NEWTON'S DESCENT

Matt Arenson, Ella Finstuen, Indiana Kretzschmar

APPM 4600,November 2024

Abstract

This paper explores Newton's Descent and optimization. Utilizing principles of numerical analysis, we will derive the method, discuss its strengths, limitations, and convergence properties, and then its Quasi variations, Lazy Newton and Broyden-Fletcher-Goldfarb-Shanno, now exclusively referred to as BFGS. Through numerical experimentation, we evaluate the methods' performance across different multivariate cost functions and their corresponding nonlinear systems. Additionally, in our independent extension we will demonstrate the applicability of Newton's Descent by using it to optimize stock weights in a portfolio, highlighting the method's versatility in real-world financial applications.

All code found in GitHub repository APPM4600_Project

Contents

1 Investigation of Newton's Descent

1.1 Introduction

Optimization problems algorithms are cornerstones of numerical analysis with wide-ranging applications in solving real-world problems. In this project, we investigate the optimization method of Newton's Descent. We will explore how Newton's Descent is derived, implemented, and applied to both theoretical and practical problems.

Newton's Descent is very similar to the well known method of steepest descent. This method iteratively minimizes the cost function $f(x)$ by following the gradient $-\nabla f(x)$, which points in the direction of the steepest descent, to approach the local minimum. In this way we merge root finding and steepest descent, as we are somewhat trying to find the root of the gradient. This method is mathematically expressed in (1), where λ is the step size.

$$x_{k+1} = x_k - \lambda \nabla f(x_k) \tag{1}$$

The negative sign ensures movement toward the local minimum rather than away from it. Although effective, steepest descent can be slow for poorly conditioned problems.

To address this limitation of steepest descent, Newton's method offers a faster alternative with quadratic convergence[12]. This method incorporates second-order information about the function, represented by the Hessian matrix. The Newton iteration is given in (2), where $\nabla^2 f(x_k)$ is the Hessian matrix:

$$x_{k+1} = x_k - \lambda (\nabla^2 f(x_k))^{-1} \nabla f(x_k). \tag{2}$$

In higher dimensions, the Hessian $\nabla^2 f(x_k) = H_i(x) \in \mathbb{R}^{d \times d}$ captures the curvature of the function. For a function $f : \mathbb{R}^3 \to \mathbb{R}$, the Hessian matrix $H_i(x)$ is a 3×3 symmetric matrix of second partial derivatives, shown in (3).

$$H_i(x) = \begin{bmatrix} \frac{\partial^2 f_i}{\partial x_1^2} & \frac{\partial^2 f_i}{\partial x_1 \partial x_2} & \frac{\partial^2 f_i}{\partial x_1 \partial x_3} \\ \frac{\partial^2 f_i}{\partial x_2 \partial x_1} & \frac{\partial^2 f_i}{\partial x_2^2} & \frac{\partial^2 f_i}{\partial x_2 \partial x_3} \\ \frac{\partial^2 f_i}{\partial x_3 \partial x_1} & \frac{\partial^2 f_i}{\partial x_3 \partial x_2} & \frac{\partial^2 f_i}{\partial x_3^2} \end{bmatrix} \tag{3}$$

Despite its superior convergence rate, Newton's method has several limitations. First, it requires the Hessian matrix to be positive definite, which is not guaranteed in all cases. It also is very computational expensive to compute with a time complexity of $O(n^2)$, due to its n^2 entries, and using the Hessian in Newton's Descent for solving the linear system in the problem, which will be introduced in the derivation, is $O(n^3)$[5]. Second, the method may fail to converge, potentially entering a cycle with multiple points. Finally, Newton's method may exhibit false convergence to saddle point or flat region rather than a local minimum under certain conditions.

We will now derive the method and present the mathematical background needed to understand Newton's Descent.

1.2 Mathematical Background and Derivation

Problem setup

We consider problem of optimizing the cost function $f : \mathbb{R}^n \to \mathbb{R}$ where the goal is to find x^* such that $\nabla f(x^*) = 0$. To achieve this, we use descent methods characterized by the iterative formula in (4), where p_k is the direction of descent, and α_k is the step size chosen to ensure $f(x_{k+1}) < f(x_k)$.

$$x_{k+1} = x_k + \alpha_k p_k \tag{4}$$

Two popular choices for p_k are the steepest descent direction, $p_k = -\nabla f(x_k)$, and the Newton direction written in (5).

$$p_k = -\big(\nabla^2 f(x_k)\big)^{-1} \nabla f(x_k) \tag{5}$$

Derivation of the Newton Direction

The Newton direction p_k is derived from a quadratic model of $f(x)$ centered at x_k. Using a second-order Taylor expansion[5], the function $f(x_k + p)$ is approximated in (6).

$$f(x_k + p) \approx f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p \tag{6}$$

We define the quadratic model $m_k(p)$ and minimize it with respect to p, using the first order optimality condition, in order to solve for the Newton direction p_k in (7).

$$\begin{aligned} m_k(p) &= f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p \\ \nabla m_k(p) &= \nabla f(x_k) + \nabla^2 f(x_k) p = 0 \\ p_k &= -\big(\nabla^2 f(x_k)\big)^{-1} \nabla f(x_k) \end{aligned} \tag{7}$$

We note that solving for p_k is aforementioned $O(n^3)$ operation.

Comparison to Steepest Descent

For steepest descent, we determine p_k by finding the direction that maximally reduces f based on the first-order derivative. The directional derivative of f along p is in (8).

$$\frac{d}{d\alpha} f(x_k + \alpha p)\bigg|_{\alpha=0} = \nabla f(x_k)^T p \tag{8}$$

Minimizing this expression with respect to p with the constraint $\|p\| = 1$, we find the direction of steepest descent $p_k = -\nabla f(x_k)$[4]. While steepest descent is straightforward and robust, it often converges slowly, particularly for poorly scaled problems. In this way the Newton direction is superior as it incorporates second-order information via the Hessian matrix to achieve quadratic convergence near the solution under favorable conditions.

Conditions for the Newton Direction

The Newton direction is reliable when the Hessian matrix, $\nabla^2 f(x_k)$, is positive definite, which ensures (9).

$$\nabla f(x_k)^T p_k < 0 \tag{9}$$

If $\nabla^2 f(x_k)$ is not positive definite, the direction p_k may not satisfy the descent property, potentially leading to divergence. In such cases, quasi-Newton methods or modifications to the Hessian may be employed to ensure reliable convergence[4].

