# 1 Introduction

This report investigates the performance of the Simulated Annealing, and Evolutionary Strategy optimisation algorithms, on the minimisation of the *Rana's function*. For each algorithm a preliminary analysis is performed on the 2D *Rana's function*, to confirm that the basis components of the algorithm are working as desired. Following this a rigorous analysis of the algorithms application to the 5D *Rana's function* for a variety of different sub-methods and hyperparameter settings is investigated, with a focus on understanding the effects of the various components of the algorithm, as well as maximising performance. For both algorithms, simple variations to the standard steps were proposed, and shown to improve performance. Finally an overall comparison is performed between the 2 algorithms, using random search as a baseline. All code was written from scratch, and is available in the Appendix.

# 2 The Problem: *Rana's function*

The optimisation problem this report deals with is the minimisation of *Rana's function* (Whitley et al. 1996), defined by

$$\begin{aligned} &\text{Minimize} \\ &f(\mathbf{x}) = \sum_{i=1}^{n-1} x_i \cos\left(\sqrt{|x_{i+1} + x_i + 1|}\right) \sin\left(\sqrt{|x_{i+1} - x_i + 1|}\right) + (1 + x_{i+1}) \cos\left(\sqrt{|x_{i+1} - x_i + 1|}\right) \sin\left(\sqrt{|x_{i+1} + x_i + 1|}\right) \\ &\text{subject to} \quad x_i \in [-500, 500] \text{ for } i = 1, \ldots, n \end{aligned} \tag{1}$$

The 2 dimensional form of *Rana's function* is shown in Figure 1, shows that the problem has many local minima - making it a difficult optimisation problem. To find the best solution the following 2 elements are required: (1) a thorough exploration of the space, in order to find promising local minima, and (2) finding the lowest zone within a promising local minima. (1) and (2) have to be balanced, as they require the algorithm to perform different functions, where (1) requires testing many solutions far apart in control parameter space, and (2) requires a focuses search within a narrow zone of control parameter space. This trade-off is referred to as exploration-exploitation trade-off, and is a common theme throughout this report. Because the *Rana's function* has an especially large number of local minima, it requires an especially strong amount of exploration.

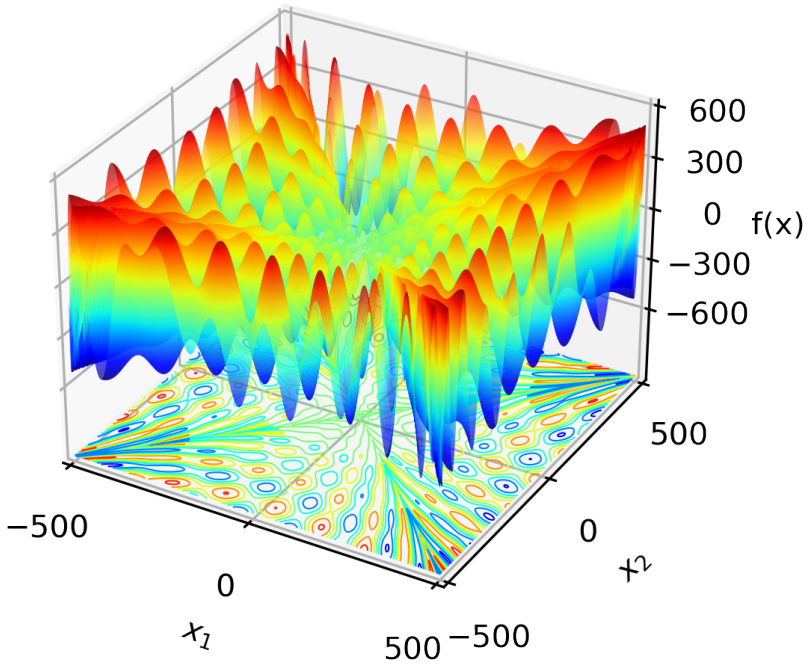


Figure 1: *Rana's function* 2D

# 3 Simulated Annealing

## 3.1 Implementation details

For each component of the Simulated Annealing algorithm the following methods were implemented, noting any deviations from the lecture slides by inserting a "$\star$" at the start of where they are mentioned.

**Annealing schedule:** Simple exponential cooling, Kirkpatrick initialisation (Kirkpatrick 1984) **Control variable perturbation methods**: (1) Constant spherical noise (`Simple-step`), (2) full adaptive covariance step matrix (referred to as `Cholesky-step`, as the Cholesky decomposition is utilised),(3) adaptive diagonal covariance step matrix (`Diagonal-step`).
$\star$ Both adaptive control variable perturbation methods are prevented from having a determinant below $10^{-16}$ to prevent the step size, and each element of the step size matrix was clipped to have it's absolute value within 10% of the control parameter range, to prevent the step size from becoming too small or to large in any direction, making the program more stable.
$\star$ Significant experimentation with non-standard rules for **when** the updates to the step size control matrix occur was performed - as described in Section 3.3.2.
**Convergence criteria:** Absolute difference in accepted objective function $<= 1e-6$ for last last 1000 objective function evaluations. Maximum 10000 iterations. **Bound enforcement:** Repeated sampling of perturbation until control variable within bounds.

## 3.2 Model Baseline on 2D Rana function

To demonstrate that the Simulated Annealing algorithm is working as desired: (1) the basis components of the algorithm ( solution generation, solution acceptance, temperature scheduling) need to be shown to be working, (2)

the algorithm has to demonstrate both exploration of the space, and exploitation of the most promising local zones of the space (exploration-exploitation trade-off).

For simulated annealing to work as desired, in the initial iterations the algorithm needs to focus on exploring as much of the space as possible, while towards the end of run, instead performing a focused search of the best "zone".

In this section a preliminary presentation of the simulated annealing optimisation algorithm is given on the 2D Rana's function, to confirm that the algorithms are working correctly, and to demonstrate the search patterns corresponding to each of the control parameter step methodologies.

Figure 2 shows the temperature and corresponding probability of acceptances for generated solutions, over the course of the program. As the temperature decreases, the probability of accepting solutions that increase the objective function become lower - focusing the optimisation on a narrower band of space. The effect of the temperature on the accepted solutions is shown in Figure 3, where the variance in the accepted objection function values is initially high - with many acceptances of objective values that were higher than the previous iteration, as the iterations continue and the temperature is annealed, both the running mean and variance of the accepted objective values generally decreases until convergence is reached. Convergence is conservatively defined as less than $10^{-8}$ of improvement over the last 1000 iterations. For the `Simple-step` method the step size remains fixed throughout the run, as is shown by the constant width of generated function evaluations in Figure 3. The solution archives (Figure 5) show that most (but not all) of the local min have been explored by the `Simple-step` method based Simulated Annealing program.

Further analysis and discussion of each of the control parameter step methods is given in Section , however the 2D problem is useful for a visual representation each of the search patterns for each of the step size covariance methods. Figure 6 shows the accepted solutions throughout the course of a run using the `Diagonal-covariance` method. Early in the run, the the standard deviations of each element are high, and the temperature is high, so the accepted solutions are scattered throughout the space. Later in the run (bottom left hand plot), the accepted solutions have lower variance with respect to x2 elements relative to x1. The algorithm also converges far faster - and by 2000 iterations there is virtually no variance of the accepted solutions (while with the `Simple-step` method there was still significant variance at 4000 iterations). This demonstrates the differences that the `Diagonal-covariance` method introduces, namely (1) the ability to greatly reduce the overall step size throughout the optimisation, narrowing the space of proposed solutions (not just narrowing the space of accepted solutions via temperature cooling) and (2) having a different variance for different elements of x.

Figure 7 shows the progression of accepted solutions throughout the course of a run. Early in the run the additional noise from the step size covariance matrix is large and the temperature is high so the accepted solutions are scattered throughout the space. Later in the run, the accepted solutions are confined in a narrow space with a very high correlation between each of the elements of x (in the bottom left hand plot of Figure 7, the solutions appear in diagonal line). The narrowness of the space later on in the run is a function of both the lower temperature, and the lower overall noise of the sampling (from the determinant of covariance matrix decreasing). The high correlation of between elements of x show that the off-diagonal elements of the covariance in the `Cholesky-covariance` method are playing a significant role in the search pattern - contrasting the pattern from the `Diagonal-covariance` method.

