Thomas Satterly
AAE 550
HW 1

<u>Part I</u>

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
    a. 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".
    b. 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.
    a. 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".
    b. 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):

```
function [y, dy, ddy] = f(x, K, P)
% Thomas Satterly
% AAE 550
% HW 1, Problem 1

% Make sure all provided matricies are the correct dimension
assert(all(size(x) == [2, 1]));
assert(all(size(K) == [2, 2]));
assert(all(size(P) == [2, 1]));

y = 0.5 * x' * K * x - x' * P;

% Optional: Return derivatives if requested
if nargout >= 2
    dy = aae550.hw1.gradF(x, K, P);
end
if nargout >= 3
    ddy = aae550.hw1.H(x, K, P);
end
end
```

gradient of *f*(x):

```
function dF = gradF(x, K, P)
% Thomas Satterly
% AAE 550
% HW 1, Problem 1

% Make sure all provided matricies are the correct dimension
assert(all(size(x) == [2, 1]));
assert(all(size(K) == [2, 2]));
assert(all(size(P) == [2, 1]));

x1 = x(1, 1);
x2 = x(2, 1);

dF(1, 1) = x(1, 1) * K(1, 1) + x2 * K(1, 2) - P(1, 1);
dF(2, 1) = x(2, 1) * K(2, 2) + x1 * K(1, 2) - P(2, 1);

dFF = K *x - P;

end
```

Hessian of *f*(x)

```
function h = H(x, K, P)
% Thomas Satterly
% AAE 550
% HW 1, Problem 1

% Make sure all provided matricies are the correct dimension
assert(all(size(x) == [2, 1]));
assert(all(size(K) == [2, 2]));
assert(all(size(P) == [2, 1]));

h = K; % woo

end
```

3) Part (a) Matlab Code (Results in Part 7):

```
% Thomas Satterly
% AAE 550
% HW 1, Problem 1, Part 3a

import aae550.hw1.*;
close all;
clear;

k1 = 3000;
k2 = 1000;
k3 = 2500;
k4 = 1500;
P1 = 500;
P2 = 1000;

K = [k1 + k2, -k2; -k2, k2 + k3 + k4];
P = [P1; P2];

x0 = [0; 0];

func = @(x) f(x, K, P);

options = optimoptions(@fminunc, 'Algorithm', 'quasi-newton', ...
    'GradObj', 'off', 'Display', 'iter');

[x_opt, f_opt, exitFlag, output, grad] = fminunc(func, x0, options);

fprintf('Iterations: %d\n', output.iterations);
fprintf('Function Calls: %d\n', output.funcCount);
fprintf('Exit Flag %d\n', exitFlag);
fprintf('f(x*): %0.6f\n', f_opt);
fprintf('x*: [%0.6f; %0.6f]\n', x_opt(1), x_opt(2));
```

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

```
% Thomas Satterly
% AAE 550
% HW 1, Problem 1, Part 3a

import aae550.hw1.*;
close all;
clear;

k1 = 3000;
k2 = 1000;
k3 = 2500;
k4 = 1500;
P1 = 500;
P2 = 1000;

K = [k1 + k2, -k2; -k2, k2 + k3 + k4];
P = [P1; P2];

x0 = [0; 0];

func = @(x) f(x, K, P);

options = optimoptions(@fminunc, 'Algorithm', 'quasi-newton', ...
    'GradObj', 'on', 'Display', 'iter');

[x_opt, f_opt, exitFlag, output, grad] = fminunc(func, x0, options);


fprintf('Iterations: %d\n', output.iterations);
fprintf('Function Calls: %d\n', output.funcCount);
fprintf('Exit Flag %d\n', exitFlag);
fprintf('f(x*): %0.6f\n', f_opt);
fprintf('x*: [%0.6f; %0.6f]\n', x_opt(1), x_opt(2));
```

4) Part (a) Matlab Code (Results in Part 7):

```
% Thomas Satterly
% AAE 550
% HW 1, Problem 1, Part 4a

import aae550.hw1.*;
close all;
clear;

k1 = 3000;
k2 = 1000;
k3 = 2500;
k4 = 1500;
P1 = 500;
P2 = 1000;

K = [k1 + k2, -k2; -k2, k2 + k3 + k4];
P = [P1; P2];

x0 = [0; 0];

func = @(x) f(x, K, P);

options = optimoptions(@fminunc, 'Algorithm', 'quasi-newton', ...
    'GradObj', 'off', 'Display', 'iter');

[x_opt, f_opt, exitFlag, output, grad] = fminunc(func, x0, options);


fprintf('Iterations: %d\n', output.iterations);
fprintf('Function Calls: %d\n', output.funcCount);
fprintf('Exit Flag %d\n', exitFlag);
fprintf('f(x*): %0.6f\n', f_opt);
fprintf('x*: [%0.6f; %0.6f]\n', x_opt(1), x_opt(2));
```

