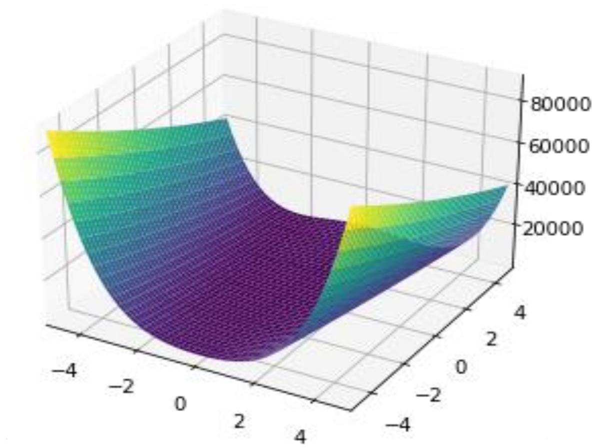
Exercise 2

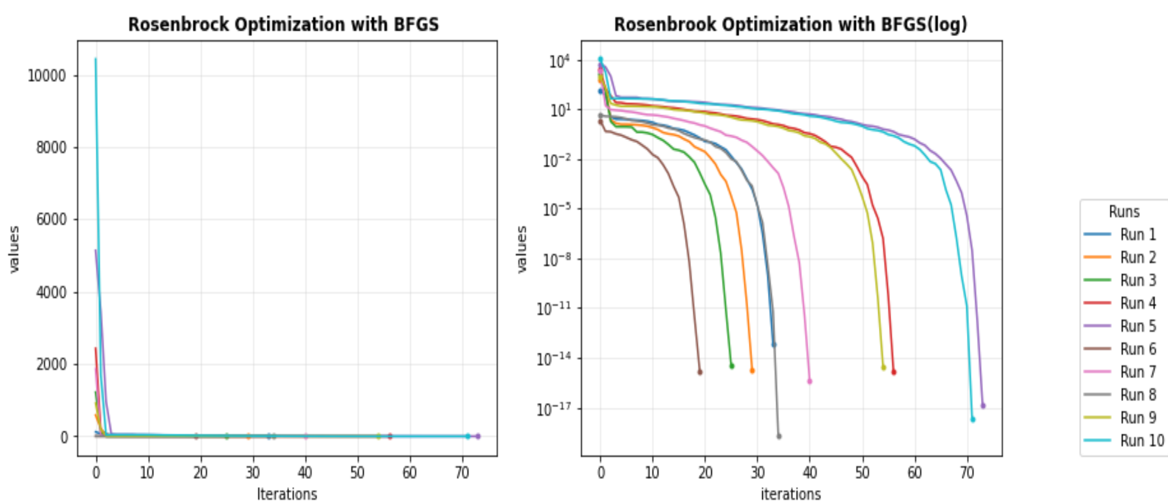
- The Banana (Rosenbrock) Function

In minimizing Rosenbrock function, the gradient descent approach (implemented in python) was used within bounds [-4, 4]. In all cases, the global minimum was found within 6 significant digits.

This a feasible solution as we can observed on the python file, for each run the algorithm is converging toward the global optimum

Run	Starting point	Optimal point	Optimal value	Number of iterations	Feasible	Time (sec)
1	[1.66250252 -0.90712071]	[1.00000021 1.00000042]	0	34	Yes	0.0260787010192871 1
2	[1.34471794 -2.61502665]	[0.99999996 0.9999992]	0	30	Yes	0.0258681774139404 3
3	[1.49774056 -3.51780251]	[0.99999995 0.9999999]	0	26	Yes	0.0208878517150878 9
4	[-3.15569399 -0.41769905]	[0.99999998 0.99999996]	0	57	Yes	0.029264211654663086
5	[-3.52320854 -0.93248649]	[1. 1.]	0	74	Yes	0.047692060470581055
6	[1.22956255 2.46509778]	[0.99999997 0.99999993]	0	20	Yes	0.0260801315307617 2
7	[2.36148569 -2.64056721]	[1.00000002 1.00000003]	0	41	Yes	0.025845766067504883
8	[-1.18806422 1.04054283]	[1. 1.]	0	35	Yes	0.023450374603271484
9	[-1.66251644 -2.82015772]	[0.99999999 0.99999998]	0	55	Yes	0.0456044673919677 7
10	[-3.72101968 -2.94140858]	[1. 1.]	0	72	Yes	0.051743507385253906