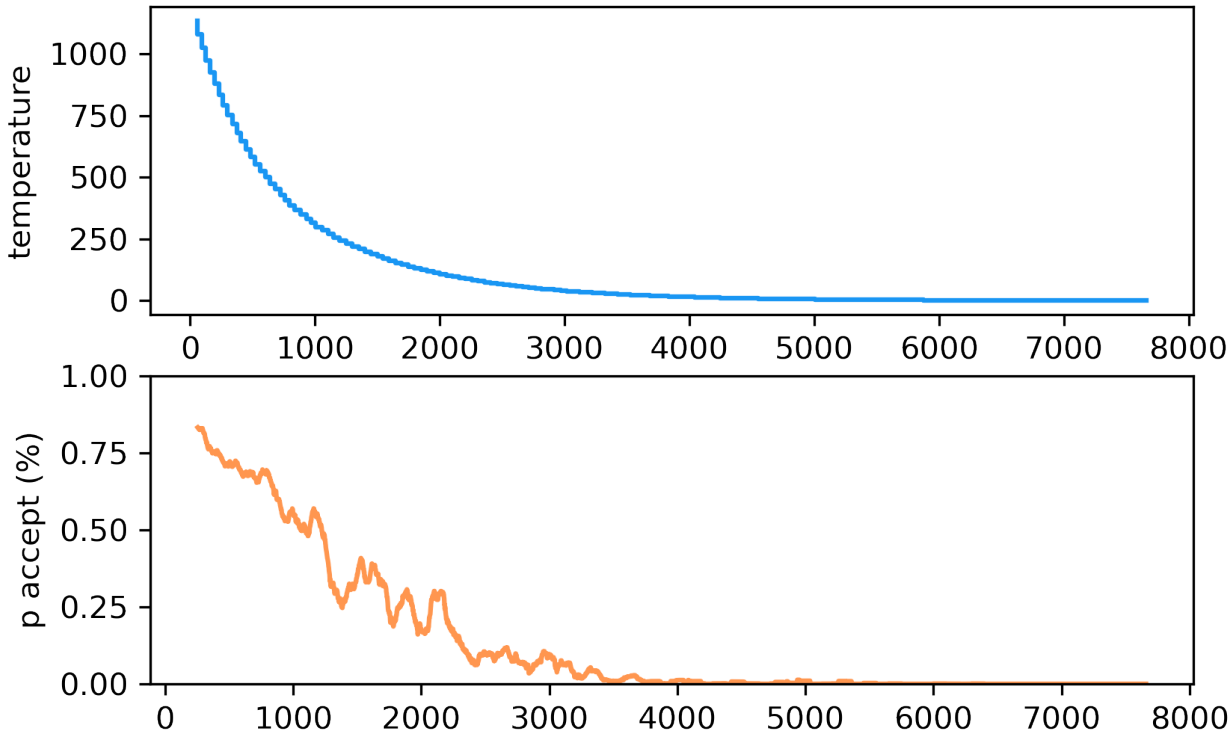


Figure 2: Temperature annealing and probability of acceptance

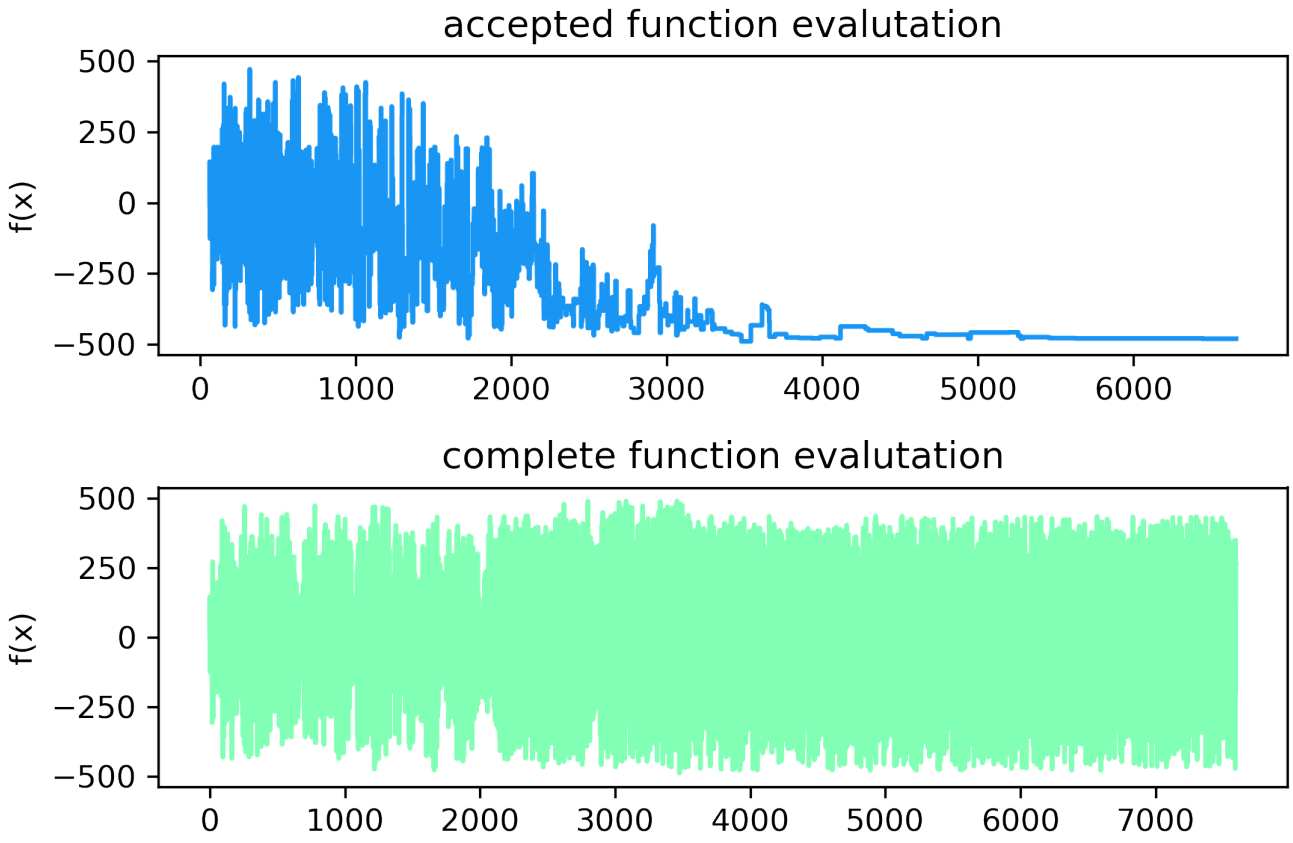


Figure 3: Performance history

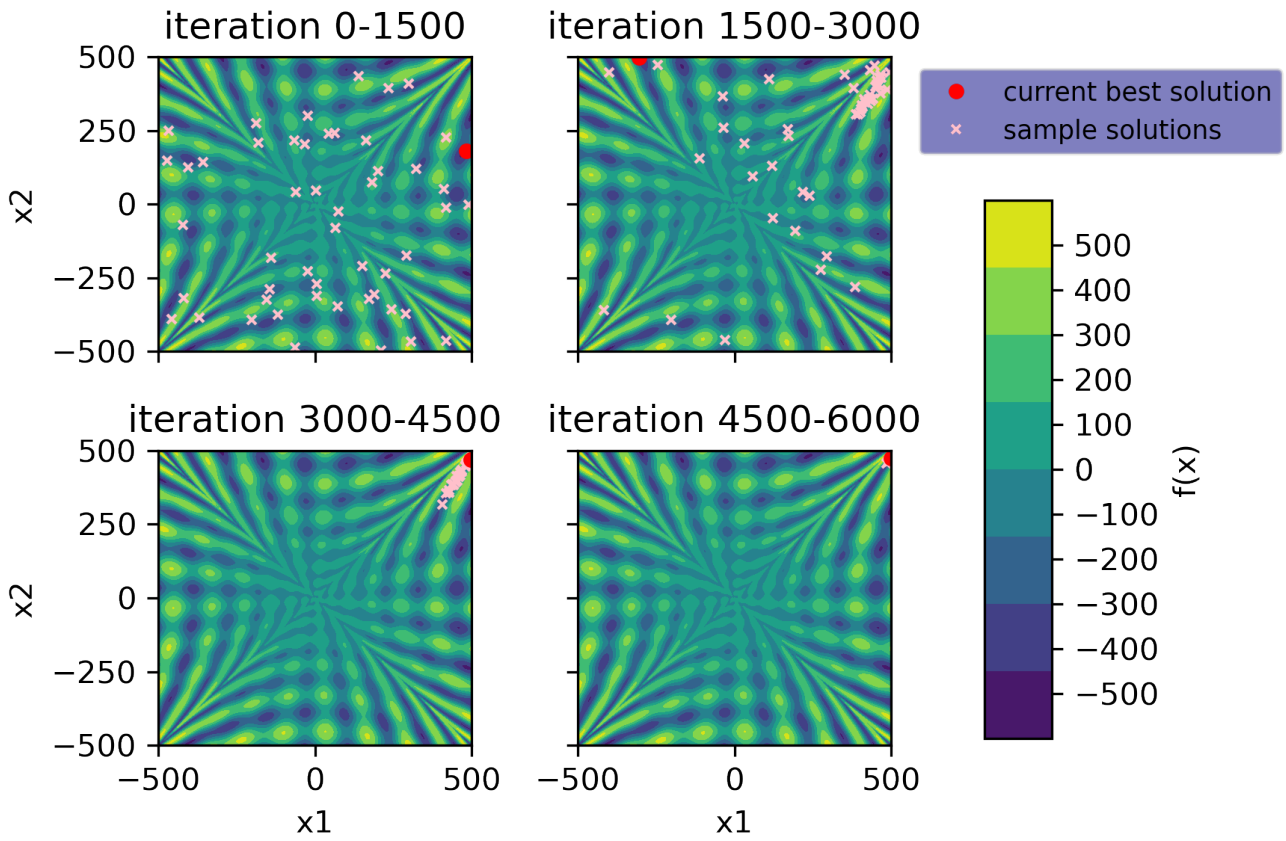


Figure 4: Sample solutions throughout optimisation. Early in the optimisation the whole space is explored. As the optimisation progresses, a progressively smaller subset of the space is focused on

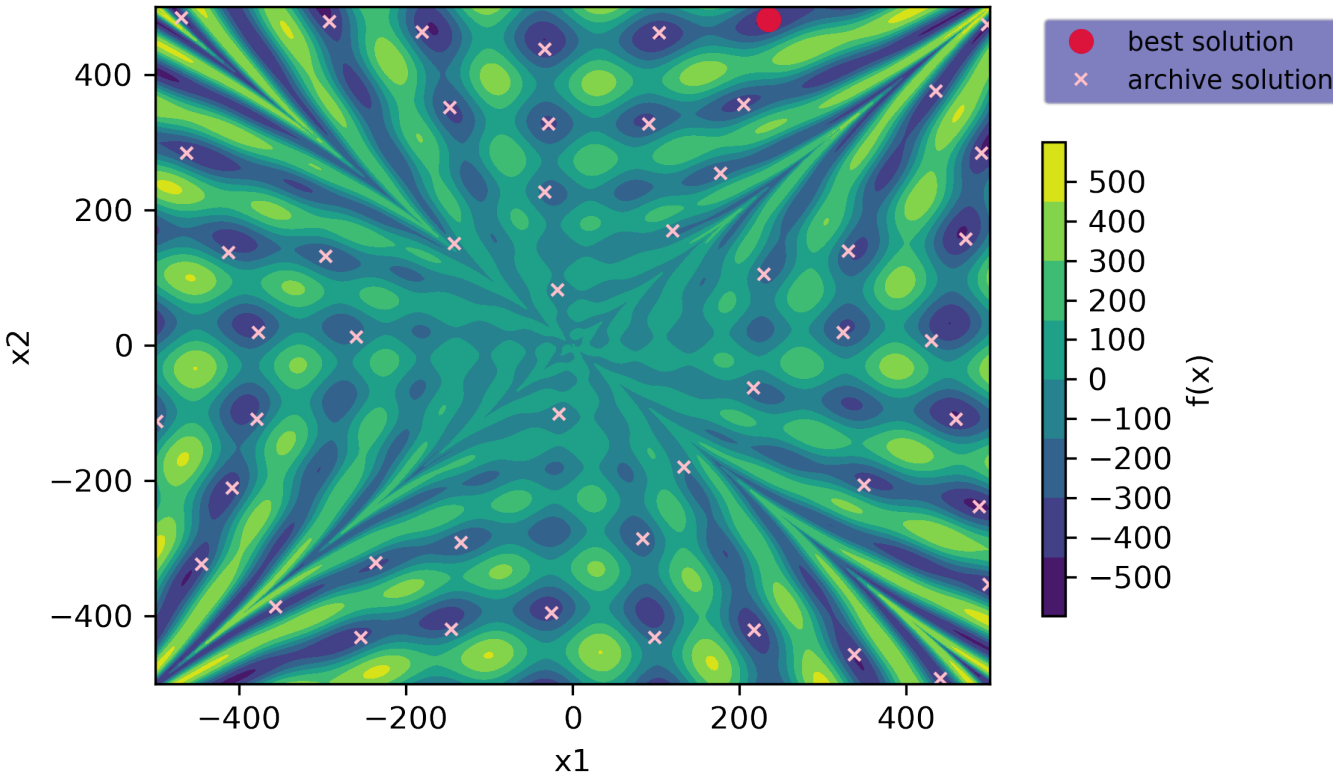


Figure 5: Archives from optimisation. The archive solutions are distributed throughout the space, indicating that the space is thoroughly explored

