Stochastic Optimisation

Practical Optimisation

Abhishek Shenoy

Abstract

This report investigates the use of a genetic algorithm (GA) and simulated annealing (SA) to find the global minimum of the 2D and the 6D eggholder function.

1 Introduction

Finding the global optimisation of complicated functions can be extremely difficult due to there being many local minima. Often a gradient-based approach is not tractable and hence we resort to stochastic optimisation methods. In this report, we investigate a genetic algorithm (GA) and simulated annealing (SA) to attempt to find the global minimum of the 6D eggholder function.

The n-dimensional eggholder function is given in Equation 1.

$$\begin{aligned} f(\boldsymbol{x}) = \sum_{i=1}^{n-1} \Big[& -x_i \sin\left(\sqrt{|x_i - x_{i+1} - 47|}\right) \\ & - (x_{i+1} + 47) \sin\left(\sqrt{\left|\frac{x_i}{2} + x_{i+1} + 47\right|}\right) \Big] \\ \text{Subject to } & -512 \leq x_i \leq 512 \text{ for } i \in \{1, ..., n\} \end{aligned} \tag{1}$$

To be able to configure our hyperparameters for the optimisation algorithms, we need to be able to evaluate their performance which can be plausibly done by testing on the 2D eggholder function.

The 2-dimensional eggholder function is given in Equation 2.

$$\begin{aligned} f^{(2d)}(x_1, x_2) = & -x_1 \sin\left(\sqrt{|x_1 - x_2 - 47|}\right) \\ & - (x_2 + 47) \sin\left(\sqrt{\left|\frac{x_1}{2} + x_2 + 47\right|}\right) \\ \text{Subject to } & -512 \leq x_i \leq 512 \text{ for } i \in \{1, 2\} \end{aligned} \tag{2}$$

For this case, we know the global minimum as being the following:

$$\text{Global Minimum} \quad f^{(2d)}(512, 404.2319) = -959.6407$$

We can visualise the surface and contour plot of the 2D function in Figure 1 to observe the nature of the function and the location of the global minimum.

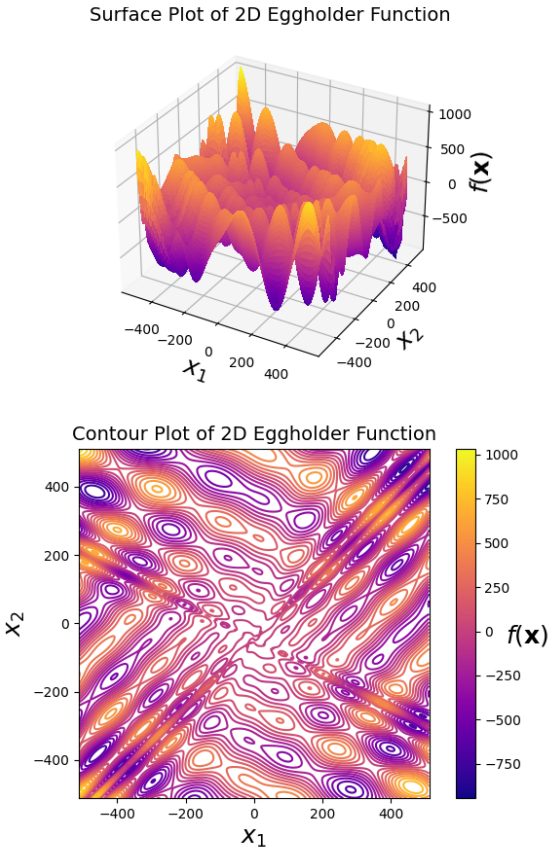


Figure 1. 2D Eggholder function

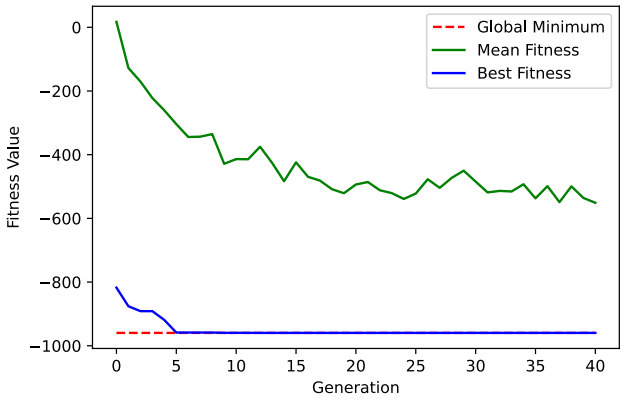
2 Genetic Algorithm

The genetic algorithm is a population-based optimisation approach in which multiple candidates are tested and evaluated. We consider $N = 500$ individuals in the population. The cost of adding more individuals in the population is minor in comparison to running more generations and it is clear that if we increase N, we are asymptotically more likely to find the global minimum although for higher dimensions the required population size increases .

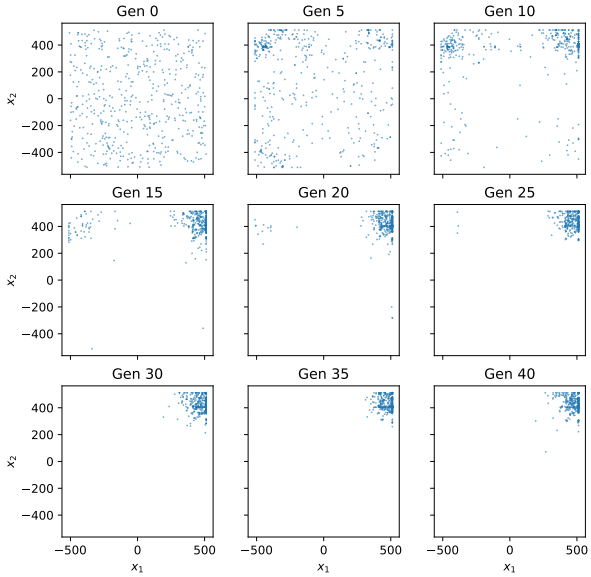
For every generation, we select the top individuals in the pool (using an elitism factor of 0.1) and then generate child candidates using a reproduction method and a mutation method both occurring independently with specific probabilities P_c and P_m respectively. We investigate the impact of varying the parameters to attempt to find the global minimum of the 6D eggholder function.

2.1 Visualisation

Figure 2 shows a single run of SA on the 2D function. The algorithm successfully locates the global minimum in the top right corner (Fig 2b) with only 20 generations. 40 generations have been simulated to show the effect of convergence on the mean fitness of the whole population (Fig 2a).



