1 Introduction

2 Simulated Annealing

2.1 Implementation details

2.2 Model Baseline on 2D Rana function

[table of each models average runtime etc] For simulated annealing to work as desired, in the initial iterations it needs to focus on exploring as much of the space as possible, and then as the algorithm progresses to focus more on a focused search of the best "zone". The variations to the control parameter step method are experimented with: (1) The `Simple-step` method samples steps from a fixed multivariate Gaussian a diagonal covariance. (2) The `Diagonal-covariance` method which samples from a diagonal covariance matrix, where each diagonal element is an adaptive parameter [define equation]. The `Cholesky-covariance` method samples from a multivariate Gaussian distribution (with covariance between control parameters), where the full matrix is adaptive [define]. In this section a preliminary presentation of the simulated annealing optimisation algorithm is given on the 2D Rana fuction, to confirm that the algorithms are working correctly, and to demonstrate the search patterns corresponding to each of the control parameter step methodologies.

Figure 1 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 2, 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 2. The solution archives (Figure 4) 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 5 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 6 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 6, 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.

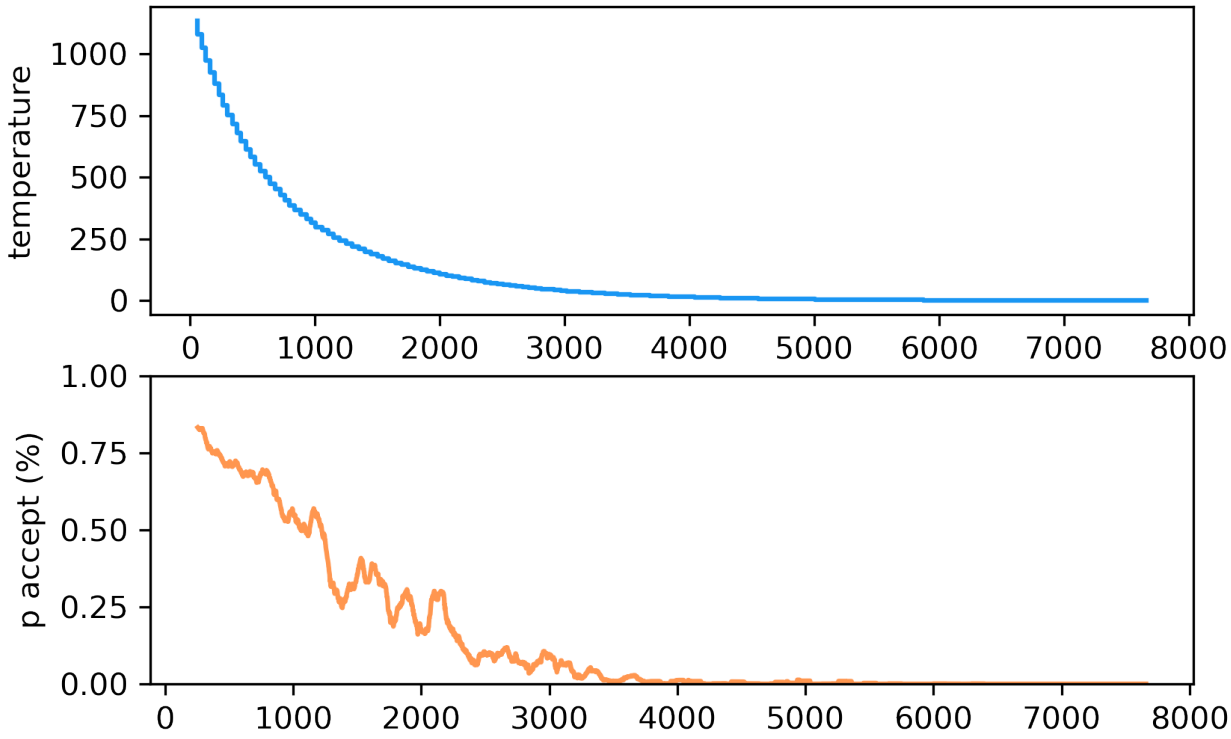

Figure 1: Temperature annealing and probability of acceptance

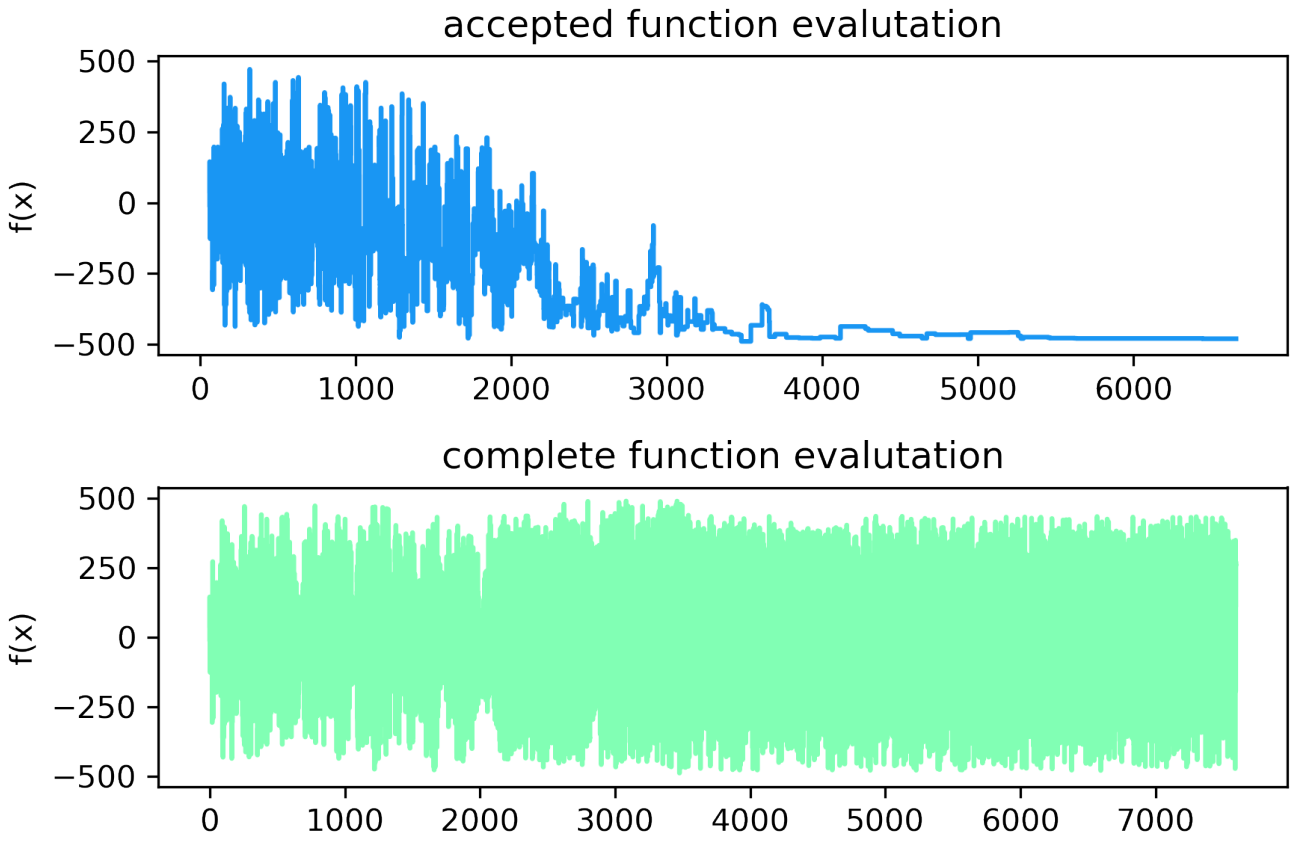

Figure 2: Performance history

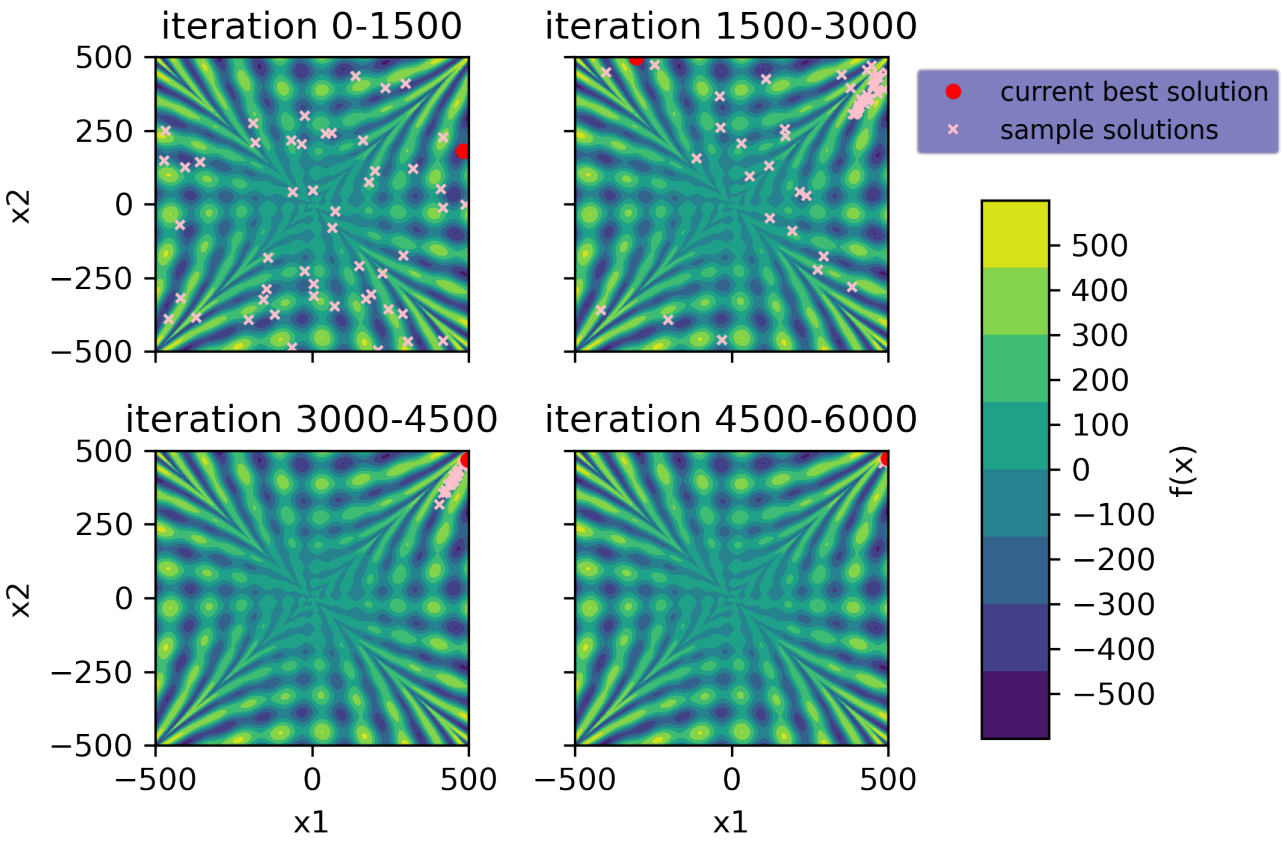

Figure 3: Sample solutions throughout optimisation. Early in the optimisation the whole space is explored. As the optimisation progresses, a progressively smaller subset of the space is focused on

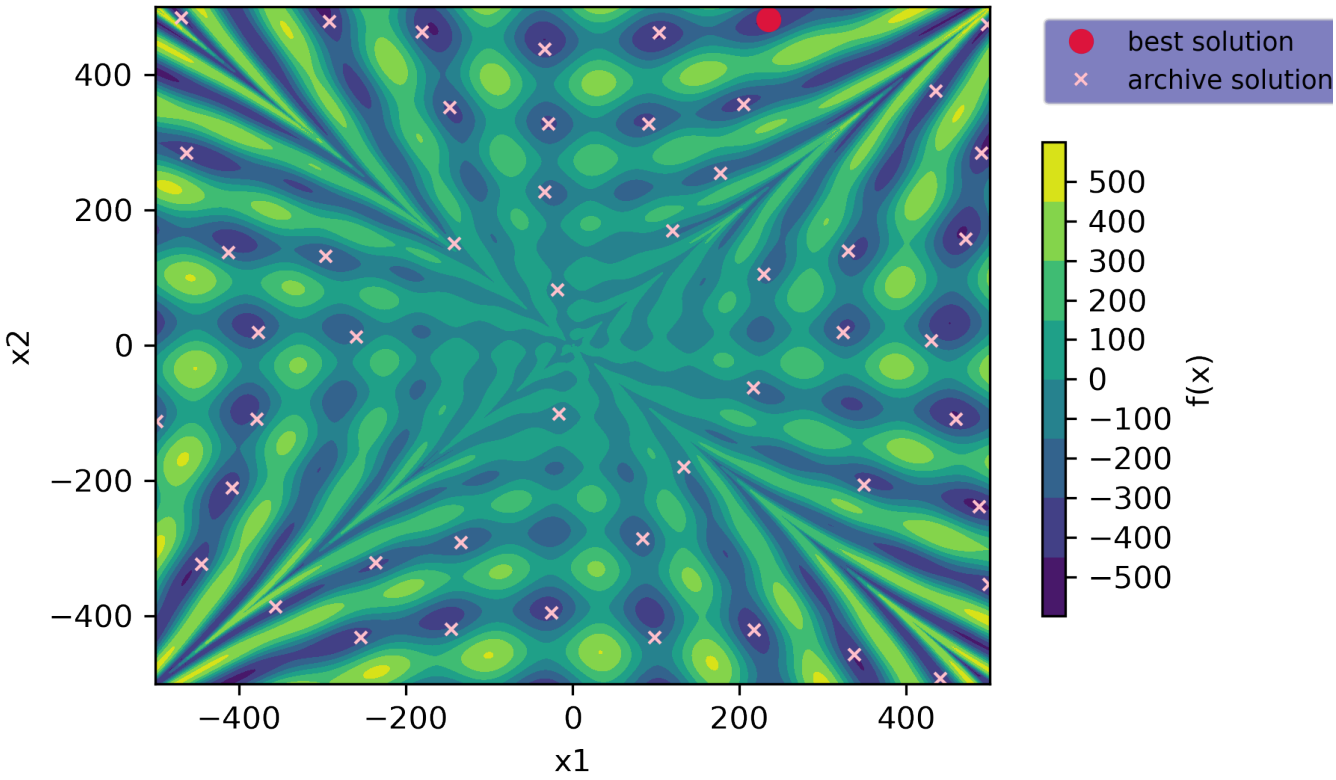

Figure 4: Archives from optimisation. The archive solutions are distributed throughout the space, indicating that the space is thoroughly explored

