Random Search

Patrick Chao

May 14th, 2018

1 Introduction

The purpose of this analysis is to empirically experiment with the Basic Random-Search algorithm (BRS). We assume we have an oracle where we can query the the function f with some given noise, meaning we receive $f(x) + (0, \sigma^2)$. With step-size α, N directions sampled per iteration, and standard deviation of noise ν, the BRS algorithm utilizes the following update rule:

$$\theta_{j+1} = \theta_j - \frac{\alpha}{N} \sum_{k=1}^{N} \left[\frac{f(x_j + \nu\delta_{i,j}) - f(x_j - \nu\delta_{i,j})}{\nu} \right] \delta_{i,j}. \tag{1}$$

One change we investigate is the effect of dividing by the standard deviation of the function query σ_f. This normalizes the step size per iteration, so that α represents an intrinsic step size meaning, as the step size in equation 1 relies on the variance in the queried function calls as well. Thus, we define σ_f and the modified update rule in the following fashion:

$$\begin{aligned} \sigma_f &= \sqrt{\frac{1}{N} \sum_{i,j} (f(x_j \pm \nu\delta_{i,j}) - \mu)^2} \\ \theta_{j+1} &= \theta_j - \frac{\alpha}{N} \sum_{k=1}^{N} \left[\frac{f(x_j + \nu\delta_{i,j}) - f(x_j - \nu\delta_{i,j})}{\nu\sigma_f} \right] \delta_{i,j}. \end{aligned} \tag{2}$$

This attempts to alleviate the dependence on both the shape of the function around θ_j and provides consistent step sizes.

2 Methods

To experiment with these two approaches, we wanted to compare convergence rates of these two update rules.

2.1 Pipeline

The experimentation pipeline is as follows:

1. Parse arguments (matrix size,α, number of iterations, initialization magnitude, ν, maximum noise, number of iterations, number of directions to sample per iteration, boolean to normalize or not, boolean to create surface plots or not, verbosity argument to display convergence path, boolean for displaying plots)
2. For each iteration:
   i. Initialize a random positive definite matrix
   ii. Initialize a random initialization for θ_0
   iii. Perform the BRS and modified BRS algorithm for four noise functions
   iv. Record the loss per optimization
3. With the loss per each iteration for each noise function, average the overall losses and save plots

To experiment with the various parameters and how the two methods perform with respect to each other, I decided to focus on tuning four hyperparameters and four different noise functions.
Four key parameters to test: α, ν, maximum noise, number of samples.
Four different noise functions: no noise, constant noise, increasing quadratic noise, decreasing quadratic noise.

Testing Parameters: Let j represent the current iteration.

$\boldsymbol{\alpha}$: step size for optimization algorithm

$\boldsymbol{\nu}$: noise to explore around θ_j

Maximum Noise: the maximum noise for the quadratic noise functions. They are defined to be $\frac{j^2 \text{max noise}}{\#\text{ iter}^2}$, and $\frac{(\#\text{ iter}-j)^2 \text{max noise}}{\#\text{ iter}^2}$, as these increase from 0 to max noise or decrease from max noise to 0. The constant noise function is the average noise for the increasing and decreasing quadratic noise functions, which is $\frac{1}{3}$max noise as $\int_0^{\#\text{ iter}} \frac{j^2 \text{max noise}}{\#\text{ iter}^2} dj = \frac{1}{3}$ max noise. The no noise function is just constant 0 noise.

Number of Samples: The number of samples to query per θ_j, this is the N in equation 2. Larger values represent better estimates for the true gradient, and lower values are more noisy.

Covariance Noise Functions:

These functions represent the noise in the function query $x^T Q x + (0, \sigma_j)$.

No Noise: $\sigma_j = 0$, constant zero noise

Constant Noise: $\sigma_j = \frac{1}{3}$max noise

Increasing Quadratic Noise: $\sigma_j = \frac{j^2 \text{max noise}}{\#\text{ iter}^2}$

Decreasing Quadratic Noise: $\sigma_j = \frac{(\#\text{ iter}-j)^2 \text{max noise}}{\#\text{ iter}^2}$

2.2 Matrix Initialization

I initialize a positive definite matrix Q by considering a random matrix A, and let $Q = AA^T$. I check that for matrices of size two, the eigenvalues must be at least 0.5.

2.3 Parameter Initialization

For θ_0, I initialize a constant magnitude value to ensure consistency over all trials. This is captured in the `'initialization magnitude'` parameter. For the experiments, I found that about 200 was a good initialization, as it showed sufficient distance to travel to converge. I also include an optional argument `'quad'` which represents a specific quadrant to initialize θ_0. For graphical purposes in the surface plots, the second quadrant gives the best and most visible path.

2.4 Function Query

The function query utilizes the quadratic form $x^TQx+\mathcal{N}(0,\sigma_j)^2$, with added various covariance noise functions mentioned above.