1.3 Numerical Experiments

We will use 3 different functions and their linear system gradients to showcase Newton's Descent method to optimization problems. We will use a strictly convex function, a Gaussian function, and the Rosenbrock function. Newton's descent requires a 'good' initial guess, meaning it must be within the basin of convergence and a positive definite Hessian to converge, to find the optimizer with quadratic convergence. The basin of convergence is relative to the set maximum number of iterations and tolerance, as that alters which initial guesses may be converged from. Newton's convergence may very depending on function behavior, initial guess, and the conditioning of the Jacobian and Hessian.

In the following experiments the maximum number of iterations allowed is 100, the tolerance required is 1×10^{-12}. We define the necessary notation in Table 1.

Table 1

Symbol	Meaning
x^*	Optimizer we seek
$x^\diamond$	True optimizer of $f(x)$
x_0	Initial guess
$\|x^\diamond - x^*\|$	Norm of difference between $x^\diamond$ and x^*
$f(x)$	Cost function
$\nabla f(x)$	Gradient of $f(x)$
$\|\nabla f(x)\|$	Norm of $\nabla f(x)$
$J(x)$	Jacobian of $\nabla f(x)$
$H_i(x)$	i^{th} Hessian of $\nabla f(x)$
α	Asymptotic Convergence order
e_k	error at the k^{th} iteration

1.3.1 Strictly convex function

We will first consider the strictly convex cost function $f : \mathbb{R}^3 \to \mathbb{R}$:

$$f(x) = e^{x_1} - x_1 + e^{x_2} - x_2 + e^{x_3} - x_3. \tag{10}$$

The necessary components of $f(x)$ for Newtons Descent are:

$$\nabla f(x) = \begin{bmatrix} e^{x_1} - x_1 \\ e^{x_2} - x_2 \\ e^{x_3} - x_3 \end{bmatrix}, \quad J(x) = \begin{bmatrix} e^{x_1} & 0 & 0 \\ 0 & e^{x_2} & 0 \\ 0 & 0 & e^{x_3} \end{bmatrix},$$

$$H_1(x) = \begin{bmatrix} e^{x_1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad H_2(x) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & e^{x_2} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad H_3(x) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & e^{x_3} \end{bmatrix}.$$

When running Newton's code on this $f(x)$, we are trying to find the local optimizer of the cost function $f(x)$, which we can trivially see is $x^\diamond = [0, 0, 0]$, as $f(x^\diamond) = 3$. Due to the convex structure of the function we predict that any initial guess will eventually converge, and for it to converge in 100 iterations it must be in some neighborhood of $x^\diamond$. We will take the initial guess of $x_0 = [1.5, 1.6, 0.8]$ and perform Newton's Descent to find x^*. Table 2 lists the results from the Newton's Descent performed on $f(x)$ while the semiology plot of error in Figure 1 and table of error in Table 3 illustrate the quadratic convergence.

Table 2

	Results
x^*	$[4.59182384e-17, -5.00438520e-17, 1.06632741e-16]$
$f(x^*)$	3.0
$\|\nabla f(x^*)\|$	0.0
α	1.9842062164967555

Figure 1

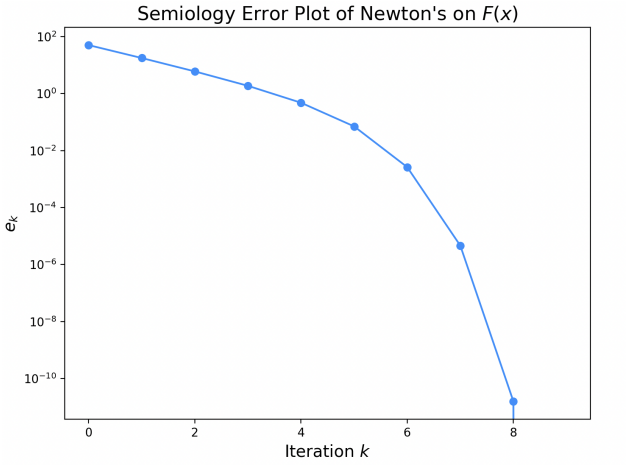


Table 3

$\boldsymbol{k}$	$\|x^\diamond - x^*\|$
0	2.3345235059857505
1	1.6322072186562357
2	1.0195837759600181
3	0.5294631445857241
4	0.19343118236516146
5	0.03406582076993969
6	0.0013137852399133783
7	2.313970266202027e-06
8	7.907506712927505e-12
9	1.2642552423221103e-16

As seen in Table 2, the α found numerically is $1.98 \approx 2$, and this makes sense based off the dramatic and swift change from x_0 to x^* through the iterations, further depicted in Figure 1 and Table 3. As $f(x^*) = 3.0$ and $\|\nabla f(x^*)\| = 0.0$ exactly, Newton's method converged not only quickly but almost exactly to $x^\diamond$. The quadratic convergence and results align with the cost function's smoothness, positive definite Hessian, and simple gradient. The convergence may also be seen in the 3D plot of Figure 2 and the Contour map of Figure 3, which depicts the simple path that the algorithm takes to find the solution with the blue dots on the contour.

Figure 2

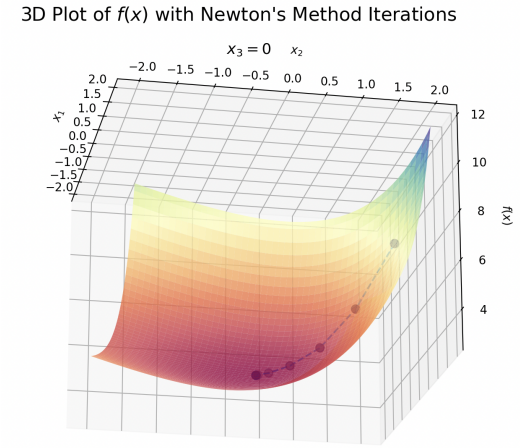
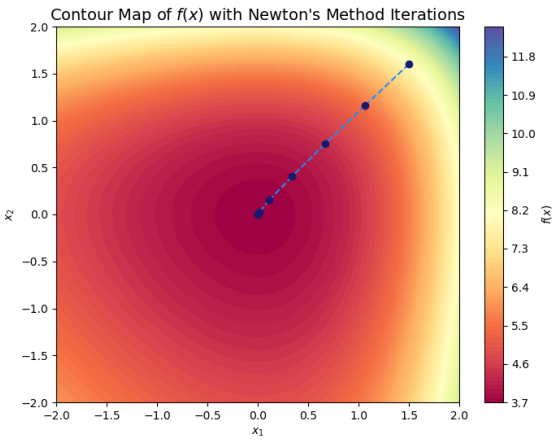


Figure 3



While this is an optimizing problem, its also important to illustrate how the optimizing done in Newton's Descent relates to root finding. As explained previously, finding the local minimums or maximums means finding where $\|\nabla f(x)\| = 0$. Thus we can interpret this problem as trying to find the point where the the functions of $\nabla f(x)$ intersect, called $x^{\diamond}$, such that $\|\nabla f(x^{\diamond})\| = 0$. The solution we find to this problem is x^*. Therefore we are doing a form of root finding on $\nabla f(x)$. Figure 4 illustrates how x^* found previously in Table 2 finds the 'root' of the functions of $\nabla f(x)$ and Figure 5 illustrates how the functions of $\nabla f(x)$ intersect at x^*.