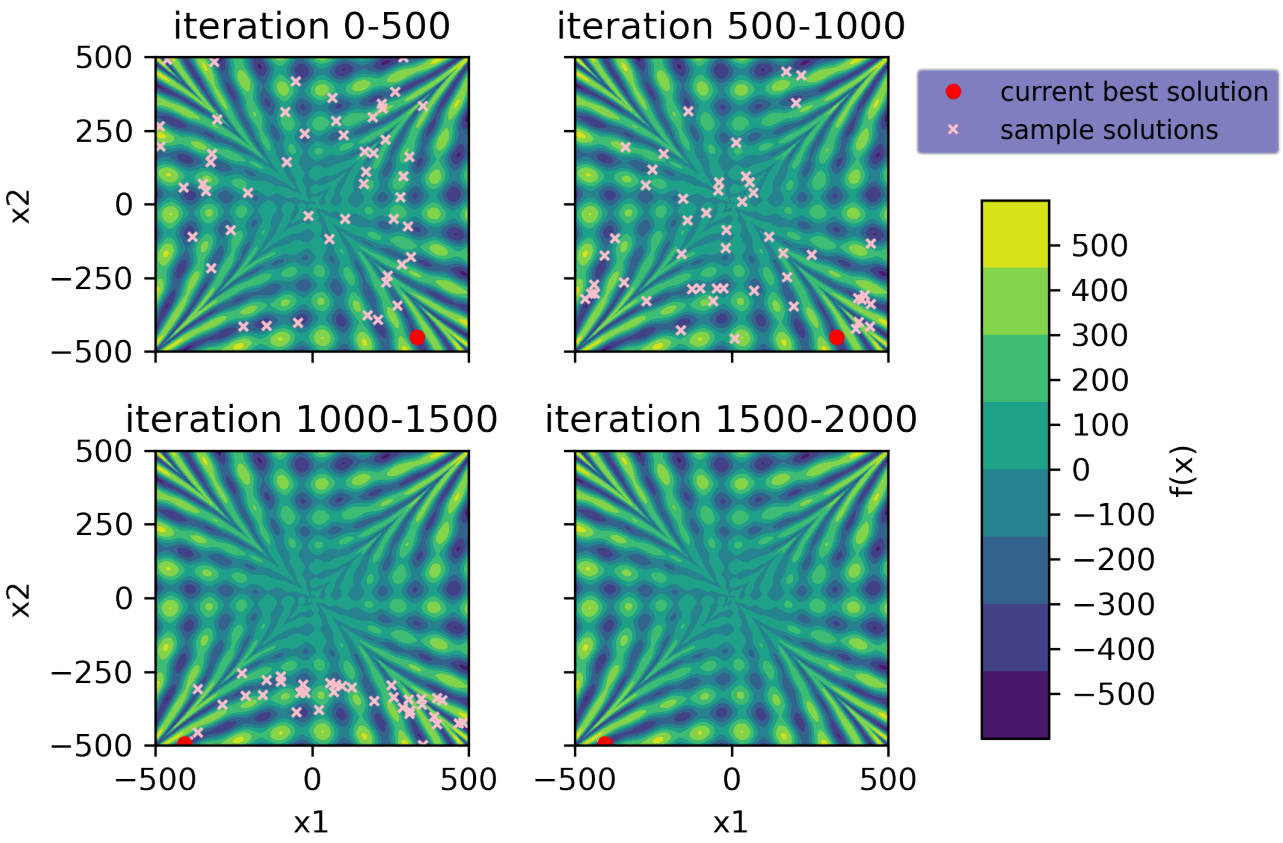

Figure 5: Sample solutions throughout optimisation for Diagonal step method

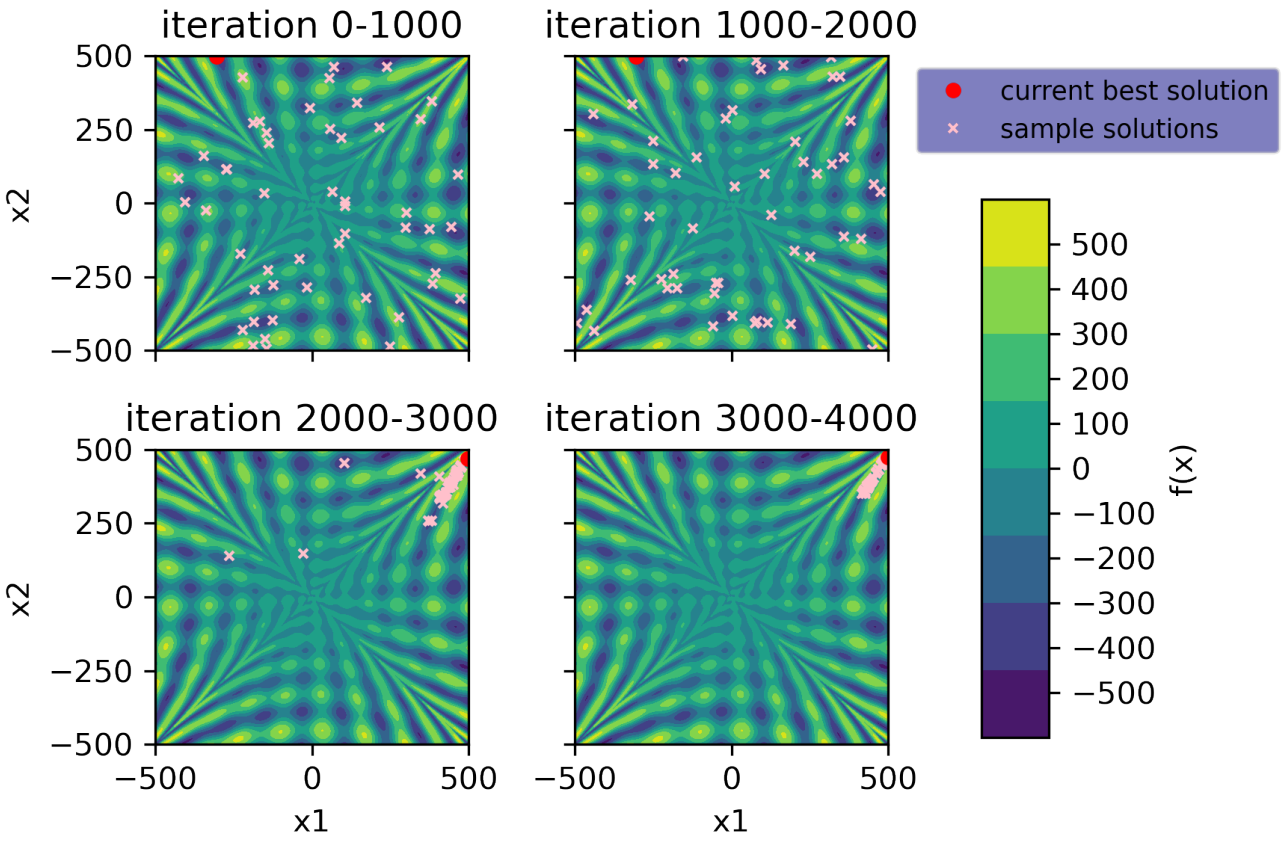

Figure 6: Sample solutions throughout optimisation for Cholesky step method

2.3 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.

2.3.1 Step Method

Various sub-plot describing the performance of the `Diagonal step method` are given in Figure 7. 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 7 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).

Various sub-plots describing the performance of the `Cholesky step method` for a single example run, is shown in Figure 8 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).