2.5 Optimization

The optimization for BRS and the modified BRS algorithms perform as described in equations 1 and 2. The only major change was that I realized it was impossible to calculate σ_f in equation 2, since each iteration depends on the value of σ_f, but σ_f depends on the evaluated values over all j. Thus, I use a modified form of σ_f:

$$\begin{aligned}
\mu_j &= \frac{1}{2N}\sum_{i=1}^{N} f(x_j \pm \nu\delta_{i,j}) \\
\sigma_{f,j} &= \sqrt{\frac{1}{2N}\sum_{i=1}^{N}(f(x_j \pm \nu\delta_{i,j}) - \mu_j)^2} \\
\theta_{j+1} &= \theta_j - \frac{\alpha}{N}\sum_{k=1}^{N}\left[\frac{f(x_j+\nu\delta_{i,j}) - f(x_j-\nu\delta_{i,j})}{\nu\sigma_{f,j}}\right]\delta_{i,j}.
\end{aligned} \tag{3}$$

There are two major differences. First, I divide by $\frac{1}{2N}$ instead of $\frac{1}{N}$ to account for the positive and negative directions. Secondly, there is instead a $\sigma_{f,j}$ for each iteration rather than a σ_f overall.

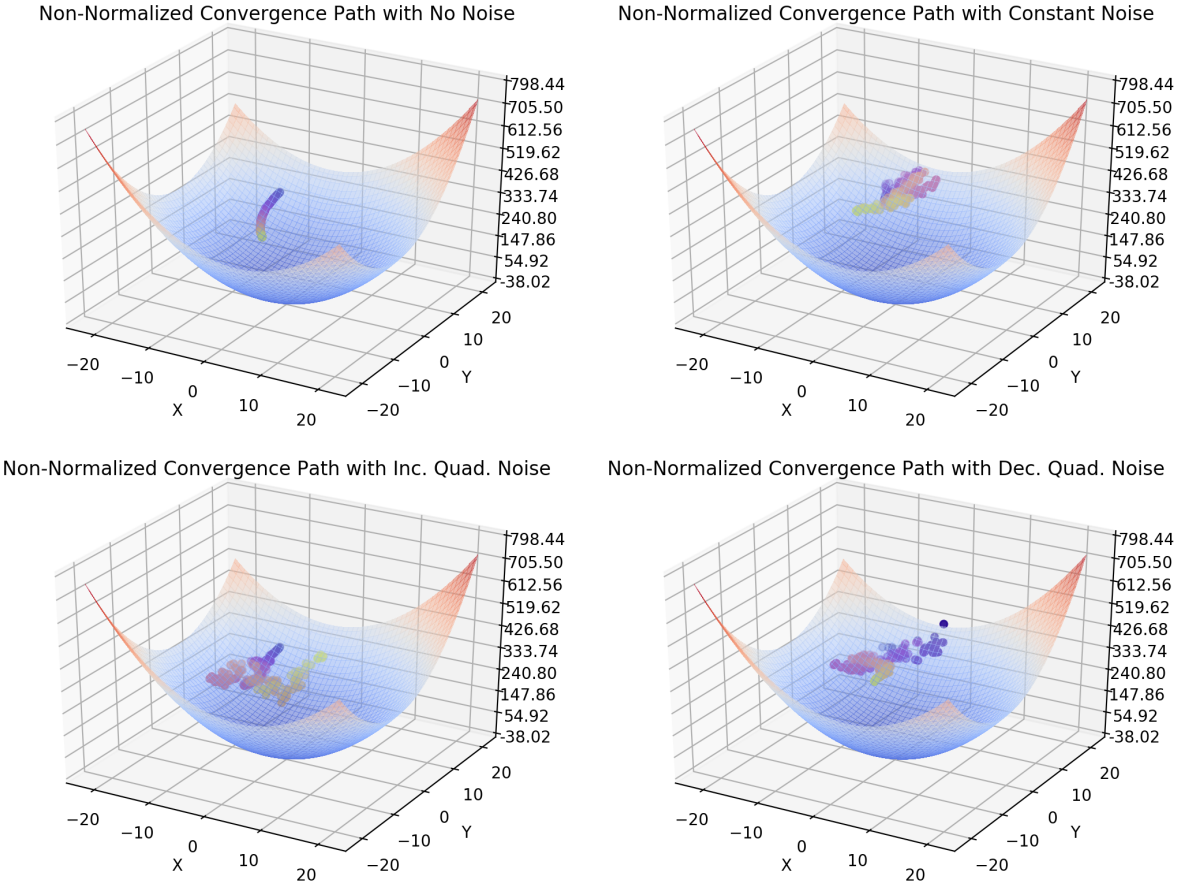
3 Analysis

First to begin, I determined that a good baseline example that had decent convergence was with the following parameters: matrix size = 2, $\alpha = 0.01$, iterations = 200, initialization magnitude = 20, $\nu = 0.01$, max noise= 30, number of initializations = 30, number of directions to sample = 20. All of the parameters are relatively self-explanatory or explained above in the methods section except for the number of initializations. This represents the number of individual trials, meaning new initializations of Q and θ_0. This was chosen to be 30 as it just needs to be sufficiently large to accurately represent the true convergence path.

