exitflag”.
4. Matlab offers two other first-order methods, the DFP update and steepest descent. Explore these to solve the problem for the equilibrium postion of the cars. Pay close attention to “exitflag” and “message” to determine if the algorithm has converged.
   1. Solve the problem using analytic gradients with the DFP update. Record x0, x\_star, f(x\_star), grad\_f(x\_star), the number of iterations needed, the number of function evaluations needed, and the value of “exitflag”.
   2. Solve the problem using analytic gradients with the steepest descent update. Record x0, x\_star, f(x\_star), grad\_f(x\_star), the number of iterations needed, the number of function evaluations needed, and the value of “exitflag”.
5. Use Matlab’s solver with the modified Newton’s method and user-supplied gradient and Hessian values. Record x0, x\_star, f(x\_star), grad\_f(x\_star), the number of iterations needed, the number of function evaluations needed, and the value of “exitflag”. Record x0, x\_star, f(x\_star), grad\_f(x\_star), the number of iterations needed, the number of function evaluations needed, and the value of “exitflag”.
6. The Excel Solver add-in can also be used to solve this problem. Solve the problem using the default options in Solver. Record x0, x\_star, f(x\_star), grad\_f(x\_star), and the number of iterations needed.
7. Make a table of the various approaches. Use a reasonable number of significant digits. What conclusions can be made about these unconstrained minimization approaches for this problem? Is there a significant difference in cost and/or accuracy when using numerical derivatives and when using analytic derivatives? How or why might the form of this problem be better suited to one of the above techniques?
8. State the optimality conditions for an unconstrained minimization problem and show why the displacements of the carts are usually found by solving **Kx = P** for **x**. Using this strategy, what are the optimal displacements of the two carts and the resulting potential energy of the system? How does this answer compare to the answers found previously?

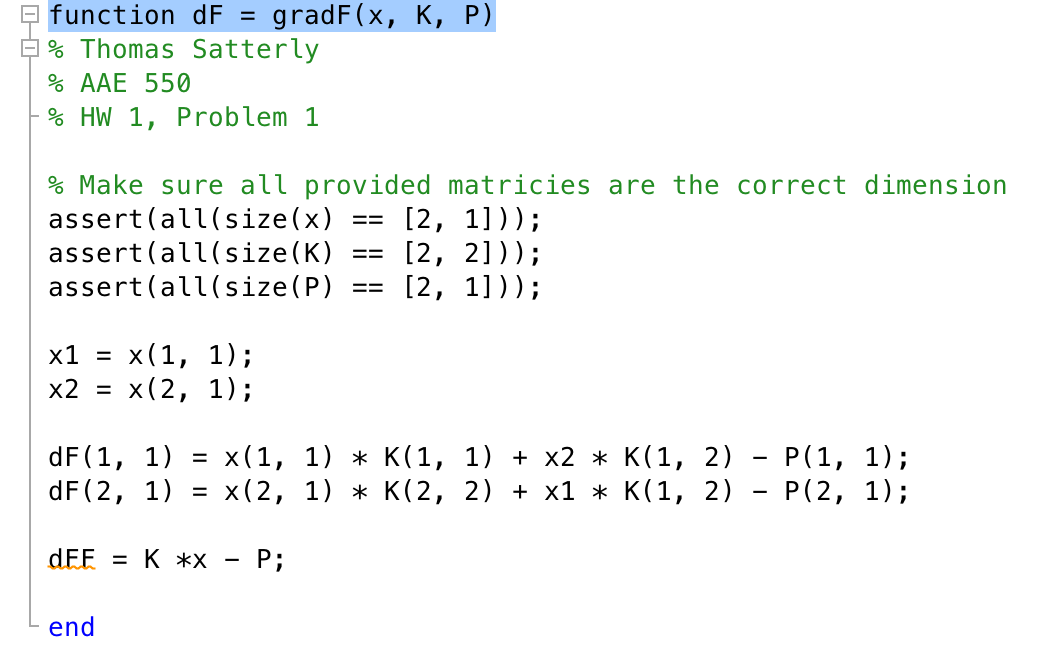
Solution:

1. *See attached for solution to Part I: Problem 1*
2. Matlab code for *f*, grad\_*f*, and *H*:

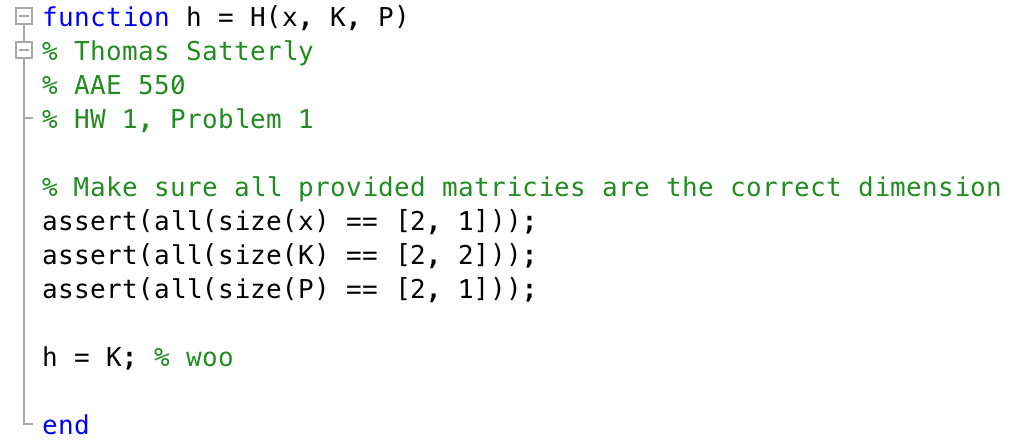
*f*(x):



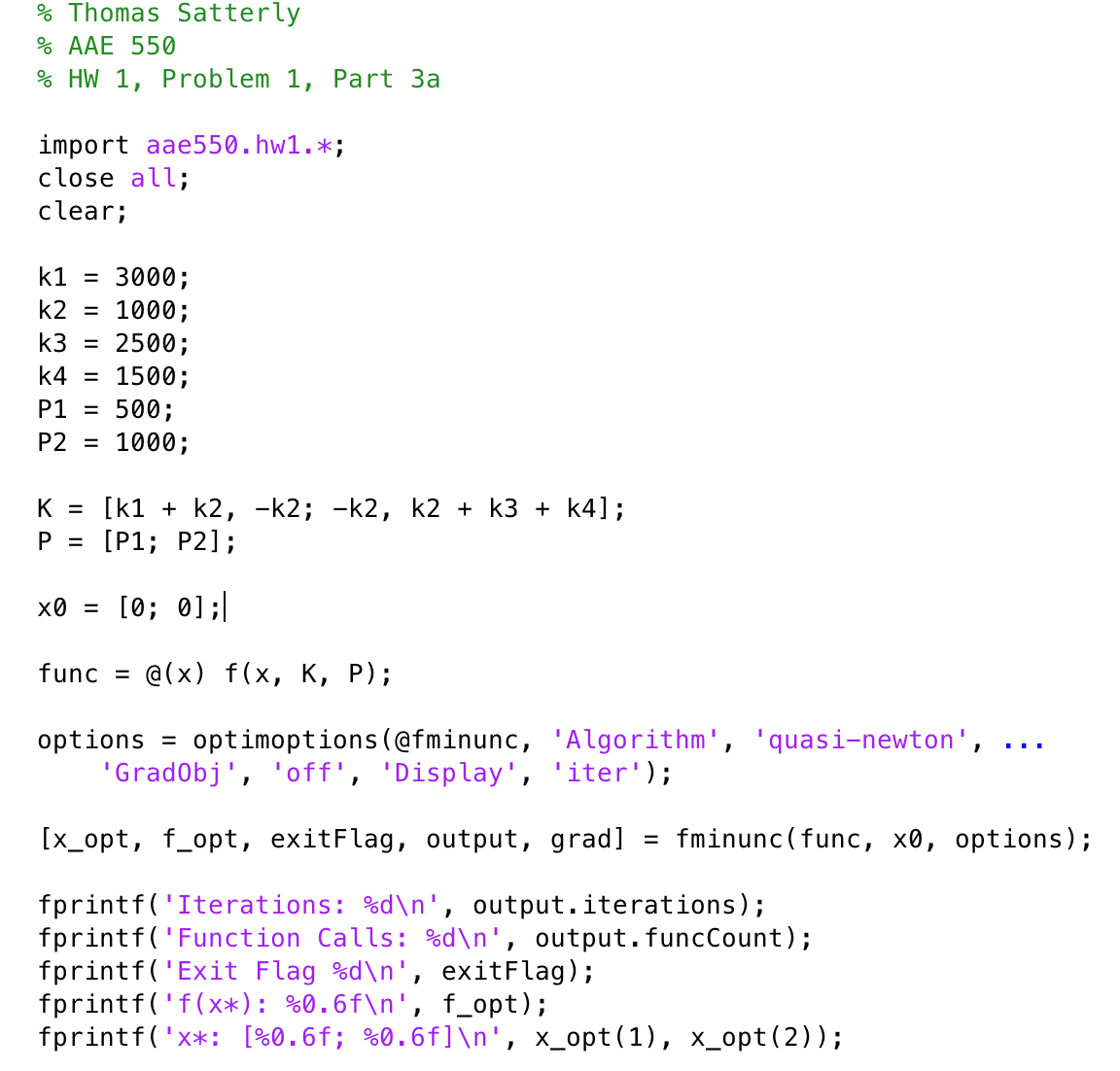
gradient of *f*(x):