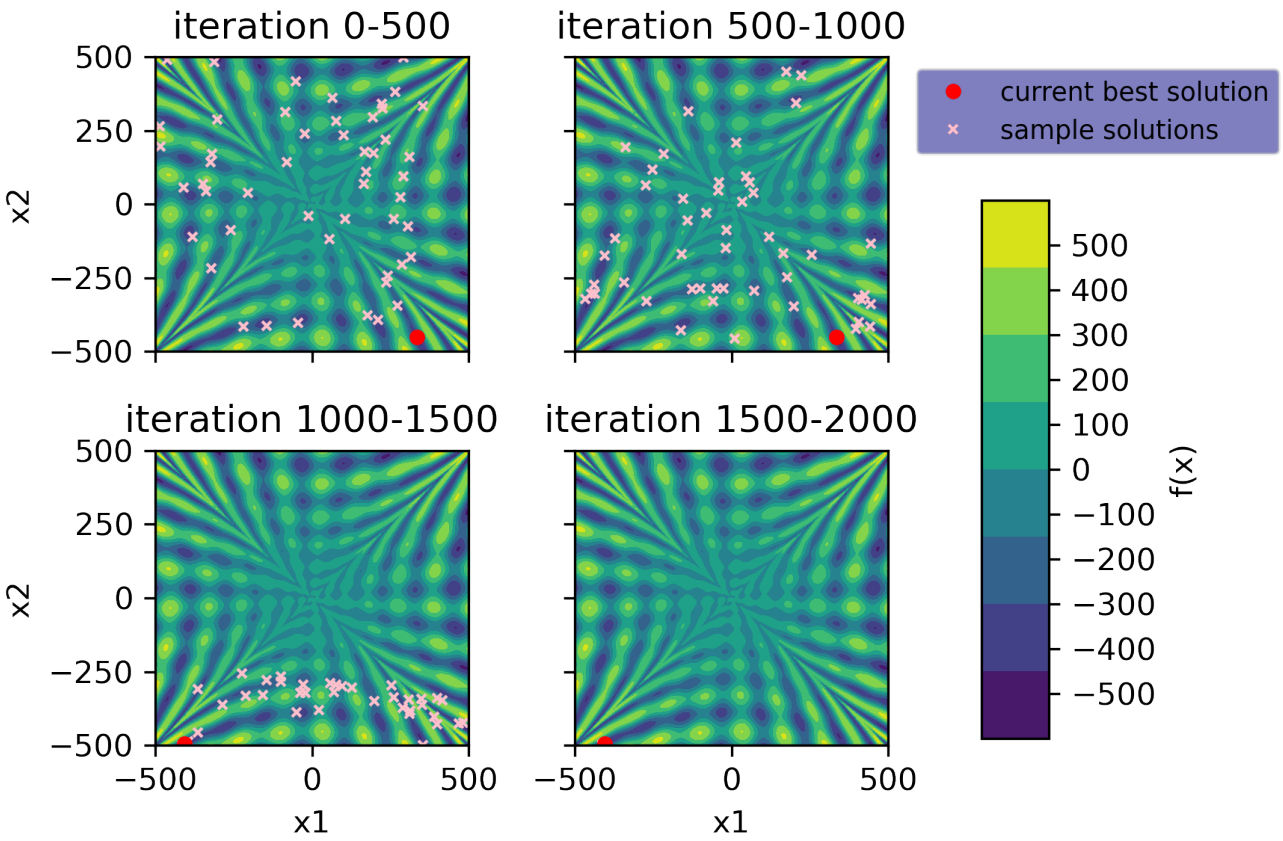


Figure 6: Sample solutions throughout optimisation for Diagonal step method

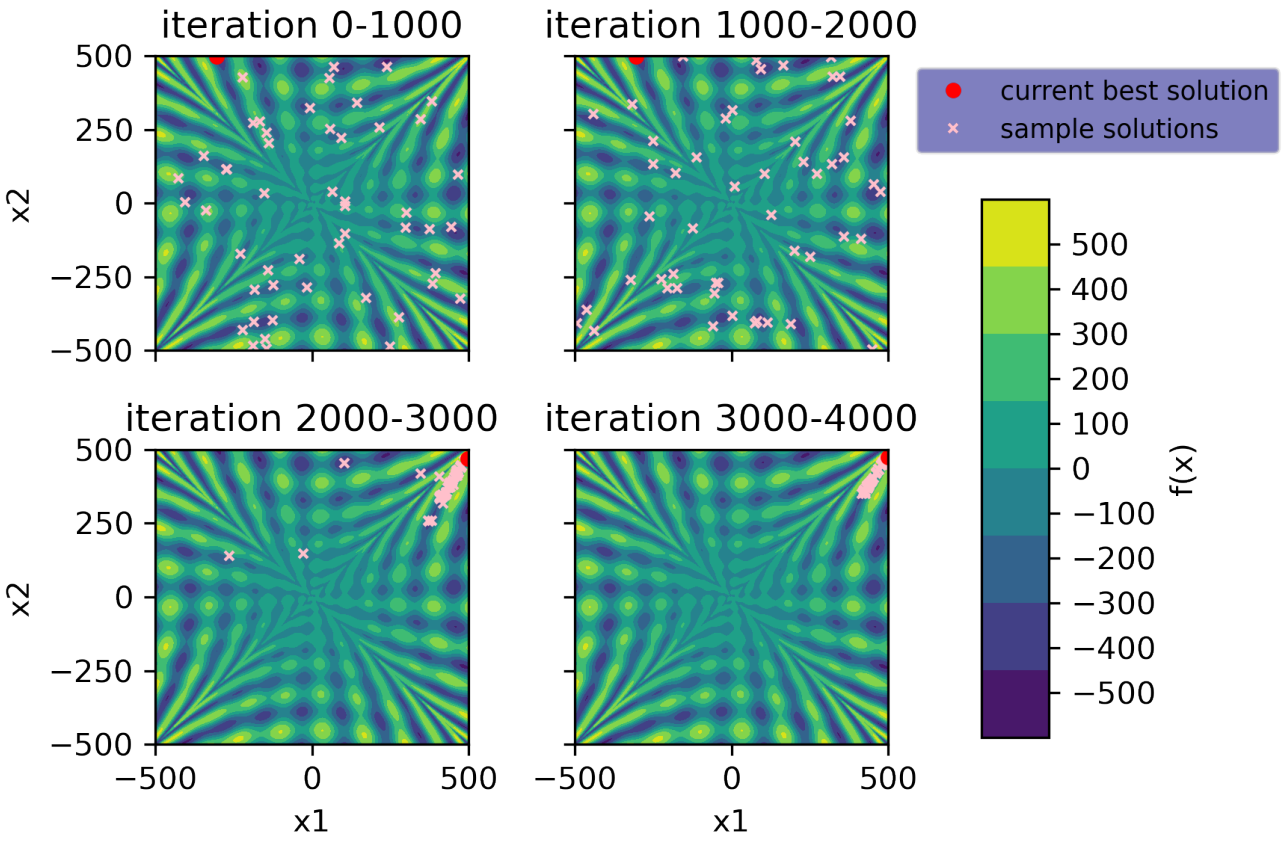


Figure 7: Sample solutions throughout optimisation for Cholesky step method

## 3.3 Analysis on 5D Rana function

In this section of the report, the effects of (1) each of the step-methods (simple, diagonal, cholesky), (2) the step-updating rule (a proposed addition to the SA algorithm) and (3) the temperature annealing schedule hyperparameters (maximum markov chain length, alpha) are investigated. In this section of the report, each effect (1,2,3) is initially discussed and analysed separately. Because these changes to the algorithm have interacting effects, they cannot be optimised individuall. Therefore a simple grid optimisation of combinations of each method and hyperparameters is performed. Finally, using the tuned hyperparameters, the overall performance of the Simulated Annealing algorithm, with the different step-size-updating rule and step-methods are compared.

### 3.3.1 Step Method

Various sub-plot describing the performance of the `Diagonal step method` are given in Figure 8. The `Diagonal step method` allows the standard deviations along each element of the control parameters to adapt over the course of the run - allowing the algorithm to focus it's search along some dimensions more than others. This is visible in the bottom subplot of Figure 8 where each of the 5 standard deviations varies. Correspondingly, the variance in the generated function evaluations visibly varies (e.g. becomes wider as the standard deviation of the purple line becomes large). The `Diagonal step` method has an additional advantage, which is in that it allows for the acceptance probability to be adjusted to include division by the actual step size, which scales the probability of accepting perturbations according to how large the step size was (with large step sizes having increased probability of acceptance) - thus encouraging exploration.

Various sub-plots describing the performance of the `Cholesky step method` for a single example run, is shown in Figure 9 below. To demonstrate the effects of having a full covariance matrix (as opposed to the `Diagonal step method`): (1) the magnitude of the eigenvalues of the covariance are plotted - indicating the magnitude of the variance along each eigenvector of the covariance matrix and (2) the minimum angle between the eigenvector corresponding to the largest eigenvalue and each of the 5 axis is plotted in order to give an indication of how much covariance between different elements of the control parameters there is.
The covariance's eigenvectors decrease throughout the program, lowers the amount of variance in x, and correspondingly in the generated objective function values (Figure **??** subplot 2 and 3). The minimum angle between the axes, and the eigenvector corresponding to the largest eigenvalue is significant throughout the course of the optimisation - this indicates that there is a significant correlation between different elements of x (i.e the non-diagonal elements of the covariance are having an effect).