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

```matlab
% Thomas Satterly
% AAE 550
% HW 1, Problem 1, Part 4b

import aae550.hw1.*;
close all;
clear;

k1 = 3000;
k2 = 1000;
k3 = 2500;
k4 = 1500;
P1 = 500;
P2 = 1000;

K = [k1 + k2, -k2; -k2, k2 + k3 + k4];
P = [P1; P2];

x0 = [0; 0];

func = @(x) f(x, K, P);

options = optimoptions(@fminunc, 'Algorithm', 'quasi-newton', ...
    'GradObj', 'on', 'Display', 'iter', 'HessUpdate','steepdesc');

[x_opt, f_opt, exitFlag, output, grad] = fminunc(func, x0, options);


fprintf('Iterations: %d\n', output.iterations);
fprintf('Function Calls: %d\n', output.funcCount);
fprintf('Exit Flag %d\n', exitFlag);
fprintf('f(x*): %0.6f\n', f_opt);
fprintf('x*: [%0.6f; %0.6f]\n', x_opt(1), x_opt(2));
```

5) Matlab Code (Results in Part 7):

```
% Thomas Satterly
% AAE 550
% HW 1, Problem 1, Part 5

import aae550.hw1.*;
close all;
clear;

k1 = 3000;
k2 = 1000;
k3 = 2500;
k4 = 1500;
P1 = 500;
P2 = 1000;

K = [k1 + k2, -k2; -k2, k2 + k3 + k4];
P = [P1; P2];

x0 = [0; 0];

func = @(x) f(x, K, P);

options = optimoptions(@fminunc, 'Algorithm', 'trust-region', ...
    'GradObj', 'on', 'Display', 'iter', 'Hessian', 'on');

[x_opt, f_opt, exitFlag, output, grad] = fminunc(func, x0, options);

fprintf('Iterations: %d\n', output.iterations);
fprintf('Function Calls: %d\n', output.funcCount);
fprintf('Exit Flag %d\n', exitFlag);
fprintf('f(x*): %0.6f\n', f_opt);
fprintf('x*: [%0.6f; %0.6f]\n', x_opt(1), x_opt(2));
```

6) Excel Workbook Setup (Results in Part 7)

f(x)	=0.5*D5^2*(D8+D9)+0.5*D6^2*(D9+D10+D11)-D5*D6*D9-D5*D13-D6*D14
x1	0.184210526352777
x2	0.236842105256143
k1	3000
k2	1000
k3	2500
k4	1500
P1	500
P2	1000

7) Results Table:

Method	x0	x*	f(x*)	grad_f(x*)	# of Iterations	# of Function Calls	exitflag
BFGS, numerical gradient	[0; 0]	[0.184211; 0.236842]	-164.473684	1e-12 * [-0.113587; -0.113687]	3	15	1
BFGS, analytic gradient	[0; 0]	[0.184211; 0.236842]	-164.473684	1e-12 * [-0.113587; -0.113687]	2	5	1
DFP, analytic gradient	[0; 0]	[0.184211; 0.236842]	-164.473684	1e-4 * [-0.303985; -0.372604]	3	15	1
Steepest Descent, analytic gradient	[0; 0]	[0.184210; 0.236842]	-164.473684	1e-3 * [-0.156247; -0.312502]	10	48	1
Modified Newton's Method, analytic gradient	[0; 0]	[0.184211; 0.236842]	-164.473684	1e-12 * [-0.3979039; 0]	1	2	1
Quasi-Newton, Excel	[0; 0]	[0.184210526; 0.236842105]	-164.4736842	1e-5 * [-0.01; -0.1]	2		

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.

8) 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:

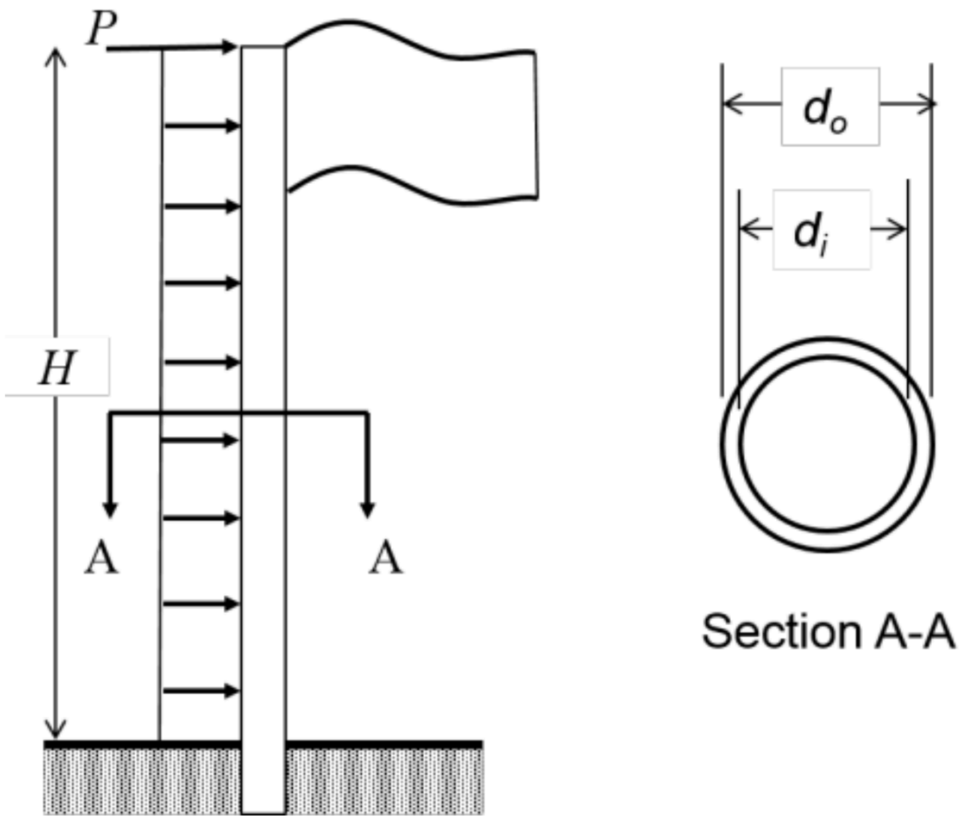
```
% Thomas Satterly
% AAE 550
% HW 1, Problem 1, Part 8

k1 = 3000;
k2 = 1000;
k3 = 2500;
k4 = 1500;
P1 = 500;
P2 = 1000;

K = [k1 + k2, -k2; -k2, k2 + k3 + k4];
P = [P1; P2];

x_opt = inv(K) * P;
fprintf('x*: [%0.6f; %0.6f]\n', x_opt(1), x_opt(2));
```

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:

Design a minimum-mass flag pole of fixed height, H. The pole is made of a uniform hollow circular tubing with d_o and d_i as outer and inner diameters, respectively. The pole must not fail under the action of high winds. The pole is treated as a cantilever beam that is subjected to a uniform lateral wind load of w (kN/m). In addition to the uniform load, the wind induces a concentrated load of P (kN) at the top of the pole, as illustrated in the figure.



The design variables have bounds of $5 \le d_o \le 50$ cm and $4 \le d_i \le 45$ cm. The tubing thickness must be between 0.5 and 2 cm.

The flag pole is subject to the following constraints:

1. The calculated shear stress, τ, shall not exceed the allowable shear stress, τ_a.
2. The calculated bending stress, σ, shall not exceed the allowable bending stress, σ_a.
3. The deflection at the top of the flag pole should not exceed 10 cm.
4. The ratio of mean diameter to thickness must not exceed 60.

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.

a. Use the exterior penalty method. Comment on values for *c_j*, if any
b. Use the interior penalty method. Comment on values for *c_j*, if any
c. Use the extended-linear penalty method. Comment on values for *c_j*, if any
d. Use the Augmented Lagrange Multiplier for inequality-constrained method. Comment on values for *c_j*, if any