(a) Fitness over 40 generations for seed=0



(b) Population over 40 generations for seed=0

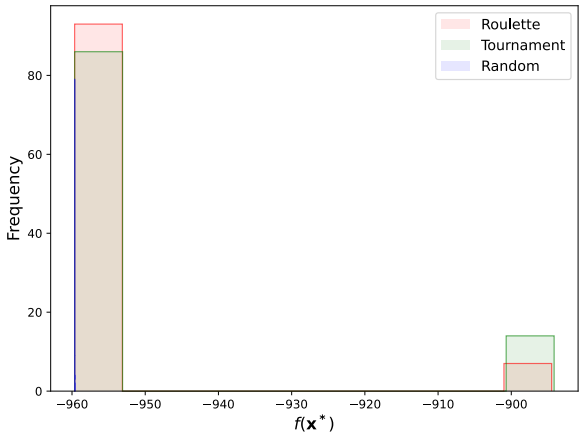
Figure 2. Genetic algorithm for 2D problem (40 generations, seed=0)

2.2 Parameter Variation

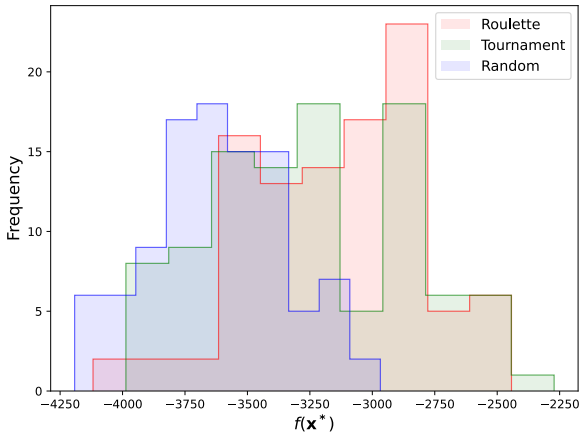
To find the best parameters, we first try to obtain the best selection and reproduction methods with unbiased crossover and mutation probability ($P_c = P_m = 0.5$). Once determining the methods that work best, we explore the probabilities to obtain the best parameters.

2.2.1 Selection Method

To select the population, we consider 3 methods specifically a roulette, a tournament and random selection. In a roulette, a parent is chosen from a distribution that is representative of the proportion of individuals with a specific fitness. In a tournament, some proportion (tournament size = 0.1) of the current population is chosen randomly and the best of the selection is chosen as a parent. In random selection, a candidate is chosen as a parent randomly from the current population with no bias. This is repeated N times to get a population of N parents which is then used to generate the new population by reproduction and mutation.



(a) 2D Problem



(b) 6D Problem

Figure 3. GA Histogram of solutions using different selection methods (100 seeds per method)

Figure 3 shows that both the tournament and roulette perform inferior to a random selection. This is likely due to the fact that the added bias in the tournament and roulette method usually used to obtain better estimates forces a greater population to fall into local minima rather than exploring the rest of the region. The random selection scheme works significantly better in the 2D problem where all solutions fall in the correct local minima and most find the global minimum (all function values are < -955 shown by the thin histogram bar/line in Figure 3a) whereas in the 6D problem, the selection produces many candidates that fall in local minima (Fig 3b). This is because a random selection in higher dimensions explores less of the hyperspace. Nonetheless the added bias from tournament and roulette selection appear to add no advantage in comparison to random selection. Therefore for the continued investigation below, we have opted to use the random selection scheme.

2.2.2 Reproduction Method

Given two individuals $\underline{p}_1$ and $\underline{p}_2$ from the population (parents), we need to generate a third candidate (child) $\underline{c}$. This is known as a reproduction or crossover method. Note that there is a probability P_c to determine whether this takes place or not. If the crossover method does not take place then the best parent is used as a candidate instead.

We consider 2 reproduction/crossover methods in this investigation.

Method 1 - Stochastic Arithmetic Recombination
This is a stochastic version of the whole arithmetic recombination method.

$$\gamma \sim \mathcal{N}(0.5, 0.15)$$
$$\underline{c} = (1-\gamma)\underline{p}_1 + \gamma \underline{p}_2$$

We do not further explore the parameters of the distribution from which γ is sampled. This is irrelevant to the investigation since these parameters are only used to generate candidates using a normal distribution so the key technique is the method of recombination and not the value of the recombination. Here we consider a scalar recombination (the child candidate lies on a straight line somewhere between the 2 parents) whereas in method 2, we consider a vector recombination. Note that it is better to use a normal distribution since we do not want our candidates to be too close to the parents (preferring it to be around the midpoint) and hence a small standard deviation is used.

Method 2 - Uniform Crossover
The uniform crossover method involves deciding each component of the candidate (gene) using a coin flip to determine whether to take the component from the first or the second parent (hence the use of the Hadamard product notation below).

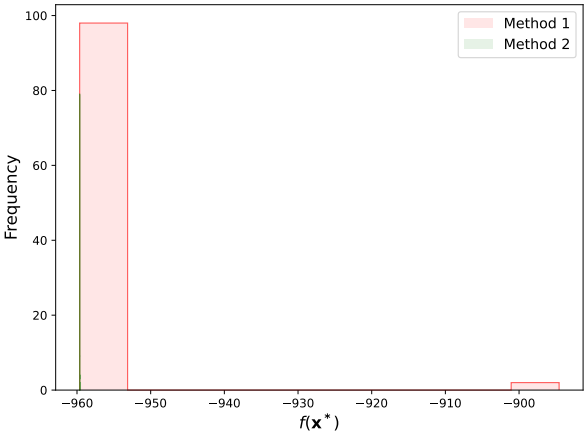
$$\underline{u} \sim \text{Bern}(0.5 \cdot \underline{1}) \qquad \underline{u} \in \mathbb{R}^D$$
$$\underline{c} = (\underline{1} - \underline{u}) \circ \underline{p}_1 + \underline{u} \circ \underline{p}_2$$

However in the case of 2 dimensions, we enforce equal crossover meaning that each component is taken from separate parents. This is simply to ensure that the child candidate is not a replica of its parents to avoid wasting computation.