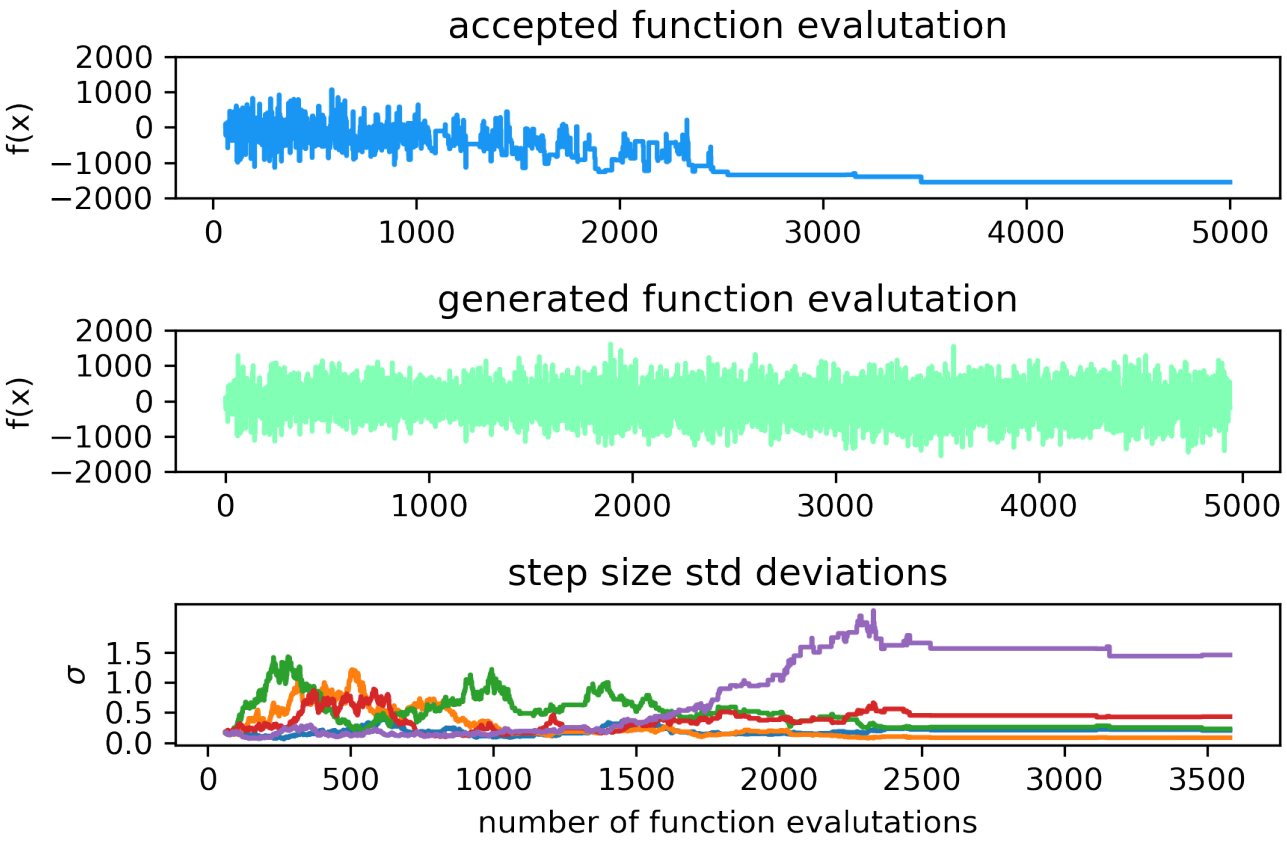


Figure 8: Diagonal step-method example performance

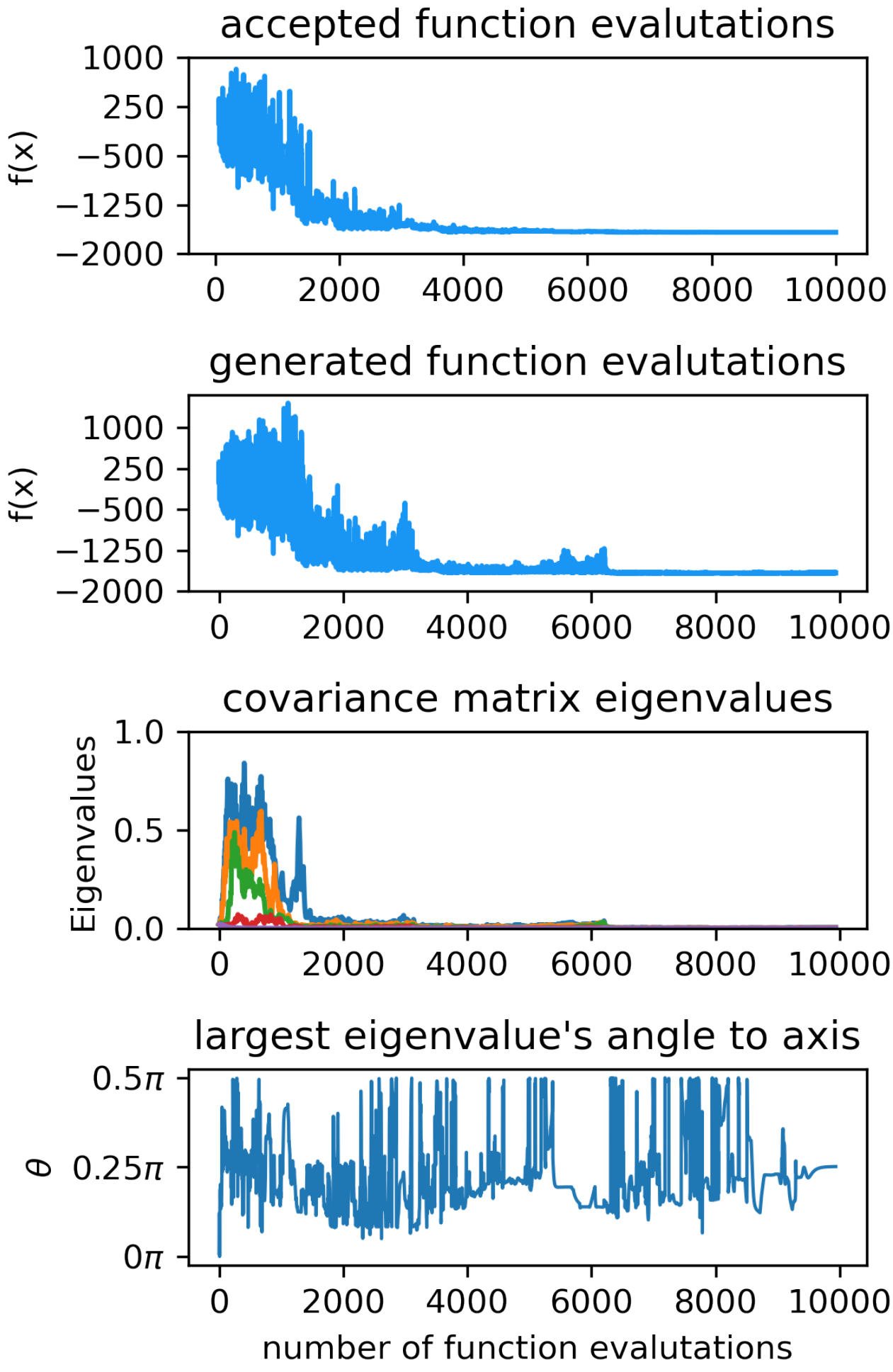




Figure 9: Cholesky step-method example performance

### 3.3.2 Step updating rule

For both of the adaptive step size methods the following issue was often observed: If the step size is only updated when new values are accepted (as specified in the [ref slides as well as original), then if a reasonably good solution is found while the temperature is relatively cool and the step size is relatively large, then new solutions are almost always rejected (as the step size is too big) but because the step size is only updated when new solutions are accepted, the step size doesn't get updated, causing the program to get stuck at the current solution. I.e. there is a catch 22 where new solutions aren't accepted because the step size is too big, and the step size is not updated as there are no new solutions being accepted. An example of such a situation is shown in Figure 10, where after 2500 updates, for a good new solution to be discovered (such that it is accepted), the step size needs to be reduced, however because there are no new solutions that are good enough are discovered (because the step size is too big), the step size remains fixed and there is no improvement for the rest of the run

This issue can be dealt with by instead updating the step size every time a perturbation is accepted **and** and at an interval (e.g. every 5 steps) when the perturbation is not accepted. The reason for updating the step size is to adjust the covariance matrix to better fit the local topology - and this addition to the algorithm allows for information on the local topology to be folded into the step size matrix at a rate corresponding to both the number of acceptances and the number of steps total steps - preventing over dependence of the rate of acceptances for the step size to match the local topology. This is referred to as the `diversified-step-update-rule` for the rest of the report.

The effect of the size of the interval between step updates using the proposed rule in shown in Figure 11. For an interval of 10000, this method becomes identical to the original step-size update rule of only updating the step-size-matrix after acceptances - and Figure 11 therefore shows that this rule has inferior performance and that the proposed rule for updating the step size has clear benefits. A step-update-interval of 4 and 45 are preferred for the Diagonal and Cholesky covariance methods respectively using an initial configuration for the other hyperparameter settings[1].

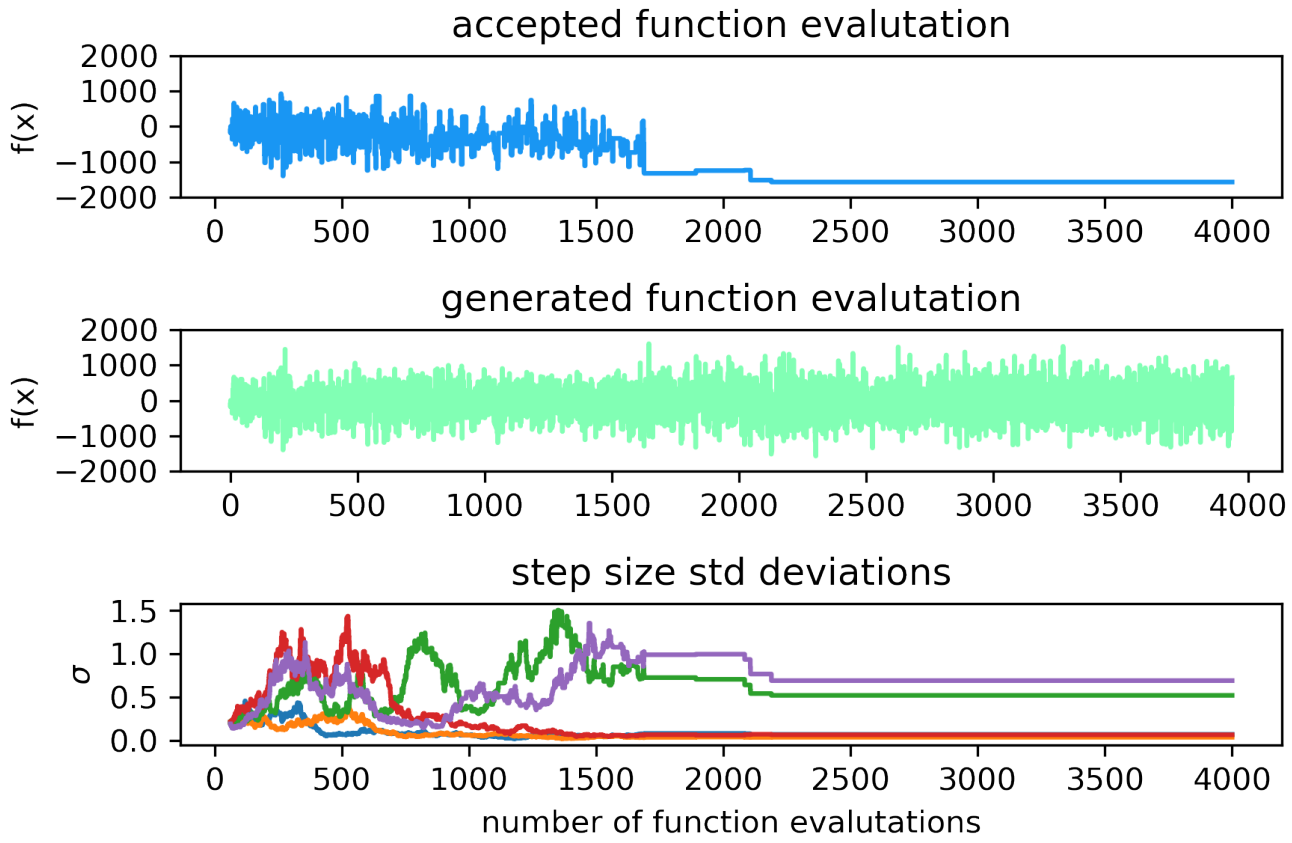


Figure 10: Illustration of issue where algorithm gets stuck, due to dependency on acceptences for step size updates

[1] hyperparameter setting of markov-chain length = 50, annealing alpha = 0.95

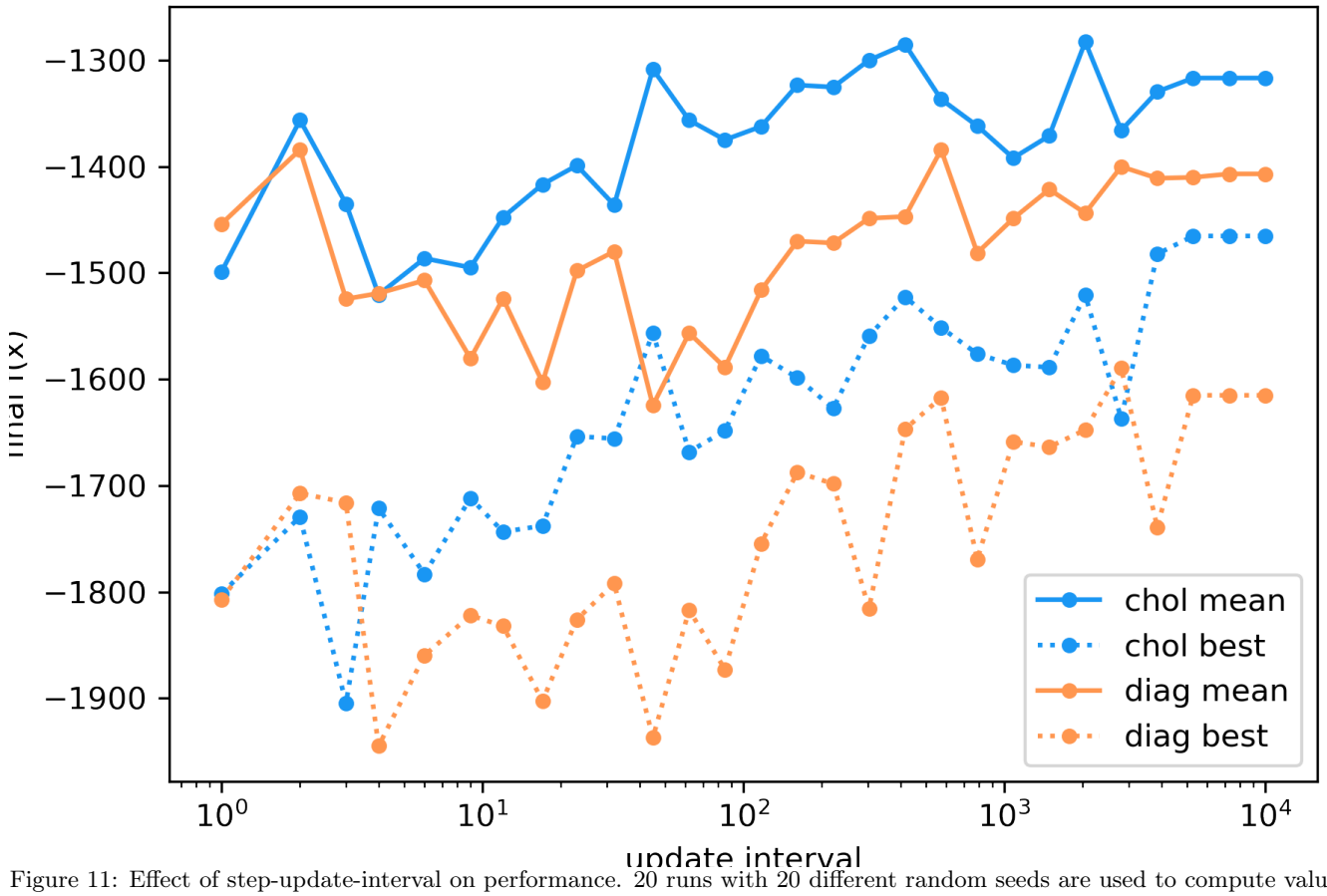


Figure 11: Effect of step-update-interval on performance. 20 runs with 20 different random seeds are used to compute values at each point