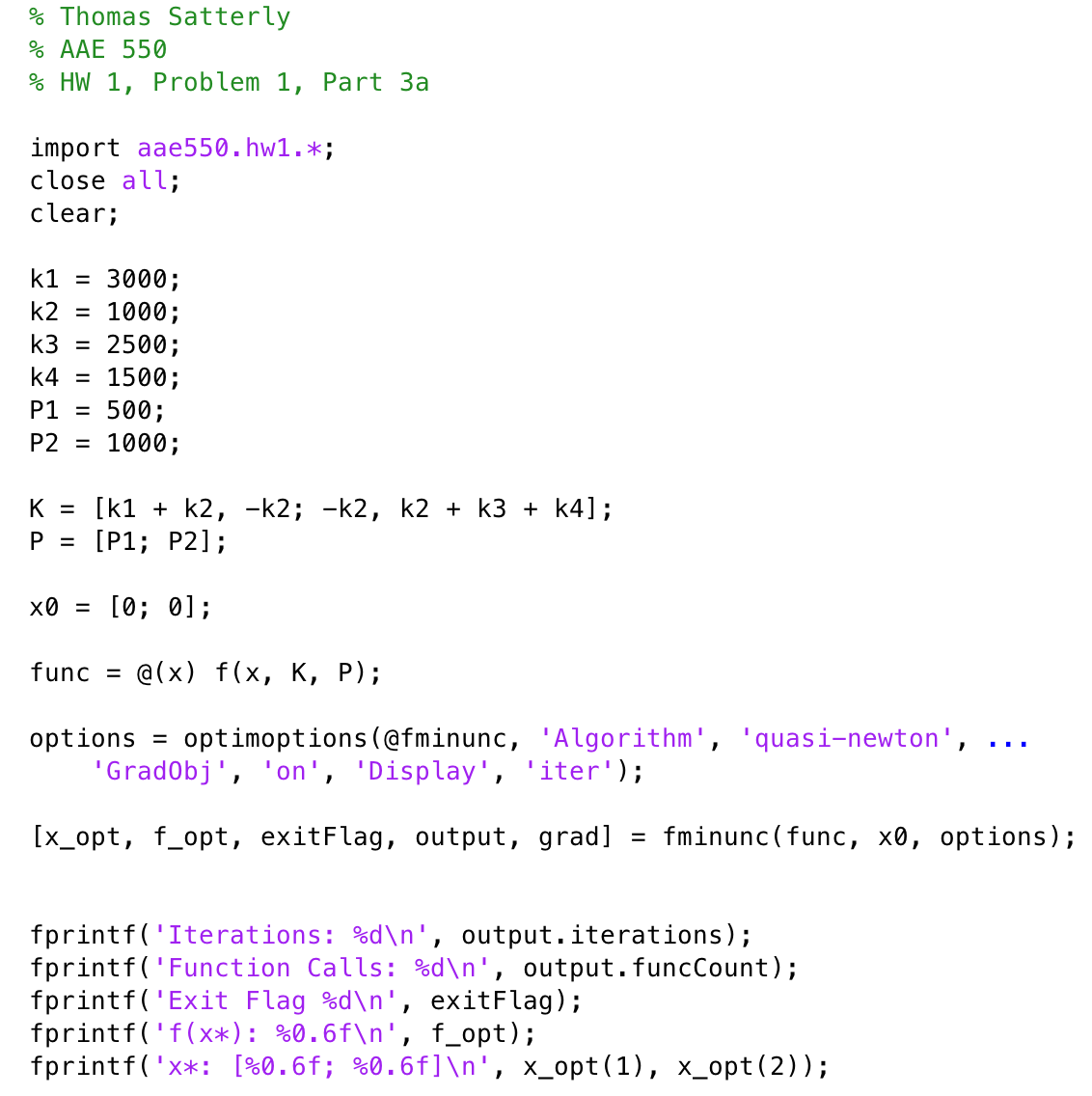
Hessian of *f*(x)



1. Part (a) Matlab Code (Results in Part 7):



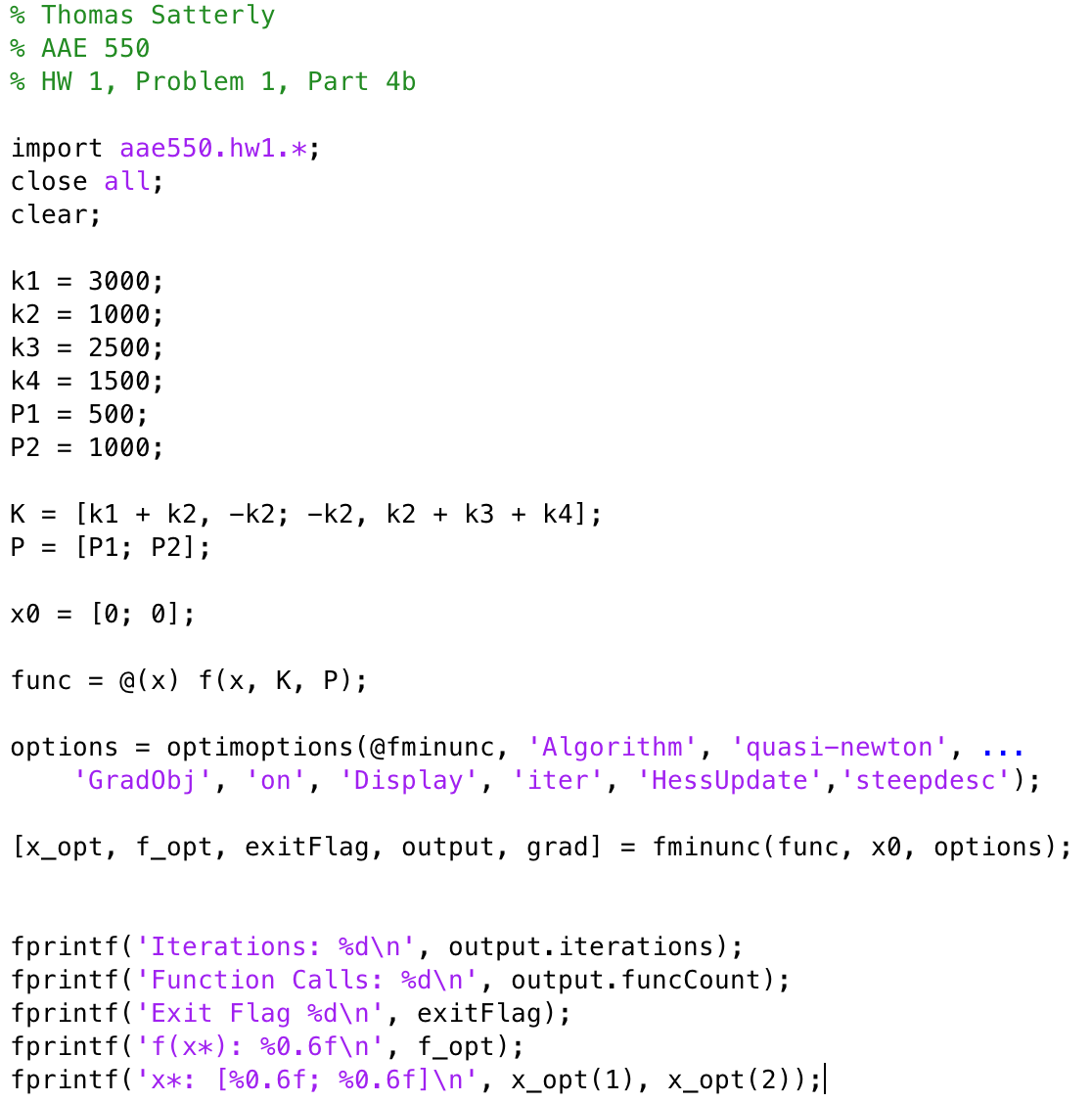
Part (b) Matlab Code (Results in Part 7):