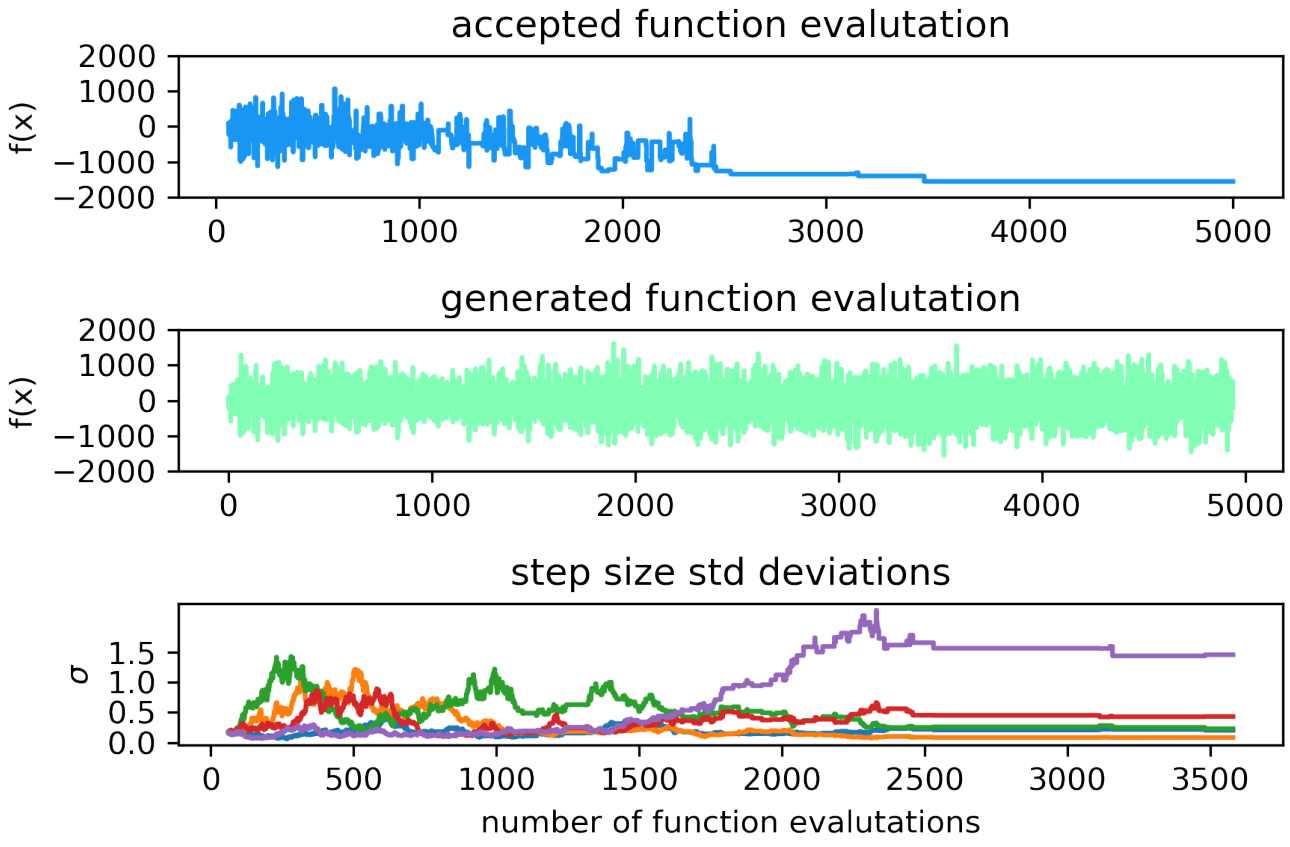

Figure 7: Diagonal step-method example performance

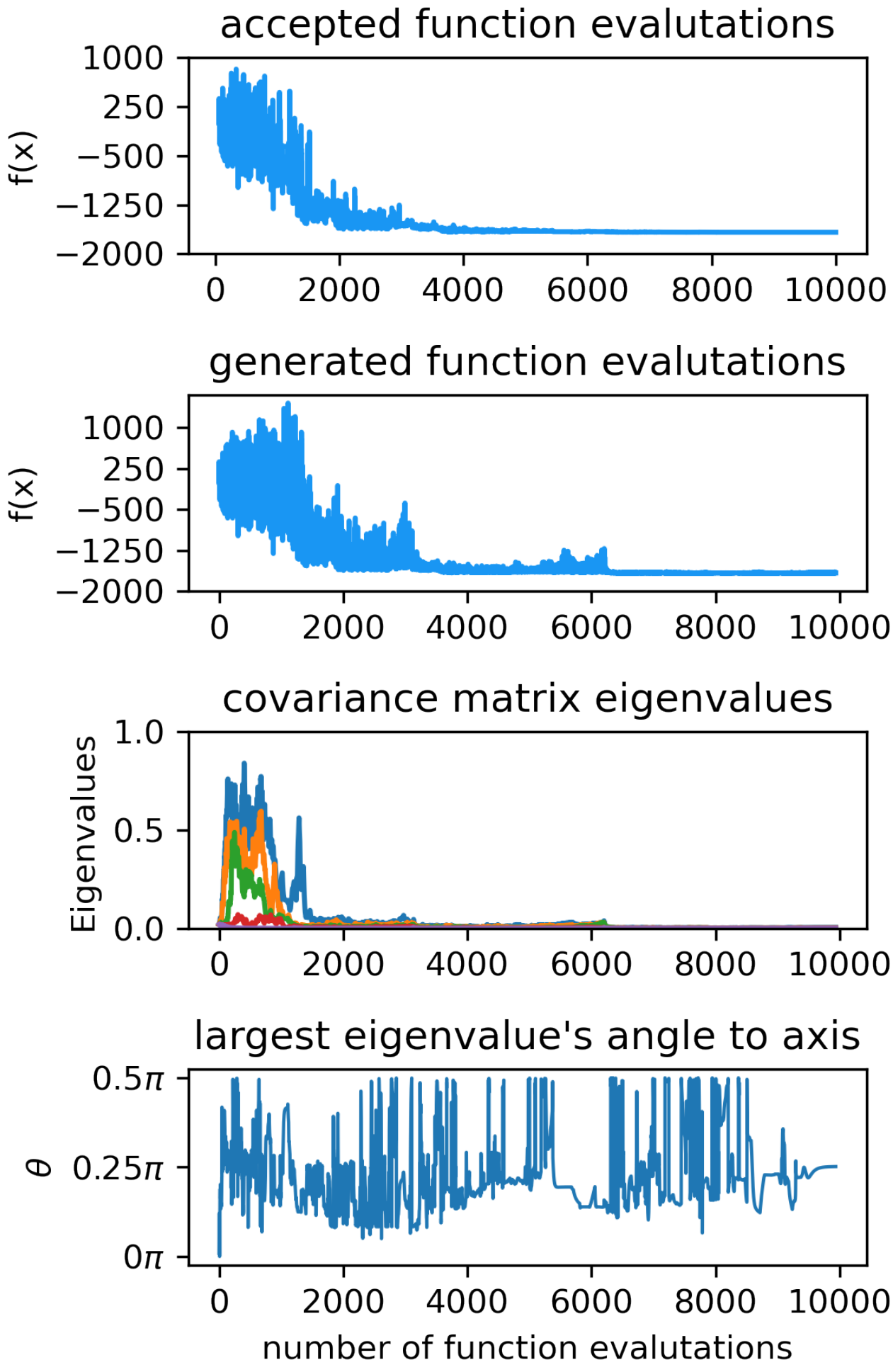

Figure 8: Cholesky step-method example performance

2.3.2 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 9, 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 10. 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 10 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].

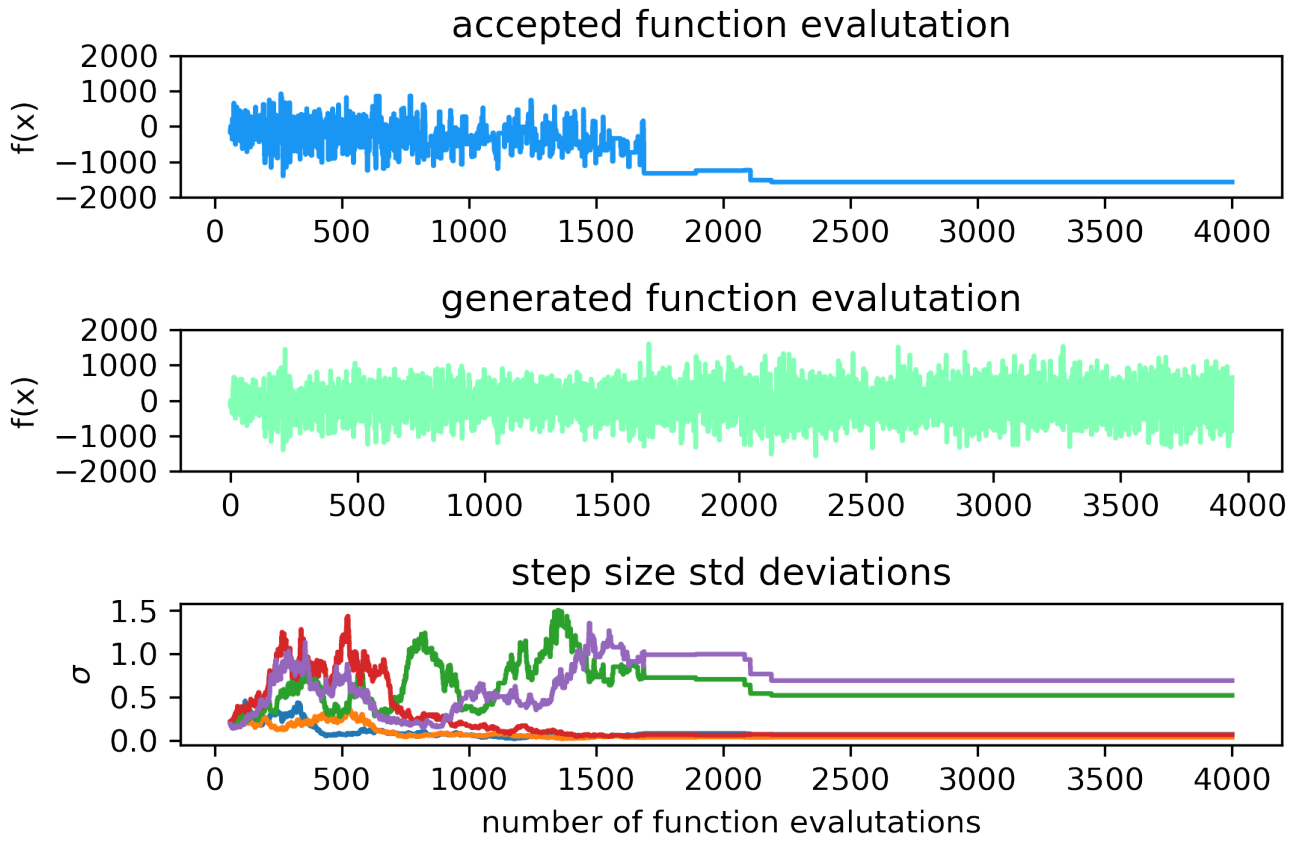

Figure 9: Illustration of issue where algorithm gets stuck, due to dependency on acceptences for step size updates