3) 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:

```
% Thomas Satterly
% AAE 550
% HW 1, Part II

% Define known constants

E = 210000000; % kPa
sigma_a = 155000; % kPa
tau_a = 50000; % kPa
rho = 7800; % kg/m^3
w = 1.7; % kN/m
H = 8; % m
P = 5; % kN
do_max = 0.5; % m
do_min = 0.05; % m
di_max = 0.45; % m
di_min = 0.04; % m
dRat_max = 60;
t_max = 0.02; % m
t_min = 0.005; % m
delta_a = 0.1; % m

g1 = @(x) 1 - x(1) / 0.05;
g2 = @(x) x(1) / 0.5 - 1;
g3 = @(x) 1 - x(2) / 0.04;
g4 = @(x) x(2) / 0.45 - 1;
g5 = @(x) ((x(1) + x(2)) / (2 * (x(1) - x(2)))) / 60 - 1;
g6 = @(x) 1 - (x(1) - x(2)) / 0.005;
g7 = @(x) (x(1) - x(2)) / 0.02 - 1;
g8 = @(x) ((16 * (P + w * H) / (pi * (x(1)^4 - x(2)^4))) * ...
    (x(1)^2 + x(1) * x(2) + x(2)^2)) / tau_a - 1;
g9 = @(x) ((32 * (P * H + 0.5 * w * H^2) / (pi * ...
    (x(1)^4 - x(2)^4))) * x(1)) / sigma_a - 1;
g10 = @(x) ((64 / (pi * E * (x(1)^4 - x(2)^4))) * ...
    ((P * H^3) / 3 + (w * H^4) / 8)) / delta_a - 1;

% Define objective function
f = @(x) max(rho * (pi / 4) * (x(1)^2 - x(2)^2) * H, 0);

% Setup constraint equations
gs = {g1, g2, g3, g4, g5, g6, g7, g8, g9, g10};
gsOrig = gs;
```

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

a. 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.

Minimization	r_p	x_0		x_star		f(x_star)	g1(x_star)	g2(x_star)	g3(x_star)
1	1000	0.3805	0.3697	0.45159	0.44448	312.251	-8.032	-0.097	-10.112
2	5000	0.45159	0.44448	0.45653	0.44906	331.281	-8.131	-0.087	-10.227
3	25000	0.45653	0.44906	0.45658	0.44901	336.245	-8.132	-0.087	-10.225
4	125000	0.45658	0.44901	0.4566	0.44899	338.064	-8.132	-0.087	-10.225
5	625000	0.4566	0.44899	0.45661	0.44898	338.488	-8.132	-0.087	-10.225
6	3125000	0.45661	0.44898	0.45661	0.44898	338.529	-8.132	-0.087	-10.225
7	15625000	0.45661	0.44898	0.45661	0.44898	338.524	-8.132	-0.087	-10.225
8	78125000	0.45661	0.44898	0.45661	0.44898	338.532	-8.132	-0.087	-10.225
9	390625000	0.45661	0.44898	0.45661	0.44898	338.528	-8.132	-0.087	-10.225
10	1953125000	0.45661	0.44898	0.45661	0.44898	338.528	-8.132	-0.087	-10.225

Minimization	g4(x_star)	g5(x_star)	g6(x_star)	g7(x_star)	g8(x_star)	g9(x_star)	g10(x_star)	# of Iterations	Exit Flag
1	-0.012	0.050	-0.422	-0.644	-0.554	0.095	-0.346	15	1
2	-0.002	0.011	-0.493	-0.627	-0.580	0.022	-0.397	17	5
3	-0.002	-0.004	-0.515	-0.621	-0.586	0.007	-0.406	4	2
4	-0.002	-0.009	-0.523	-0.619	-0.588	0.001	-0.409	3	2
5	-0.002	-0.011	-0.525	-0.619	-0.589	0.000	-0.410	2	2
6	-0.002	-0.011	-0.526	-0.619	-0.589	0.000	-0.410	1	2
7	-0.002	-0.011	-0.526	-0.619	-0.589	0.000	-0.410	1	2
8	-0.002	-0.011	-0.526	-0.619	-0.589	0.000	-0.410	1	2
9	-0.002	-0.011	-0.526	-0.619	-0.589	0.000	-0.410	1	2
10	-0.002	-0.011	-0.526	-0.619	-0.589	0.000	-0.410	1	5
							Total Iterations	46	

b. 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:

Minimization	r_p	x_0		x_star		f(x_star)	g1(x_star)	g2(x_star)	g3(x_star)
1	800	0.4575	0.4497	0.44854	0.43958	389.973	-7.971	-0.103	-9.989
2	880	0.44854	0.43958	0.44756	0.43803	413.794	-7.951	-0.105	-9.951
3	968	0.44756	0.43803	0.44803	0.43757	453.809	-7.961	-0.104	-9.939
4	1064.8	0.44803	0.43757	0.44593	0.43535	457.026	-7.919	-0.108	-9.884
5	1171.28	0.44593	0.43535	2.3524	2.3522	64.626	-46.049	3.705	-57.804

Minimization	g4(x_star)	g5(x_star)	g6(x_star)	g7(x_star)	g8(x_star)	g9(x_star)	g10(x_star)	# of Iterations	Exit Flag
1	-0.023	-0.174	-0.792	-0.552	-0.643	-0.113	-0.467	21	1
2	-0.027	-0.226	-0.907	-0.523	-0.663	-0.162	-0.495	12	1
3	-0.028	-0.294	-1.091	-0.477	-0.693	-0.235	-0.540	5	2
4	-0.033	-0.306	-1.116	-0.471	-0.695	-0.236	-0.538	10	2
5	4.227	138.871	0.944	-0.986	1.155	0.000	-0.885	2	5

c) 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.