Figure 4

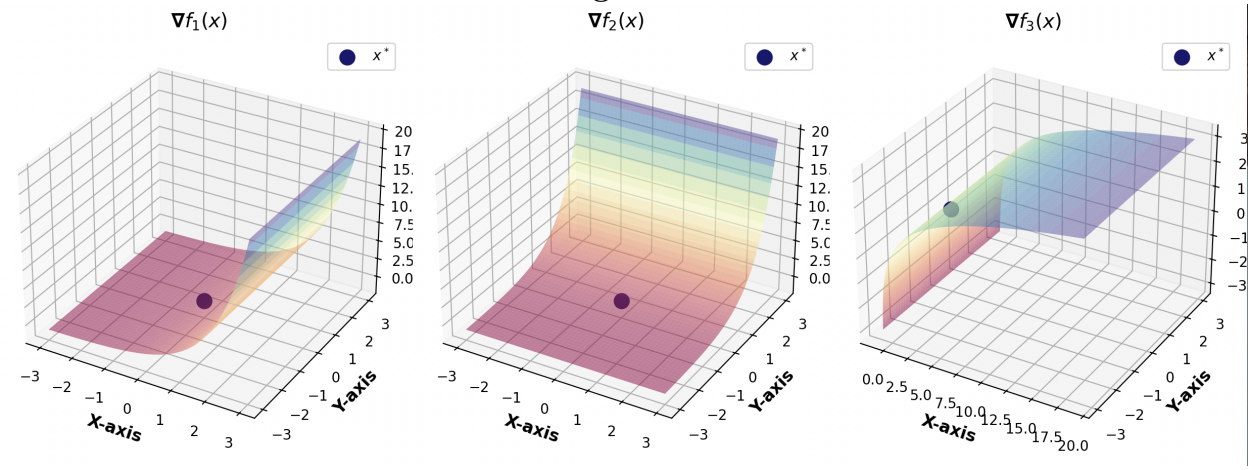
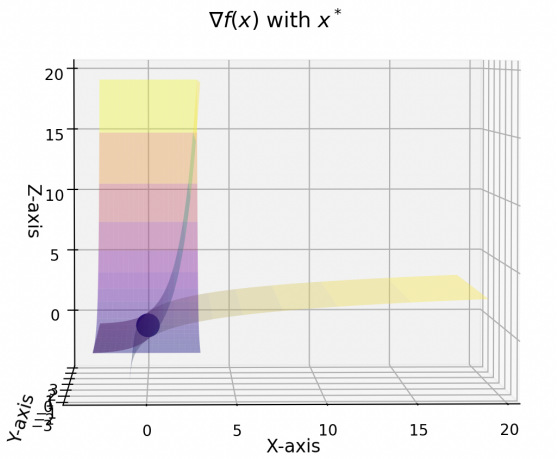
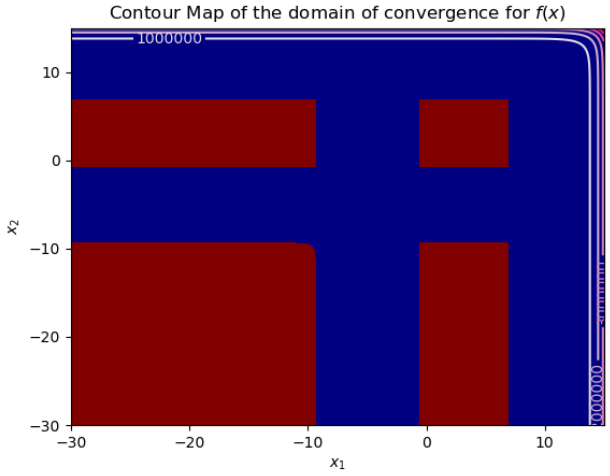


Figure 5



We would also like to determine which initial guesses are 'good' enough. The basin of convergence illustrated in Figure 6 allows for a maximum of 20 iterations. A plot depicting the basin of convergence like Figure 6, where the red points will converge and the blue points will diverge, is important in an investigation of this sort because it illustrates the impact of initial guess on convergence. These plots may also be helpful in finding a converging initial guess for poorly conditioned functions, which will be experimented with later.

Figure 6



Lastly, because Newton's Method is suitable $\forall n \in \mathbb{N}$ such that $f : \mathbb{R}^n \to \mathbb{R}$, we will show an example for another n. We will show Newton's Descent on the strictly convex function $f : \mathbb{R}^6 \to \mathbb{R}$:

$$f(x) = e^{x_1} - x_1 + e^{x_2} - x_2 + e^{x_3} - x_3 + e^{x_4} - x_4 + e^{x_5} - x_5 + e^{x_6} - x_6. \tag{11}$$

We will use the initial guess of $x_0 = [1.9, 1.7, 0.8, 1.4, 0.6, 1.3]$ and suspect that $x^\diamond = [0, 0, 0, 0, 0, 0]$. Newton's Descent produces the results seen in Table 4, along with the semiology error plot in Figure 7 and the table of error in Table 5.

Table 4

	Results
x^*	$[-2.33023264e-17, 9.41748815e-17, 1.06632741e-16,$ $-1.14967087e-17, 3.32415301e-17, 7.84273788e-17]$
$f(x^*)$	6.0
$\|\nabla f(x^*)\|$	$1.5099033134902243e-14$
α	1.9743320737319414

Figure 7

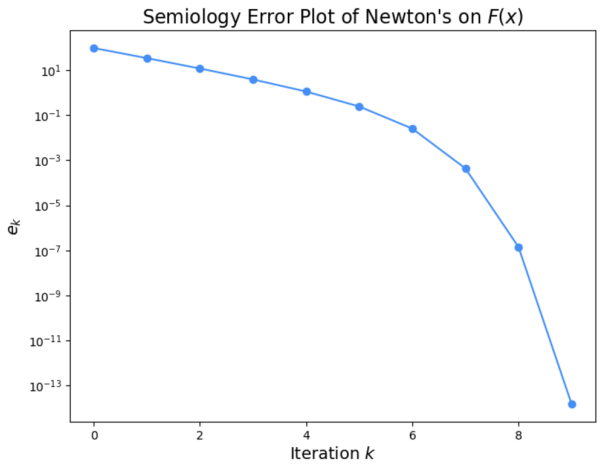


Table 5

$\boldsymbol{k}$	$\|x^\diamond - x^*\|$
0	3.3391615714128
1	2.373477506465968
2	1.5348474346626941
3	0.8584766346925584
4	0.37668051368735417
5	0.10699605761362846
6	0.012500038231001993
7	0.00021752161662409784
8	7.0683433462097e-08
9	7.52804943393937e-15

The results seen in Table 4, Figure 7, and Table 5 correspond to the quadratic convergence and characteristics of the function we discussed for Newton's on the convex function $f : \mathbb{R}^3 \to \mathbb{R}$. Newton's Descent effectively found a suitable x^* even for $f : \mathbb{R}^6 \to \mathbb{R}$, showing the algorithm's variability and capabilities.