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

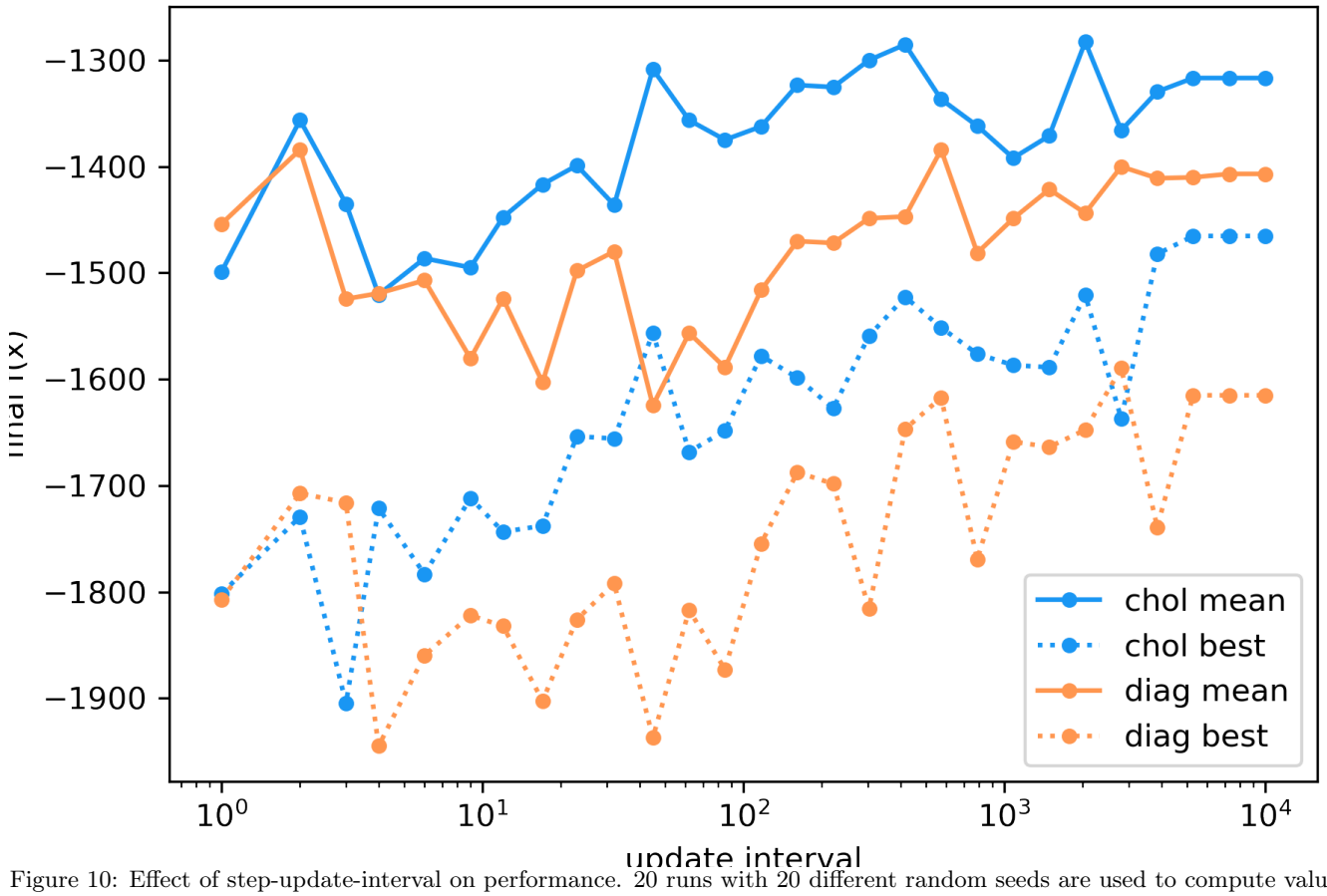

Figure 10: Effect of step-update-interval on performance. 20 runs with 20 different random seeds are used to compute values at each point

2.3.3 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 11 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 12. 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.

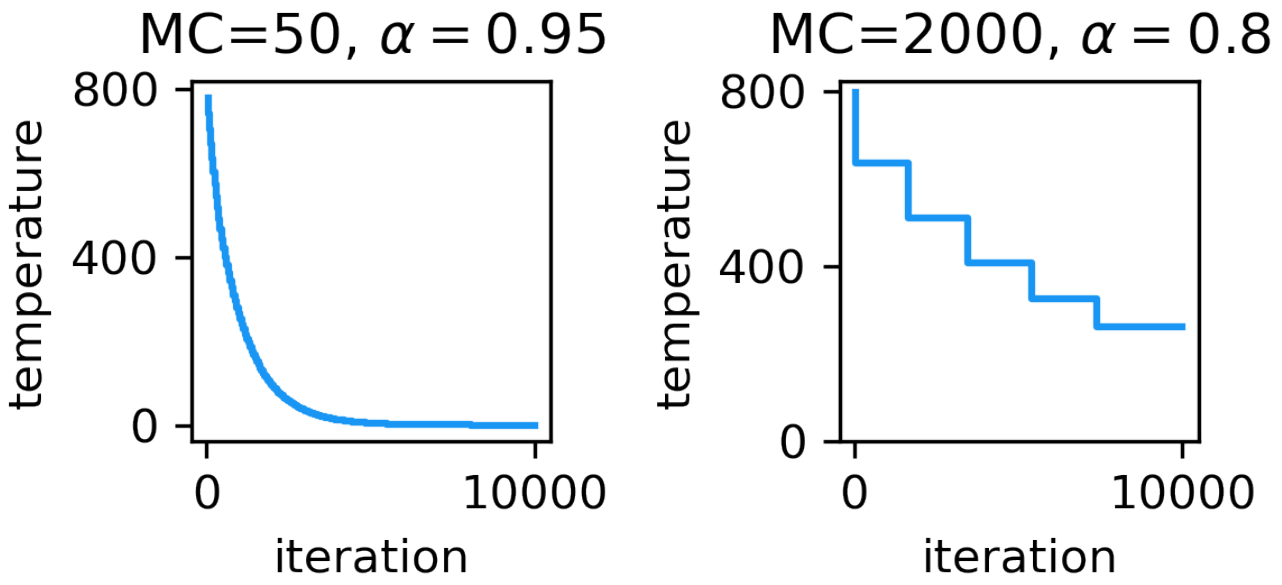

Figure 11: Effect of alpha and markov chain length (MC) on annealing schedule

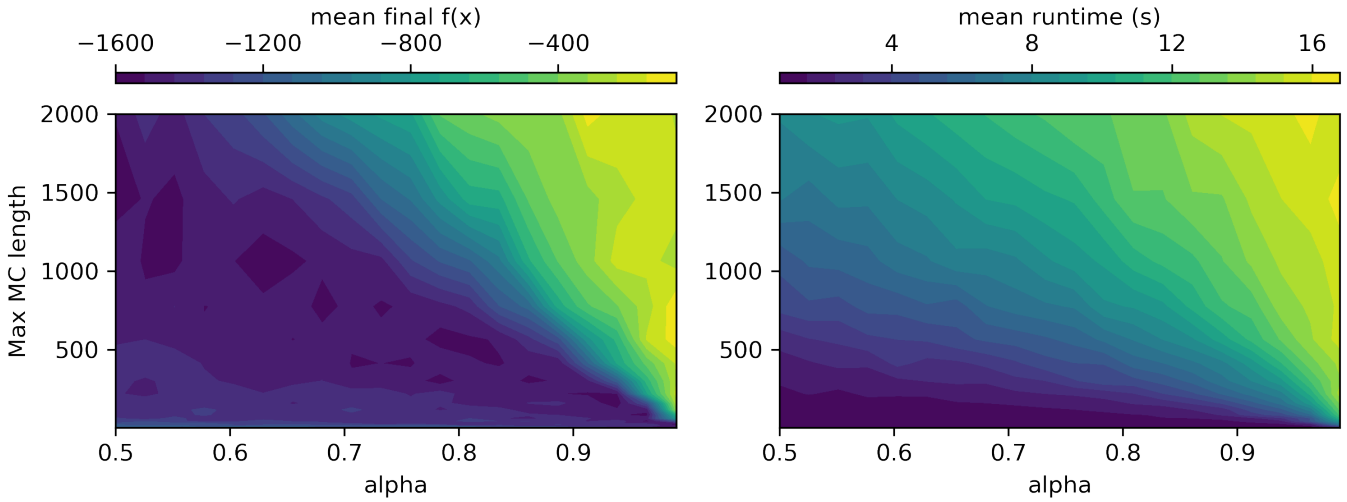

Figure 12: Contour plot for grid search of maximum markov chain length, and alpha value, for Diagonal-step-method and `diversified-step-update-rule` (interval of 20). 30 runs with 30 different random seeds are used of estimation of mean objective value and mean time

2.3.4 Results with tuned hyperparameters

[rewrite properly] diversified-step-update-rule has clear benifits. Diagonal best in terms of final value of the function, and in terms of runtime. Outperforms simple method in terms of runtime, as it converges quickly. Outperforms Cholesky in terms of runtime because it has less computationally expensive operations (i.e doesn't have to calcualte Cholesky decomposition). Cholesky method (with diversified-step-update-rule) has higher variance and lower best final performance across seeds) suggesting that it is worse at exploration and better at exploration than the Diagonal method: it is good at minimisation within a local zone but it sometimes does not find a good local zone to exploit - so when it has a relatively "lucky" random seed it finds very low point, but is less likely to find a good local zone.

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
best final performance (across seeds)	-1678.31	-1914.31	-1808.93	-1858.22	-1745.54
mean best objective (within run)	-1477.22	-1588.84	-1599.03	-1666.58	-1524.57
average runtime (s)	16.13	7.98	11.4	7.23	12.13

3 Evolutionary strategies

3.1 Implementation details

For each component of the evolutionary strategies the following methods were implemented, noting any deviations from the lecture slides with a "$\star$".

Mutation methods:

1. `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. 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) ($\star$).
2. $\star$ `simple mutation`: Spherical Gaussian noise
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).

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

3.2 Model Baseline on 2D Rana function

Because fixed-diagonal Guassian mutation is used, the parents don't reach the convergence criterion (as they have non-negligible variance) even though functionally the algorithm has converged (the other mutation methods did in practice converge).

