Hw5

Monday, March 3, 2025 9:13 AM

5.1 KKT conditions

$$\min \; f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$$

subject to $g_1(x) = x^2+y^2-10 \le 0$

$g_2(x) = x+y-3 \le 0$

Solve for the optimal point and the Lagrange multipliers that satisfy the KKT conditions

$$L(x,y,\sigma_1,s_1,\sigma_2,s_2) = (x+2y-7)^2 + (2x+y-5)^2 + \sigma_1(x^2+y^2-10+s_1^2) + \sigma_2(x+y-3+s_2^2)$$

$$\frac{\partial L}{\partial x} = 2(x+2y-7) + 2(2x+y-5)2 + 2\sigma_1 x + \sigma_2$$

$$2x+4y-14+8x+4y-20 \rightarrow \boxed{10x+8y-34+2\sigma_1 x+\sigma_2}$$

$$\frac{\partial L}{\partial y} = 2(x+2y-7)2 + 2(2x+y-5) + 2\sigma_1 y + \sigma_2$$

$$4x+8y-28+4x+2y-10 \rightarrow \boxed{8x+10y-38+2\sigma_1 y+\sigma_2}$$

$$\frac{\partial L}{\partial \sigma_1} = 0+0+x^2+y^2-10+s_1^2$$

$\sigma \neq 0 \,, \; s = 0$

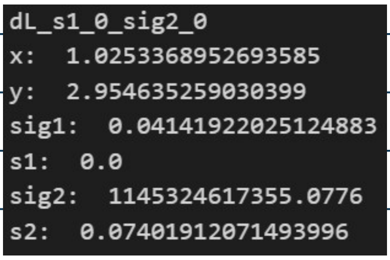
$$\frac{\partial L}{\partial s_1} = 0+0+2\sigma_1 s_1$$

$$\frac{\partial L}{\partial \sigma_2} = 0+0+0+x+y-3+s_2^2$$

$$\frac{\partial L}{\partial s_2} = 0+0+0+2\sigma_2 s_2$$

See code for work to solve:

Results:



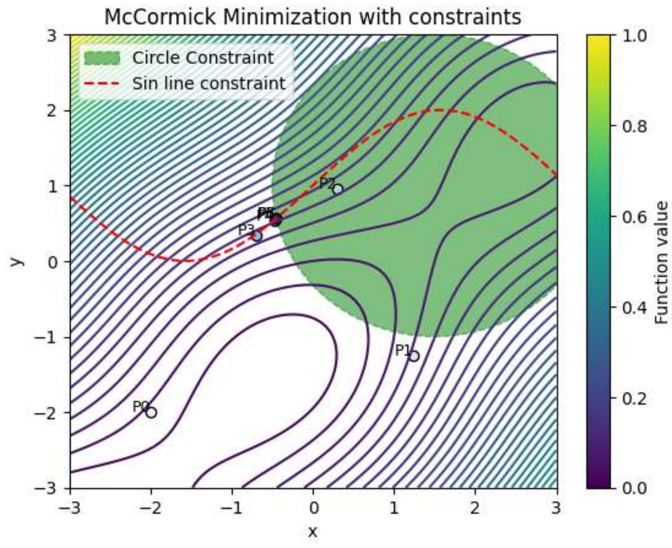
```
dL_s1_0_sig2_0
x:  1.0253368952693585
y:  2.954635259030399
sig1:  0.04141922025124883
s1:  0.0
sig2:  1145324617355.0776
s2:  0.07401912071493996
```

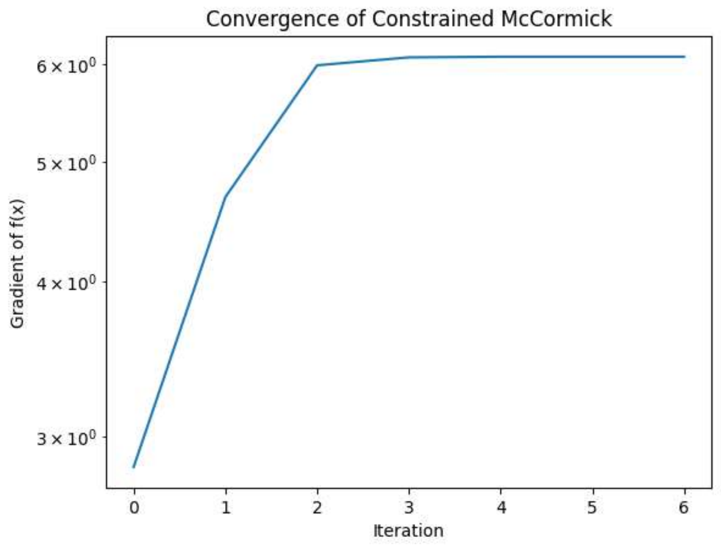
The optimal point is (1.025, 2.95) and this was the only feasible sig1, s1, sig2, s2 combo. Sigma 1 is positive, s1 is 0, sigma 2 is 0 and s2 is positive.