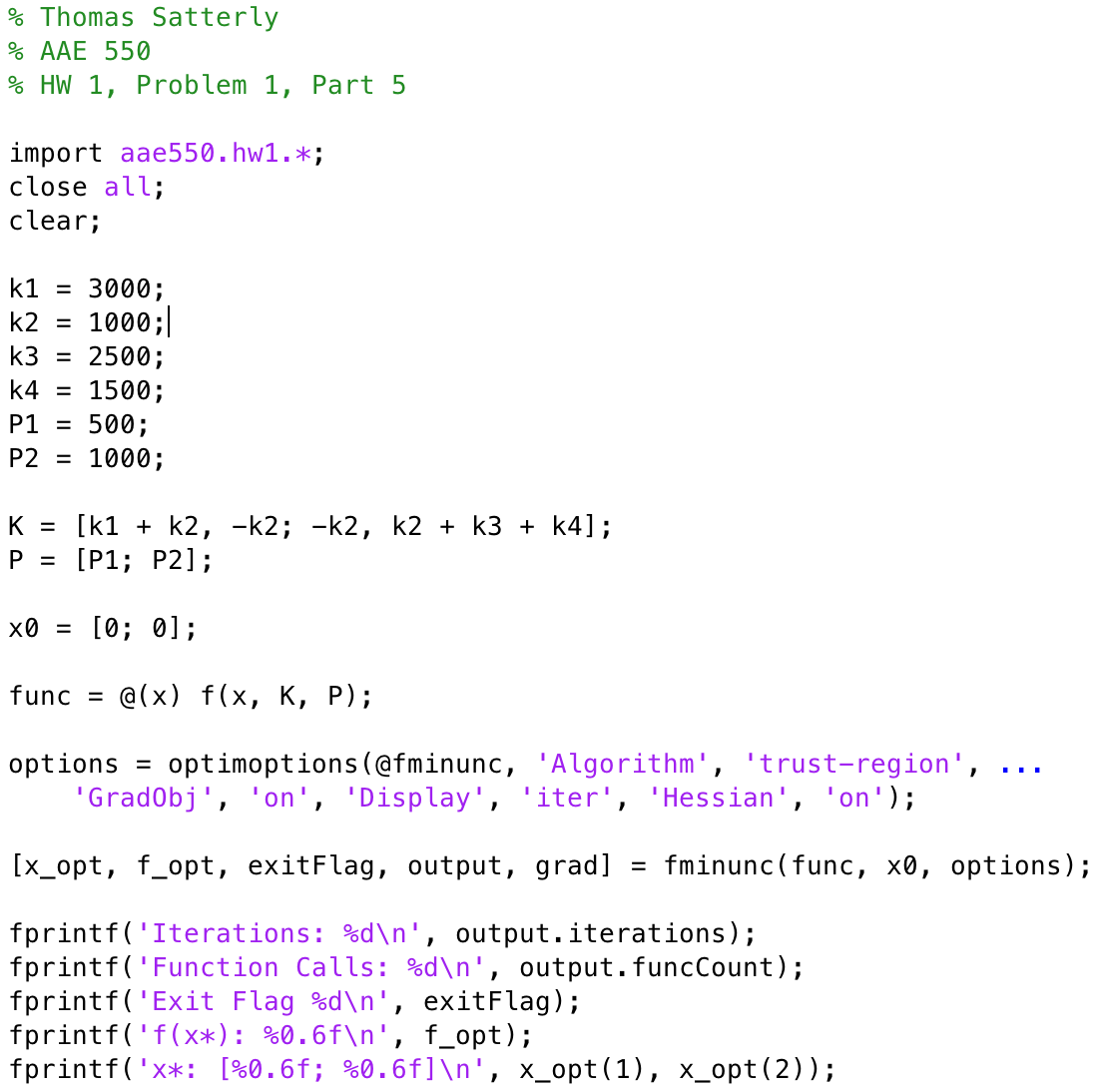
1. Part (a) Matlab Code (Results in Part 7):



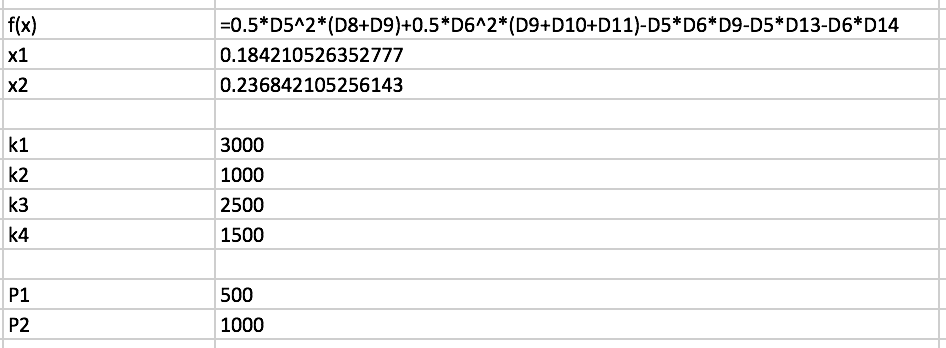
Part (b) Matlab Code (Results in Part 7):



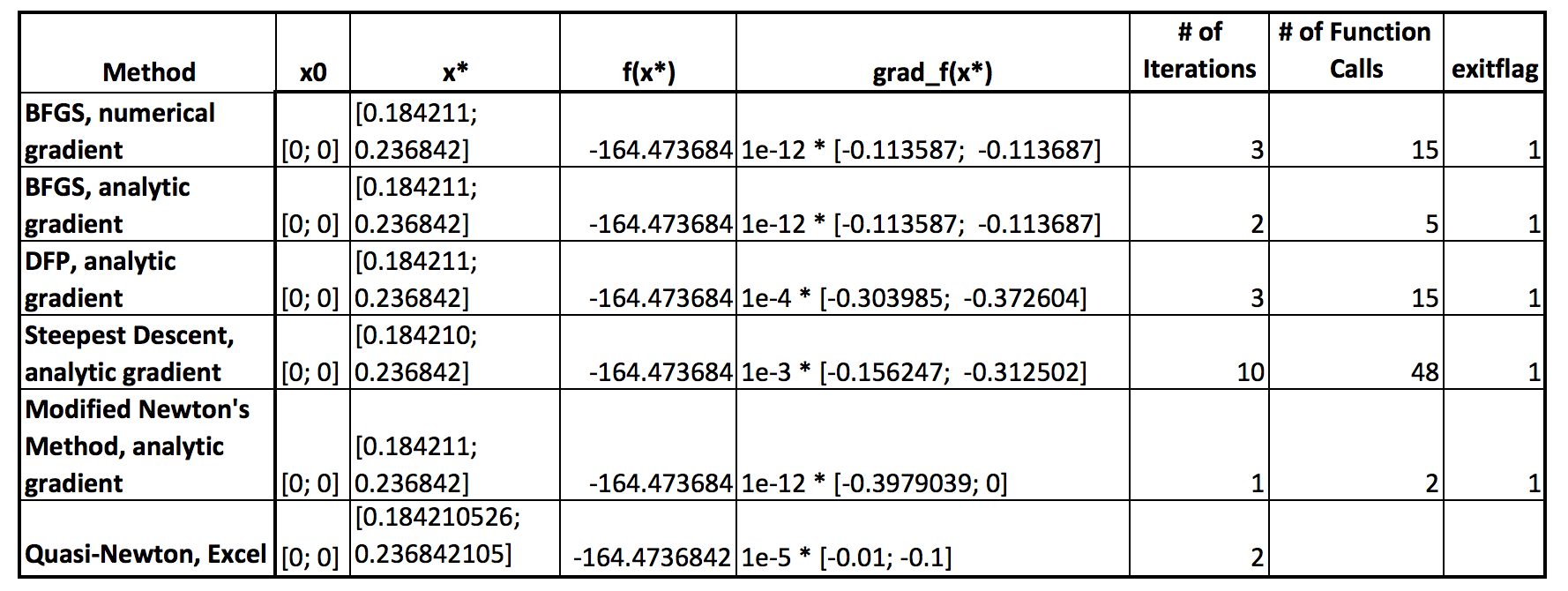
1. Matlab Code (Results in Part 7):



1. Excel Workbook Setup (Results in Part 7)

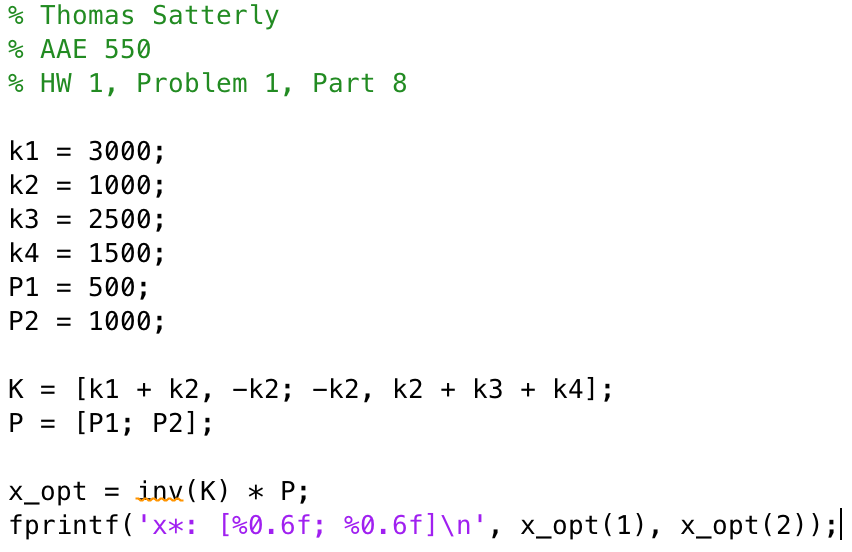


1. Results Table:



The Modified Newton’s Method proved to generate the results with the smallest end gradient, least number of iterations, and least number of function calls, making it the most efficient of the used algorithms. However, the BFGS algorithm with analytic and numerical derivatives were a close second and third, respectively. The Steepest Descent method was unsurprisingly the least efficient, and also had the largest end gradient. This problem in particular is well suited for the Modified Newton’s Method, as it takes full advantage of the analytic gradient and Hessian terms. Based on the results of both BFGS methods, there is not a significant difference in the end result whether analytic or numerical derivatives are used. It is made apparent that numerical derivatives required significantly more function calls, and with costly function calculations, this could be problematic.