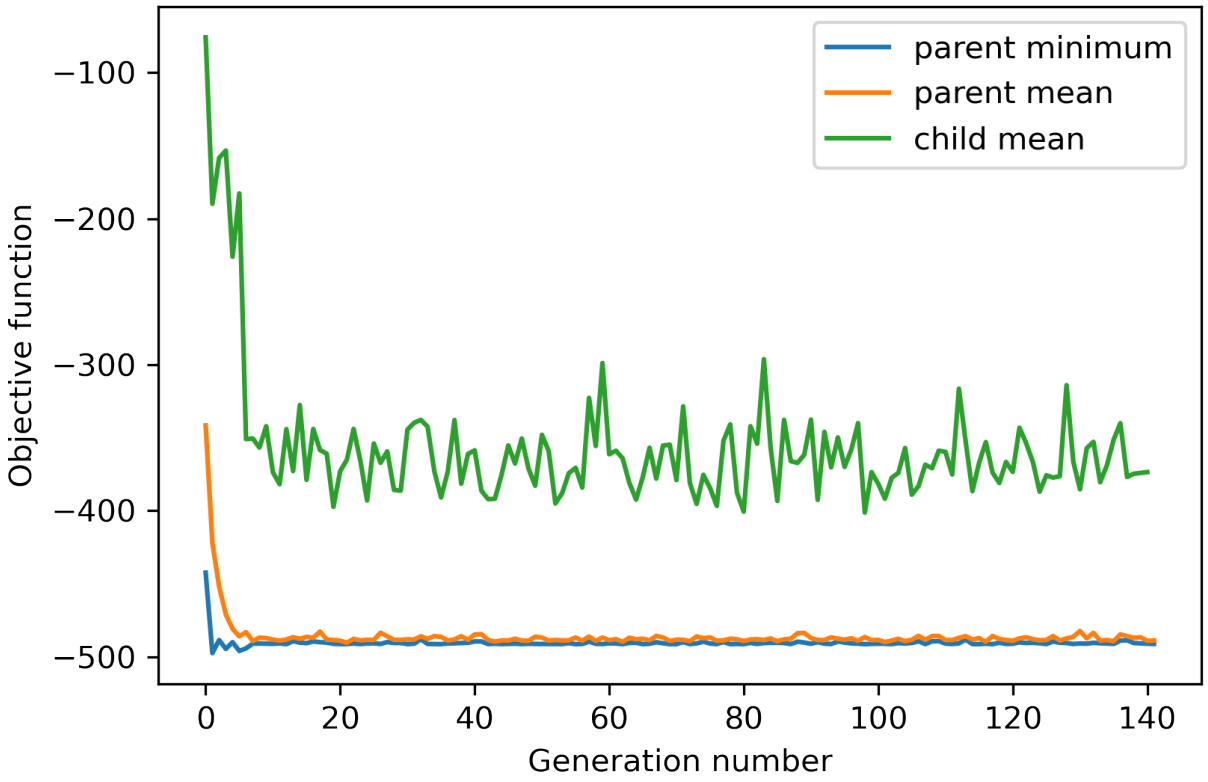

Figure 13: Evolution strategies method example performance on 2D rana function

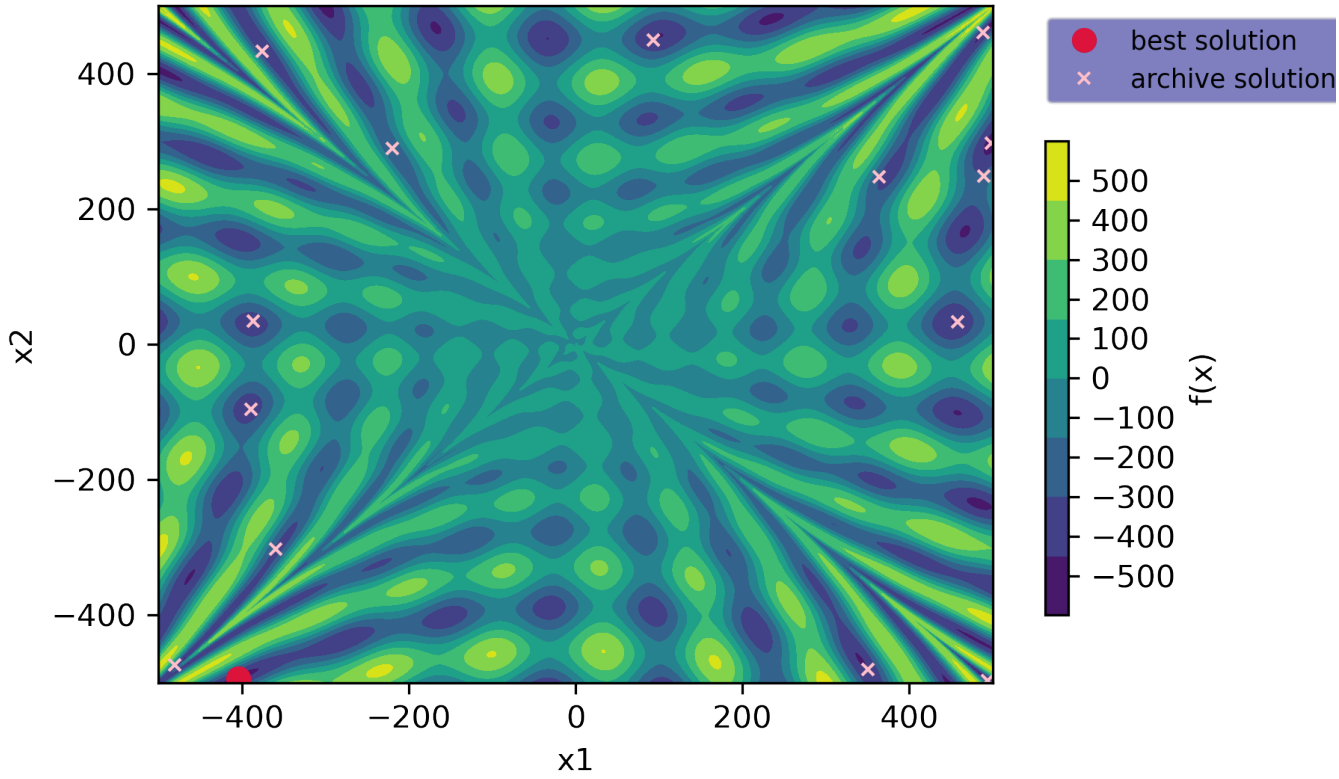

Figure 14: Evolution strategies method archive example from run on 2D rana function