See Python file

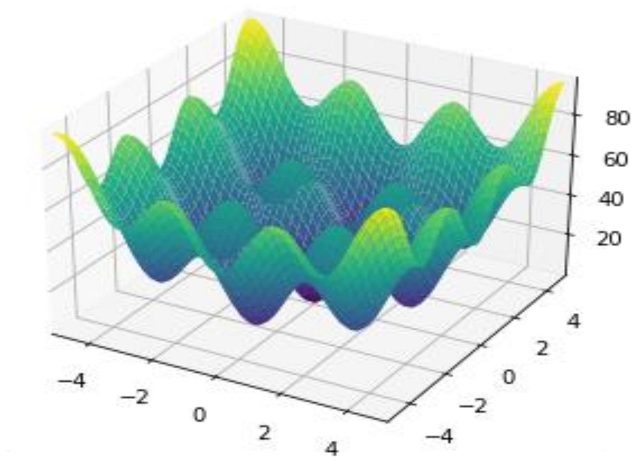
- The eggcrate function

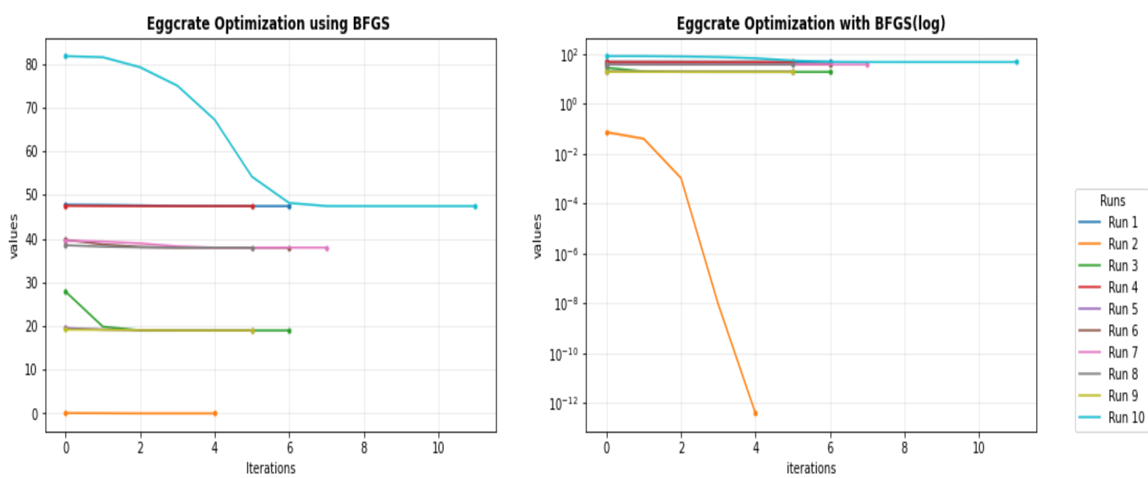
For eggcrate function, the same algorithm was employed and it was noticed that gradient- based optimizer were stuck in the local optima, depending on the starting point that were chosen randomly within bounds [-2π, 2π].

Out of 10 runs, 9 was stuck in a local minimum while 1 found the global optima.

In this case we still have feasible solutions. Although the optimum at each run is feasible, it is still unlikely to catch the global optimum.