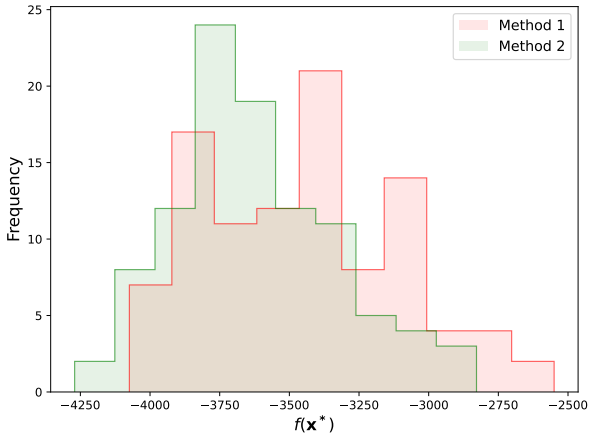
This method uses vector recombination equivalent to traversing the corners of a hyper-cuboid. We can visualise this by considering the zero vector $\underline{0}$ and the one vector $\underline{1}$ in 3d space and recombining the 2 parent vectors using the uniform crossover method resulting in the traversal of the corners of the cube. Using alternate parent vectors is equivalent to simply using a different basis.

Method Evaluation

In Figure 4, we see that vector recombination (Method 2) outperforms scalar recombination (Method 1) in both the 2D and 6D problem but this is more noticeable in the 2D case. It is likely that in even higher dimensions with a larger population, the advantage of vector recombination over scalar recombination is not felt since there is stochastic selection in both cases and a straight-line recombination in hyperspace could perform better when multiple reproductions/crossovers take place. Nonetheless we have opted to use vector recombination due to sufficient evidence of improvement over scalar recombination for the continued investigation below.



(a) 2D Problem



(b) 6D Problem

Figure 4. GA Histogram of solutions using different reproduction methods (100 seeds per method)

2.2.3 Crossover Probability P_c

The crossover probability is used to determine whether we generate a child from 2 parents using reproduction by applying the crossover method or whether to take the best parent as the candidate. This is very beneficial for retaining locations of local minima that can eventually be used to determine the global minima by mutation. It is clear that the value of this must be high to allow effective traversal. However with the combined effect of mutation, it may be better to have a lower crossover probability so that mutation has a greater effect on generating the new candidate. Hence we explore a wide range of probabilities P_c between 0.05 and 0.95 with $P_m = 0.5$.

Figure 5 shows that there is no particular value that performs significantly better for the 2D or the 6D case as the mean fitness line is relatively constant and the best fitness fluctuates without any clear trend. However it is notable that a lower value works marginally better than a higher value in both the 2D and 6D case. We opt to use a value of $P_c = 0.35$ which provides the minimum best solution for the 6D case over 100 seeds.

2.2.4 Mutation Probability P_m

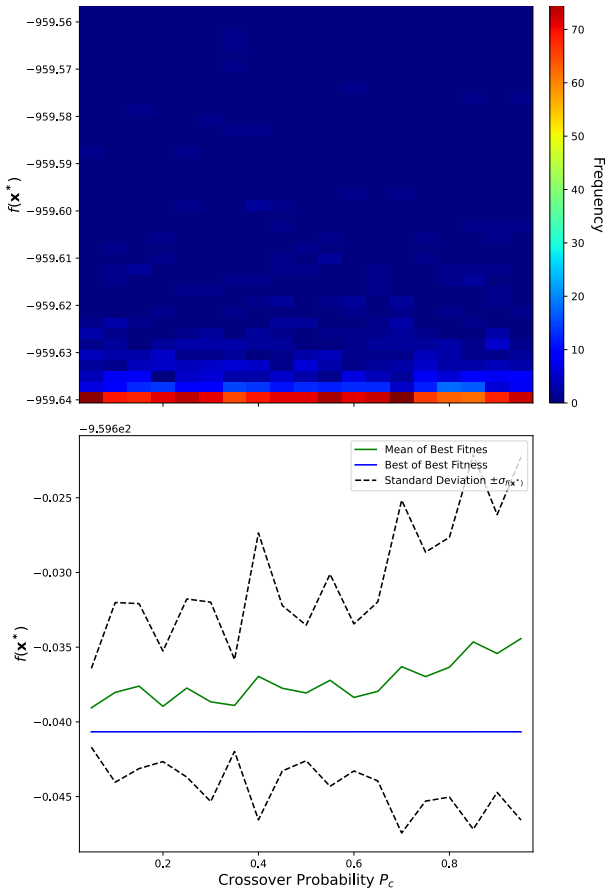
Mutation occurs with probability P_m after the crossover method has been applied with $P_c = 0.5$. We define the mutation method as the following:

$$\begin{aligned} \underline{B} &= 1024 \cdot \underline{1} \qquad \underline{B} \in \mathbb{R}^D \\ \underline{b} &\sim U(-m\underline{B}, m\underline{B}) \\ \underline{c} &= \underline{c} + \underline{b} \end{aligned}$$

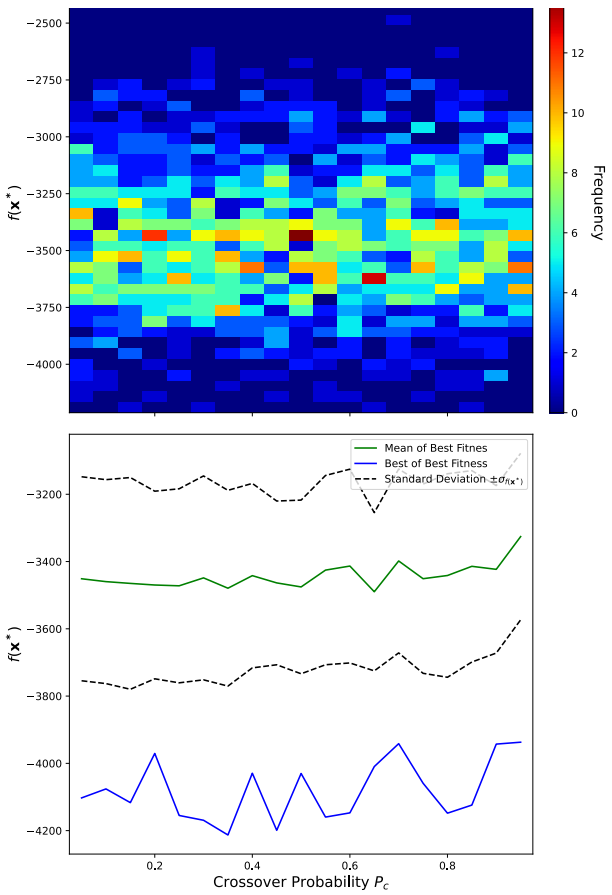
$\underline{B}$ are the bound lengths of the candidate vectors (in our case all of the bounds are -512 to 512 and hence of length 1024), m is the mutation scale which we set to 0.1 by experimentation and $\underline{c}$ is the child vector returned from the crossover method. The mutation scale must be small enough to not hop out of local minima but large enough to explore the local minima thoroughly as much as possible.