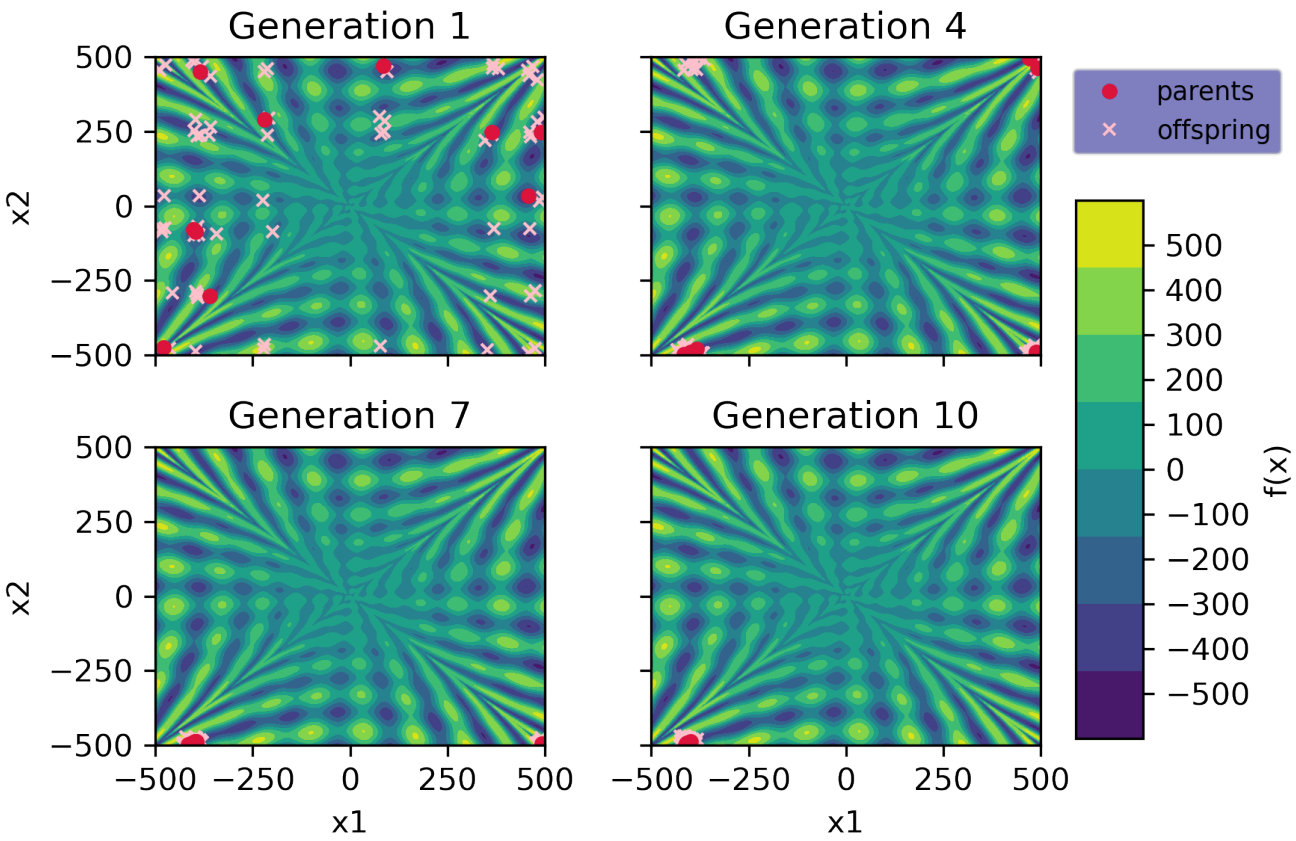

Figure 15: Evolution strategies method example search pattern on 2D rana function

3.3 Analysis on 5D Rana function

In this section, an initial presentation of the various methods for selection (μ, λ-selection and $\mu + \lambda$-selection), and mutations (fixed-diagonal Gaussian, adaptive diagonal-covariance Gaussian, adaptive full-covariance Gaussian) is given. The effects of the hyperparameters controlling the population dynamics are explored, and optimised for each combination of the selection and mutation methods. Finally, a comparison of the methods with the optimal hyperparameters is given.

3.3.1 Mutation method

Both the amount and "direction" of the mutations is controlled by the mutation-covariance matrix. The determinant is therefore indicative of the total "amount" of mutation - and in Figure 17 the effect of a smaller mutation-covariance determinant leading to a smaller amount of variation in the offspring is clear.

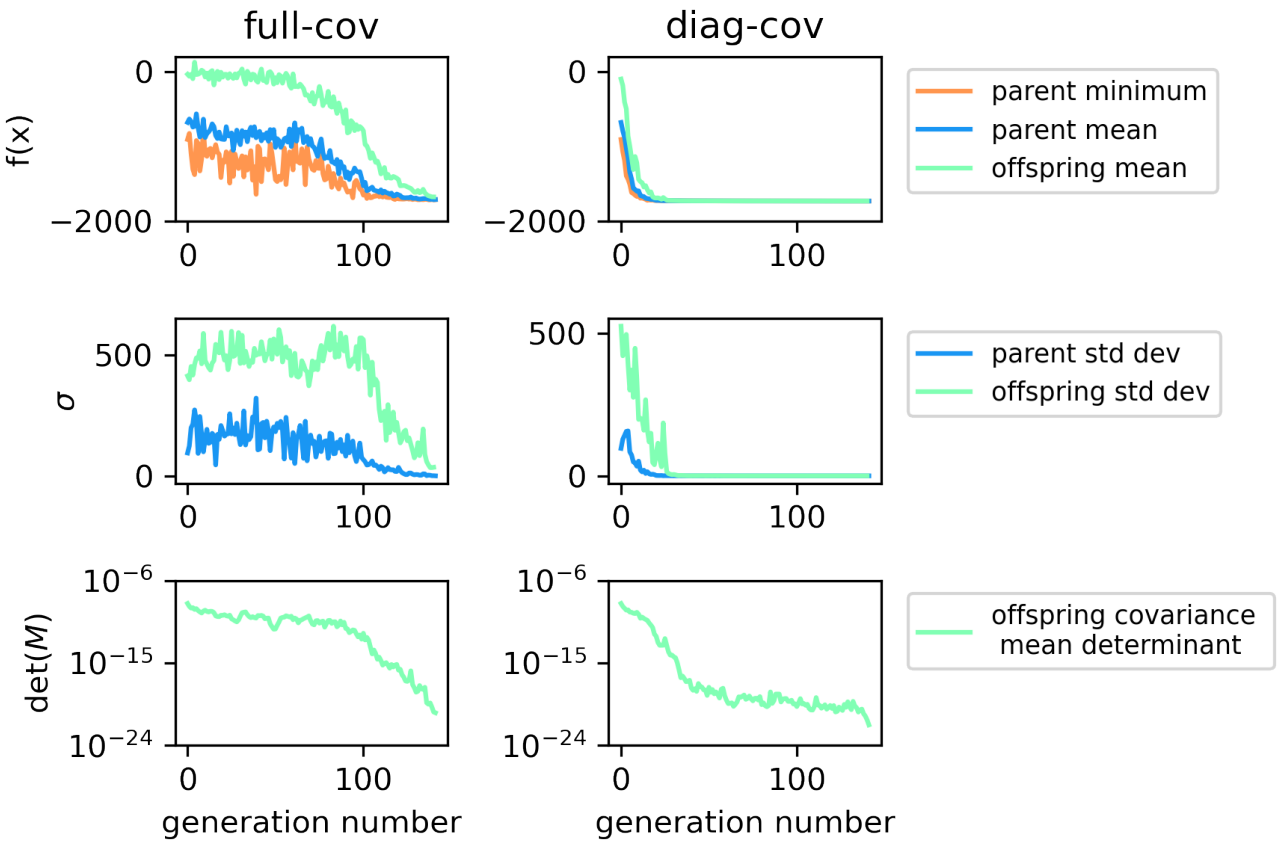

Figure 16: Performance of ES algorithm with diagonal & full mutation covariance matrices

3.3.2 Selection Method

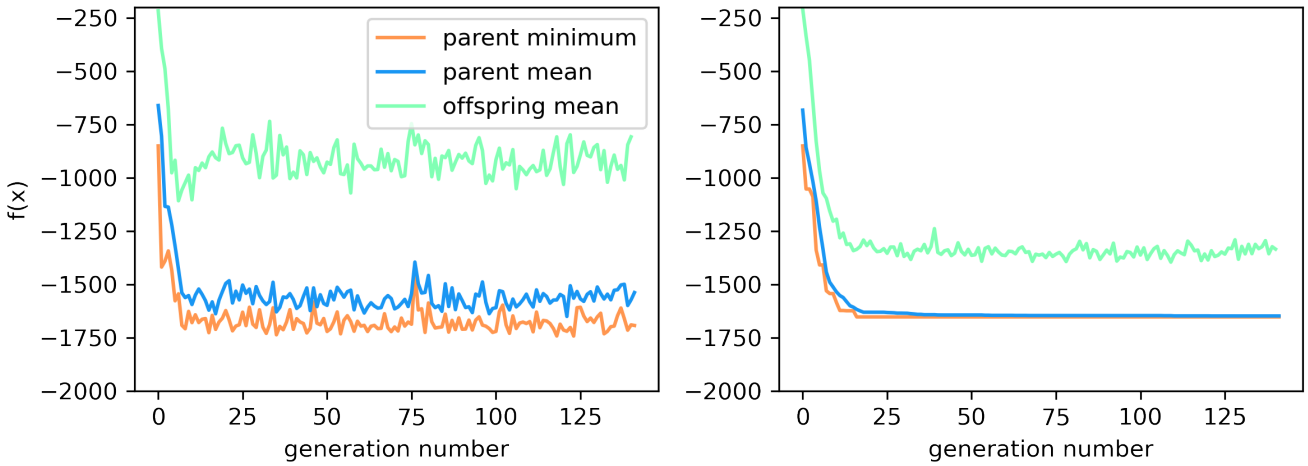

Figure 17: Performance of ES algorithm with μ, λ-selection and $\mu + \lambda$-selection

3.3.3 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 18. 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 exloitation). 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.