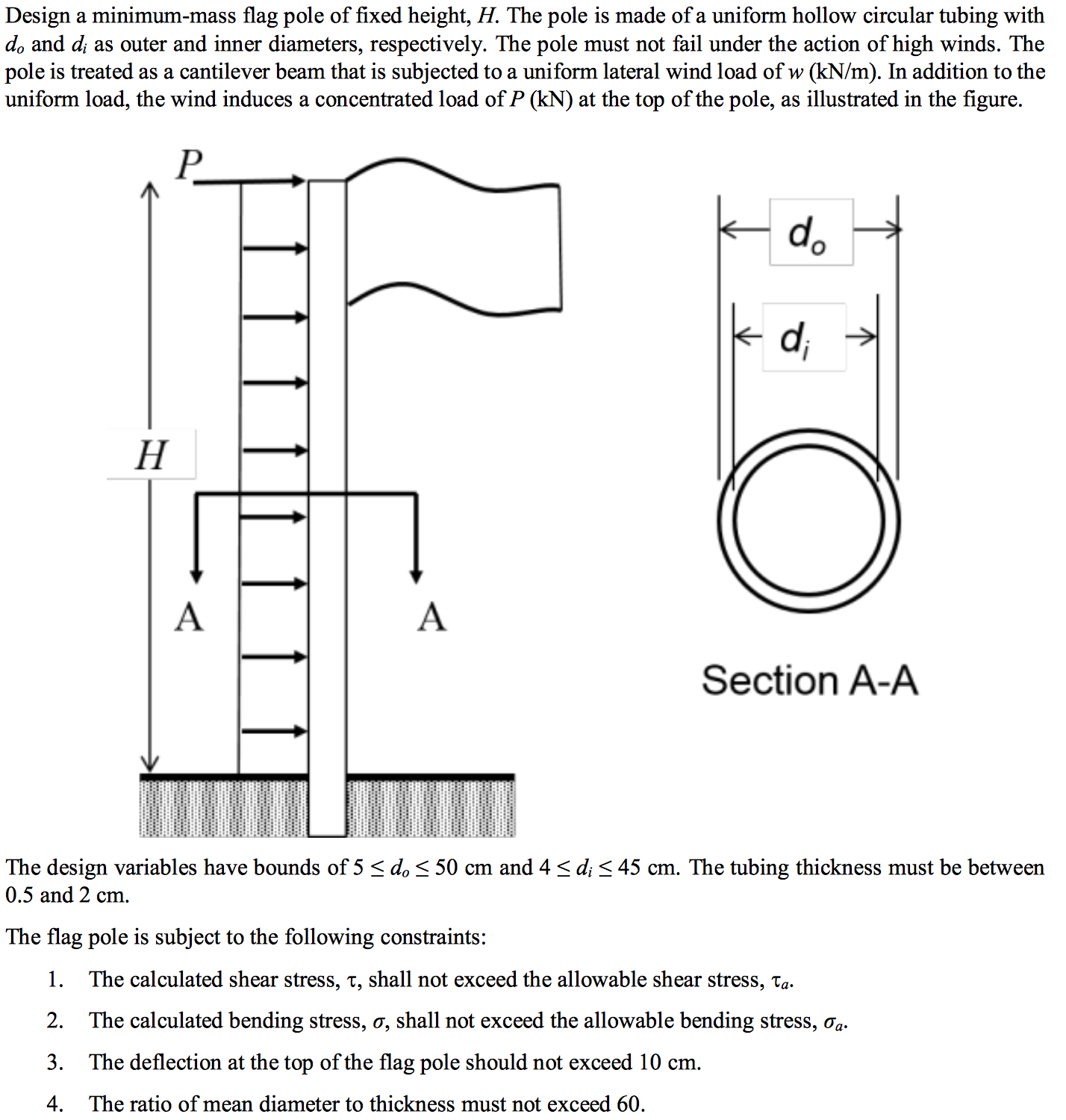
1. Optimality conditions for an unconstrained minimization problem are found when the gradient is zero and the Hessian matrix elements are positive (local minimum). In this case, the Hessian matrix is always positive, meaning that the optimal solution is found where the gradient is zero. Rearranging, the optimal solution can be found by **x = inv(K)\*P**. The Matlab script below solves this equation:



Here, the optimal solution is at x\* = [0.184211; 0.236842], and f(x) = -1.6447368. This answer is not significantly different than the “better” optimization solutions found in part (7).

**Part 2**

Given:



Find:

1. *See attached for definition of Part II: Problem 1*
2. Use the *fminunc* function in Matlab with the following penalty methods to solve the problem using the SUMT approach. Use the default BFGS update and numerical gradients. Record the total number of unconstrained minimizations, total number of iterations, final design solution, x\*, *f*(x\*), and g\_j(x\*), and exit flag for each iteration.

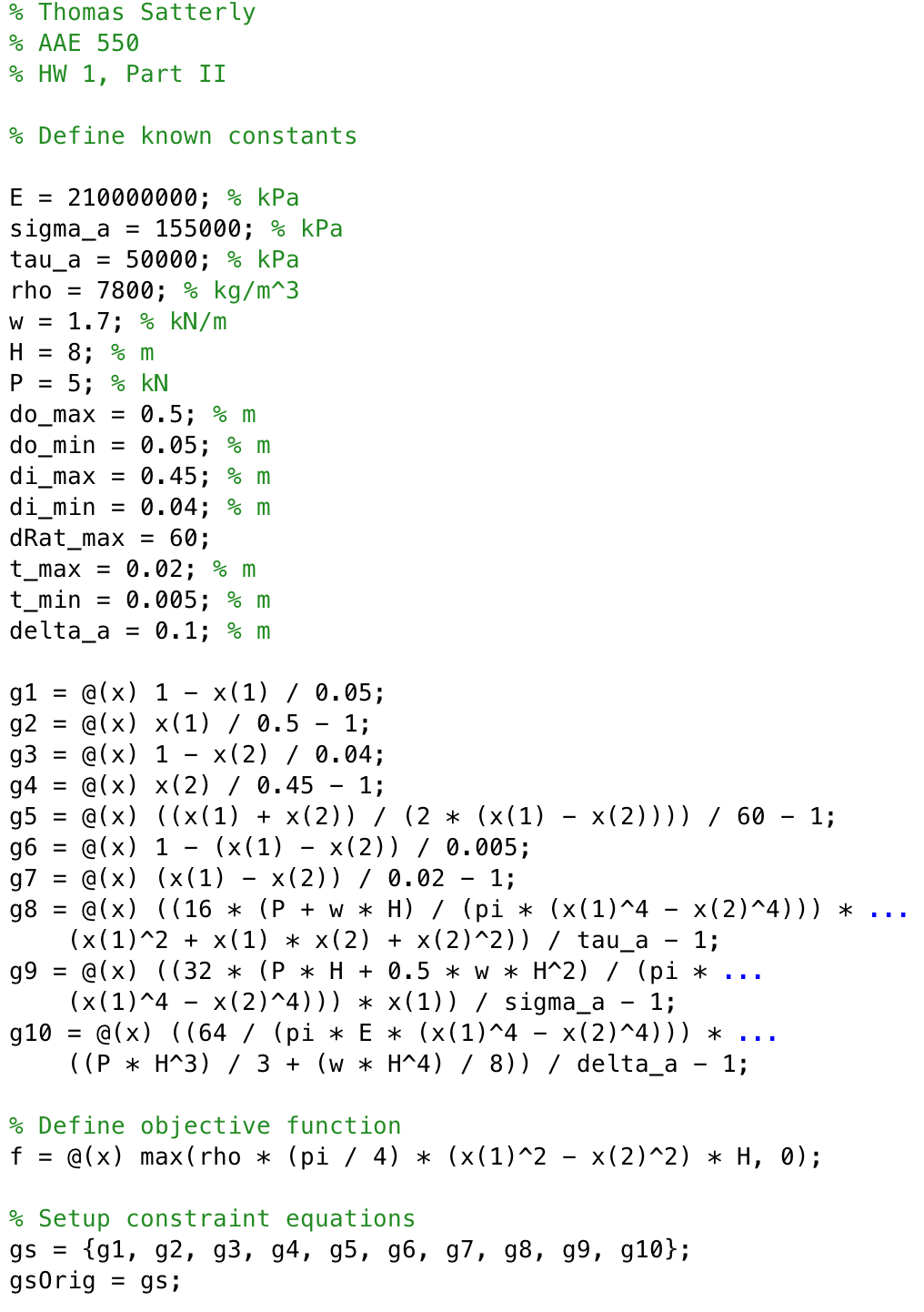
At each solution verify that the constraints, if slightly violated, are acceptable. If an optimization method is not able to solve the problem, explain what was attempted to try to make the method work and a possible reason why the method cannot be used for this particular problem.

* 1. Use the exterior penalty method. Comment on values for *c\_j*, if any
  2. Use the interior penalty method. Comment on values for *c\_j*, if any
  3. Use the extended-linear penalty method. Comment on values for *c\_j*, if any
  4. Use the Augmented Lagrange Multiplier for inequality-constrained method. Comment on values for *c\_j*, if any

1. Compare the total number of unconstrained minimizations and iterations needed for each method. Also, compare the solutions. Which method was the easiest to implement and use? Can any conclusions be made about the different penalty methods for this problem?