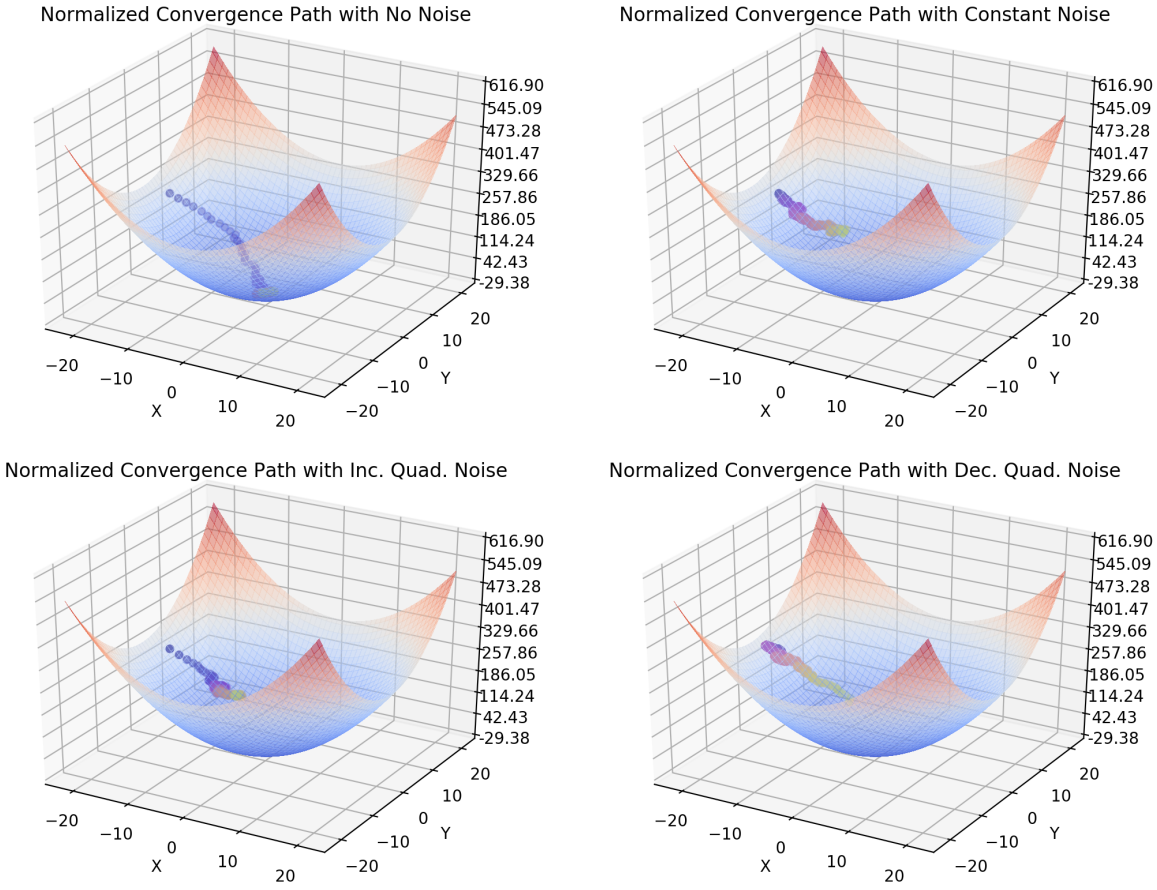
3.1 Convergence Paths

Here are plots representing the convergence paths with the given initializations, various noise functions, and with/without normalizing step sizes in the BRS algorithm. The quadratic surface is color-coded with a coolwarm map where darker blues represent lower values and lighter reds represent larger values. The path of the convergence is color-coded with a plasma map where darker blues represent earlier portions of the path, and lighter colors such as red and yellow represent later sections, farther in the optimization.

Non-Normalized Paths



Normalized Paths



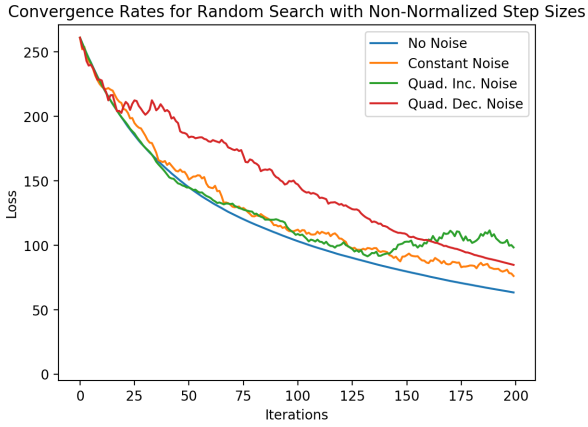
It seems that overall the normalized paths tend to converge better with the equivalent parameters, even the path with no noise remained in a relatively shallow region. Additionally, it seems that decreasing quadratic noise may be beneficial, as it allows greater 'exploration' early on, and little noise when the gradients are small.

3.2 Convergence Plots

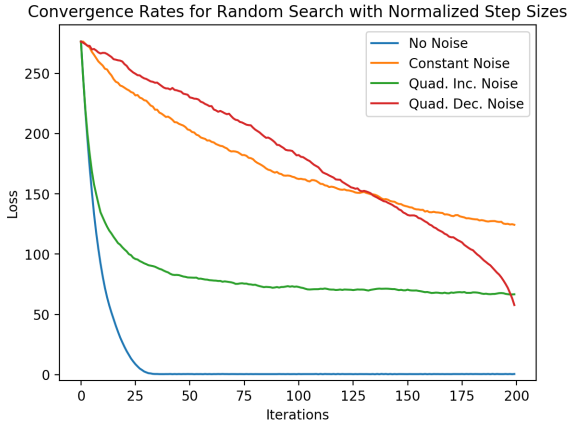
As a more rigorous exploration through convergence, I created plots to analysis overall convergence relative to the loss, $x^T Qx$ per iteration per noise function per optimization algorithm.