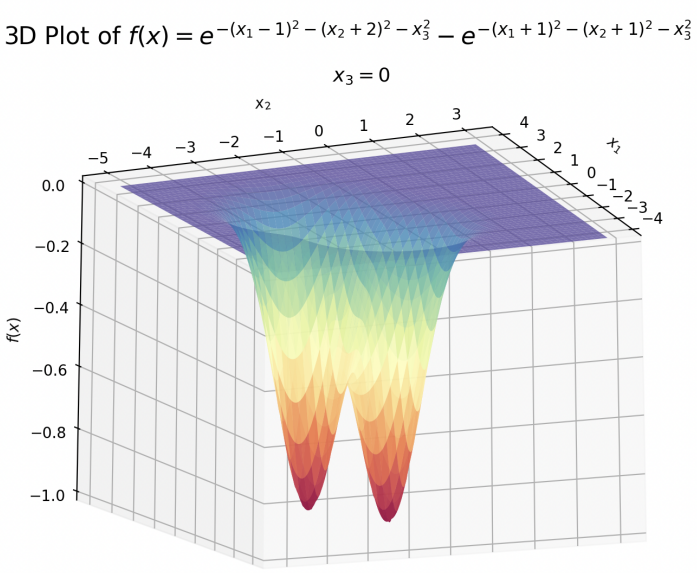
1.3.2 Gaussian Function

We will now consider the Gaussian function:

$$f(x) = e^{-(x_1-1)^2-(x_2+2)^2-x_3^2} - e^{-(x_1+1)^2-(x_2+1)^2-x_3^2}. \tag{12}$$

This function is a type of Gaussian function, where there will be a two local optimizers of $x^{\diamond_1} = [1, -2, 0]$ and $x^{\diamond_2} = [-1, -1, 0]$, where $f(x^{\diamond_1}) = f(x^{\diamond_2}) = -1$. Figure 8 illustrates the function where $x_3 = 0$.

Figure 8

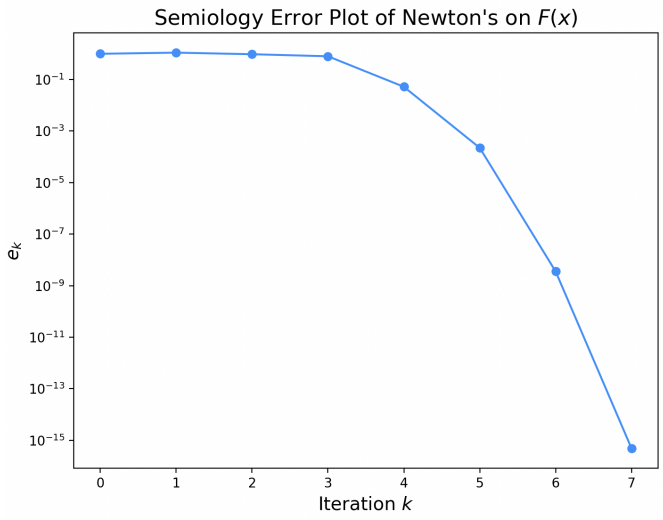


From Figure 8, we suspect that initial guesses within some neighborhood around the local minimum will converge to that respective point. We also suspect that there is possibility for unique behavior in the valley between the two minimums, and additionally there could be false convergence to the flat portion of the surface. We will first test a point we suspect to converge to one of the minimums, then a point we expect will converge falsely, and then determine the respective basins of convergence. First we will choose $x_0 = [0.85, -1., 0.01]$. We expect it will quadratically converge to $x_1^{\diamond}$ because of the smoothness of the region and it seems relatively close to the this minimum compared to the distance from $x_2^{\diamond}$. The experiment results may be seen in Table 6 and the experiment's error plot in Figure 9.

Table 6

	Results
x^*	$[0.98562387, -1.99281194, 0.0]$
$f(x^*)$	-1.0069799283897052
$\|\nabla f(x^*)\|$	$4.874285804879884e-16$
α	2.023846980444727

Figure 9



As suspected, this choice of x_0 led to the quadratic convergence towards $x_1^{\diamond}$. This corresponds to what we discussed previously about Newton's Descent's convergence in section 1.3.1. The path that Newton's Descent took is illustrated on the 3D plot in Figure 10 and on the contour plot in Figure 11.

Figure 10

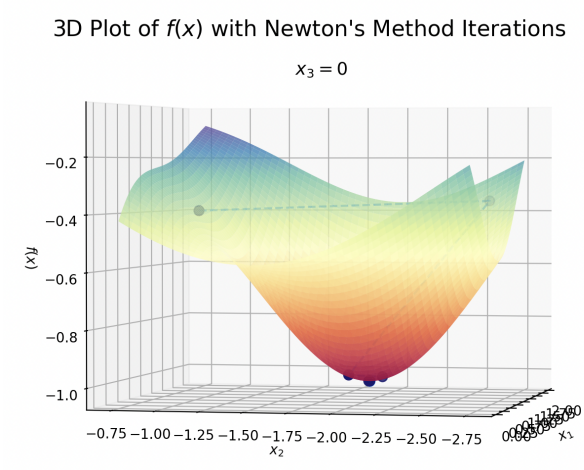
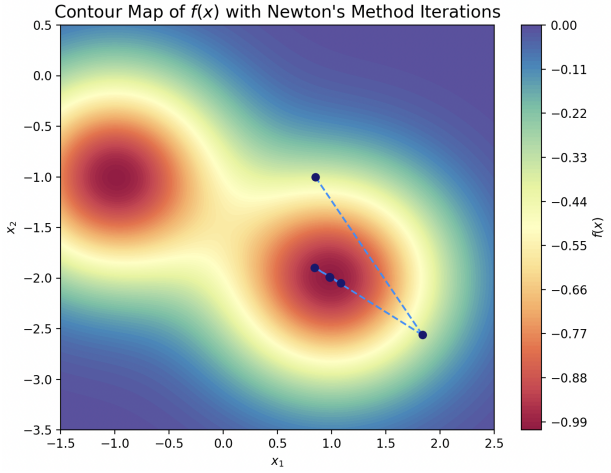