Minimization	r_p	x_0		x_star		f(x_star)	g1(x_star)	g2(x_star)	g3(x_star)
1	25000000	0.44	0.425	0.43893	0.42506	587.563	-7.779	-0.122	-9.626
2	12500000	0.43893	0.42506	0.43725	0.42268	613.900	-7.745	-0.125	-9.567
3	6250000	0.43725	0.42268	0.43458	0.42067	582.774	-7.692	-0.131	-9.517
4	3125000	0.43458	0.42067	0.43023	0.4154	614.957	-7.605	-0.140	-9.385
5	1562500	0.43023	0.4154	0.42593	0.412	572.039	-7.519	-0.148	-9.300
6	781250	0.42593	0.412	0.42422	0.4091	617.432	-7.484	-0.152	-9.228
7	390625	0.42422	0.4091	0.4218	0.40749	581.598	-7.436	-0.156	-9.187
8	195312.5	0.4218	0.40749	0.42019	0.40527	603.415	-7.404	-0.160	-9.132
9	97656.25	0.42019	0.40527	0.41835	0.40396	580.155	-7.367	-0.163	-9.099
10	48828.125	0.41835	0.40396	0.41602	0.40189	566.477	-7.320	-0.168	-9.047
11	24414.0625	0.41602	0.40189	0.4126	0.39859	557.084	-7.252	-0.175	-8.965
12	12207.03125	0.4126	0.39859	0.41102	0.39647	575.813	-7.220	-0.178	-8.912
13	6103.515625	0.41102	0.39647	0.40963	0.39521	568.794	-7.193	-0.181	-8.880
14	3051.757813	0.40963	0.39521	0.4085	0.39406	568.316	-7.170	-0.183	-8.851
15	1525.878906	0.4085	0.39406	0.40738	0.39293	566.674	-7.148	-0.185	-8.823
16	762.9394531	0.40738	0.39293	0.40624	0.3919	561.112	-7.125	-0.188	-8.797
17	381.4697266	0.40624	0.3919	0.40521	0.39053	572.357	-7.104	-0.190	-8.763
18	190.7348633	0.40521	0.39053	0.40404	0.38977	555.233	-7.081	-0.192	-8.744
19	95.36743164	0.40404	0.38977	0.40293	0.38907	537.868	-7.059	-0.194	-8.727
20	47.68371582	0.40293	0.38907	0.40207	0.38864	520.495	-7.041	-0.196	-8.716
21	23.84185791	0.40207	0.38864	0.40133	0.3887	489.176	-7.027	-0.197	-8.717
22	11.92092896	0.40133	0.3887	0.40083	0.38884	463.820	-7.017	-0.198	-8.721
23	5.960464478	0.40083	0.38884	0.40053	0.38915	440.593	-7.011	-0.199	-8.729
24	2.980232239	0.40053	0.38915	0.40037	0.38939	425.317	-7.007	-0.199	-8.735
25	1.490116119	0.40037	0.38939	0.40037	0.38969	413.588	-7.007	-0.199	-8.742
26	0.74505806	0.40037	0.38969	0.4003	0.38979	406.810	-7.006	-0.199	-8.745
27	0.37252903	0.4003	0.38979	0.40026	0.38988	402.019	-7.005	-0.199	-8.747
28	0.186264515	0.40026	0.38988	0.40023	0.38994	398.339	-7.005	-0.200	-8.748
29	0.093132257	0.40023	0.38994	0.4002	0.38998	395.643	-7.004	-0.200	-8.749
30	0.046566129	0.4002	0.38998	0.4002	0.39003	393.737	-7.004	-0.200	-8.751
31	0.023283064	0.4002	0.39003	0.4002	0.39006	392.698	-7.004	-0.200	-8.752
32	0.011641532	0.4002	0.39006	0.40019	0.39008	391.686	-7.004	-0.200	-8.752
33	0.005820766	0.40019	0.39008	0.40019	0.39009	391.049	-7.004	-0.200	-8.752
34	0.002910383	0.40019	0.39009	0.40018	0.39009	390.608	-7.004	-0.200	-8.752
35	0.001455192	0.40018	0.39009	0.40018	0.3901	390.301	-7.004	-0.200	-8.752
36	0.000727596	0.40018	0.3901	0.40017	0.3901	390.081	-7.003	-0.200	-8.753
37	0.000363798	0.40017	0.3901	0.40017	0.3901	389.984	-7.003	-0.200	-8.753
38	0.000181899	0.40017	0.3901	0.40017	0.3901	389.907	-7.003	-0.200	-8.753
39	9.09495E-05	0.40017	0.3901	0.40017	0.39011	389.831	-7.003	-0.200	-8.753
40	4.54747E-05	0.40017	0.39011	0.40017	0.39011	389.754	-7.003	-0.200	-8.753
41	2.27374E-05	0.40017	0.39011	0.40017	0.39011	389.677	-7.003	-0.200	-8.753
42	1.13687E-05	0.40017	0.39011	0.40017	0.39011	389.600	-7.003	-0.200	-8.753