Baselines

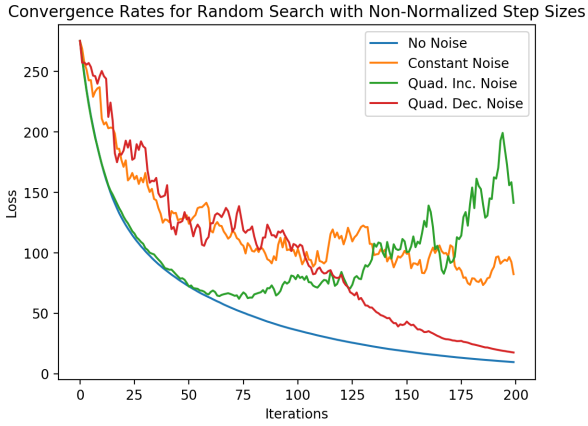
Non-Normalized

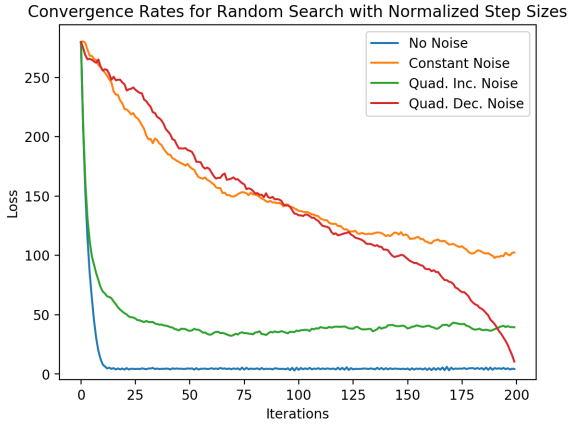


Normalized

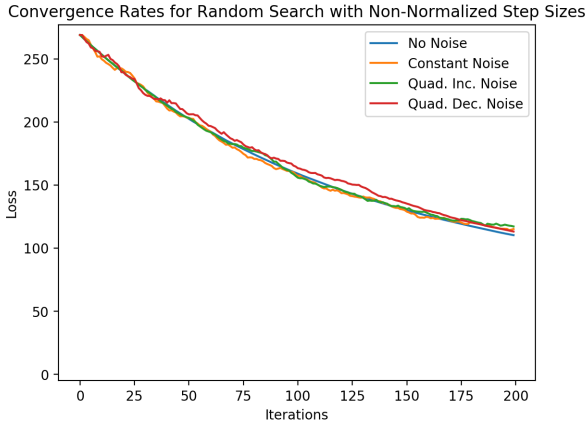


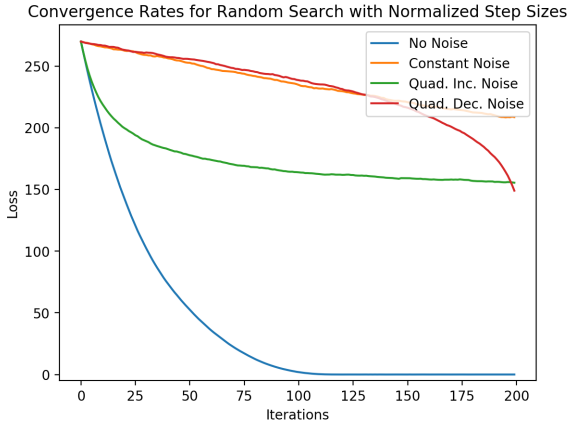
Increasing α



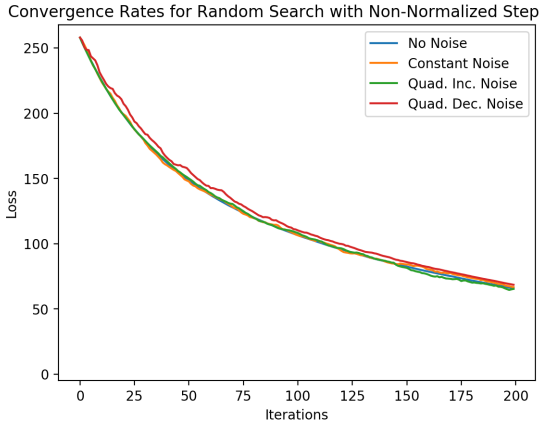


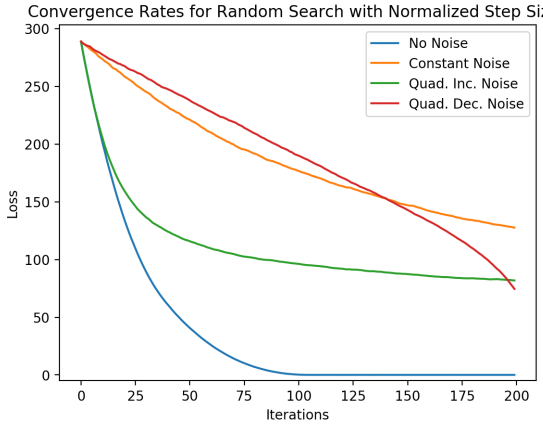
Decreasing α