Figure 11

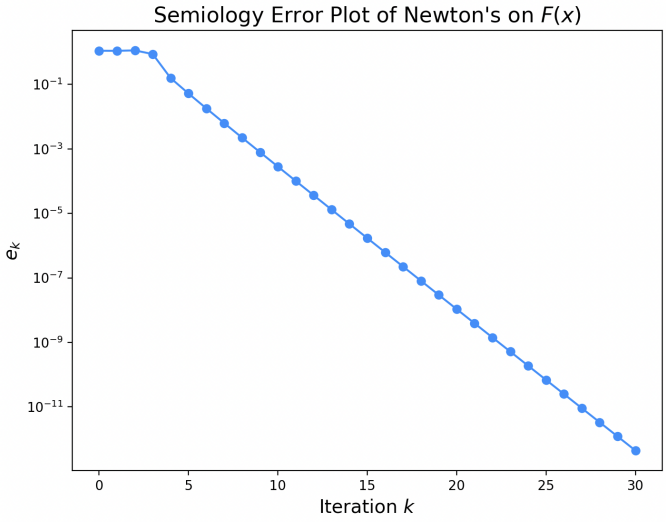


It is quite interesting that the path goes across the walls of the surface around the minimum before converging. We are interested in further discussion of the initial guess and the basin of convergence, so now we will test another point on a steeper part of the surface, $x_0 = [1.6, -1.8, 0.8]$, We suspect that this points location will cause Newton's Descent to falsely converge to the flat region because of the steepness of the function. The experiment results may be seen in Table 7 and the experiment's error plot in Figure 12.

Table 7

	Results
x^*	$[-1.52420576, -3.06044018, -3.7160689]$
$f(x^*)$	-1.1514807839195139e-08
$\|\nabla f(x^*)\|$	$4.405680964875033e-13$
α	0.9996780956076323

Figure 12



Despite the apparent convergence, this experiment does not converge to the location of a local max or min. This can be verified in two ways, firstly we can see in Table 7 that $f(x^*)$ is not approximately -1 but 0 which is not a local max or min. Secondly, the iterations can be plotted in 3D space on the surface, as seen in Figure 13.

Figure 13

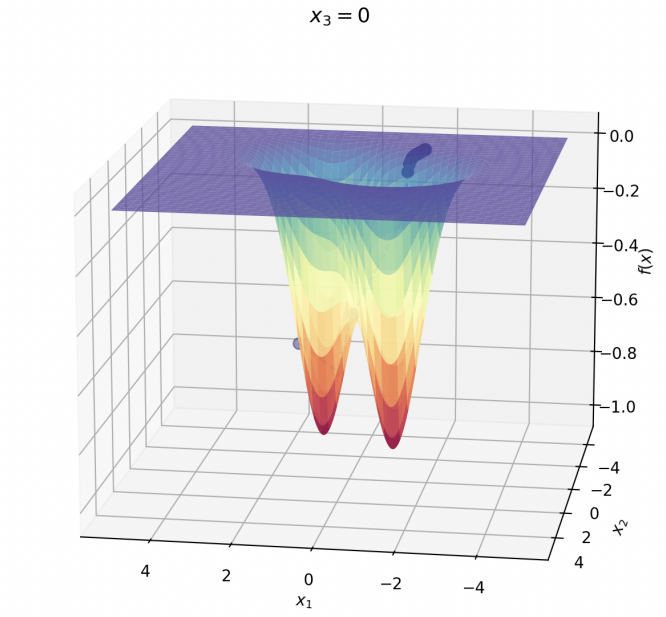
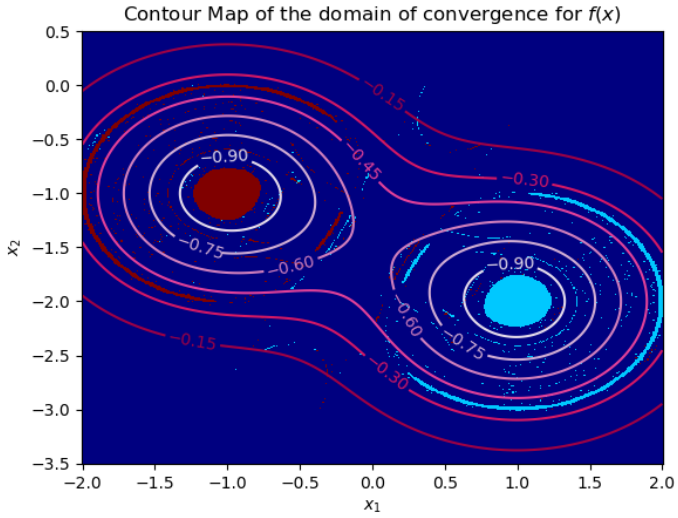


Figure 13 verifies that rather than converging to the location of one of the local minimums, Newton's Descent falsely converged to the flat surface. This is possible because our implementation does not verify that convergence is to local max or min, but rather that convergence is to a point where $\|\nabla f(x^*)\| \approx 0$, and in a flat region this is true. This example points out an aspect of optimizations problems that programmers must be careful of, it must be verified that you are converging to a local maximum or minimum for more precise results.

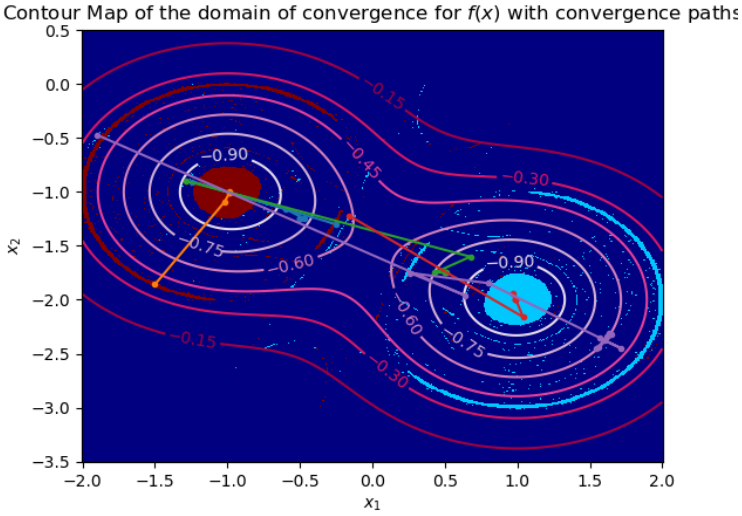
Lastly, we wish to create a plot of the basins of convergence for the minimums that only shows convergence to the minimums and not false convergence. This plot is depicted in Figure 14.