Figure 6b shows that in the 6D case as the mutation probability increases better solutions are found more frequently. In the 2D case (Fig 6a), there is no clear value that is better as the algorithm successfully finds the global minimum in majority of the cases although again there is some evidence to suggest that a higher value is preferable.

We decide to use a mutation probability of $P_m = 1$ so that mutation is guaranteed for all generated child candidates. The Monte-Carlo variation caused due to mutation actually helps to generate more diverse candidates within the surrounding local minimum therefore being able to evaluate local minima more effectively.

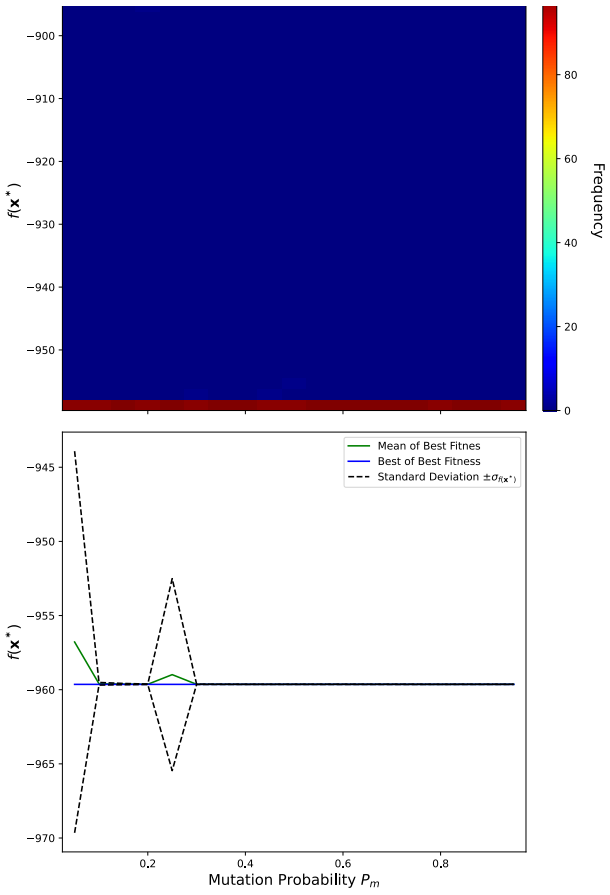


(a) 2D Problem

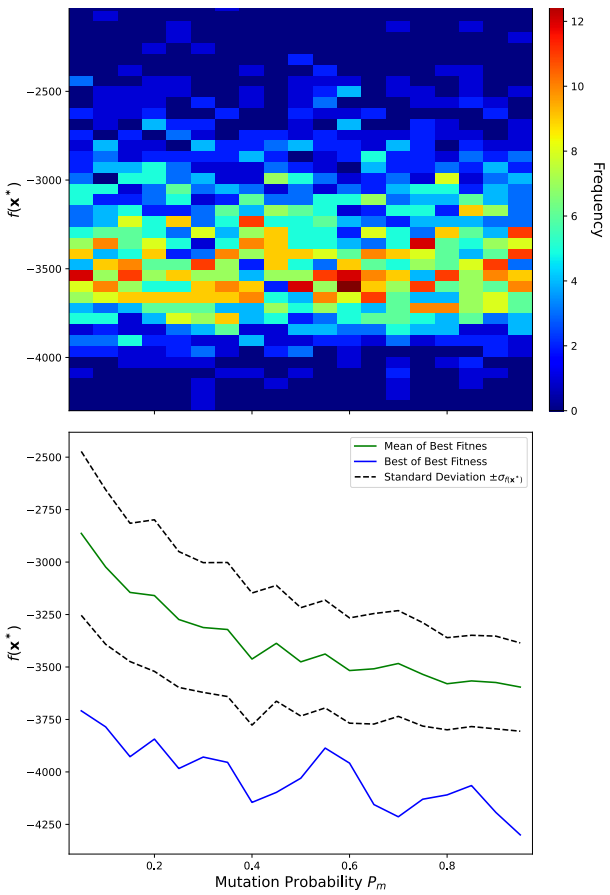


(b) 6D Problem

Figure 5. GA Histogram of solutions with variation of crossover probability (100 seeds per value)



(a) 2D Problem

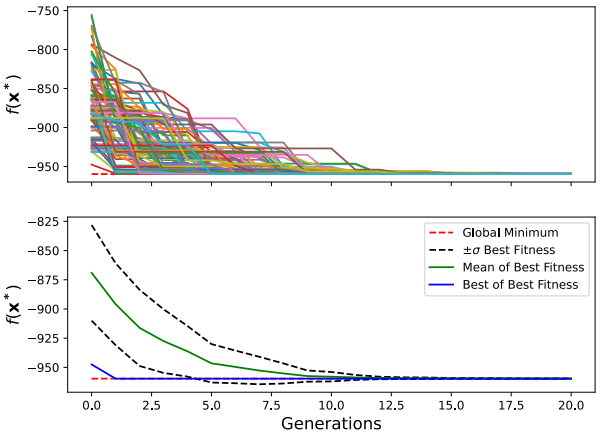


(b) 6D Problem

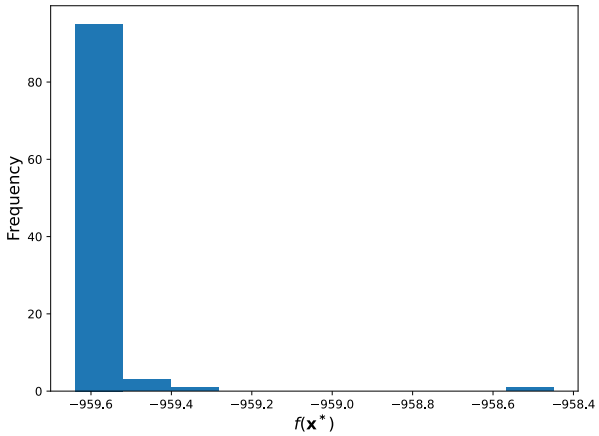
Figure 6. GA Histogram of solutions with variation of mutation probability (100 seeds per value)

2.3 Best Solutions

To measure the best performance of the algorithm, we use the investigated parameters to obtain final solutions over different seeds to see how well they converge. The best parameters from our investigation were $P_c = 0.35, P_m = 1.0$ with the random selection method and vector recombination as the reproduction method. We plot the minimisation evolution and the histogram of the final solutions of each seed for both the 2D problem (Fig 7) and the 6D problem (Fig 8).