Run	Starting point	Optimal point	Optimal value	Number of iterations	Feasible	Time (sec)
1	[-4.13925234 -5.9932668]	[-3.01960188 -6.03142402]	47.417669	7	Yes	0.007659912109375
2	[0.64868558 -0.72541048]	[-3.92466624e-08 -1.14949864e-07]	0	5	Yes	0.0062885284423828125
3	[4.39006606 2.42564939]	[3.01960187 3.01960187]	18.976395	7	Yes	0.006117582321166992
4	[5.58882228 -2.95615944]	[6.03142402 -3.01960194]	47.417669	6	Yes	0.013808488845825195
5	[-4.05061976 -2.5695851]	[-3.0196019 -3.0196087]	18.976395	6	Yes	0.005657196044921875
6	[0.6891563 5.75093381]	[8.02955849e-09 6.03142401e+00]	37.929472	7	Yes	0.013371944427490234
7	[-5.56695809 -0.63792794]	[-6.03142401e+00 3.85023245e-10]	37.929472	8	Yes	0.010630369186401367
8	[4.53745943 -5.02133461]	[-3.57050750e-10 -6.03142401 +00]	37.929472	6	Yes	0.0076673030853271484
9	[2.68352678 3.89065855]	[3.01960195 3.01960187]	18.976395	6	Yes	0.006125450134277344
10	[4.86697494 -5.98830213]	[3.01960189 -6.03142401]	47.417669	12	Yes	0.010137796401977539





See python file

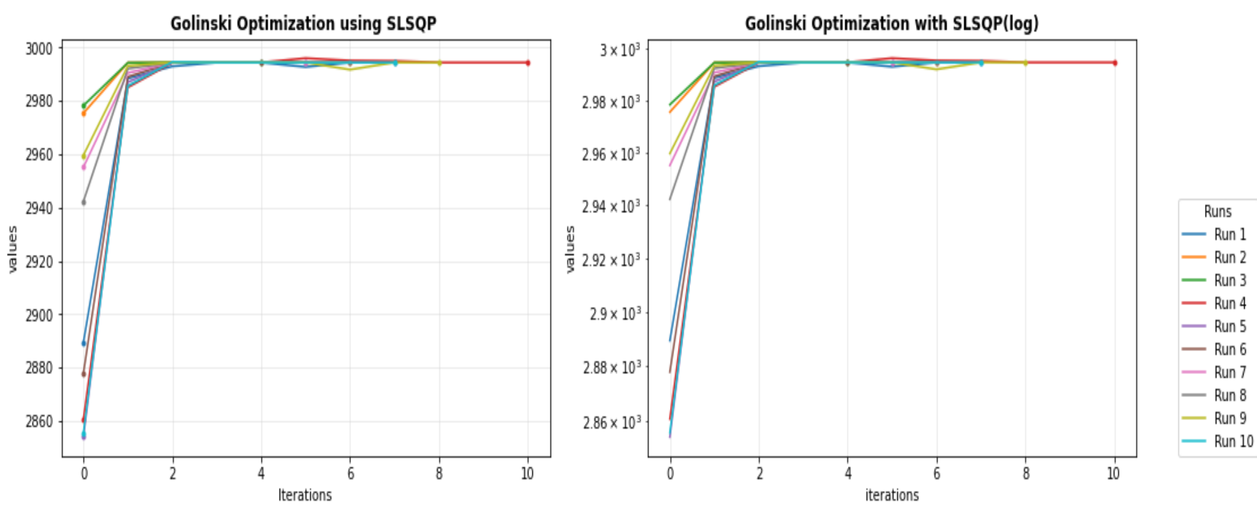
- Golinski’s Speed Reducer

SLSQP – Sequential Least Squares Programming approach has been used for this optimization problem

Finally the Golinski’s speed reducer problem was solved using sequential quadratic programming approach (implemented in Python) that handles gradient-based constrained optimization problems. This problem has 11 constraints, in addition to bound constraints and objective is to minimize the weight of the speed reducer. The 10 starting points were picked at random as before and the quickly optimizer converged to the optimum in all cases.