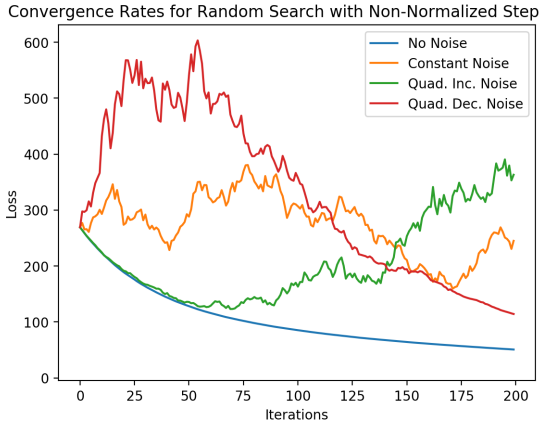


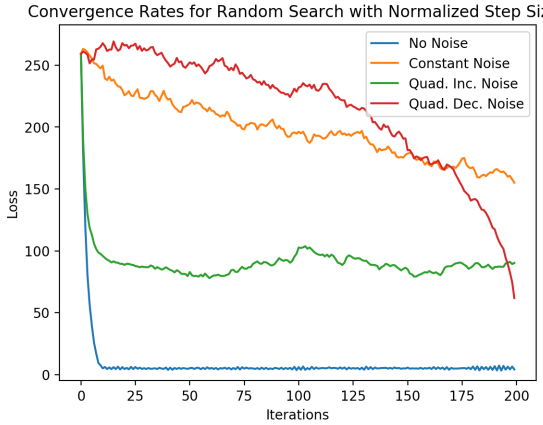
Increasing ν



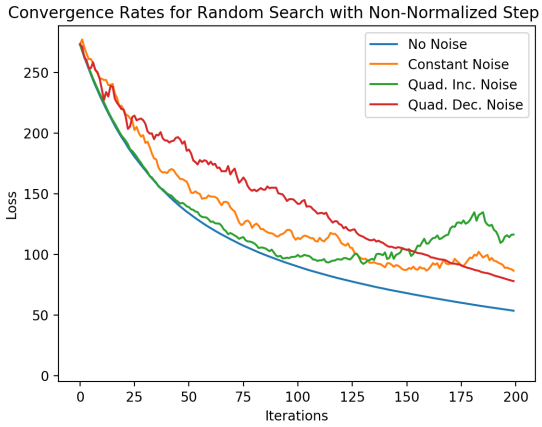


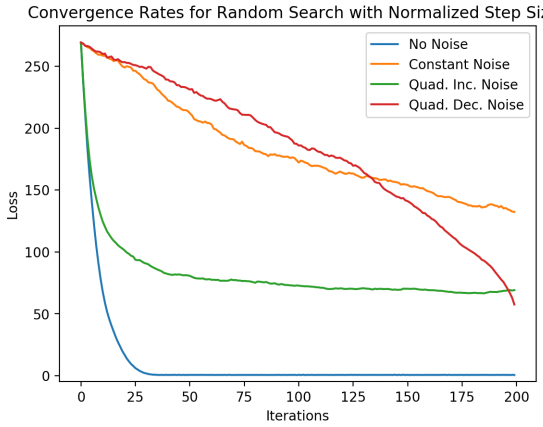
Decreasing ν



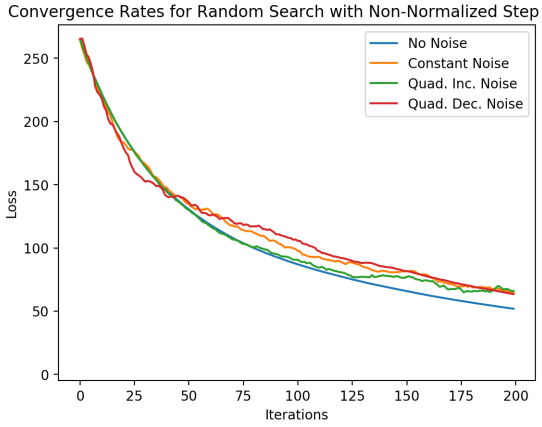


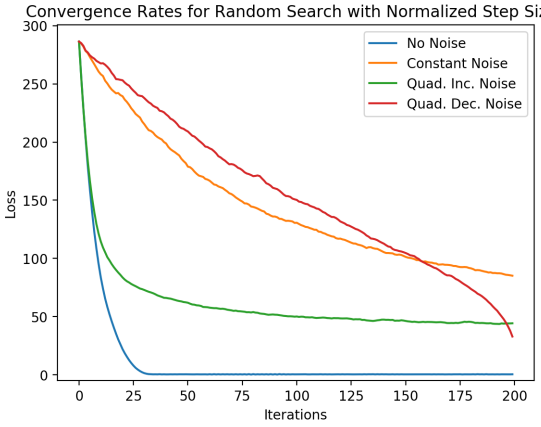
Increasing Maximum Noise