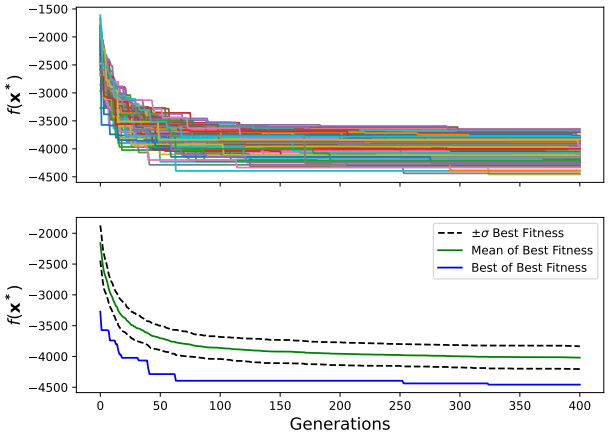
(a) Best fitness for different seeds



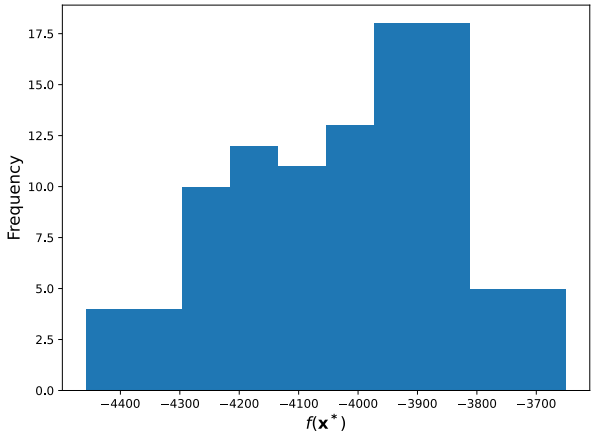
(b) Histogram of final best fitness for different seeds

Figure 7. GA Best solutions for 2D problem using 100 different seeds (20 generations/seed)

Since the parameters are tuned for the 6D problem, the 2D case performs marginally worse than if the parameters were tuned specifically for the 2D problem. Nonetheless for the 6D problem, we find many minima below -4000 with very few iterations and this is a significant achievement for this problem. The best solution found by the genetic algorithm has an objective function value of $\mathbf{-4457.3618}$.



(a) Best fitness for different seeds



(b) Histogram of final best fitness for different seeds

Figure 8. GA Best solutions for 6D problem using 100 different seeds (400 generations/seed)

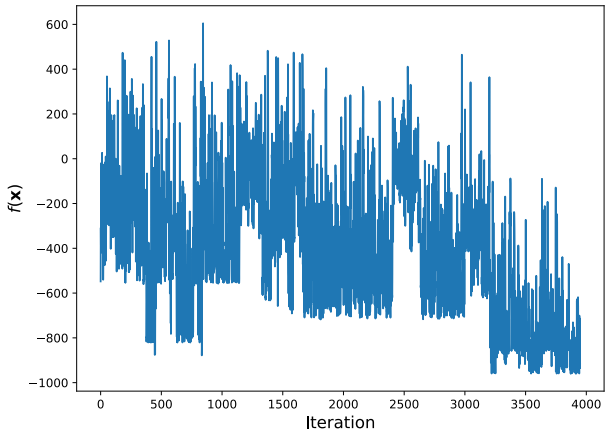
3 Simulated Annealing

Simulated annealing is a trajectory-based optimisation approach in which a single candidate is updated according to a certain rule. The phases of SA can be loosely characterised as 'melting' and 'freezing'. Melting occurs early in the search when the temperature is high. During melting, moves with a positive Δf have a high probability of being accepted and the search resembles a random walk around the feasible region without any constraints.

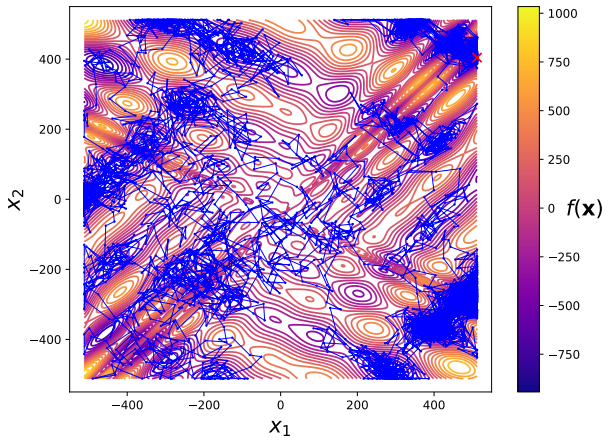
The standard deviation σ of a set of 100 random samples from $f(\underline{x})$ is used to determine the starting temperature T_0 each time SA is run (White, 1984).

The solution is updated using the update rule $\underline{x}_{k+1} = \underline{x}_k + \delta \cdot 512 \cdot \underline{u}$ where $\underline{u} \sim U(-1, 1)$. This is equivalent to using a scaled identity matrix $\boldsymbol{D} = \delta \cdot 512 \cdot \boldsymbol{I}$ as the update matrix. We also explore the Parks method which provides an alternate method of updating the solution (Parks, 1990).

3.1 Visualisation



(a) Objective function minimisation for seed=0



(b) Trajectory evolution for seed=0

Figure 9. Simulated annealing for 2D problem (4000 function evaluations, seed=0, $\delta = 0.125, \alpha = 0.95, L_{min} = 400$)

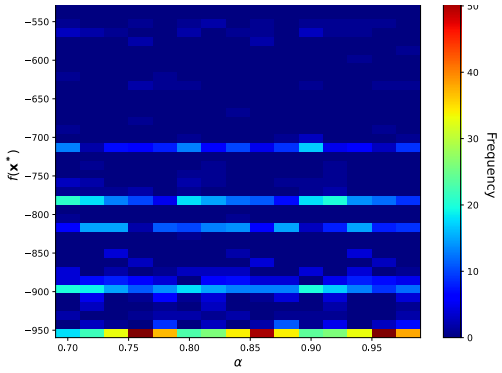
The random walk behaviour explains the disperse samples in Fig 9b and the white noise in Fig 9a. The behaviour of the algorithm changes as it transitions to the 'freezing' phase when it encounters a local minima. In this period, the temperature has been reduced significantly and upward moves with a positive Δf have a low probability of being accepted. The search steps necessarily become smaller therefore tends to converge towards the closest local minimum (as seen in Fig 9b at various points in the trajectory). This final local minimum may not be the best found in the search but the resetting procedure attempts to find a better local minimum which in this case is the global minimum (denoted by $\times$ in Fig 9b).