Solution:

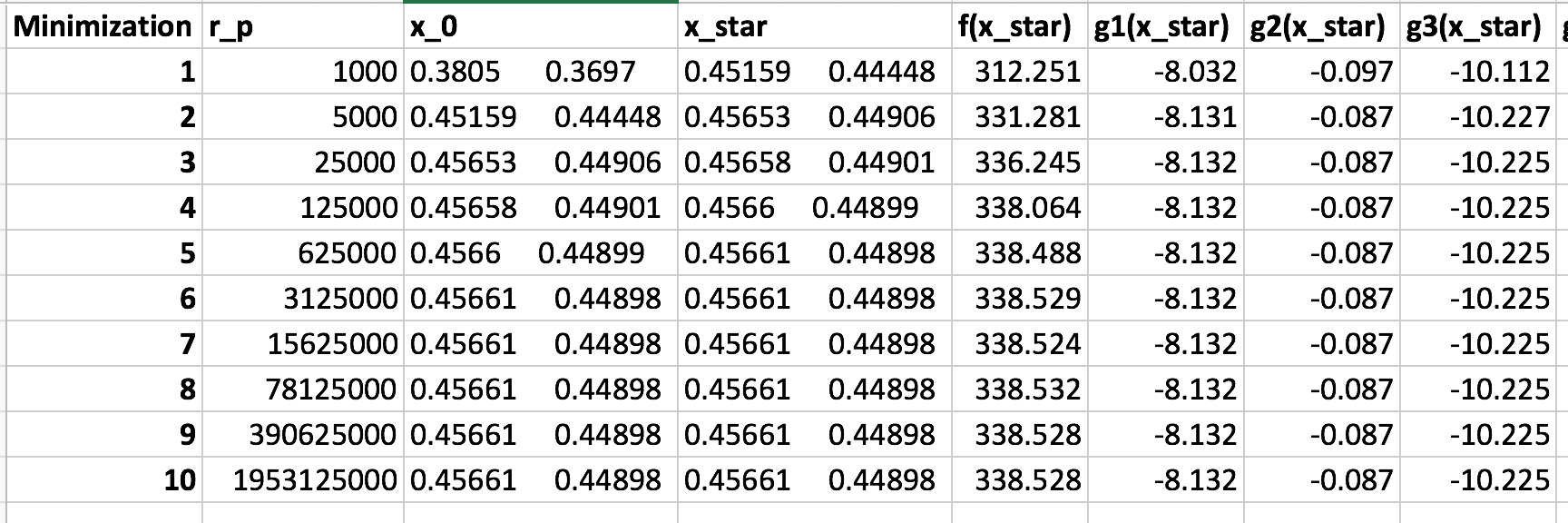
1. *See attached for solution to Part I: Problem 1*
2. The Matlab code common to all problems is as follows:

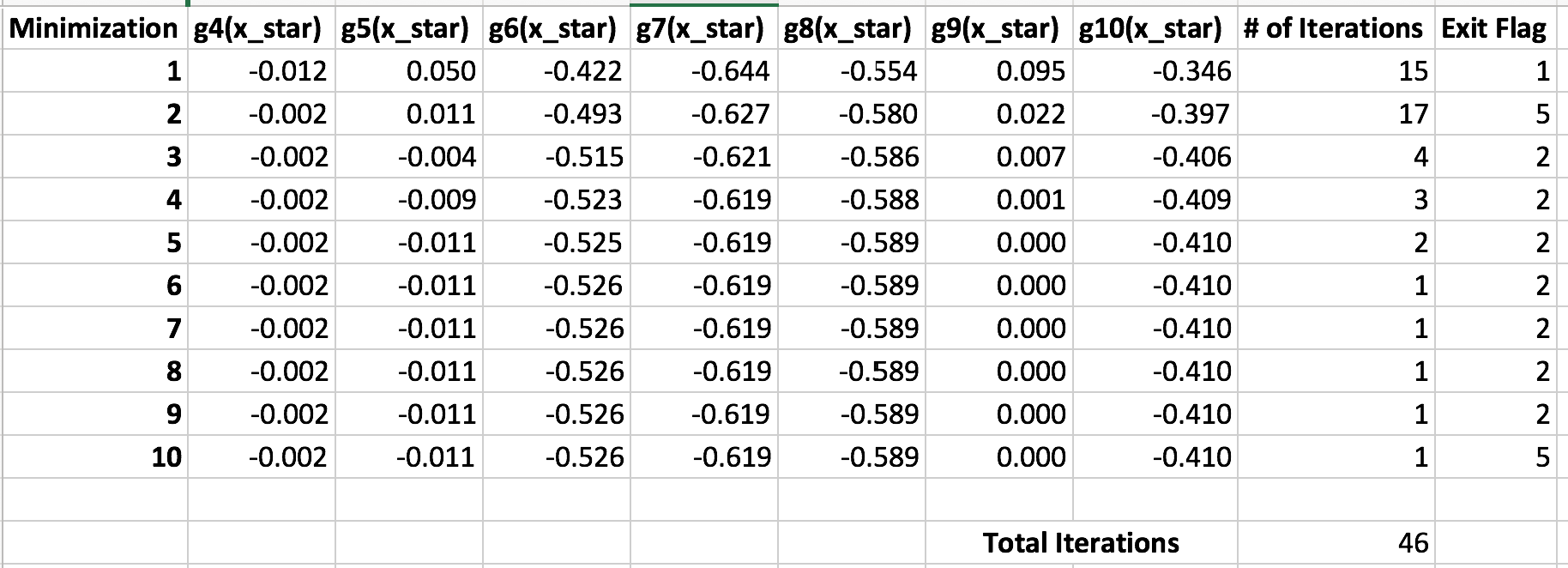
Setup Code: 

Matlab code for the individual problems are provided in attached documents due to their length. The iteration tables for each method are as follows:

* 1. Exterior Penalty Method:

No constraint coefficients (c\_j’s) were used with this method, and no further conditioning had to be performed in order to find a solution. The optimal solution was restricted to violate any constraint function by no more the 1e-4, which the 9th constraint (the only violation in this case) was under. Successive optimal error was limited to 1e-6 for a final solution to be generated.





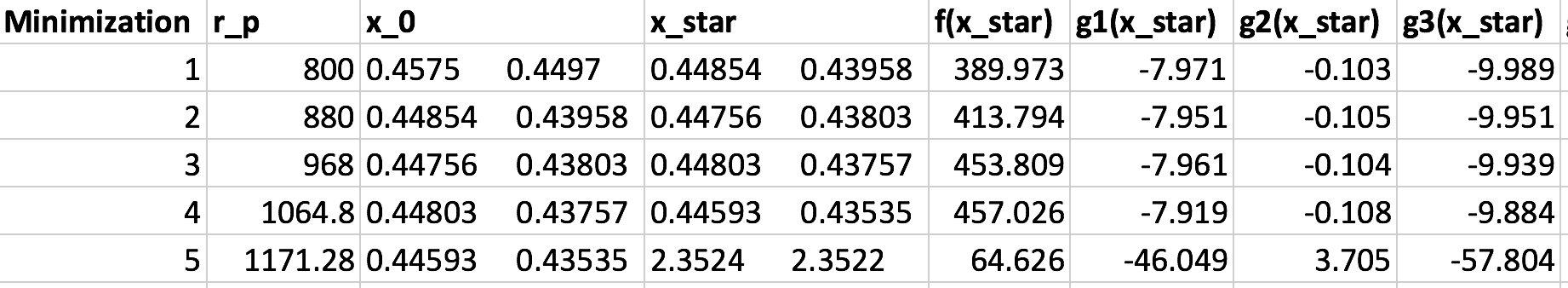
* 1. Interior Penalty Method:

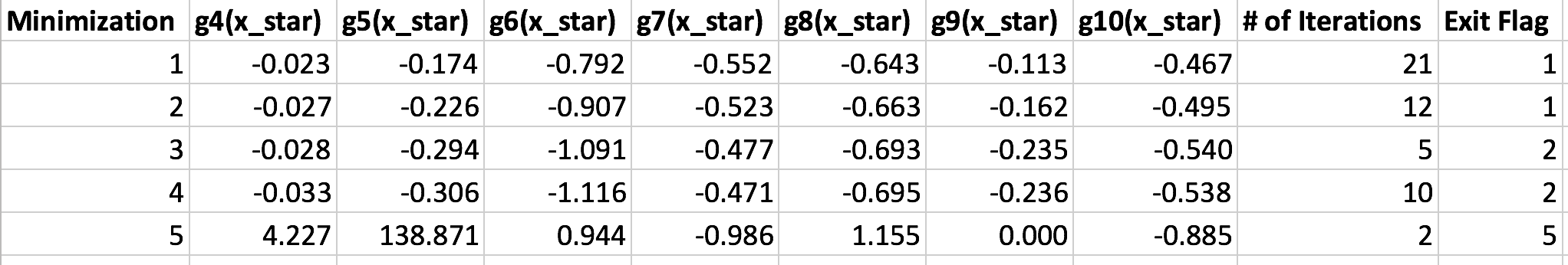
I could not get the classical interior penalty method to work. Attempts I made to condition the problem better were:

1. Swept across valid starting points for x0 in both x(1) and x(2)
2. Updated c\_j values at the start of each minimization
3. Removed design variables from the denominators of constraint functions

It seems this method is not successful at optimizing the problem because it approaches a pinch point of constraint functions near the optimal value that, when combined with non-perfect line search, throws the successive optimization values outside the feasible bounds.