Minimization	r_p	x_0		x_star		f(x_star)	g1(x_star)	g2(x_star)	g3(x_star)
43	5.68434E-06	0.40017	0.39011	0.40017	0.39011	389.593	-7.003	-0.200	-8.753
44	2.84217E-06	0.40017	0.39011	0.40017	0.39011	389.585	-7.003	-0.200	-8.753
45	1.42109E-06	0.40017	0.39011	0.40017	0.39011	389.578	-7.003	-0.200	-8.753
46	7.10543E-07	0.40017	0.39011	0.40017	0.39011	389.570	-7.003	-0.200	-8.753
47	3.55271E-07	0.40017	0.39011	0.40017	0.39011	389.563	-7.003	-0.200	-8.753
48	1.77636E-07	0.40017	0.39011	0.40017	0.39011	389.555	-7.003	-0.200	-8.753
49	8.88178E-08	0.40017	0.39011	0.40017	0.39011	389.551	-7.003	-0.200	-8.753
50	4.44089E-08	0.40017	0.39011	0.40017	0.39011	389.550	-7.003	-0.200	-8.753
51	2.22045E-08	0.40017	0.39011	0.40017	0.39011	389.549	-7.003	-0.200	-8.753
52	1.11022E-08	0.40017	0.39011	0.40017	0.39011	389.548	-7.003	-0.200	-8.753
53	5.55112E-09	0.40017	0.39011	0.40017	0.39011	389.548	-7.003	-0.200	-8.753
54	2.77556E-09	0.40017	0.39011	0.40017	0.39011	389.547	-7.003	-0.200	-8.753
55	1.38778E-09	0.40017	0.39011	0.40017	0.39011	389.547	-7.003	-0.200	-8.753
56	6.93889E-10	0.40017	0.39011	0.40017	0.39011	389.547	-7.003	-0.200	-8.753
57	3.46945E-10	0.40017	0.39011	0.40017	0.39011	389.546	-7.003	-0.200	-8.753
58	1.73472E-10	0.40017	0.39011	0.40017	0.39011	389.546	-7.003	-0.200	-8.753
59	8.67362E-11	0.40017	0.39011	0.40017	0.39011	389.546	-7.003	-0.200	-8.753
60	4.33681E-11	0.40017	0.39011	0.40017	0.39011	389.546	-7.003	-0.200	-8.753
61	2.1684E-11	0.40017	0.39011	0.40017	0.39011	389.546	-7.003	-0.200	-8.753
62	1.0842E-11	0.40017	0.39011	0.40017	0.39011	389.546	-7.003	-0.200	-8.753
63	5.42101E-12	0.40017	0.39011	0.40017	0.39011	389.546	-7.003	-0.200	-8.753
64	2.71051E-12	0.40017	0.39011	0.40017	0.39011	389.546	-7.003	-0.200	-8.753
65	1.35525E-12	0.40017	0.39011	0.40017	0.39011	389.546	-7.003	-0.200	-8.753
66	6.77626E-13	0.40017	0.39011	0.40017	0.39011	389.546	-7.003	-0.200	-8.753