3.2 Parameter Variation

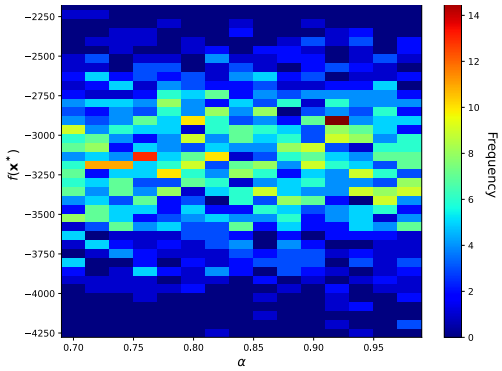
Determining the parameter values prior to variation was extremely difficult for SA since the quality of the solutions is dependent on the combined effect of all the variables. We explored multiple initialisations of the investigated variables before performing single-variable variation and we decided to use $\delta = 0.125, \alpha = 0.95, L_{min} = 400$ for exploration.

3.2.1 Temperature Cooling Factor α

The cooling factor α controls the ratio between the duration of the melting phase and the duration of the freezing phase. Figure 10 shows an optimal value of around 0.95 (as suggested in Kirkpatrick et al. (1983)) for both the 2D and 6D problem since there are a greater number of best solutions less than -4000 as the value of α increases. Since $\alpha < 1$, a value of 0.95 is appropriate and by experimentation, no benefit can be seen for making any changes to this.



(a) 2D Problem



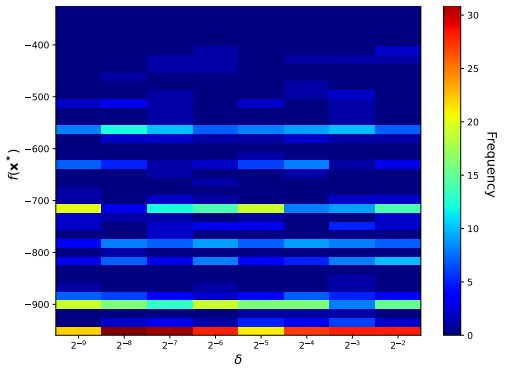
(b) 6D Problem

Figure 10. SA Histogram of solutions with variation of α (100 seeds per value)

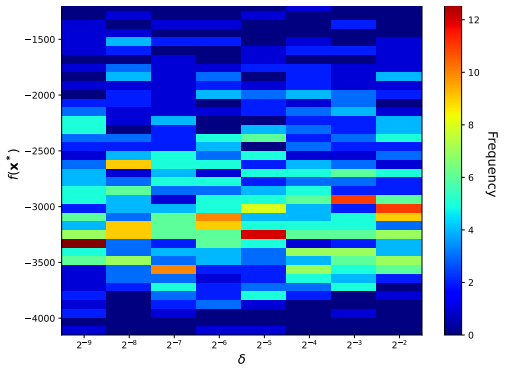
3.2.2 Trajectory Scale δ

The value of δ is a normalised value between 0 and 1 chosen according to some characteristic length scale of the objective function. δ controls the size of the hyper-cube of possible moves. If δ is too small then the feasible region will not be explored sufficiently and will fall into the closest local minima. If δ is too large, the samples will be very dispersed in the melting phase. However, in the freezing phase, the search will hop between local minima, which may be undesirable. Furthermore, when the search has 'frozen' into a local minimum, many samples will be rejected (due to many positive Δf moves).

In our problem, we find that hopping is very useful (due to the presence of many local minima) hence a relatively large length is required. Our characteristic length scale equals 512 since these are the bounds of the problem, hence a $\delta = 0.25$ corresponds to a length-scale of 128 which is quite large. In Figure 11, we investigate a range of δ values along a log base 2 scale to attempt to find an optimal value.



(a) 2D Problem



(b) 6D Problem

Figure 11. SA Histogram of solutions with variation of δ (100 seeds per value)

From Figure 11, it is generally difficult to discern any particular value and the best value for 2D is not necessarily the same as 6D especially for the length-scale. Nonetheless in the 6D case, we see a much larger density around -3600 for $\delta = 2^{-3} = 0.125$ also found experimentally by iteration. A smaller δ tends to result in getting stuck in local minima which we want to avoid. A larger δ can find the global minimum in the 2D case but unlikely in higher dimensions.

3.2.3 Markov Chain Length $L_{\min}$

In the 2D case (Fig 12a), the mean solution plateaus as $L_{\min}$ increases however in the 6D case, there is no clear chain length that provides the best results. $L_{\min} = 950$ is a local minimum but there are plausibly many more such values beyond $L_{\min} = 1000$, therefore the more feasible value $L_{\min} = 450$ is used instead in the below investigation.

3.3 Parks Random Step

We implemented and explored the Parks random step method defined in Parks (1990). Tuning the parameters required for best convergence is extremely difficult and hence we do not achieve a better performance over the standard SA for either the 2D or the 6D problem.

Generally the cost of the Parks computation is also higher but does perform better marginally better with a smaller α and a larger ω in the 2D case (here α is the Parks parameter and not the SA cooling factor). The computation however was extremely slow and hence not tested for the 6D case.