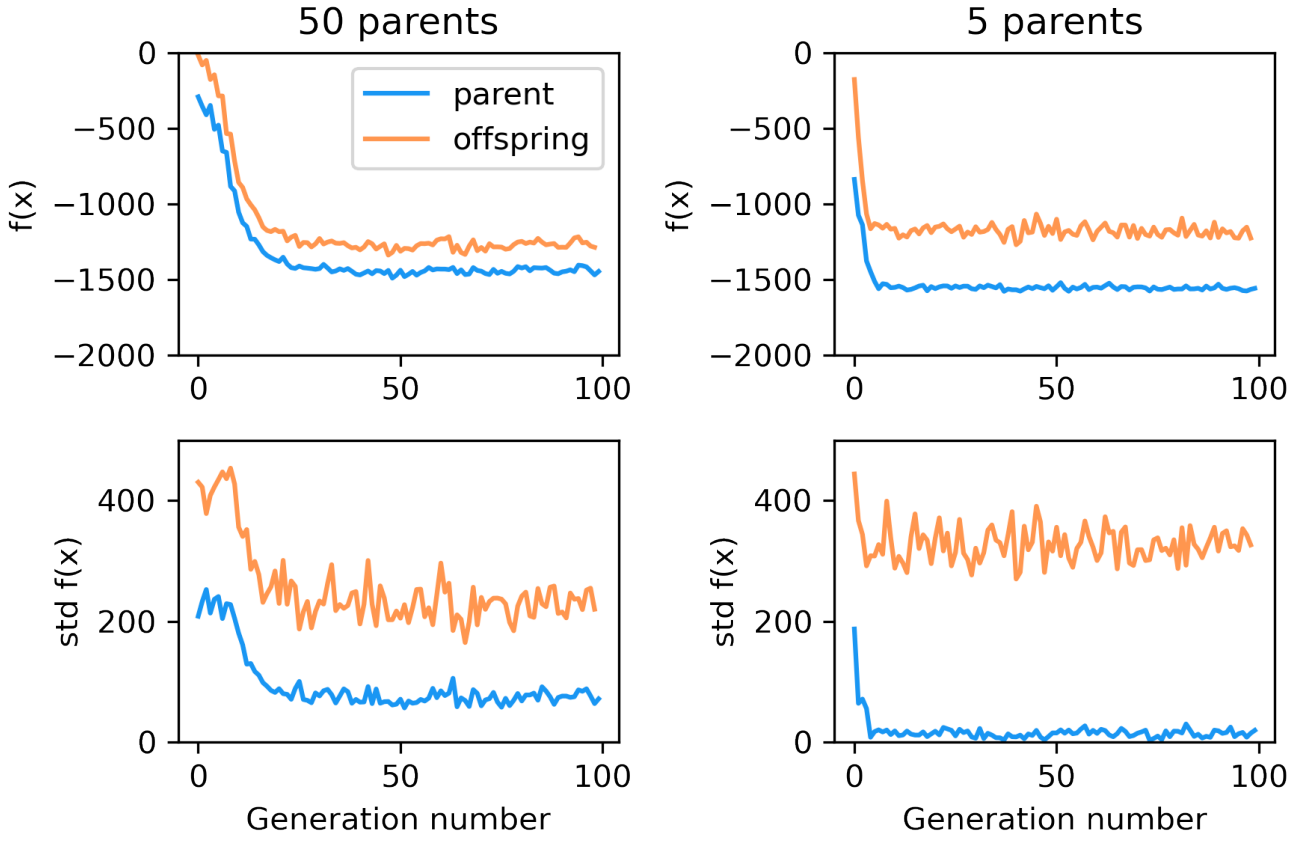

Figure 18

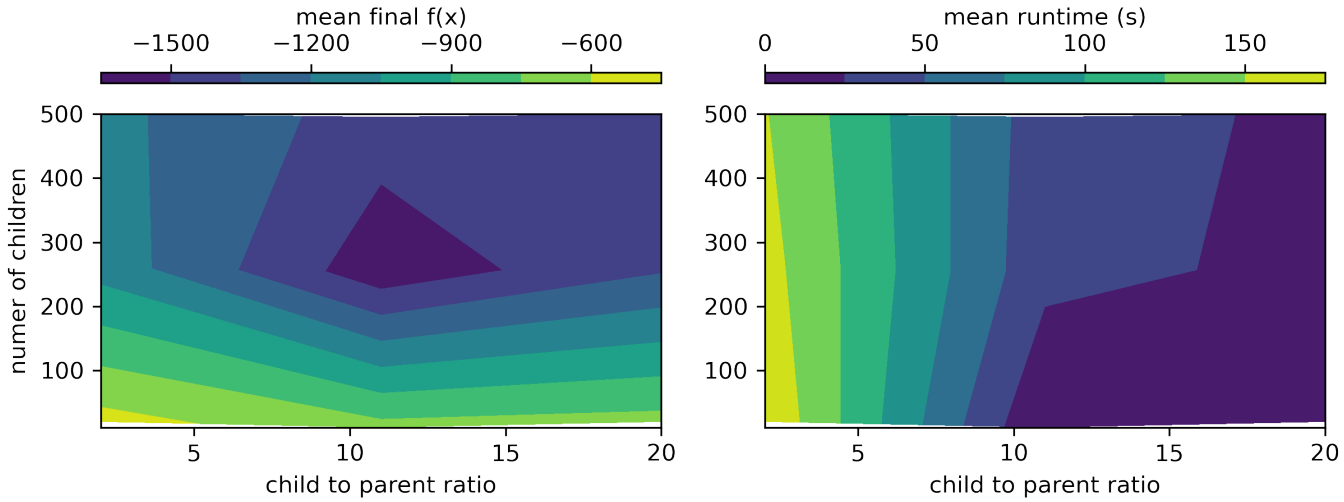

Figure 19: Evolution stratergies contour plot with X hyperparameter settings

3.3.4 Final Comparison

[still need to update this with final values and include mew plus lambda] Runtimes are very long, maybe need to rewrite some code to improve this]

mutation method	complex	diagonal	simple
selection_method	mew,lambda	mew,lambda	mew,lambda
Offspring per parent	11	11	11
number of offspring	253	495	495
mean final performance	-1596.72	-1894.53	-1844.63
std dev final performance	16.13	22.89	28.42
best final performance (across seeds)	-1612.85	-1917.43	-1873.05
mean best objective (within run)	-1636.23	-1896.73	-1858.58
average runtime (s)	32.09	31.19	31.24

4 Overall

Table with best perfoming models results and runtimes

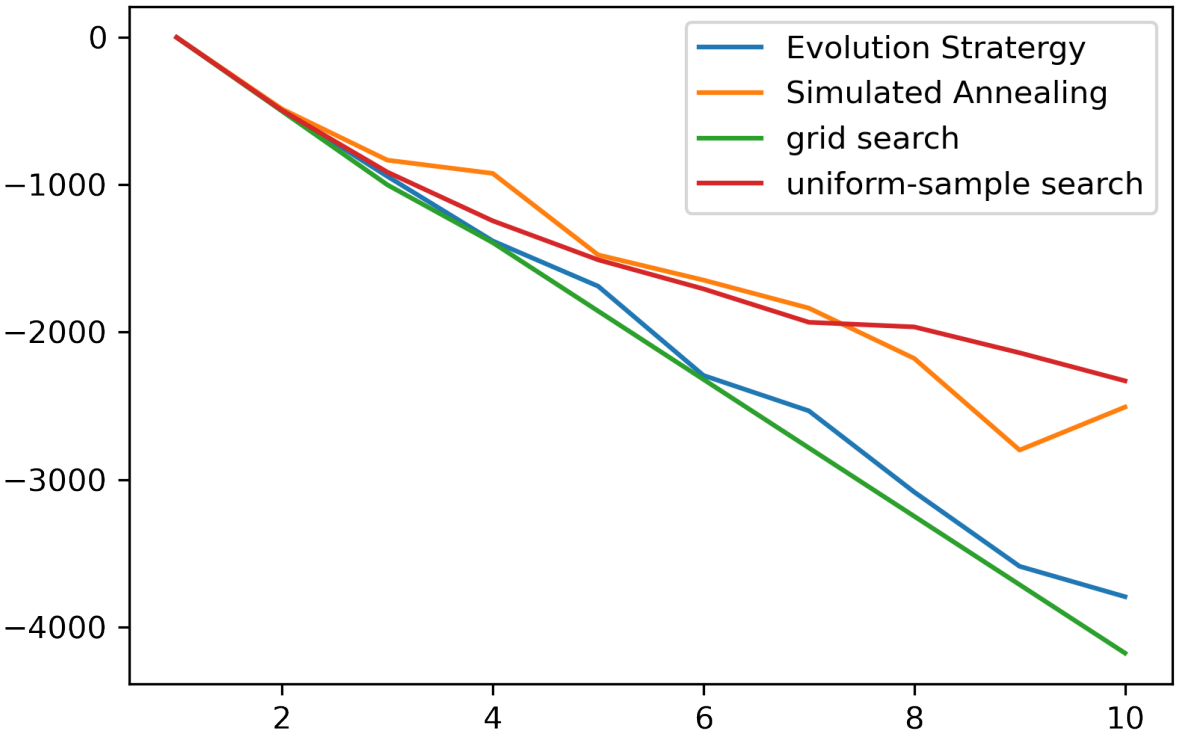
0
−1000
−2000
−3000
−4000
2
4
6
8
10
Evolution Stratergy
Simulated Annealing
grid search
uniform-sample search

Figure 20

5 Appendix

5.1 Simulated Annealing

5.1.1 Markov Chain length and Alpha Parameter Optimisation