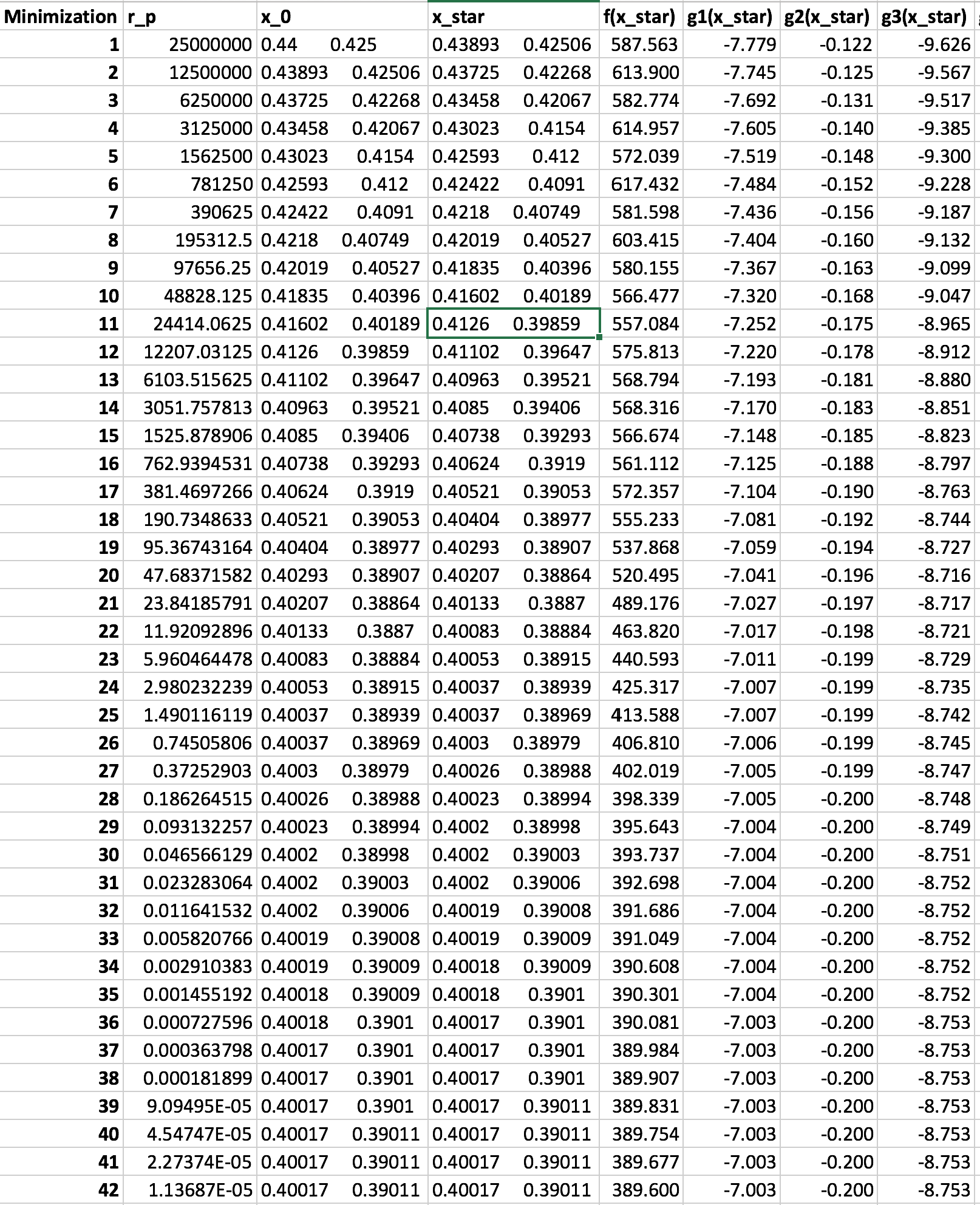
Shown in the tables below, the method is stable for the first few minimizations, but then proceeds to produce invalid solutions, dooming itself for failure:

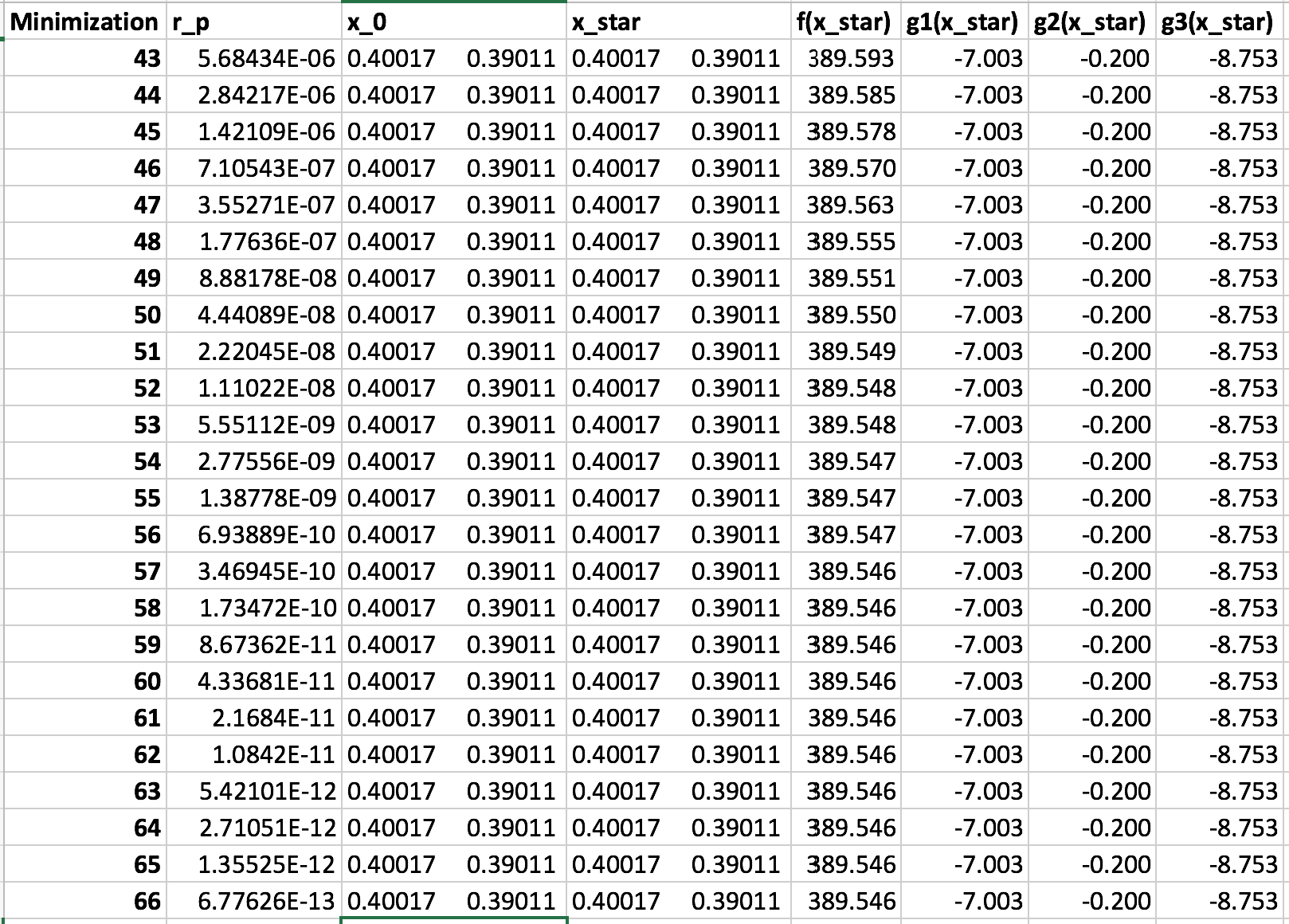


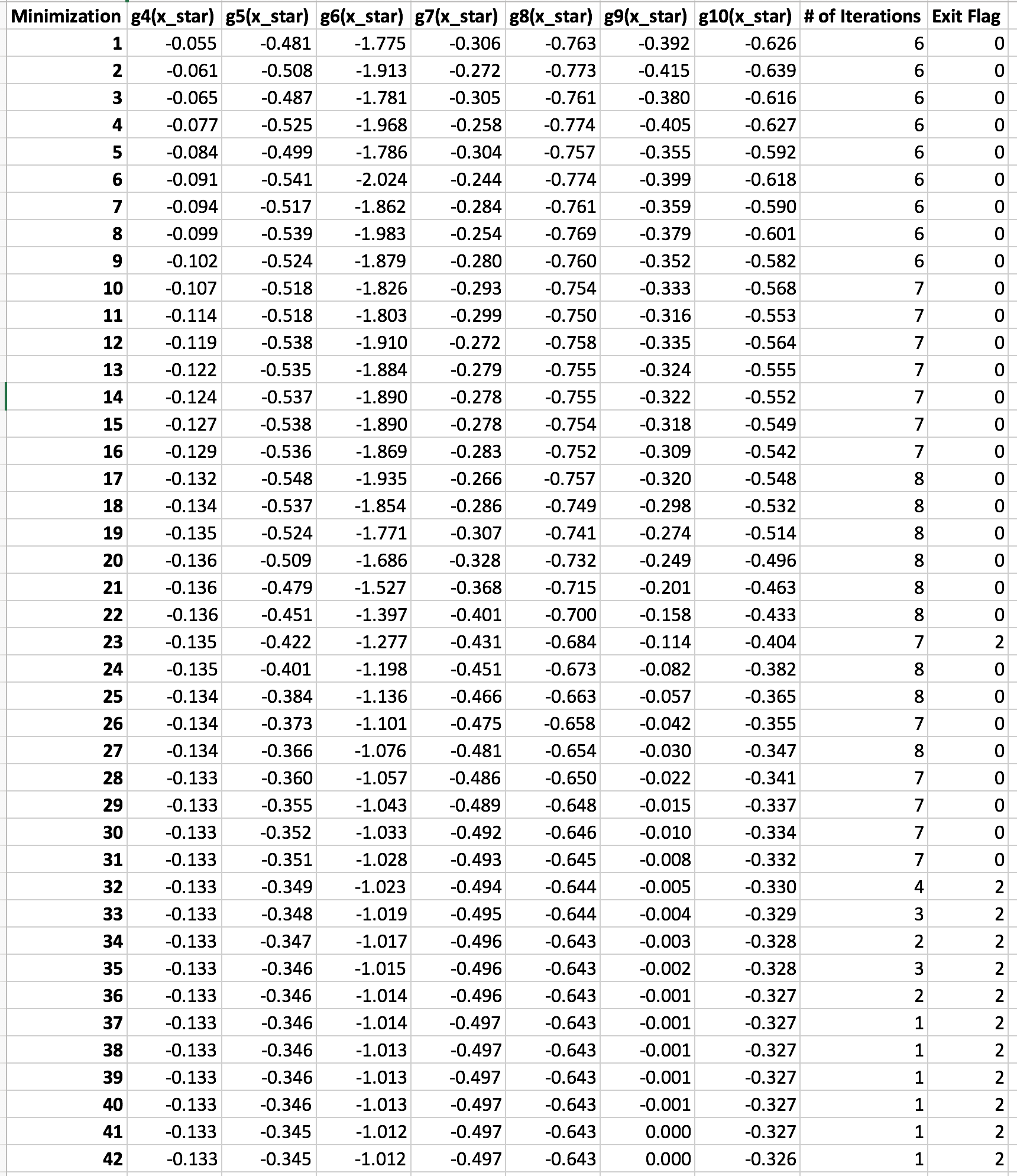


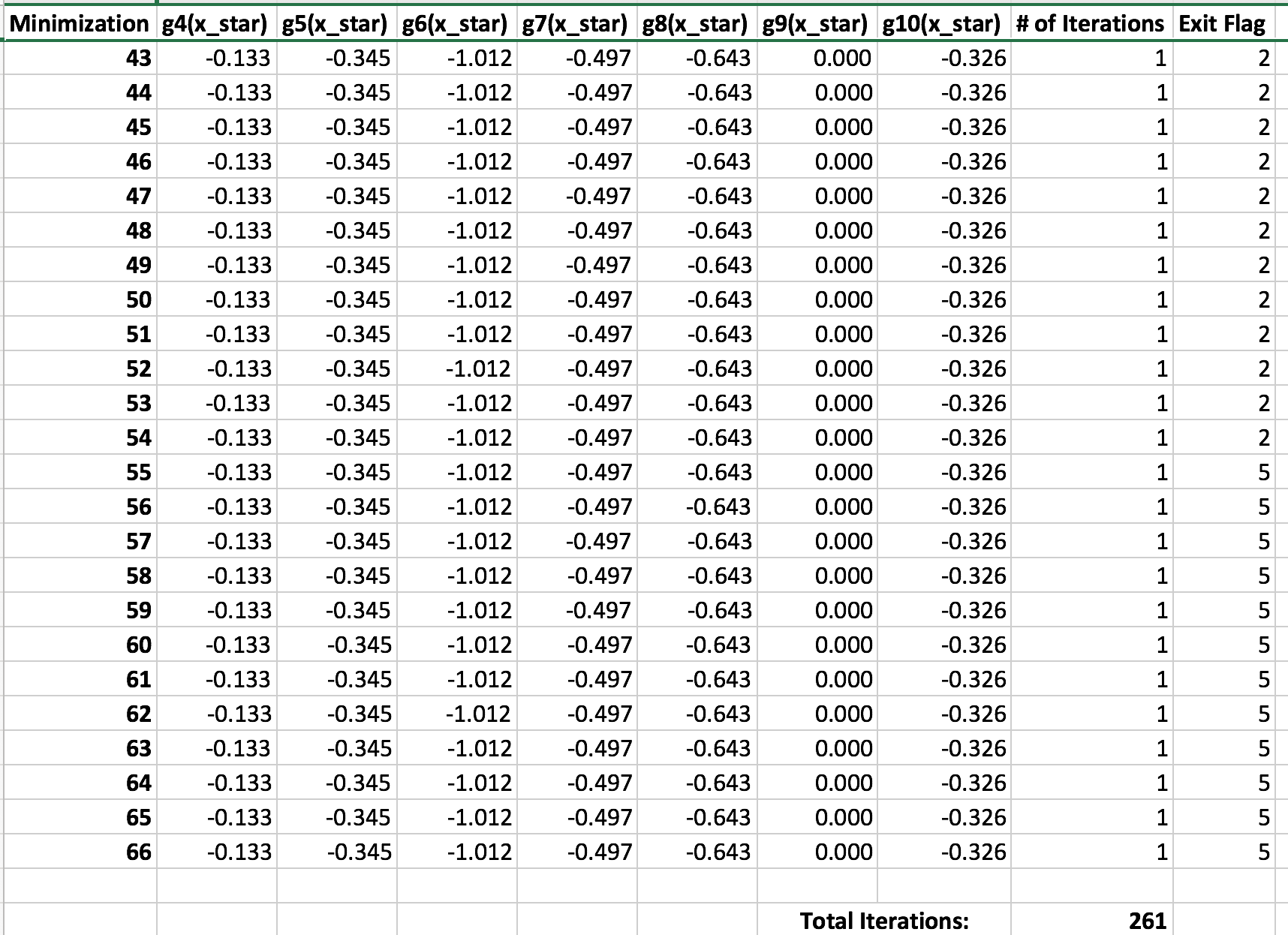
1. Linear Extended Interior Penalty Method

Constraint coefficient values (c\_j’s) were update via the numerical gradient method at the start of each minimization. R\_p values were limited to decrease by a factor of 2 at each iteration. Otherwise, the method proved to be unstable. Successive optimal error was limited to 1e-6 for a final solution to be generated.