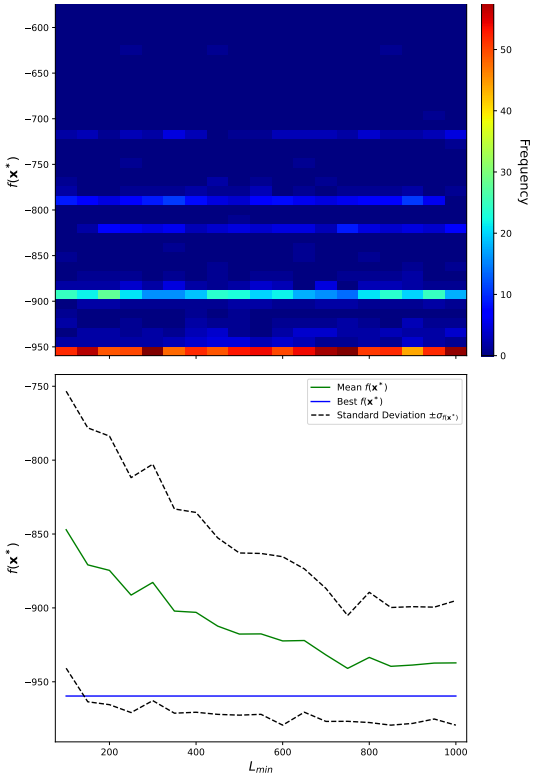
We demonstrate the histogram of best solutions over different seeds in Figure 13.

3.4 Best Solutions

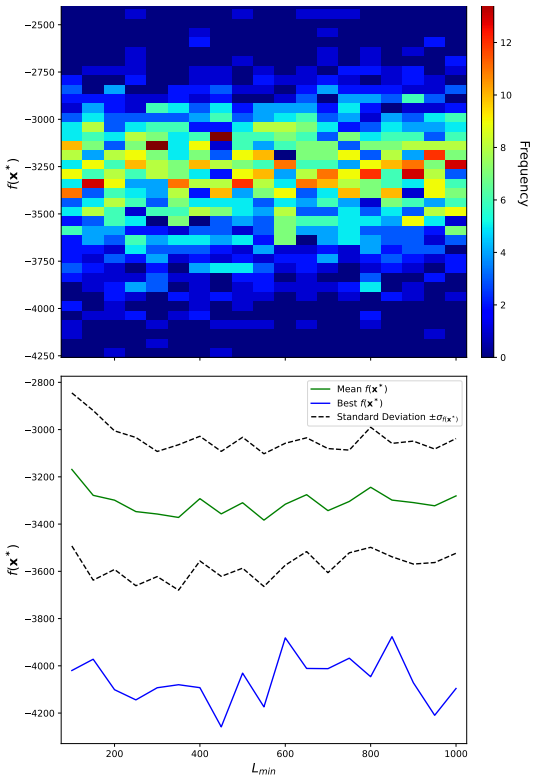
To measure the best performance of the algorithm, we use the investigated parameters to obtain final solutions over different seeds to see how well they converge. The best parameters from our investigations were $\alpha = 0.95$, $\delta = 0.125$ and $L_{min} = 450$ for both 2D and 6D case. We plot the minimisation evolution and the histogram of the final solutions of each seed for both the 2D problem (Fig 14) and the 6D problem (Fig 15).

Figure 14 has many outlier solutions unlike the genetic algorithm hence suggesting that there are other parameters that could be optimised for the 2D case.

The algorithm performs relatively well for the 6D case finding a few minima below -4000 (Fig 15b). The mean solution does eventually reach a plateau but the best solution can and often does continue to decrease with asymptotically larger number of objective function evaluations. The best solution found by simulated annealing has an objective function value of $\mathbf{-4258.4970}$.

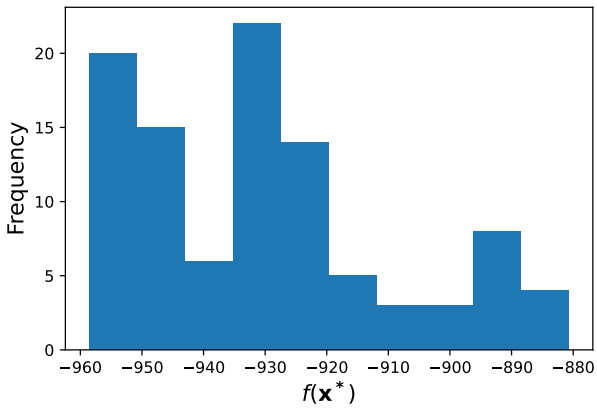


(a) 2D Problem

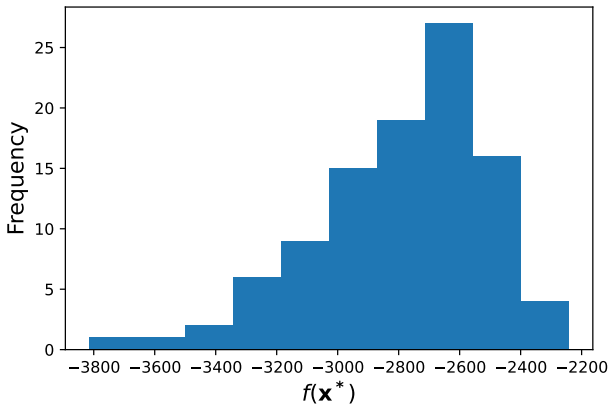


(b) 6D Problem

Figure 12. SA Histogram of solutions with variation of $L_{\min}$ (100 seeds per value)

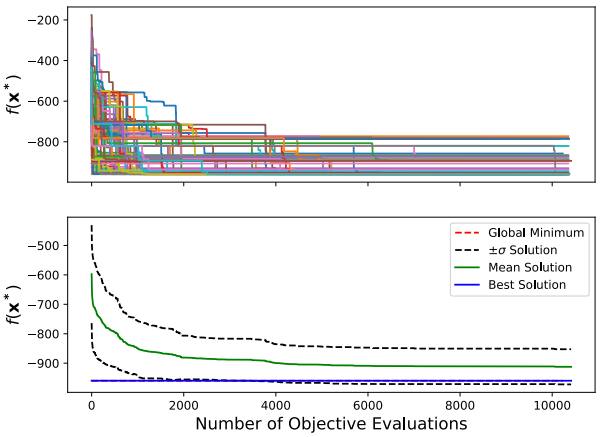


(a) 2D Problem

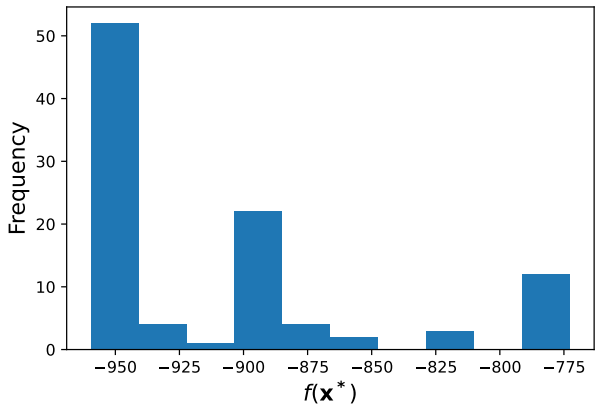


(b) 6D Problem

Figure 13. SA with the Parks method using 100 different seeds (10000 function evaluations/seed)