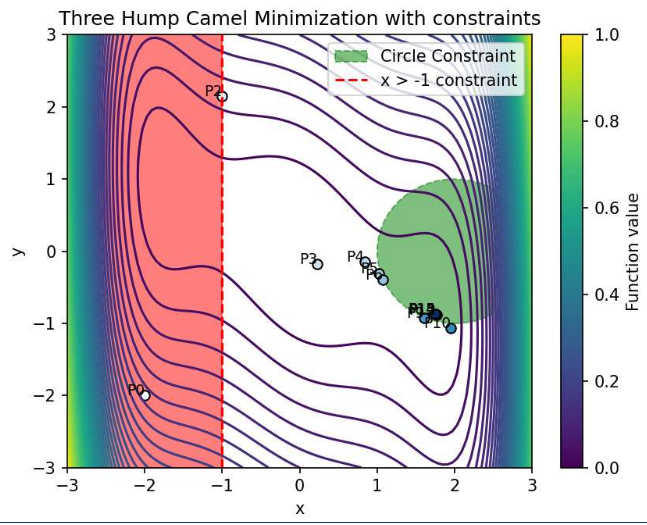
5.2 Constrained Optimization

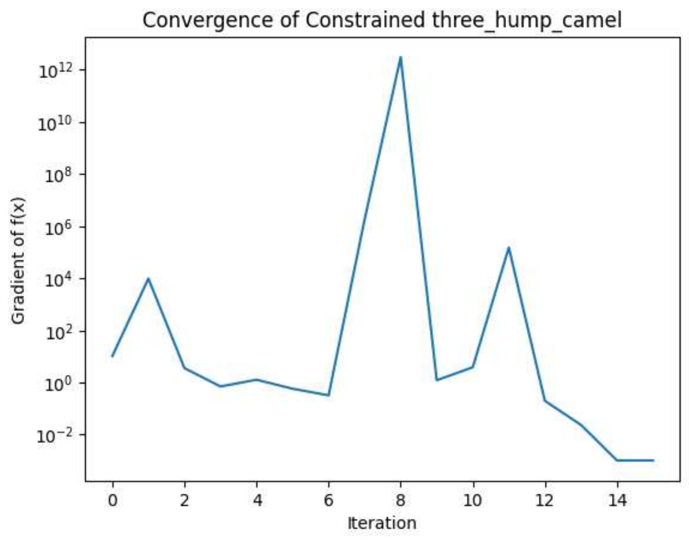
Select **three** functions found here:
https://en.wikipedia.org/wiki/Test_functions_for_optimization
Don't select the Booth function (that's 5.1) and don't worry about the multi-objective functions right now. For each function, identify two constraints that would make your optimization somewhat exciting (i.e. at least one of those constraints should be active at x^*). Do not choose two equality constraints if the design space is two-dimensional. (That's a trivial solution. Do you know why?)

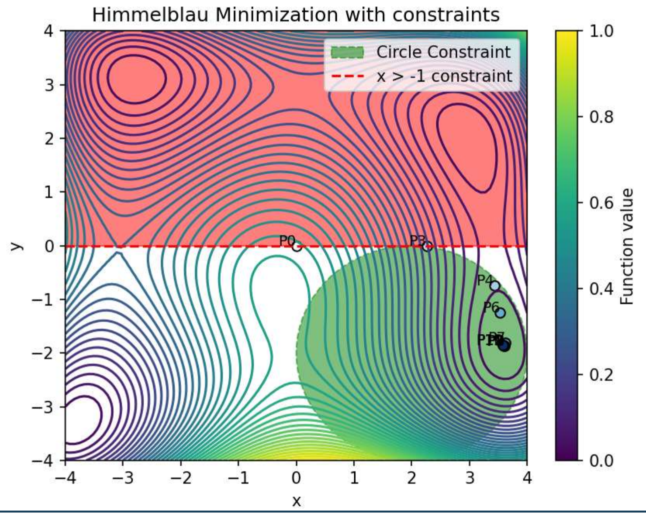
Solve your three problems using an existing optimizer. Show the **contour plot for each** (with constraints, the iteration history, and the optimal point) and a **convergence plot for each**. You may want to start from different initial guesses. Note, you can make adjustments to the functions (like changing coefficients), but make sure the function is still exciting, interesting, or somewhat challenging.