Minimization	g4(x_star)	g5(x_star)	g6(x_star)	g7(x_star)	g8(x_star)	g9(x_star)	g10(x_star)	# of Iterations	Exit Flag
1	-0.055	-0.481	-1.775	-0.306	-0.763	-0.392	-0.626	6	0
2	-0.061	-0.508	-1.913	-0.272	-0.773	-0.415	-0.639	6	0
3	-0.065	-0.487	-1.781	-0.305	-0.761	-0.380	-0.616	6	0
4	-0.077	-0.525	-1.968	-0.258	-0.774	-0.405	-0.627	6	0
5	-0.084	-0.499	-1.786	-0.304	-0.757	-0.355	-0.592	6	0
6	-0.091	-0.541	-2.024	-0.244	-0.774	-0.399	-0.618	6	0
7	-0.094	-0.517	-1.862	-0.284	-0.761	-0.359	-0.590	6	0
8	-0.099	-0.539	-1.983	-0.254	-0.769	-0.379	-0.601	6	0
9	-0.102	-0.524	-1.879	-0.280	-0.760	-0.352	-0.582	6	0
10	-0.107	-0.518	-1.826	-0.293	-0.754	-0.333	-0.568	7	0
11	-0.114	-0.518	-1.803	-0.299	-0.750	-0.316	-0.553	7	0
12	-0.119	-0.538	-1.910	-0.272	-0.758	-0.335	-0.564	7	0
13	-0.122	-0.535	-1.884	-0.279	-0.755	-0.324	-0.555	7	0
14	-0.124	-0.537	-1.890	-0.278	-0.755	-0.322	-0.552	7	0
15	-0.127	-0.538	-1.890	-0.278	-0.754	-0.318	-0.549	7	0
16	-0.129	-0.536	-1.869	-0.283	-0.752	-0.309	-0.542	7	0
17	-0.132	-0.548	-1.935	-0.266	-0.757	-0.320	-0.548	8	0
18	-0.134	-0.537	-1.854	-0.286	-0.749	-0.298	-0.532	8	0
19	-0.135	-0.524	-1.771	-0.307	-0.741	-0.274	-0.514	8	0
20	-0.136	-0.509	-1.686	-0.328	-0.732	-0.249	-0.496	8	0
21	-0.136	-0.479	-1.527	-0.368	-0.715	-0.201	-0.463	8	0
22	-0.136	-0.451	-1.397	-0.401	-0.700	-0.158	-0.433	8	0
23	-0.135	-0.422	-1.277	-0.431	-0.684	-0.114	-0.404	7	2
24	-0.135	-0.401	-1.198	-0.451	-0.673	-0.082	-0.382	8	0
25	-0.134	-0.384	-1.136	-0.466	-0.663	-0.057	-0.365	8	0
26	-0.134	-0.373	-1.101	-0.475	-0.658	-0.042	-0.355	7	0
27	-0.134	-0.366	-1.076	-0.481	-0.654	-0.030	-0.347	8	0
28	-0.133	-0.360	-1.057	-0.486	-0.650	-0.022	-0.341	7	0
29	-0.133	-0.355	-1.043	-0.489	-0.648	-0.015	-0.337	7	0
30	-0.133	-0.352	-1.033	-0.492	-0.646	-0.010	-0.334	7	0
31	-0.133	-0.351	-1.028	-0.493	-0.645	-0.008	-0.332	7	0
32	-0.133	-0.349	-1.023	-0.494	-0.644	-0.005	-0.330	4	2
33	-0.133	-0.348	-1.019	-0.495	-0.644	-0.004	-0.329	3	2
34	-0.133	-0.347	-1.017	-0.496	-0.643	-0.003	-0.328	2	2
35	-0.133	-0.346	-1.015	-0.496	-0.643	-0.002	-0.328	3	2
36	-0.133	-0.346	-1.014	-0.496	-0.643	-0.001	-0.327	2	2
37	-0.133	-0.346	-1.014	-0.497	-0.643	-0.001	-0.327	1	2
38	-0.133	-0.346	-1.013	-0.497	-0.643	-0.001	-0.327	1	2
39	-0.133	-0.346	-1.013	-0.497	-0.643	-0.001	-0.327	1	2
40	-0.133	-0.346	-1.013	-0.497	-0.643	-0.001	-0.327	1	2
41	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.327	1	2
42	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2

Minimization	g4(x_star)	g5(x_star)	g6(x_star)	g7(x_star)	g8(x_star)	g9(x_star)	g10(x_star)	# of Iterations	Exit Flag
43	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
44	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
45	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
46	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
47	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
48	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
49	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
50	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
51	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
52	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
53	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
54	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	2
55	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
56	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
57	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
58	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
59	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
60	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
61	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
62	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
63	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
64	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
65	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
66	-0.133	-0.345	-1.012	-0.497	-0.643	0.000	-0.326	1	5
						Total Iterations:		**261**	

d) 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.