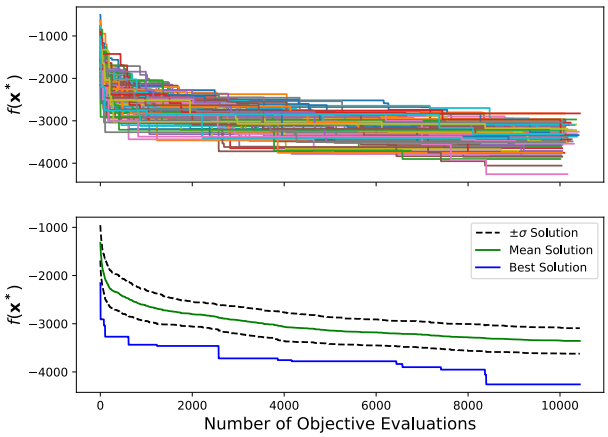


(a) Objective function minimisation for different seeds

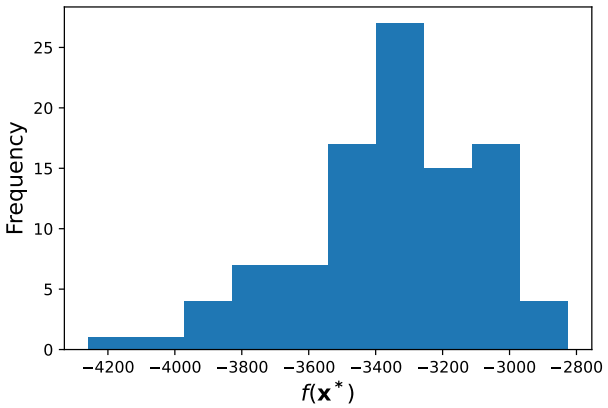


(b) Histogram of final solutions for different seeds

Figure 14. SA for 2D problem using 100 different seeds (10000 function evaluations/seed)



(a) Objective function minimisation for different seeds



(b) Histogram of final solutions for different seeds

Figure 15. SA for 6D problem using 100 different seeds (10000 function evaluations/seed)

4 Conclusion

We explored a genetic algorithm and simulated annealing for both the 2D and 6D eggholder function and we find that optimising the 2D problem is trivial in comparison to solving the 6D problem. The sparsity of random sampling in high dimensional space means that the global minimum is difficult to find. Nonetheless we do find a minimum using the genetic algorithm that could plausibly be the global minimum at a value of $\mathbf{-4457.3618}$. The best local minimum found by simulated annealing has a value of $\mathbf{-4258.4970}$ which although is a good solution, it is still far from the solution found by the genetic algorithm.

The GA histogram for the 6D problem in Figure 8 has a much greater mean solution than the SA histogram in Figure 15 while finding a much better estimate of the global minimum. Hence it is clear that the genetic algorithm performs significantly and with much fewer configurations required. Simulated annealing required a lot of testing and tuning whereas the genetic algorithm performs significantly better just by using an intelligent form of sampling.

4.1 Analysis

The genetic algorithm outperforms simulated annealing by a significant margin. There are many reasons for this, primarily that more candidates are tested out in the genetic algorithm and the samples are more diverse. The SA trajectory often tests points close to the local minima resulting in increased computation with little benefit to reaching the global minimum. Furthermore SA has a greater number of hyperparameters that can be varied and this in itself could be considered an optimisation problem often with no well-defined global minimum. GA generally outperforms SA when sample selection (population-based approach) provides an advantage over sample generation (path/trajectory-based approach). The temperature annealing requires a larger number of iterations and hence becoming costly near local minima.

References

Steve R. White. Concepts of scale in simulated annealing. In *AIP Conference Proceedings*, volume 122, pages 261–270. AIP, 1984. doi: 10.1063/1.34823. URL http://aip.scitation.org/doi/abs/10.1063/1.34823.

Geoffrey Thomas Parks. An intelligent stochastic optimization routine for nuclear fuel cycle design. *Nuclear Technology*, 89(2):233–246, 1990. doi: 10.13182/NT90-A34350. URL https://doi.org/10.13182/NT90-A34350.

Scott Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science (New York, N.Y.)*, 220:671–80, 06 1983. doi: 10.1126/science.220.4598.671.

A Appendix

All Python files are available on Google Drive.

(Overleaf Memory Limit prevents attachment of more code - 3 Python Files - Visualisation, Genetic Algorithm, Simulated Annealing)

```python
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt

def f2(x1,x2):
    total = 0
    a = -(x2+47)*np.sin(np.sqrt(np.abs(x2 + 0.5*x1 + 47)))
    b = - x1*np.sin(np.sqrt(np.abs(x1 - x2 - 47)))
    total += a + b
    return total

x = np.linspace(-512,512,1000)
y = np.linspace(-512,512,1000)
X,Y = np.meshgrid(x, y) # grid of point
Z = f2(X, Y) # evaluation of the function on the grid

fig = plt.figure(figsize=(5,10))

ax = fig.add_subplot(2, 1, 1, projection='3d')
surf = ax.plot_surface(X, Y, Z, rstride=5, cstride=5, cmap=cm.plasma, linewidth=0, antialiased=False)
ax.set_title('Surface Plot of 2D Eggholder Function', fontsize=12)
ax.set_xlabel('$x_1$', fontsize=12)
ax.set_ylabel('$x_2$', fontsize=12)
ax.set_zlabel('$f(\mathbf{x})$', fontsize=12)

ax = fig.add_subplot(2, 1, 2)
ax.set(aspect=1)
ax.contour(X, Y, Z, 20, cmap=cm.plasma)
ax.set_title('Contour Plot of 2D Eggholder Function', fontsize=12)
ax.set_xlabel('$x_1$', fontsize=12)
ax.set_ylabel('$x_2$', fontsize=12, labelpad=-5)

cbar = plt.colorbar(surf, aspect=10)
cbar.set_label('$f(\mathbf{x})$', rotation=0, fontsize=12)

plt.savefig("plot2d.svg")
plt.show()
```

Listing 1. Eggholder Plot - Generate Figure 1