### 3.3.3 Markov Chain length and Alpha Parameter Optimisation

---

Maximum markov chain length and alpha have both control the temperature schedule over the course of simulated annealing. Markov chain controls the number of temperature steps, while alpha controls the size of the steps. If the markov chain length is large, then there are less total steps during the run, and thus the total temperature is reduced less. If alpha is small, the size of each temperature reduction step is large, and the temperature is reduced more over the run. This effect is shown in Figure 12 Having a well calibrated temperature schedule is important for the optimisation, as temperature controls the exploration exploitation trade-off of the program. Early in the program the temperature has to be set sufficiently high in order for the space to be explored, while later in the program the temperature has to be sufficiently low for the "best zone" to be narrow on. If the temperature scheduler reduces the temperature too slow or fast, then the program will not explore or exploit.

An example of the effects of maximum markov chain length and alpha on performance for Diagonal-method and the `diversified-step-update-rule` are shown in Figure 13. Because both of these parameters effect the temperature schedule, there is rough equivalence in performance between certain schedules, for example (high MC length, low alpha) and (low MC length, high alpha) correspond to similar explore/exploit levels - this can be seen by the downwards sloping bands of colour in the contour plot. The average runtime is lowest for low values of both alpha and maximum markov chain length, as these lead to the system temperature becoming low quickly, resulting in convergence in lower number of steps.

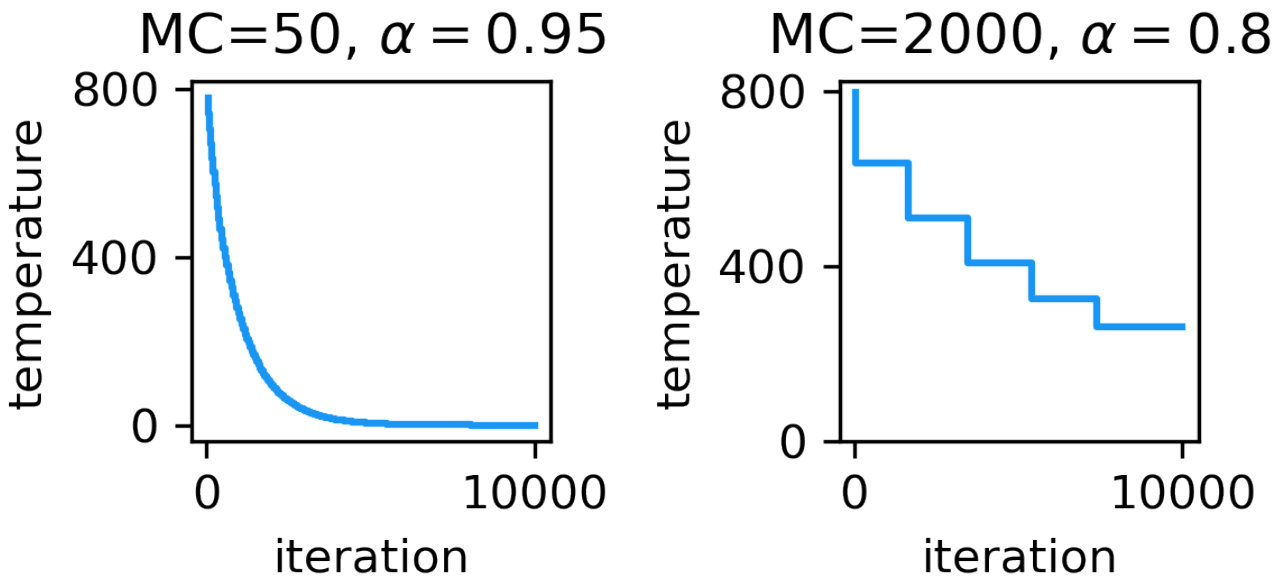


Figure 12: Effect of alpha and markov chain length (MC) on annealing schedule

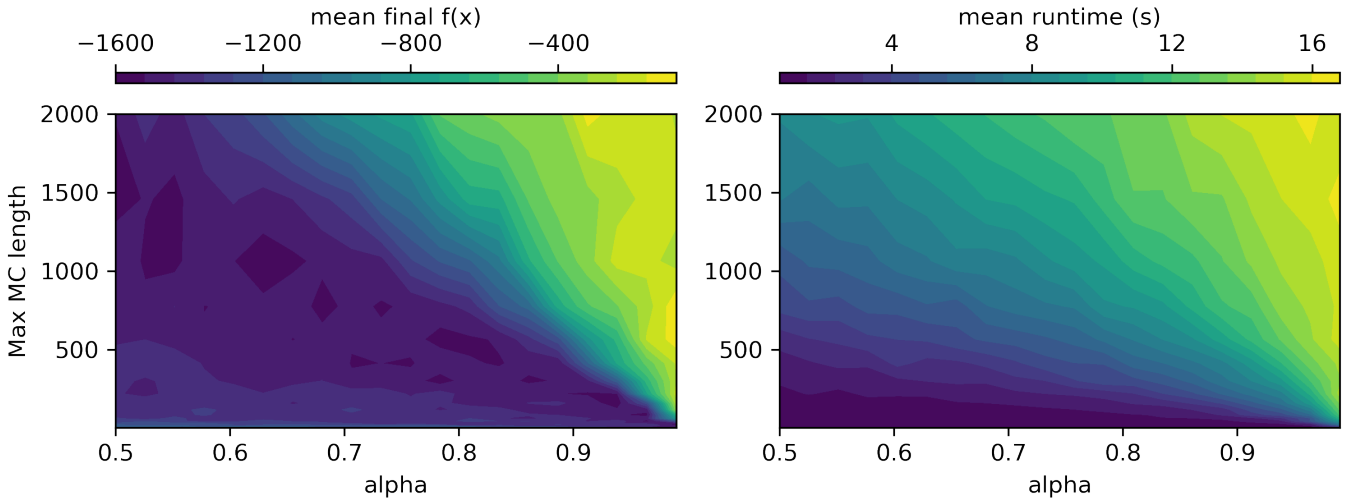


Figure 13: Contour plot for grid search of maximum markov chain length, and alpha value, for Diagonal-step-method and `diversified-step-update-rule` (interval of 20). 30 runs with 30 different random seeds are used of estimation of mean objective value and mean time

### 3.3.4 Results with tuned hyperparameters

---

The `diversified-step-update-rule` has clear benefits, significantly improving both the `Cholesky step` and `Diagonal step` method's mean objective function by over 1000. Both the `Cholesky step` and `Diagonal step` methods outperformed the `Simple step`, showing the utility of the adaptive step size matrix. The `Diagonal step` method had the best performance, both in terms of final value of the function, and in terms of runtime. Because the Rana function encourages exploration strongly, the ability of the `Cholesky step` method to more richly match the local topology was did not give it a strong advantage over the `Diagonal step` method. Furthermore, the `Diagonal step` method included useful adjustments of the acceptance probability, incorporating the step size to encourage exploration. The `Diagonal step` method outperforms the `Cholesky step` in terms of runtime because it has less computationally expensive operations (i.e doesn't have to calculate Cholesky decomposition). Although the `Diagonal step` method requires more computation per step of the algorithm, it was also able to outperform the `Simple step` method, by converging faster, and therefore requiring less steps.

Table 1: Results for different Simulated Annealing performance for different step-methods, with and without the diversified-step-update-rule, using tuned hyperparemeters (maximum markov chain length, alpha, and diversified-step-update-rule interval (when used). Performance metrics calculated over 30 runs with 30 different random seeds

| **step-method** | **Cholesky** | | **Diagonal** | | **Simple** |
|---|---|---|---|---|---|
| **diversified-step-update-rule interval** | - | 14 | - | 20 | - |
| **max markov chain length** | 776 | 776 | 1064 | 9 | 776 |
| **annealing alpha** | 0.77 | 0.5 | 0.55 | 0.99 | 0.61 |
| **mean final performance** | -1409.06 | -1578.82 | -1551.19 | -1666.23 | -1481.48 |
| **std dev final performance** | 116.75 | 156.9 | 144.45 | 129.67 | 157.47 |
| **average runtime (s)** | 16.13 | 7.98 | 11.4 | 7.23 | 12.13 |

# 4 Evolutionary strategies

## 4.1 Implementation details

For each component of the evolutionary strategies the following methods were implemented, noting any deviations from the lecture slides by inserting a "$\star$" at the start of where they are mentioned.

**Mutation methods:**

1. $\star$ `simple mutation`: Spherical Gaussian noise (non-adaptive)
2. `full covariance mutation`: Gaussian noise sampled from a covariance matrix formed with standard deviation and rotation angle strategy parameters. To ensure positive-definiteness, a small diagonal matrix is added repeatedly until the covariance is positive definite. $\star$ Off-diagonal elements of the covariance matrix are clipped to have their absolute values bounded by the minimum of the diagonal elements corresponding to the row and column number (e.g. element in row 2, column 3 is clipped to have it's absolute value bounded by the minimum of diagonal element 2 and diagonal element 3) - as this is a condition that covariance matrices hold, and helps encourage positive definiteness in a less computationally expensive manner than the original method of positive definiteness enforcement which sometimes require repeated operations (the original method is still used in conjunction with the clipping). $\star$ Standard deviations clipped to be above $10^{-6}$ and below 1 to prevent mutation becoming too large or too small.
3. $\star$ `diagonal covariance mutation`: Same as `full covariance mutation` but with standard deviation strategy parameters only (i.e with rotation angles fixed at 0). Standard deviations clipped to be above $10^{-6}$ and below 1 to prevent mutation becoming too large or too small.

**Recombination methods:** Global discrete recombination for control parameters. Global intermediate recombination on strategy parameters. **Selection methods:** $\mu, \lambda$ and $\mu + \lambda$ **Convergence criteria:** Absolute difference in objective function of parents $<= 1e-6$. Maximum 10000 iterations. **Bound enforcement:** Repeated sampling of mutation until offspring within bounds.

## 4.2 Model Baseline on 2D Rana function

To show that the Evolution Strategy Algorithm is working as desired, an initial exploratory analysis is performed using `simple mutation` and $\mu, \lambda$ selection[2], on the 2D Rana function, as it allows for visualisation of the search patterns. The key areas that are indicative of desired performance are (1) minimisation of the objective function (2) balance of explore-exploit tradeoff (3) visibility of the components of the Evolution Strategy Algorithm: Mutation, Selection and Recombination.