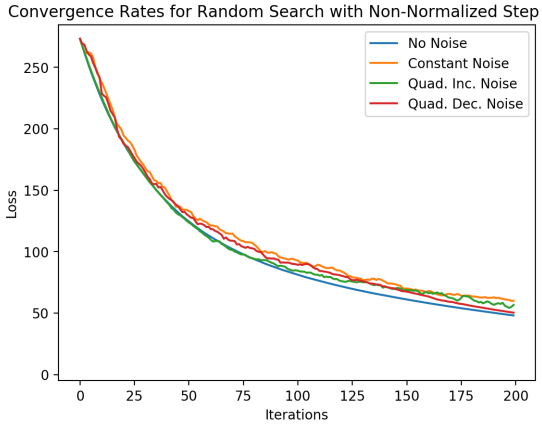


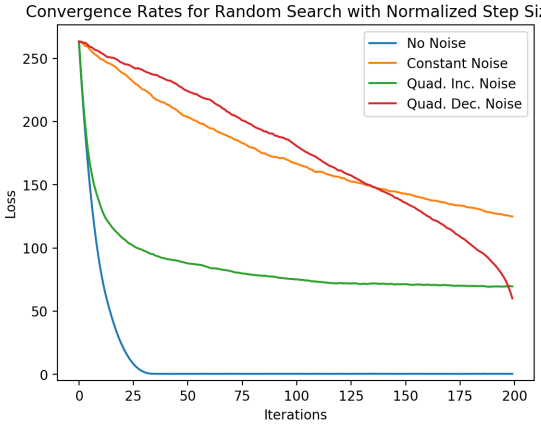
Decreasing Maximum Noise



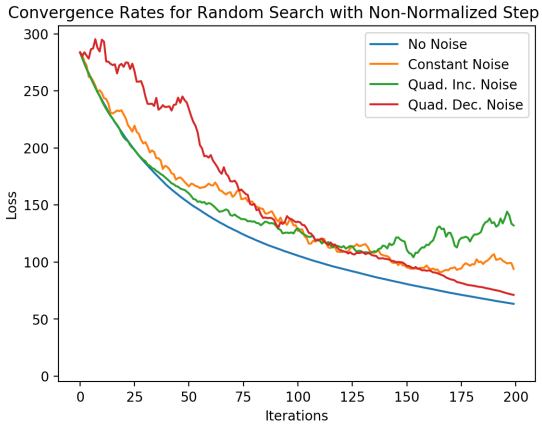


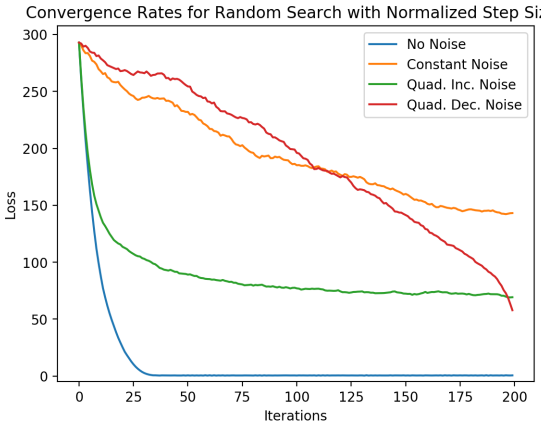
Increasing Number of Samples





Decreasing Number of Samples





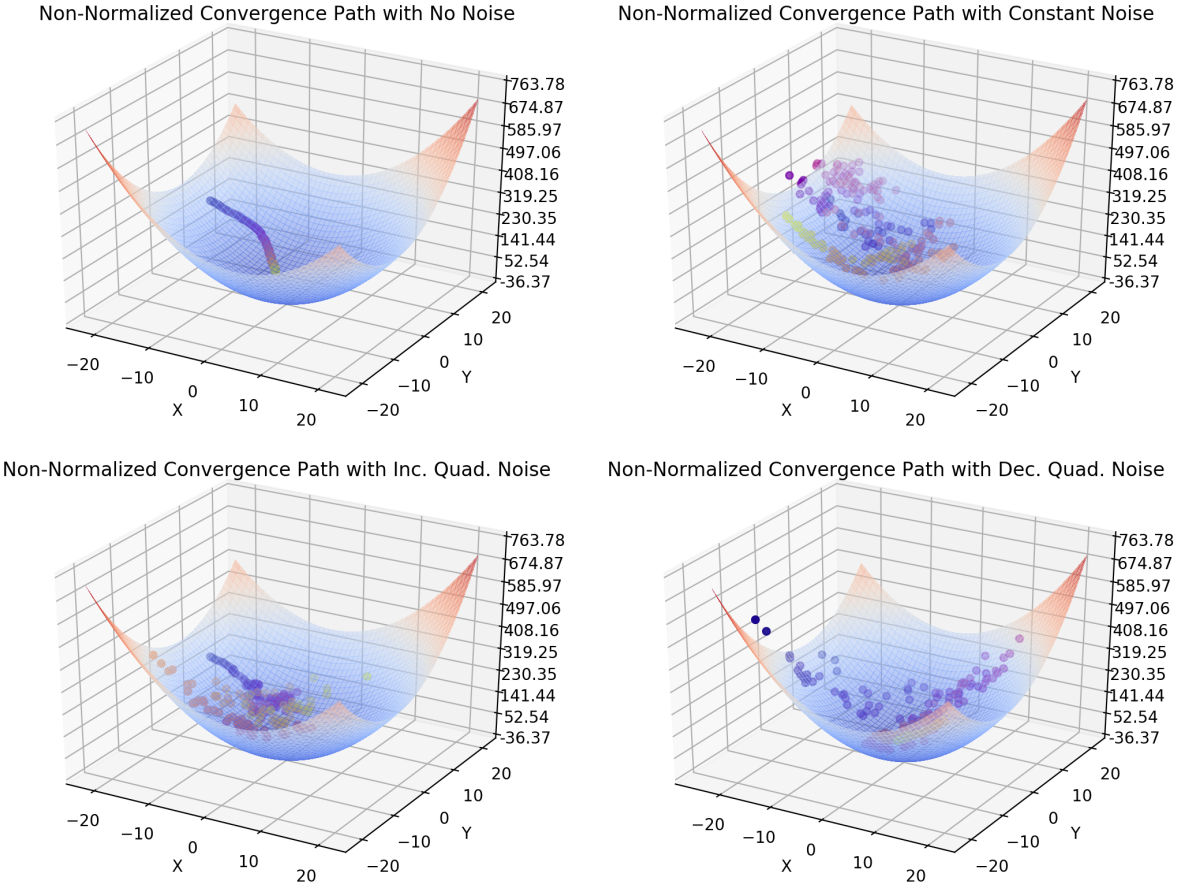
3.3 Parameter Analysis