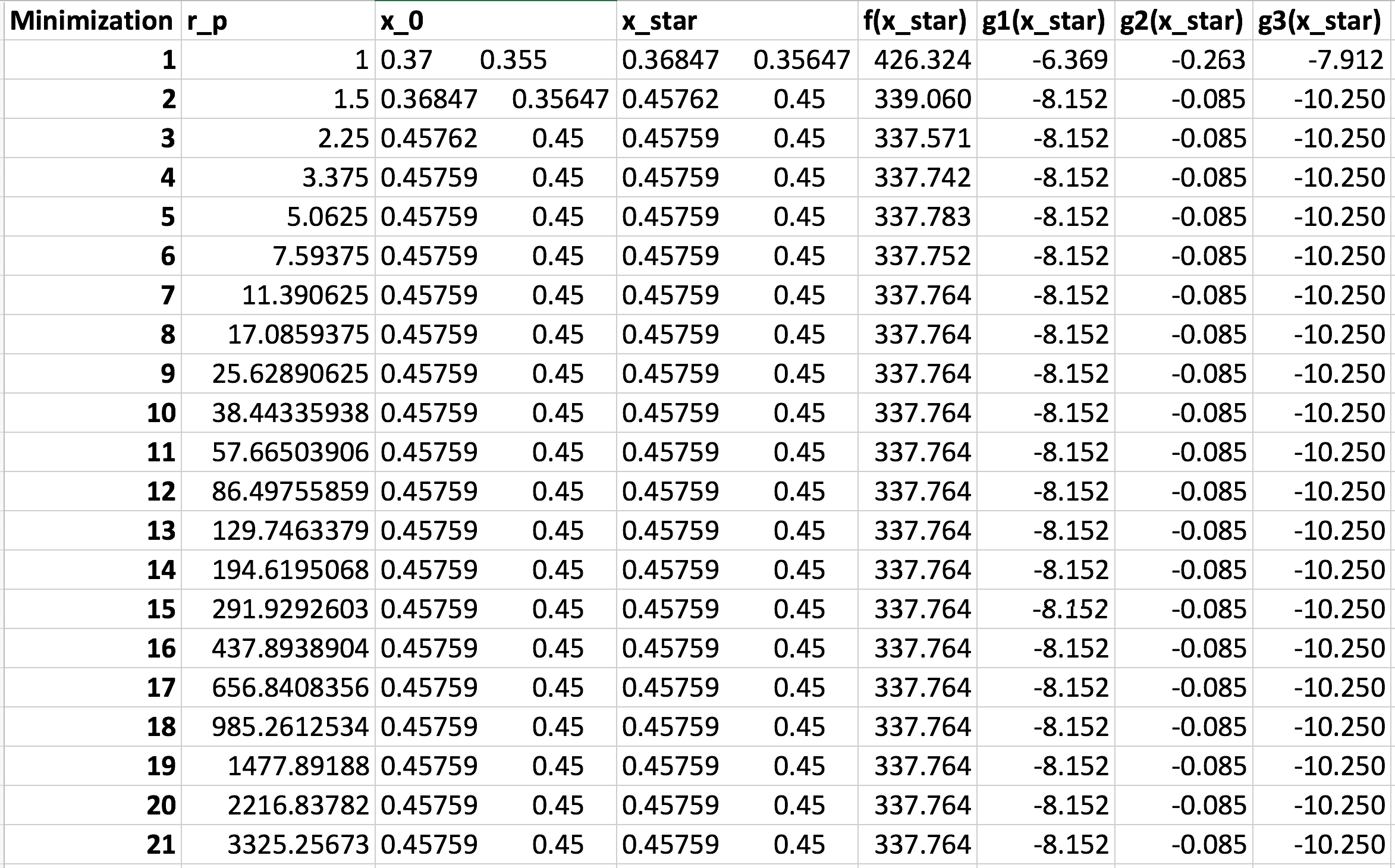


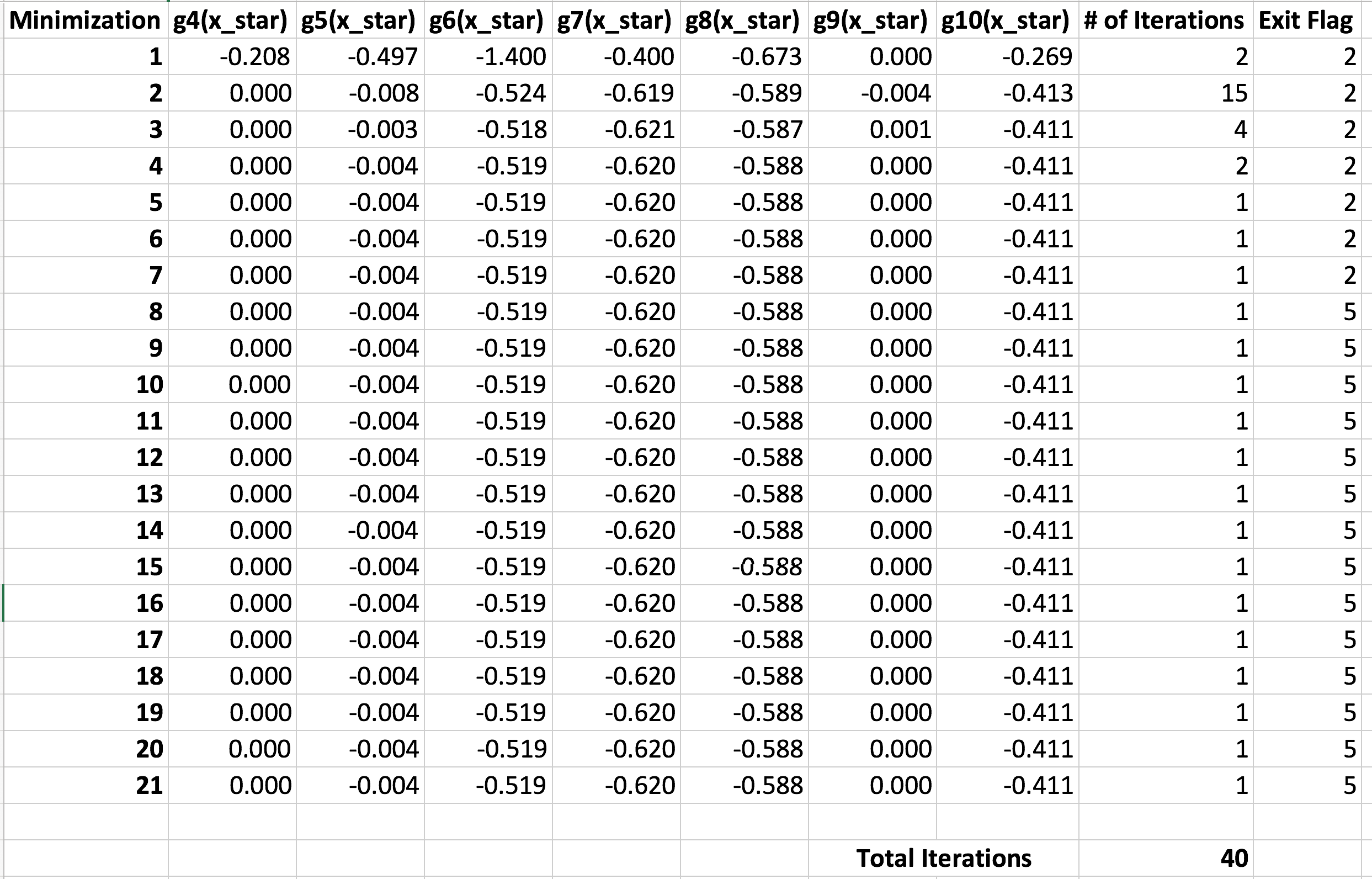




1. Augmented Lagrange Multiplier

Constraint coefficient values (c\_j’s) were update via the numerical gradient method at the start of each minimization. Successive optimal error was limited to 1e-6 for a final solution to be generated.



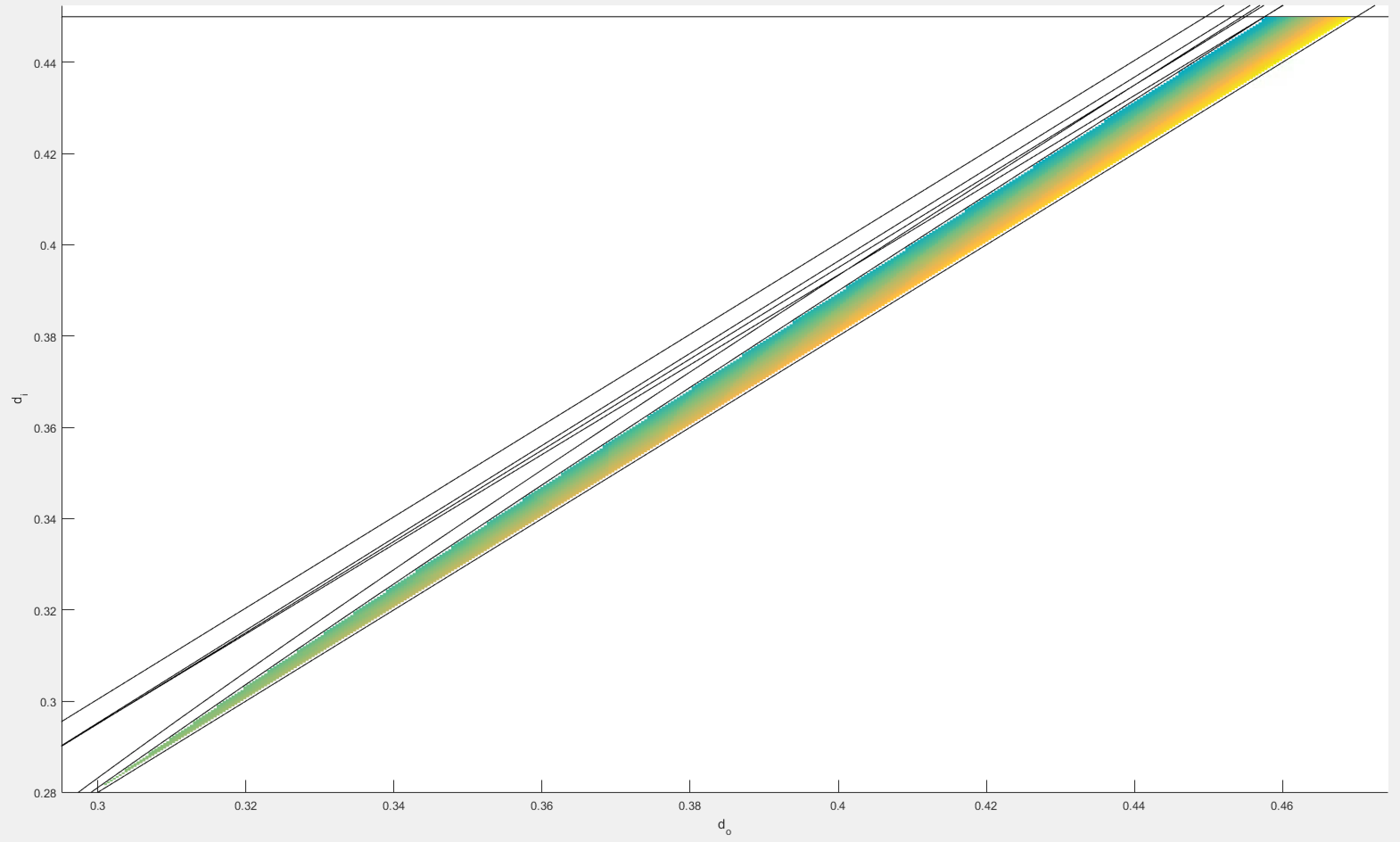


1. Both the exterior penalty method an augmented Lagrange multiplier methods had a relatively low number of total iterations, with 46 and 40 iterations (respectively). In terms of compute power, the augmented Lagrange multiplier method did require constraint coefficients to be updated at the start of each minimization, which does take additional function calls. The interior penalty method did not require such conditioning, which makes the compute time required closer to the augmented Lagrange multiplier method. Additionally, the exterior penalty method only required 5 minimizations, while the augmented Lagrange multiplier took 21 minimizations. The interior penalty method failed to converge entirely (as described in part 2b), which the linearly-extended penalty method did eventually converge after 66 minimizations and 266 total iterations.

Each method also found a different optimal solution. While the solutions found by the exterior penalty method and augmented Lagrange multiplier are very similar (pole weight differing by 0.2%), the extended-linear interior penalty method produced a much different solution that resulted in a pole weight 15% greater than the solutions provided by other methods. Ultimately, the augmented Lagrange multiplier method produced the most optimal solution with x\* = [0.45759, 0.45] and a pole weight of 337.764 kg.

The easiest method to implement was the exterior penalty method, as it required the least conditionals for the penalty method itself, as well as the least conditioning to generate a solution. For a little extra work, the augmented Lagrange method provided better results and little to no change in compute time. Both interior penalty methods took time to condition until they worked (if at all).

Given how the interior penalty methods performed on this problem, it appears that interior penalty methods are inherently poor at solving an optimization problem where multiple constraints appear on the same “side” near the optimal solution. A graphical representation of this problem is produced below:



Optimal solution

Multiple constraint boundaries

With interior penalty methods, the penalty for approaching very close to a constraint boundary rises sharply, and unlike the exterior or augmented Lagrange methods, the penalties always exist for all constraints. When multiple constraints appear near each other at the optimal point, they may have an additive effect on each other and prevent the solution from being found quickly or at all.