Figure 14 shows a sample run, in which the objective function is successfully minimised (final value -491). Mutation and crossing over are the sources of variation of objective values - and their cumulativie effects can be seen by the varying mean objective values of the offspring across time. As only the best children are selected as parents, the mean objective of the parents (per generation) is significantly below the mean objective of the offspring - Figure 14 is therefore also indicative of the selection working as desired. Because `simple mutationn` is used, the parents don't reach the convergence criterion (as they have non-negligible variance) even though functionally the algorithm

[2]Configuration: 10 parents, 70 offspring, standard devation of each x element = 0.01

has converged by 20 generations (the other mutation methods did in practice converge on the 2D rana problem).

The plot of the archived solutions (Figure 15) are reasonably spread throughout the space - indicating a reasonable amount of exploration has taken place. This is confirmed in the search pattern plot (Figure 16), where early in the run the solutions are spread throughout the space. The effect of selection is clear, with solutions at higher areas of the terrain becoming less prevalent as the program continues, as they are removed from the gene pool. The search pattern also shows that later in the algorithm, there is a strong focus on exploitation of the best solutions (with only 2 and 1 local zones focused on by generation 7, and 10 respectively). The symmetry of generation 4 along each axis is most likely a result of cross-over (e.g. as it allows for values of each element of various solutions to "swap", causing symmetry). Lastly, the effects of mutation are clear in the search patterns, where there are many values close (but distinctly separate) bunched at local minima.

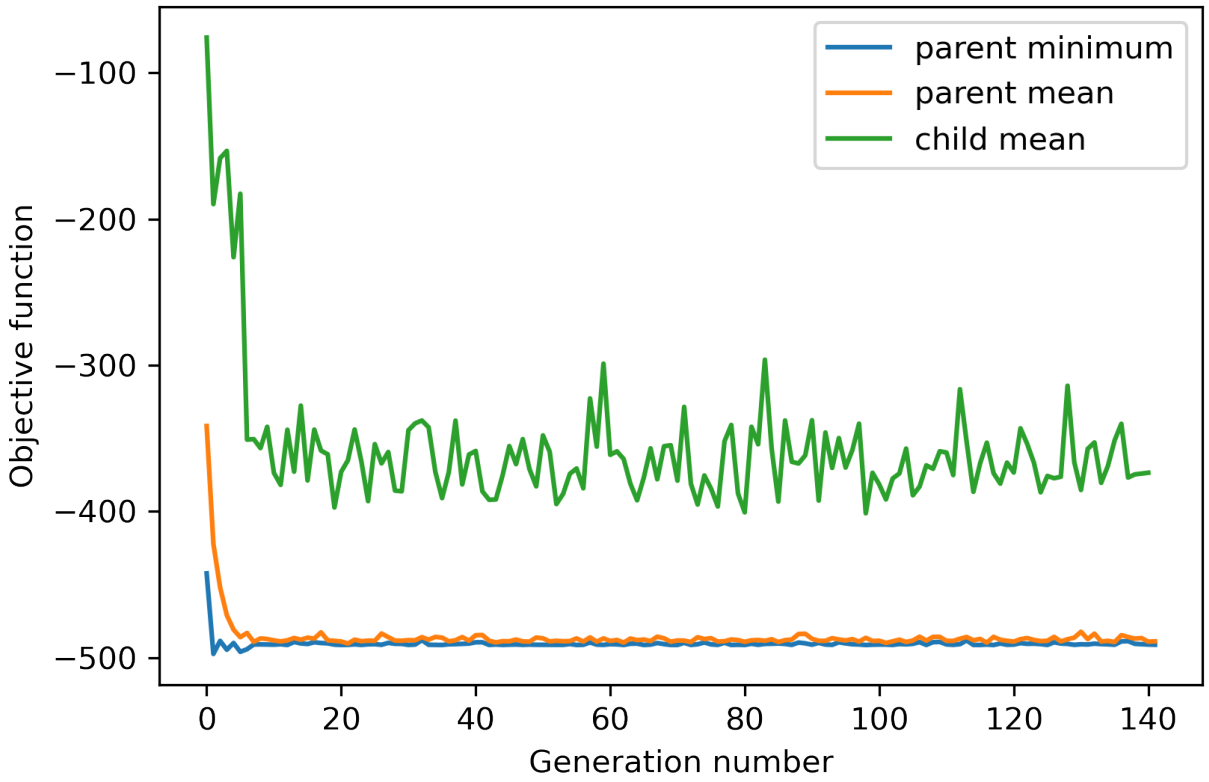


Figure 14: Evolution strategies method example performance on 2D rana function. `simple mutation`, $\mu, \lambda$ selection

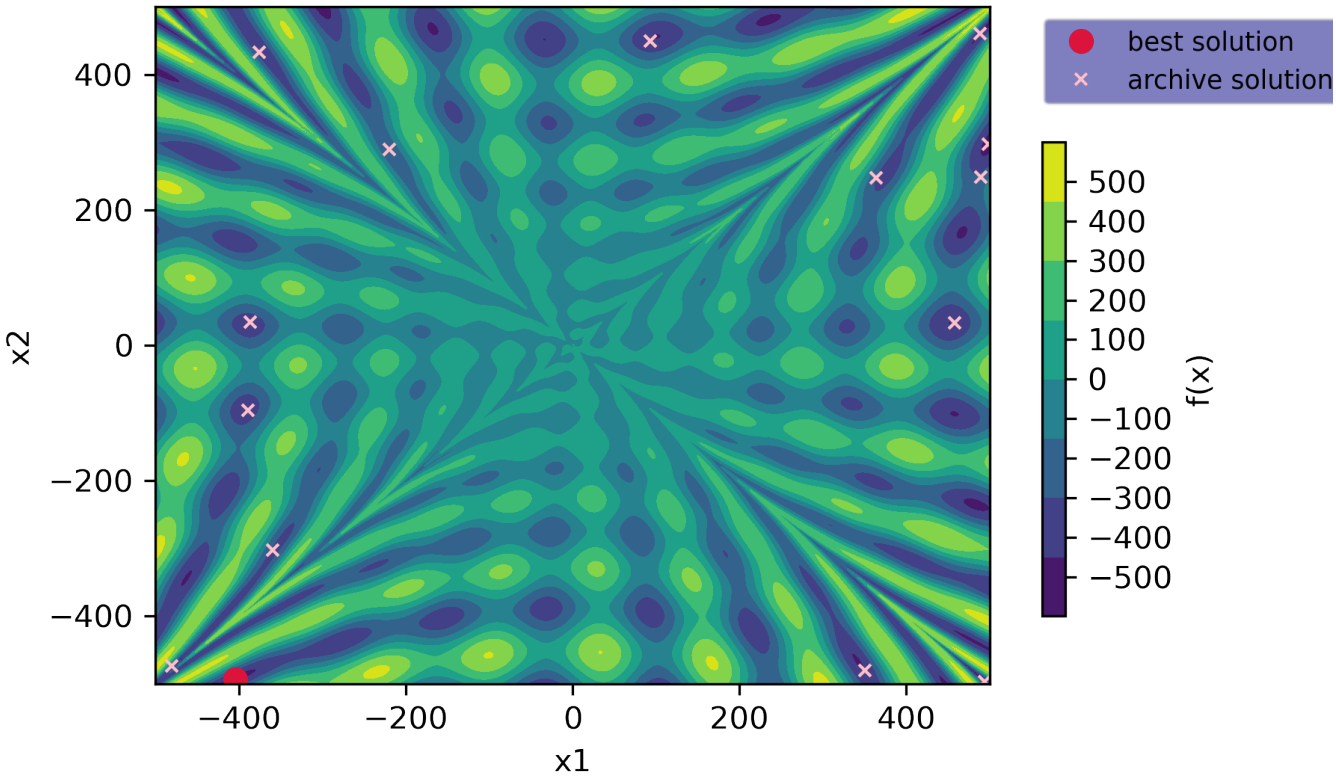


Figure 15: Evolution strategies method archive example from run on 2D rana function. `simple mutation`, $\mu, \lambda$ selection

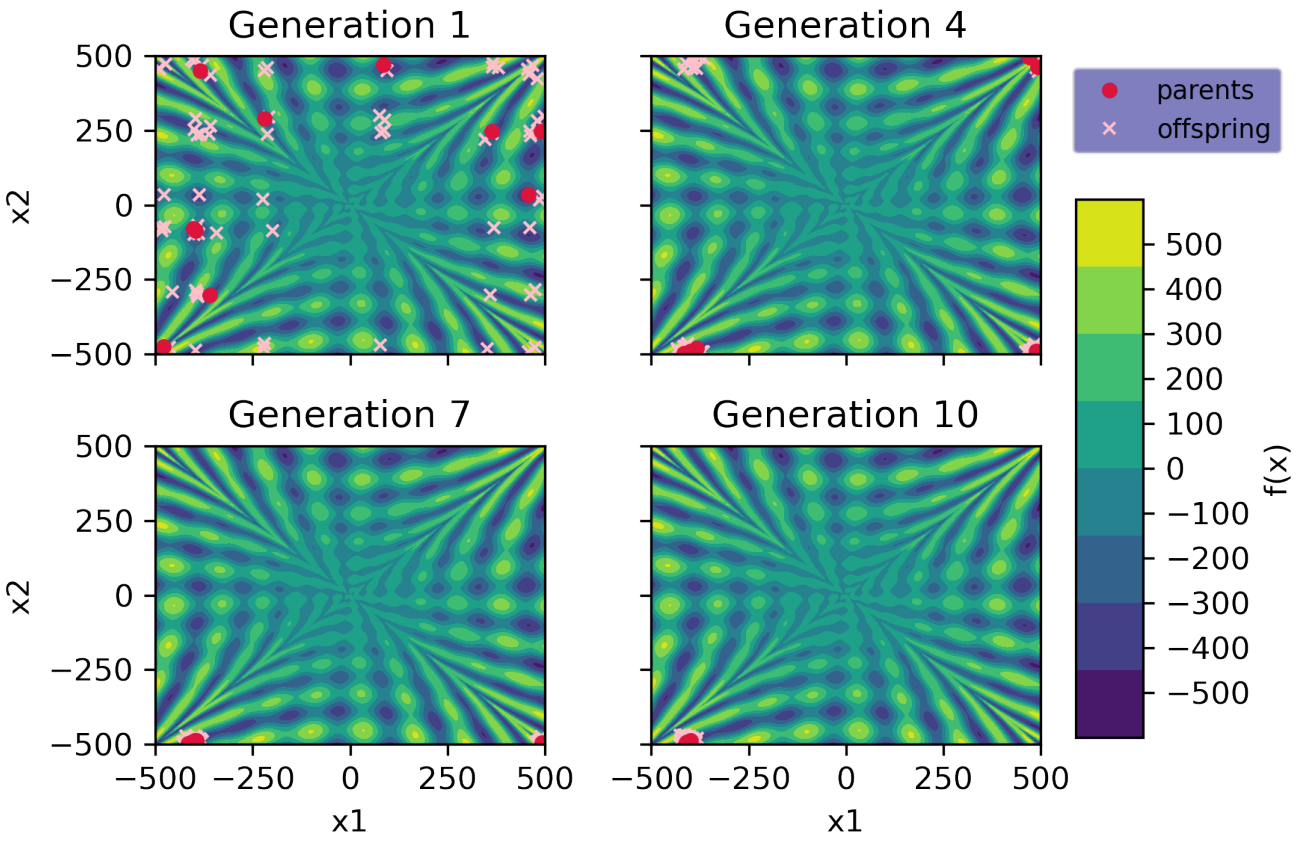


Figure 16: Evolution strategies method example search pattern on 2D rana function. `simple mutation`, $\mu, \lambda$ selection

## 4.3 Analysis on 5D Rana function

In this section, an initial presentation of the various methods for selection ($\mu, \lambda$-selection and $\mu + \lambda$-selection), and mutations (`simple mutation`, `diagonal covariance mutation`, `full covariance mutation`) is given, with an emphasis on explaining their effects (performance comparisons are rather done after hyperparameter optimisation). The effects of the hyperparameters controlling the population dynamics are explored, and optimised (by grid search) for each combination of the selection and mutation methods. Finally, a comparison of the methods with the optimal hyperparameters is given.