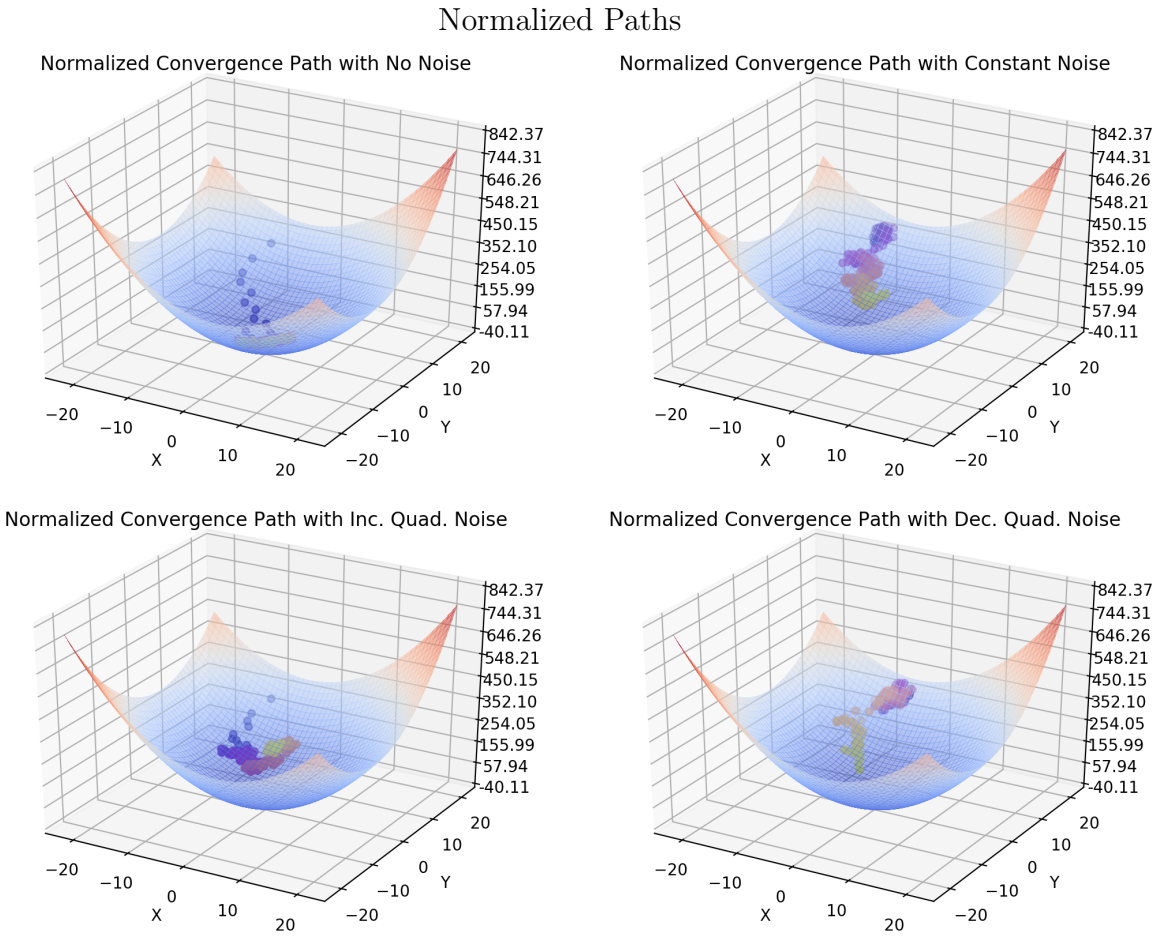
Baseline The first row of plots represents the baseline convergence rates. The normalized step sizes allow very rapid convergence for no noise situations, in fact, it converges to zero in all examples I explored. Thus this implies that normalizing step sizes is very helpful in low noise environments.