Figure 14



In Figure 14, the light blue points on the plot converge to $x^{\diamond 1} = [1, -2, 0]$, the red points converge to $x^{\diamond 2} = [-1, -1, 0]$, and the dark blue points diverge or converge falsely to the flat region. We notice that some points that we would expect to converge to one of the minimums due to their location and proximity to that minimum, converges to the other. Figure 15 depicts the convergence paths of various interestingly converging points on the contour map.

Figure 15



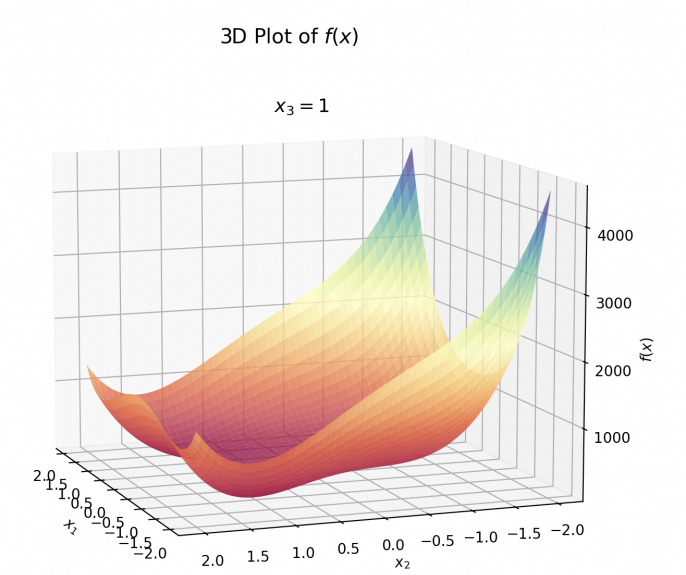
1.3.3 The Rosenbrock Function

Lastly, we will briefly consider the Rosenbrock Function:

$$f(x) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2 + 100(x_3 - x_2^2)^2 + (x_1 - 1)^2. \tag{13}$$

There will be two local optimizers, $x^{\diamond_1} = [1, 1, 1]$ and $x^{\diamond_2} = [-1, 1, 1]$, where $f(x^{\diamond_1}) = f(x^{\diamond_2}) = 0$. Figure 16 illustrates the function where $x_3 = 1$.

Figure 16



From Figure 8, we suspect that it will be very difficult to get Newton's descent to converge on $f(x)$ due to its extreme flatness and valley like characteristics near the minimums and dramatic steepness at the walls of the surface. The hypothesis is that the algorithm could get stuck bouncing around in the valley and may never converge or take many iterations to. We will test this hypothesis with $x_0 = [1.1, 0.9, 0.8]$. As this is in the flat valley at the bottom, we expect this will diverge. The experiment results may be seen in Table 8, the experiment's error plot in Figure 17, and the contour plot of the algorithm's diverging path in Figure 18.

Table 8

	Results
x^*	$[1.0, 1.0, 1.0]$
$f(x^*)$	$2.824368559724104e - 28$
$\|\nabla f(x^*)\|$	$7.169026506947216e - 11$
Error Message	1

Figure 17

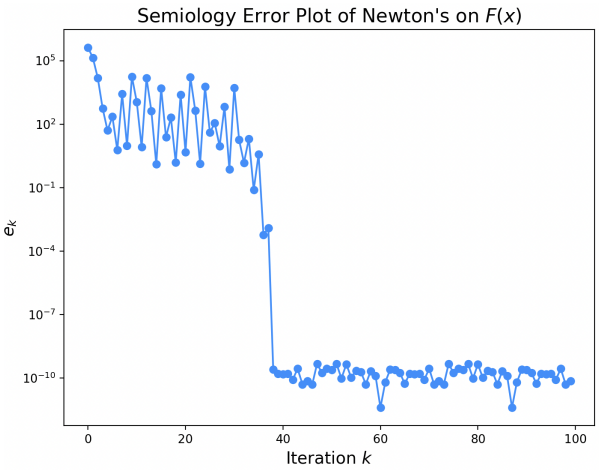
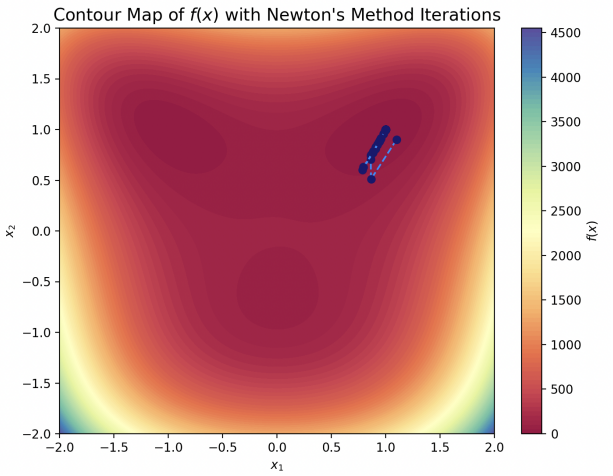
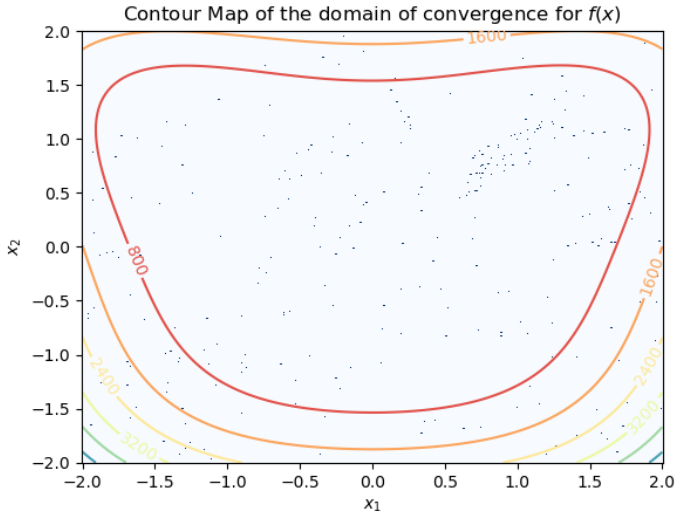


Figure 18



It is apparent from the error message of 1 meaning divergence in Table 8, the sporadic plot of error in Figure 17, and the points bouncing around on the contour in Figure 18, that the experiment diverged. The hypothesis that the path would bounce around in the flat valley was correct and can be seen in Figure 18 and inferred from the low errors after approximately iteration 40 in Figure 17. This example demonstrates the difficult to converge on this function. Even though x_0 was relatively close $x^{\diamond_1}$, this function has poor behavior and the Hessians will be poorly conditioned. This makes us curious about what the basin of convergence may be. Figure 19 displays the plot of the basin of convergence.