Minimization	r_p	x_0		x_star		f(x_star)	g1(x_star)	g2(x_star)	g3(x_star)
1	1	0.37	0.355	0.36847	0.35647	426.324	-6.369	-0.263	-7.912
2	1.5	0.36847	0.35647	0.45762	0.45	339.060	-8.152	-0.085	-10.250
3	2.25	0.45762	0.45	0.45759	0.45	337.571	-8.152	-0.085	-10.250
4	3.375	0.45759	0.45	0.45759	0.45	337.742	-8.152	-0.085	-10.250
5	5.0625	0.45759	0.45	0.45759	0.45	337.783	-8.152	-0.085	-10.250
6	7.59375	0.45759	0.45	0.45759	0.45	337.752	-8.152	-0.085	-10.250
7	11.390625	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
8	17.0859375	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
9	25.62890625	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
10	38.44335938	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
11	57.66503906	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
12	86.49755859	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
13	129.7463379	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
14	194.6195068	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
15	291.9292603	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
16	437.8938904	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
17	656.8408356	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
18	985.2612534	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
19	1477.89188	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
20	2216.83782	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250
21	3325.25673	0.45759	0.45	0.45759	0.45	337.764	-8.152	-0.085	-10.250

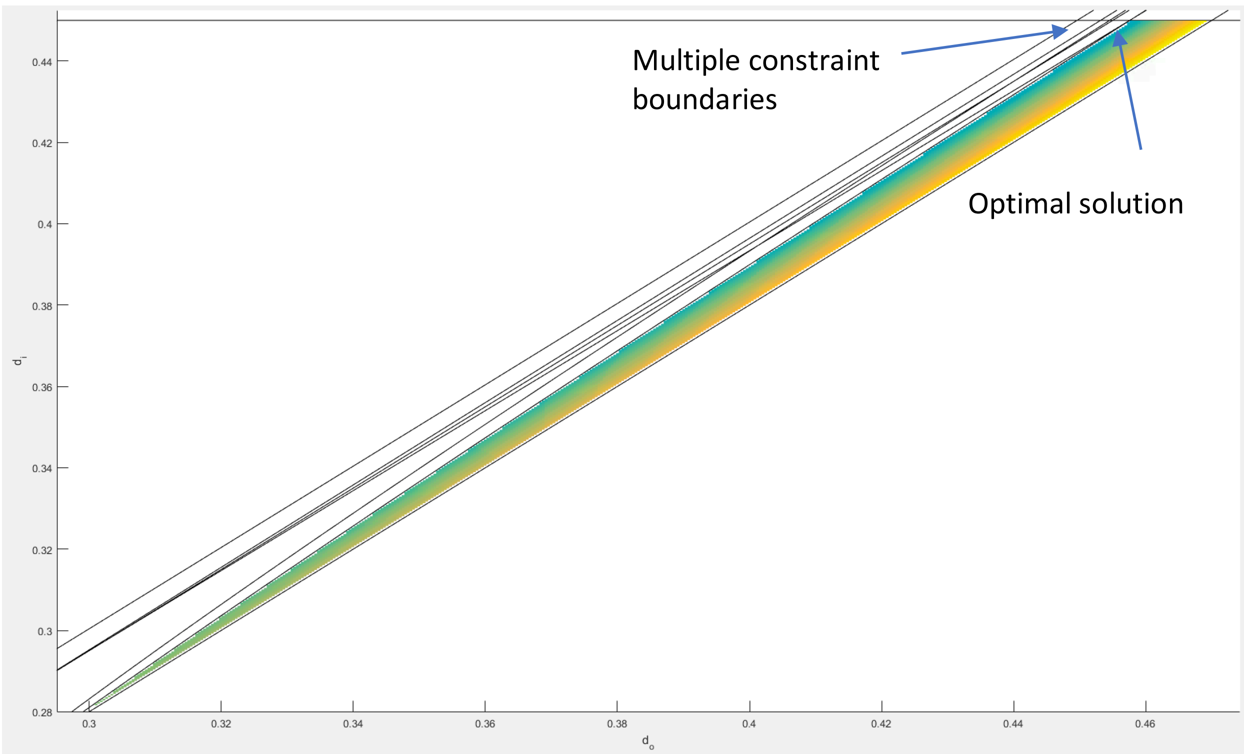
Minimization	g4(x_star)	g5(x_star)	g6(x_star)	g7(x_star)	g8(x_star)	g9(x_star)	g10(x_star)	# of Iterations	Exit Flag
1	-0.208	-0.497	-1.400	-0.400	-0.673	0.000	-0.269	2	2
2	0.000	-0.008	-0.524	-0.619	-0.589	-0.004	-0.413	15	2
3	0.000	-0.003	-0.518	-0.621	-0.587	0.001	-0.411	4	2
4	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	2	2
5	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	2
6	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	2
7	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	2
8	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
9	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
10	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
11	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
12	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
13	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
14	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
15	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
16	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
17	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
18	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
19	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
20	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
21	0.000	-0.004	-0.519	-0.620	-0.588	0.000	-0.411	1	5
						Total Iterations		**40**	

3. 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:



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.