### 4.3.1 Mutation method

In evolution, dynamic or stressfull environments favour high levels of mutation, as useful adaptions are important for survival. On the other hand, stationary environments favour lower levels favour lower levels of mutation, as the parents typically are well suited to the environment, so children similar to the parents tend to have the highest fitness [[cite]]. This has a direct parallel to the Evolutionary strategy algorithm, where the current values of the control parameters determines the "environment" (i.e. environment represented by location on control parameter landscape). Initially the control variables are in random locations that are relatively high in the landscape. As the run continues the algorithm will move significantly lower in the landscape (removing offspring similar to the original parents from the gene pool in the process, and thus the "environment" is initially dynamic, favouring high levels of mutation. Towards the end of the program, the parents are typically located in a good local minima, and change between generations becomes very small (as it is hard to find a lower point in the space). Thus later in the optimisation, the "environment" is stable, and lower levels of mutation are preferred. The adaptive covariance methods are therefore useful as they allow the amount of mutation to adapt to the stage of the otpimisation program. Furthermore, the adaptability of the magnitude mutation not only in total, but also with respect to specific elements of the control parameters (or with respect to linear combinations of control parameters in the case of `full covariance mutation`), allowing for higher levels of mutation along "directions" that are more promising.

The determinant of the covariance matrix is indicative of the total "amount" of mutation - and in Figure 18 shows that the smaller mutation-covariance determinant leads to a smaller amount of variation in the offspring.

[**could maybe speak more about pros and cons of each method?**]

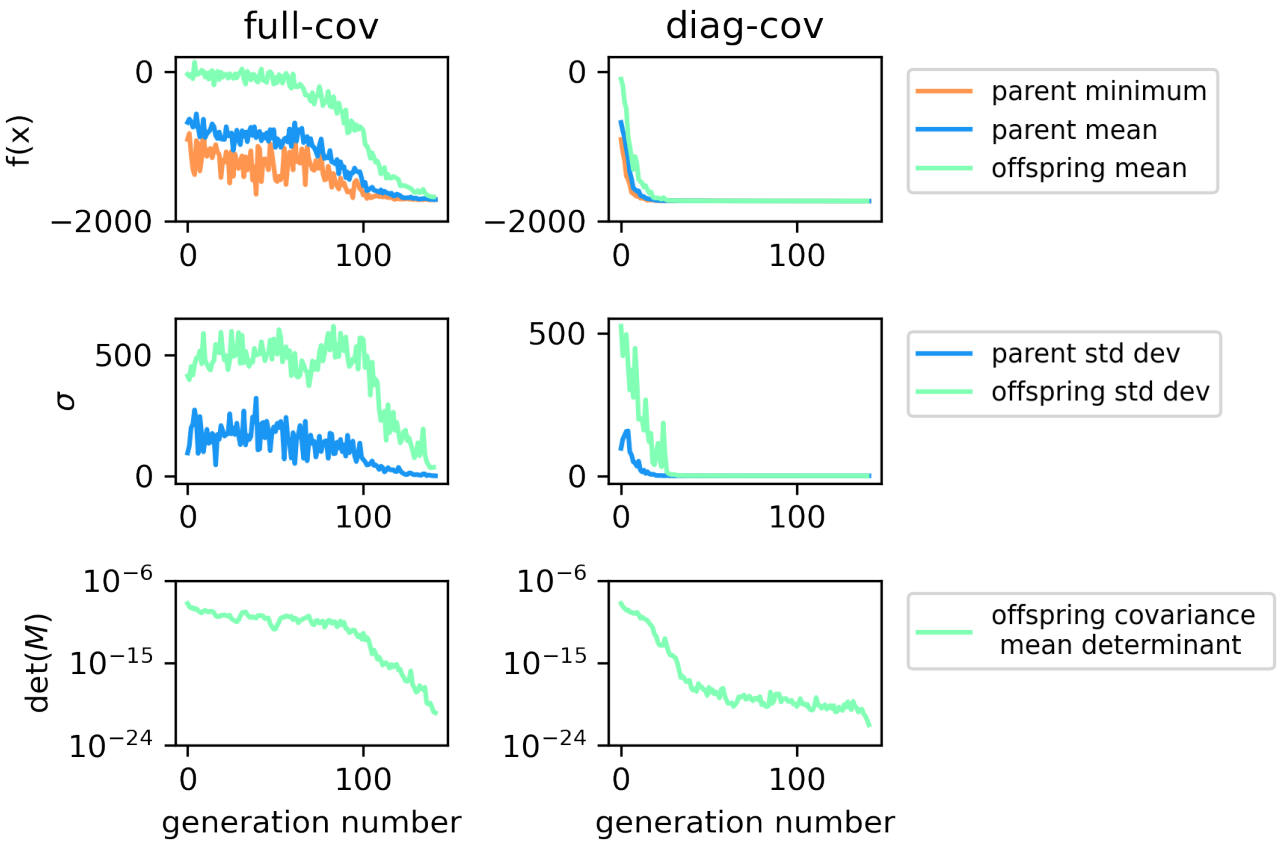


Figure 17: Performance of Evolution Strategy algorithm with `full covariance mutation` and `diagonal covariance mutation`

#### 4.3.2 Selection Method

---

$\mu, \lambda$-selection is more reflective of evolution by natural selection (in which the parents of each generation die) and has the benefit of being better suited to "dynamic"[3] environments (more new solutions are tried each step, giving a greater probability of useful new adaption). $\mu + \lambda$-selection has the advantage of never losing the "best" current solution, guaranteeing that the objective function will stay the same of decrease each generation - however it results in less exploration taking place, as some of the new offspring in each generation are often merely clones of the previous parent generation. In Figure 18, which shows an example run of both selection methods, the minimum parent of the $\mu, \lambda$-selection often increases, where with $\mu, \lambda$-selection the minimum parent objective is always lower or the same as the previous generation. In the given example run, the $\mu, \lambda$-selection has a better final objective value, due to its superior exploration of the space.

[3] dynamic evolutionary environments defined in Section 4.3.1

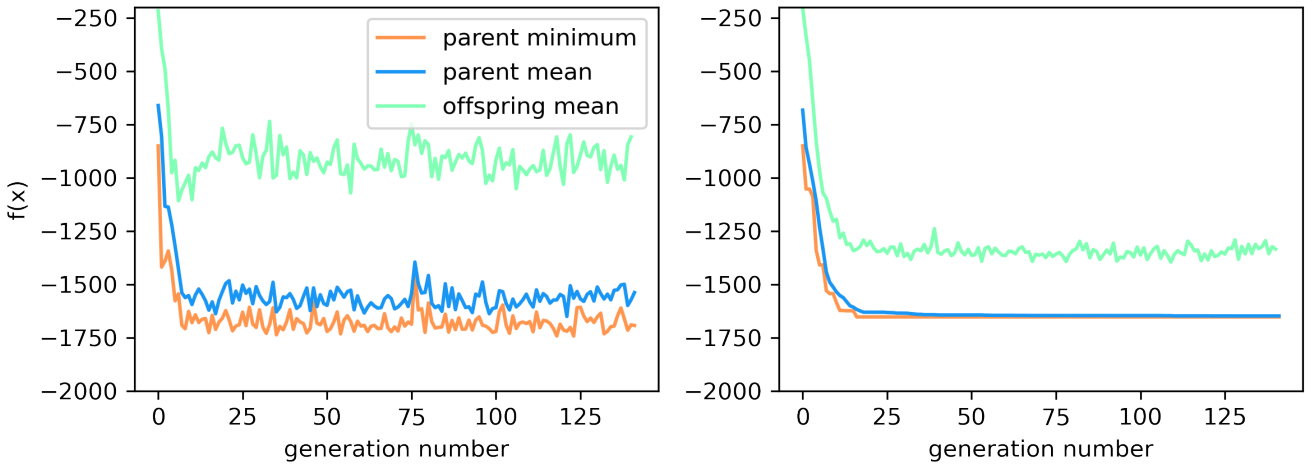


Figure 18: Performance of Evolution Strategy algorithm with $\mu, \lambda$-selection and $\mu + \lambda$-selection

### 4.3.3 Population dynamics

---

The child to parent ratio controls the selective pressure on the system. If the child to parent ratio is high, because only a small fraction of children survive each generation, the selective pressure is high (only the very fittest survive), while if the child to parent ratio is relatively low, then many offspring survive and the selective pressure is relatively low. This relates the exploration exploitation trade-off - higher selective pressure causes a larger focus on narrow searches in the current best local optima (**exploitation**), while lower selective pressure allows for a greater **exploration** of the space. At the extreme of a 1-1 parent to child ratio, the algorithm becomes a random walk ("pure" exploration).

The effect of child-to-parent ratio was explored by holding the number of offspring at 100, and changing the number of parents per generation - as this allows for the number of generations, and the amount of mutation per generation to be held constant.

The effect of child to parent ratio is demonstrated in Figure 19. The fixed-diagonal Gaussian mutation was used for this figure - as this prevents interaction effects between the mutation method and the offspring-to-parent-ratio, so allows for a simpler isolation of the effects of offspring-to-parent-ratio. In the left hand side plot, the child-to-parent-ratio is relatively low (2:1) and the following trends are clear (1) the decrease in the parents' average objective function over generations is relatively slow - from lower selection pressure creating less downwards force on the parents objectives (some parents with only mildly low objective functions are accepted each iteration). (2) The variance of the parent population is relatively high - because there are more parents, it is more likely that a parent near a different local minima to the current best will be selected. Conversely, in the right had plot the child-to-parent-ratio is high (20:1) (1) the parents' average objective function decreases very rapidly (and then plateus) (only the lowest objective values are selected, so the objective function quickly decreases). (2) the variance in the parents' objective functions quickly becomes very low, as the high selective pressure causes the parents to bunch around a single local optimal. Overall it is clear that the low child-to-parent ratio focuses more on exploration, and the high child to parent ratio of exploitation.

Holding the parent to child ratio and the maximum number of function evaluations constant, the number of population size (both parents and offspring) controls the exploration-exploitation in a different way. If the number of offspring is large then there is a lower number of total generations (as the 1000 function evaluations are "consumed" at a higher rate per generation) and if the number of offspring is small then there are many generations. As early generations focus more of exploration, and later generations on explotation, if most of the children are contained in early generations (if there are many offspring), there is a greater focus on exploration (and correspondingly less children -> more exploitation). Another way this has an effect is through the number of parents - if there are many parents, then a higher number of promising local zones are past from generation to generation. Thus for a large space with many local minima, a larger population is preferred.