Figure 19



In the contour plot in Figure 19, white points are points that diverged and the navy points are ones that converged. Figure 19 is interesting because it shows how this badly behaved function with poorly conditioned properties has a seemingly stochastic basin of convergence. It demonstrates how Newton's Descent method ping pongs around in this valley and very seldom converges. There are many interesting independent extensions possible to further investigate how to navigate this apparent randomness, but we will not delve into it in this paper.

1.4 Quasi-Newton Descent

One of the previously mentioned limitations of Newton's Descent is the computationally expensive Hessian. While the quadratic convergence of Newton's Descent is great, it is computationally expensive to compute the Hessian every iteration. As can be expected, for $f : \mathbb{R}^n \to \mathbb{R}$ it becomes more expensive to computer as n gets larger due to the aforementioned $O(n^3)$ cost of computing and using the Hessian in Newton's Descent. To combat this, we consider the Quasi Newton methods BFGS and Lazy Newton.

1.4.1 BFGS

BFGS is very similar to Newton's method with a difference of how it computes the Hessian. Rather than directly computing the Hessian, BFGS approximates the Hessian with a matrix B, that uses rank one updates at every iteration. B must be positive definite, satisfy (14) at every iteration, and be relatively "close" to the previous B, determined by the norm of their difference.

$$B_{k+1}\left[\mathbf{x}_{k+1} - \mathbf{x}_k\right] = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) \tag{14}$$

The rank one update for BFGS is written in (15) [7].

$$B_{k+1} = B_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \Delta \mathbf{x}_k} - \frac{B_k \Delta \mathbf{x}_k \Delta \mathbf{x}_k^T B_k}{\Delta \mathbf{x}_k^T B_k \Delta \mathbf{x}_k} \tag{15}$$

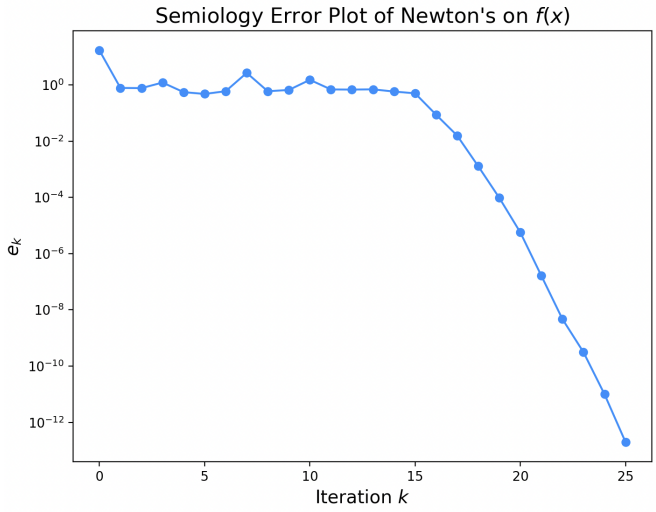
Because (15) is an approximation, we do need to worry about numerical instability. To combat this we incorporate line search, which is an algorithm that determines the appropriate step size, appropriateness based on a sufficient decrease and improving numerical stability [8]. The benefit of this is that for systems with non positive definite Hessians or bad initial guesses, BFGS can work around this and most likely converge [10] by approximating the Hessian instead. The caveat of this method is that it converges much slower, as it does not use the precise Hessian to find a solution.

We can compare the implementation of BFGS on the strictly convex function $f : \mathbb{R}^3 \to \mathbb{R}$ in (10). We will use the same $x_0 = [1.5, 1.6, 0.8]$ from the original experiment with this $f(x)$ found in (10). When testing BFGS on $f(x)$, we suspect that the convergence will be slower than before as the information used in the direction is now approximated. Table 9 and Figure 20 depict the performance of BFGS on $f(x)$.

Table 9

	Results
x^*	$[3.19846847e-16, -5.91823608e-16, 6.71028948e-16]$
$f(x^*)$	3.0
$\|\nabla f(x^*)\|$	$1.985257587845444e-13$
α	1.2719824150869175

Figure 20



Recall that the order of converge for this experiment using regular Newton's Descent was quadratic, seen in Table 2. As expected and verified by Table 7 and Figure 20, the convergence of BFGS on $f(x)$ with same initial guess and tolerance has much slower convergence and takes many more iterations. From Table 7 we deduce that BFGS exhibits super linear convergence as the convergence order is greater than 1. This example demonstrates how even though it is favorable to perform less computationally expensive algorithms, this typically means slower convergence and more iterations because it uses more robust techniques.

1.4.2 Lazy Newton

The difference between Lazy Newton's Descent and Newton's Descent is how often the Hessian is computed. Rather than computing the Hessian every iteration, Lazy Newton uses a criterion, in this example the tolerance being when $\|\nabla f(x)\| > 1 \times 10^{-4}$, to determine when to calculate the Hessian. The point of using this 'lazy' method rather than regular Newton's Descent is to limit the expensive Hessian computation. The downside of this is significantly slower convergence and more possible divergence since the Hessian may not be close to what it should be at that iteration [10]. The consequence of this is that the Hessian may not be very effective at capturing the curvature of a function adequately because it may be out of date depending on the Hessian calculation criterion.

We will again compare the implementation of Lazy Newton on the strictly convex function $f : \mathbb{R}^3 \to \mathbb{R}$ in (10) with the same $x_0 = [1.5, 1.6, 0.8]$ from the original experiment. Again, we hypothesis that convergence will be slower and more iterations will be needed, because Lazy Newton's is may not have the up-to-date curvature of the function. Table 10 and Figure 21 depict the performance of Lazy Newton on $f(x)$.