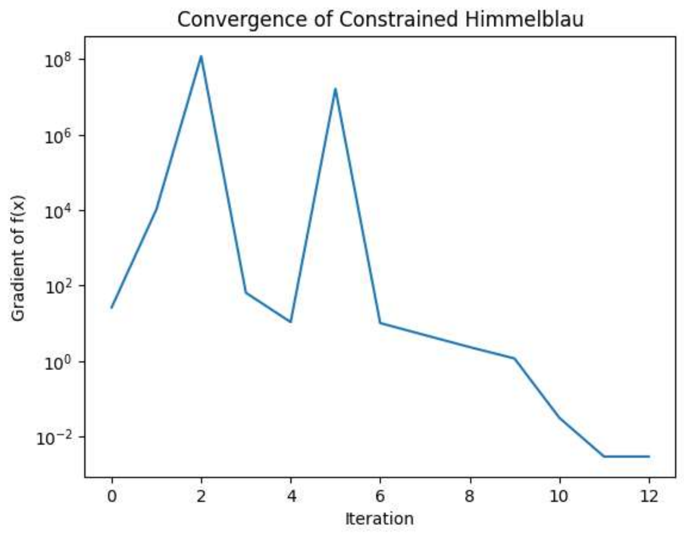












The constraints made the functions a lot more difficult. They were still solved in relatively few iterations, but in the case of Three Hump Camel and Himmelblau, a few points were thrown very far off before finding the minimum, as can be seen in the convergence plots. McCormick converges to a nonzero final gradient as it isn't actually at the minimum. The other two have small slopes and converge when their slopes don't change much either.

5.3 Student-Defined Optimizer

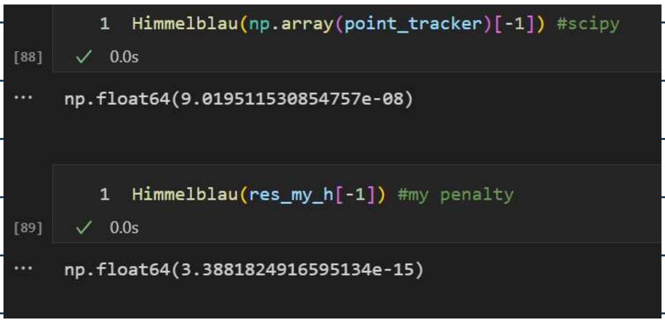
Write a *basic* constrained gradient-based optimizer using a penalty method. Test your method out on your three functions and constraints you identified in question 5.2 above. In a table, compare the solution accuracy and convergence efficiency of your own method against the optimizer you used in 5.2. Briefly describe what you learned in 400 words or less.

Solver	McCormick	Three Hump Camel	Himmelblau
SciPy function calls	7	16	13
My Penalty Method function calls	134+	163+	112+
Percent f(x) error for penalty method	-1.23%	0.000015%	99.99%**
Distance off for penalty method	0.0286	0.00009944	0.000072

**This is because the values are so small and close to zero, so in the % when divided by a small number it blows up. See the screenshot below to see they are quite close and my penalty method technically did better

I was off at first because the penalty method I coded, for inequality constraints, does maximum(0,g(x)), but g(x) returns positive to mean good for scipy, but positive here means bad, so I had to switch the signs. I am impressed how well my coded function did on getting so close. The method is quite accurate, although mine requires a lot more function calls. Interestingly enough, my penalty method technically did better on Himmelblau, as the minimum was 0 and my coded penalty method was at e-15 vs scipy at e-8, though that is just a tolerance or lucky chance thing.

On all of the functions my max_iters (100) was reached, but they only had 134, 163, and 112 recorded points respectively, so either some of the inner minimize routines ran less than 2 steps, or it didn't record all the points despite deepcopying etc. Thus I said 134+. I do wonder if SciPy is only returning outer iterations as well.



```
[88]
Himmelblau(np.array(point_tracker)[-1]) #scipy
✓ 0.0s

np.float64(9.019511530854757e-08)
```

```
[89]
Himmelblau(res_my_h[-1]) #my penalty
✓ 0.0s

np.float64(3.3881824916595134e-15)
```