Run	Starting point	Optimal point	Optimal value	Number of iterations	Feasible	Time (sec)
1	[2.7551, 0.7436, 21.1665, 7.3794, 8.0887, 2.9018, 5.2142]	[3.5000, 0.7000, 17.0000, 7.3000, 7.7153, 3.3502, 5.2867]	2994.3516	11	Yes	0.20740818977355957
2	[3.3695, 0.8088, 25.0352, 7.9858, 8.0607, 3.1946, 5.4470]	[3.5000, 0.7000, 17.0000, 7.3000, 7.7153, 3.3502, 5.2867]	2994.3516	8	Yes	0.09480667114257812
3	[3.4303, 0.7682, 19.6928, 7.3937, 7.3078, 3.3007, 5.5022]	[3.5000, 0.7000, 17.0000, 7.3000, 7.7153, 3.3502, 5.2867]	2994.3516	5	Yes	0.066616296768818848
4	[2.8209, 0.7467, 22.2140, 7.9013, 7.7479, 3.8998, 5.5598]	[3.5000, 0.7000, 17.0000, 7.3000, 7.7153, 3.3502, 5.2867]	2994.3516	10	Yes	0.205561399459838 87
5	[3.0868, 0.7663, 19.6152, 7.7766, 8.0903, 3.8417, 5.8043]	[3.5000, 0.7000, 17.0000, 7.3000, 7.7153, 3.3502, 5.2867]	2994.3516	9	Yes	0.088186979293823 24
6	[2.8003, 0.7415, 20.2288, 7.8916, 8.1696, 3.7621, 5.5744]	[3.5000, 0.7000, 17.0000, 7.3000, 7.7153, 3.3502, 5.2867]	2994.3516	8	Yes	0.059932708740234375
7	[3.5311, 0.7799, 22.3920, 7.9197, 7.4234, 3.8438, 5.2505]	[3.5000, 0.7000, 17.0000, 7.3000, 7.7153, 3.3502, 5.2867]	2994.3516	5	Yes	0.072154521942138 67
8	[3.0801, 0.7039, 17.4099, 7.9046, 8.1320, 3.7719, 5.1876]	[3.5000, 0.7000, 17.0000, 7.3000, 7.7153, 3.3502, 5.2867]	2994.3516	10	Yes	0.125942468643188 48

9	[3.4462, 0.7669, 24.0182, 8.1382, 7.8775, 2.9449, 5.4831]	[3.5000, 0.7000, 17.0000, 7.3000, 7.7153, 3.3502, 5.2867]	2994.3516	12	Yes	0.12774682044982 91
10	[2.6080, 0.7266, 25.8264, 7.9898, 7.8160, 3.7639, 5.4972]	[3.5000, 0.7000, 17.0000, 7.3000, 7.7153, 3.3502, 5.2867]	2994.3516	11	Yes	0.13335514068603516



See python file

Exercise 3

As heuristic technique, we have used the PSO - Particle Swarm Optimization approach (See Python file).

i. Dependence of answers on initial design vector (start point, initial population)

Problem Name	Gradient-Based Optimizer	Particle Swarm Otpimizer
Rosenbrock Function	low	Low
Eggcrate Function	High	Low
Golinski Speed Reducer	Low	low

ii. Computational effort (CPU time [sec] or FLOPS)

Problem Name	Gradient-Based Optimizer	Particle Swarm Otpimizer
Rosenbrock Function	0.031	0.347
Eggcrate Function	0.008	0.151
Golinski Speed Reducer	0.117	17.43

iii. Convergence history

Problem Name	Gradient-Based Optimizer	Particle Swarm Otpimizer
Rosenbrock Function	Always converged to global minimum.	Converged, but efficiency depends on the tuning parameters selected
Eggcrate Function	Always converged, but either a local or a global minimum.	Converged, but efficiency depends on the tuning parameters selected
Golinski Speed Reducer	Always converged to global minimum.	Converged, but efficiency depends on the tuning parameters selected

iv. Frequency at which the technique gets trapped in a local optimum

Problem Name	Gradient-Based Optimizer	Particle Swarm Otpimizer
Rosenbrock Function	0	0
Eggcrate Function	0.9	0
Golinski Speed Reducer	0	0

Conclusion

Gradient-based optimizers can get stuck in local optima and are sensitive to the starting point, especially if there are multiple optima in the design space.

Genetic Algorithms are computationally expensive and requires considerable "tuning" effort, especially for complex problems.