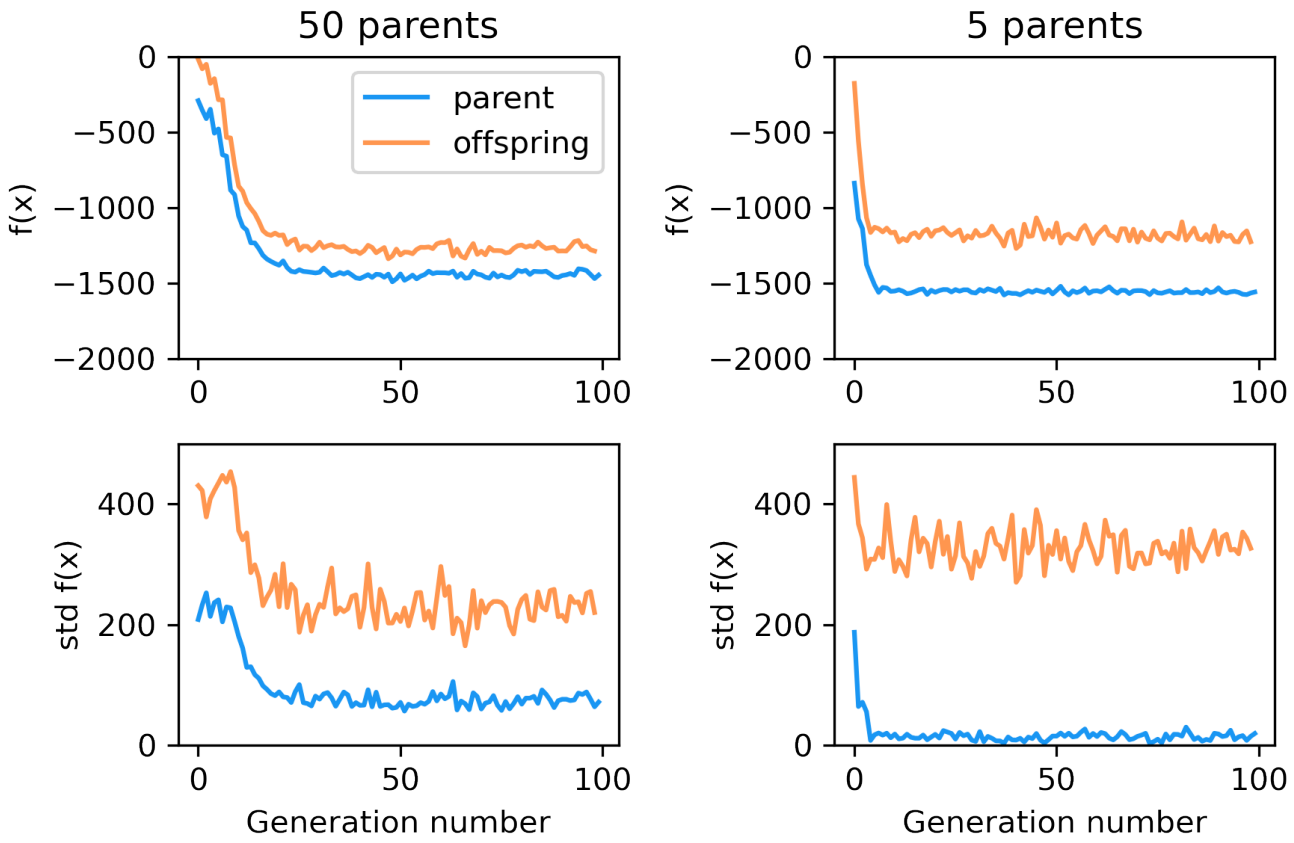


Figure 19

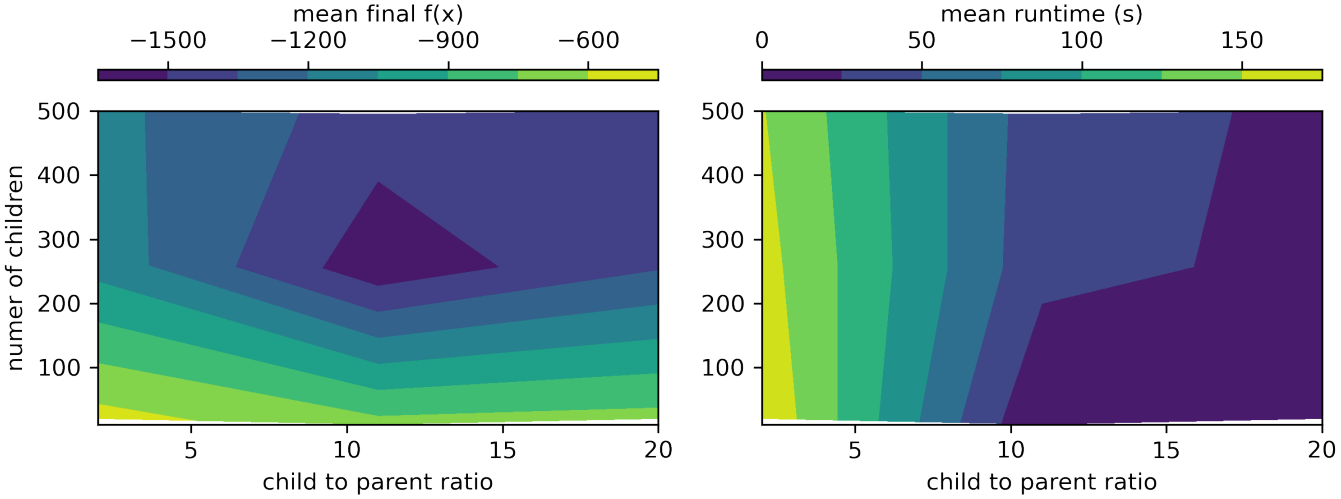


Figure 20: Evolution strategies contour plot with X hyperparameter settings

#### 4.3.4 Final Comparison with optimised population dynamics hyperparameters

A grid search optimisation was performed on the population dynamics hyperparameters (i.e. offspring number and offspring to parent ratio) for each of the mutation methods, and selection methods. Table [X] below shows the performance results with the optimised hyperparameter settings. The `diagonal covariance mutation` method and $\mu + \lambda$ selection each generally had the best performance (with their combination yielding the results). The `diagonal covariance mutation` method's high performance is most likely because (1) it has the benefit of an adaptive mutation, allowing it to adjust the level of mutation to fit the "evolutionary environment" (2) it has a lower number of parameters than the `full covariance mutation` and therefore is more robust. The `full covariance mutation` also has a significantly higher average runtime, due to the more expensive operation of required to (1)

| **mutation method** | complex | complex | diagonal | diagonal | simple | simple |
|---|---|---|---|---|---|---|
| **selection method** | $\mu, \lambda$ | $\mu + \lambda$ | $\mu, \lambda$ | $\mu + \lambda$ | $\mu, \lambda$ | $\mu + \lambda$ |
| **Offspring per parent** | 13 | 9 | 12 | 8 | 18 | 10 |
| **number of offspring** | 65 | 117 | 492 | 472 | 486 | 440 |
| **parent number** | 5 | 13 | 41 | 59 | 27 | 44 |
| **mean f(x)** | -1660.27 | -1761.65 | -1873.8 | -1893.85 | -1829.8 | -1874.89 |
| **std dev f(x)** | 107.95 | 53.44 | 38.38 | 29.49 | 53.68 | 28.62 |
| **average runtime (s)** | 7.15 | 10.03 | 1.14 | 1.15 | 1.1 | 1.12 |

calculate the covariance matrix from the rotation and standard deviation strategy parameters and (2) enforce positive definiteness. $\mu + \lambda$ selection, was preferred because of the advantage of never "losing" the best solutions throughout the optimisation, and through selecting a high number of parents, the downside of $\mu + \lambda$ selection (less exploration) is minimised.

$\mu, \lambda$

# 5 Overall Comparison

A comparison of the best performing Simulated Annealing and Evolution Strategy algorithms on the 5D *Rana's function* is shown in Table 2. Both algorithms are able to significantly outperform random search (which can be thought of an algorithm that focuses purely on exploration). Although this is not particularly impressive, it does provide a useful "sanity check" baseline, indicating that the algorithms are completing some functionality beyond randomly searching through the space. The Evolution Strategy algorithm is far superior to Simulated Annealing, both in terms of minimising the objective function and in terms of average runtime. The lower runtime of the Evolution Strategy Algorithm is due to it's ability to run function evaluations of each generation in parallel.

Figure **??** shows the performance of the Evolution Strategy algorithm is consistently superior to random search and Simulated Annealing for an increasing number of dimensions of the *Rana's function*.

[Maybe cite other models performance on rana function, or look at equation to see what optimal is]

Table 2: Simulated Annealing and Evolution Strategy performance with best performing configurations, benchmarked against a random search (uniform sampling within the space for 10000 iterations)

| **Method** | **Simulated Annealing** | **Evolution Strategy** | **Random Search** |
|---|---|---|---|
| **mean f(x)** | -1666.23 | -1893.85 | -1498.15 |
| **std dev f(x)** | 129.67 | 29.49 | 83.08 |
| **average runtime (s)** | 7.23 | 1.15 | 0.27 |

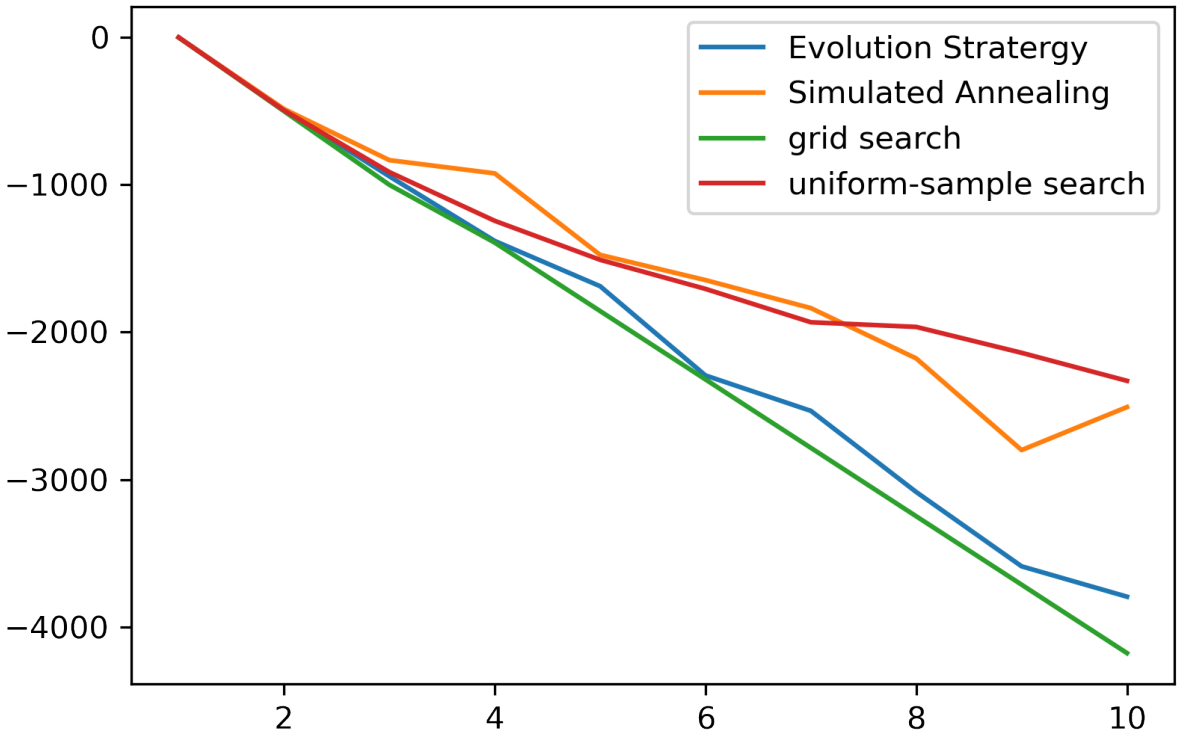


Figure 21: Performance of Simulated Annealing, Evolution Strategy and Random Search optimisation with increasing dimension of *Rana's function*

# References

[1] Scott Kirkpatrick. "Optimization by simulated annealing: Quantitative studies". In: *Journal of statistical physics* 34.5-6 (1984), pp. 975–986.

[2] Darrell Whitley et al. "Evaluating evolutionary algorithms". In: *Artificial Intelligence* 85.1 (1996), pp. 245–276. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(95)00124-7. URL: http://www.sciencedirect.com/science/article/pii/0004370295001247.

# 6 Appendix