α

From the second and third row, we find that we have the luxury of increasing alpha with normalized step sizes. This makes sense, as the algorithm can continue to make equal steps even when the surface has minimal gradients. However, when experimenting with these larger step sizes, I found surprising results in observing individual paths.

Non-Normalized Paths





For non-normalized step sizes, the paths are incredibly wild and irregular, representing that the value of alpha is more sensitive with non-normalizing step sizes as expected. The paths with normalized step sizes are also a bit noisy, but still manage to perform well on average. Thus with normalizing step sizes, we may increase the value of α, allowing for quicker convergence. Also, it seems that with normalized step sizes and no noise, there are two hotspots of activity near the minima, suggesting that it just alternatives between the two locations where there is similar loss, but not at the true minima. This suggests that possible a decreasing learning rate/step size would be advantageous.

With decreasing alpha in the third row of loss plots, we see that not normalizing the step sizes results in very similar poor performances, and the normalized step sizes all perform mediocrely. Interestingly, it seems that constant noise also performs quickly poorly compared to the quadratic noise functions, it seems to decrease linearly.

ν

With increasing ν, there seemed to not have a large difference for normalized step sizes. For non-normalized step sizes, the various noise functions all performed almost identically, which imply that the large value of ν essentially negates the various noises, as large values of ν sample a large region around θ_j.

With small values of ν, this resulted in interesting results. It seems that without normalizing step sizes, the models perform very poorly, even going as far as to diverge away from the optimum such as the quadratic increasing noise case. It seems with small values of ν, the noise from queries are greatly amplified, since the BRS algorithm divides by ν, thus can lead to larger steps by virtue of the noise. Regularizing step sizes avoids this.

Maximum Noise
For increasing/decreasing maximum noise, the results are not quite interesting. For non-normalizing step sizes, it seems that greater noise results in poorer convergence, but normalizing results in more robust convergence. With lesser noise, overall convergence is better as expected.

Number of Samples
Lastly for the number of samples, with non-normalization, a larger number of samples essentially ignores the query noise by the law of large numbers. With fewer samples, it is more important to have lower amounts of noise closer to the optima, as demonstrated by the quadratic increasing/decreasing noise functions. With normalization, it seems that the models tend to perform better with more samples, as this just decreasing the noise in the prediction for the true gradient as expected. On the other hand, decreasing the number of samples does not have drastic effects. Looking at individual samples of convergence paths does, however, result in much more sporadic and irregular paths.

4 Conclusion

Overall, there are few conclusions to draw comparing normalized step sizes and non-normalized step sizes. First, low noise environments perform exceptionally well with normalized step sizes, as they are not prone to vanishing gradients close to the extrema. Additionally, normalized step sizes seem to be more robust to overall changes in hyperparameters, such as changing ν the noise to sample for the local gradient and the number of samples.

The seemingly most significant result is that the normalized step sizes may utilize a larger value of α and result in better overall convergence compared to non-normalization. With the normalization, one useful approach may be to have large values of α at the start, then exponentially decay over time.

More work must be done to exactly model the difference in convergence rates with these normalized step sizes, and the sensitivity to various parameters. A simple first step would be to consider a higher dimensional positive definite matrix. I chose a two-dimensional matrix, as this is the simplest and can be easily visualized. Moving to higher dimensions may broaden the differences between the algorithms.

Additionally, considering non-convex problems with local minima may illustrate greater differences in the noise functions. In this elementary data exploration, it seems that having a decreasing noise function is advantageous, as it allows for greater exploration early on and then confident steps near the minima. This may be beneficial with local minima in the convex case as well. Lastly, it may be interesting to consider matrices with eigenvalues very close to

zero. For these trials, I only considered two-dimensional matrices with eigenvalues at least 0.5 to consider sufficiently convex surfaces, although flatter surfaces with lower eigenvalues may be very interesting as well.

5 Appendix

For this, I implemented many small details for improving the overall pipeline for testing models. First, there are three shell scripts to automatically generate the plots shown. These are '`norm_loss_curves.sh`', '`non_norm_loss_curves.sh`', and '`surface_plots.sh`'. The loss curves take about 20 minutes to fully generate, and the surface plots are under a minute.

The program '`RandomSearch.py`' may be called through terminal and takes in a multitude of arguments:

--matrix_size: the size of the matrix, default set to 2

--alpha: the learning rate, default set to 0.01

--iters: the number of iterations, default set to 200

--init_mag: initialization magnitude for θ_0, default set to 20

--nu: the standard deviation for the query, default set to 0.01

--max_noise: the maximum noise for the oracle, default set to 30

--num_init: number initializations to process, default set to 0.01

--num_samples: number of directions to sample to calculate approximate gradient, default set to 0.01

--no-normalize: if specified, will **not** normalize the step sizes in the optimization algorithm, default is set to false, meaning it normalizes by default

--surface_plots: if specified, will save surface plots rather than loss curves, default set to False.

--verbose: will print out (x,y) location and value, default set to false

--show_plot: if specified, will display plots after generation, default set to false