Table 10

	Results
x^*	$[-8.58146725e-17, 1.64096895e-17, 2.42678663e-17]$
$f(x^*)$	3.0
$\|\nabla f(x^*)\|$	$1.1550588755362437e-14$
α	0.9996179669122286

Figure 21

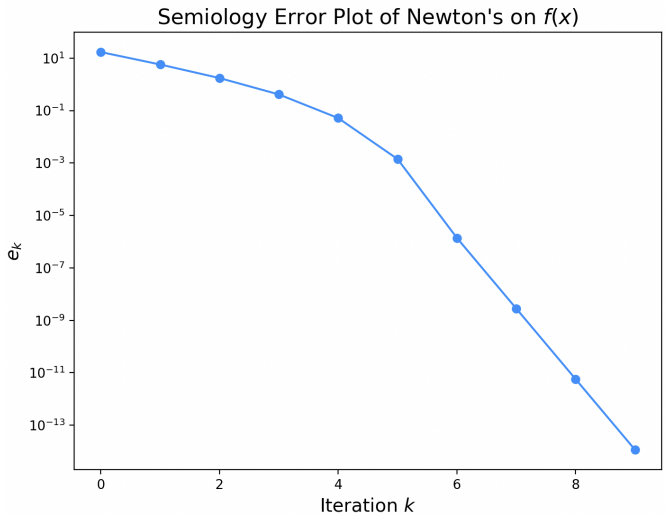


Table 10 and Figure 21 verify the slower convergence, linear in this case, of Lazy Newton compared to original Newton's Descent. This result aligns with our hypothesis that Lazy Newton's out of date information lowers the convergence order.

1.5 Conclusions

In this investigation of Newton's Descent we introduced the Newton's Descent through derivation and discussed its strengths, limitations, cost, and convergence before demonstrating these hypotheses through numerical experiments. We also experimented with Quasi newton methods such as BFGS and Lazy Newton to create solutions for Newton's Descent limitations. Through derivation we learned that the iterative method uses first and second order derivatives to approximate the root or find minimum of function by taking steps proportional to the local curvature. Additionally, through this derivation we discovered the $O(n^3)$ cost of the method and that a positive definite Hessian is necessary for convergence.

In our numerical experiments we showcased the performance on a strictly convex function, a Gaussian function, and the Rosenbrock function. The variety of examples allowed us to explore the order of convergence, basin of convergence, paths to convergence, how function attributes and initial guess affect convergence, and how the optimization problem relates to root finding. We conclude that Newton's Descent converges quadratically, is sensitive to initial guess and poorly conditioned Hessian, which is impacted by the varying steepness of the function, and can be expensive with the Hessian computation. To combat the expensive Hessian computation, we investigated the quasi newton methods BFGS and lazy Newton. These methods are less expensive but much slower, showing that efficiency and cost must be weighed when deciding which algorithms to use.

Our experiments allowed us to make the conclusion that while Newton's Descent is effective and fast for optimizing well-attributed cost functions, it may fail to converge for a poor initial guess or poorly conditioned function. Flat regions may cause false converges or divergence if the algorithm gets stuck. Newton's method efficiently optimizes the given cost function, and it is up to the user to decide whether their computational priorities lie in higher order of convergence and less expensive processes.

2 Extension to Portfolio Optimization

2.1 Introduction

We will now utilize what we have learned about Newton's Descent and its capabilities concerning optimization problems to find optimize a stock portfolio. By incorporating historical financial data into our method, our goal is to optimize portfolio weights to minimize risk while achieving desired returns, showcasing the usefulness of descent methods in real-world problems.

We introduce the Sharpe ratio, "the ratio of the excess expected return of an investment to its return volatility or standard deviation"[9]. This ratio compares investments with different levels of risk and return for the optimization of portfolios. We can use Newton's descent to minimize the negative of the Sharpe ratio to identify the best allocation of stocks for portfolio performance. In this computation, we apply the root finding problem as an optimization problem to the various combinations of weights to efficiently find the solution with the highest risk-adjusted return[1].

In this section we will showcase how the problem is built and how Newton's Descent is applied to the historical stock data, run experiments using this application, and discuss the efficiency and applicability of Newton's Descent to this real world optimization problem.

2.2 Showcasing the optimization problem

Problem Definition

In this section of our project, we will use Newton's method to optimize portfolio weights by maximizing the *Sharpe ratio*. The Sharpe ratio, defined as the ratio of excess portfolio return to portfolio risk, is an important tool metric in modern portfolio theory for analyzing risk-adjusted returns:

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p}[9], \tag{16}$$

where:

- $R_p = \mathbf{w}^T \boldsymbol{\mu}$ is the portfolio return, dependent on the weights $\mathbf{w}$ and the expected returns $\boldsymbol{\mu}$ [2],
- R_f is the risk-free rate, and
- $\sigma_p = \sqrt{\mathbf{w}^T \Sigma \mathbf{w}}$ is the portfolio volatility, determined by the weights $\mathbf{w}$ and the covariance matrix Σ [2].

Since the Sharpe ratio involves a non-linear function of the weights, we use *Newton's method* to efficiently solve this optimization problem.

Newton's Method for Sharpe Ratio Maximization

In order to apply Newton's method to optimizing portfolio weights, we set the objective to maximize the Sharpe ratio. Equivalently, we minimize its negative:

$$f(\mathbf{w}) = -\frac{R_p - R_f}{\sigma_p}.$$

At each iteration, Newton's method updates the portfolio weights ($\mathbf{w}$) giving us equation (17):

$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{current}} - H^{-1} \nabla f(\mathbf{w})[11], \tag{17}$$

where:

- $\nabla f(\mathbf{w})$ is the gradient of the objective function (negative Sharpe ratio),
- H is the Hessian matrix (second derivative of the objective function with respect to the weights),
- $H^{-1}\nabla f(\mathbf{w})$ determines the direction and step size of the weight updates.

(The Hessian and the gradient of the negative sharp ratio can be seen below.)

1. Gradient ($\nabla f(\mathbf{w})$)

Using the quotient rule, the gradient of $f(\mathbf{w})$ (the negative Sharpe ratio) is derived as equation (18):

$$\nabla f(\mathbf{w}) = \frac{-\nabla R_p \cdot \sigma_p - (R_p - R_f) \cdot \nabla \sigma_p}{\sigma_p^2} [11]. \tag{18}$$

Breaking this into components:

- $\nabla R_p = \boldsymbol{\mu}$: The gradient of the portfolio return is the vector of mean returns.
- $\nabla \sigma_p = \frac{\Sigma \mathbf{w}}{\sigma_p}$: The gradient of the standard deviation involves the covariance matrix Σ and the portfolio weights $\mathbf{w}$ [11].

Substituting these into the gradient formula:

$$\nabla f(\mathbf{w}) = -\frac{\boldsymbol{\mu}}{\sigma_p} + \frac{(R_p - R_f) \cdot (\Sigma \mathbf{w})}{\sigma_p^3}. \tag{19}$$

This provides the direction of steepest ascent or descent of the Sharpe ratio.

2. Hessian (H)

The Hessian (H) captures the second-order curvature information of the objective function:

$$H = \nabla^2 f(\mathbf{w}) [11]. \tag{20}$$

To compute the Hessian, we differentiate the gradient components:

- The gradient of $\nabla R_p = \boldsymbol{\mu}$ is zero (it is linear in $\mathbf{w}$).
- The second derivative of σ_p:

$$\nabla^2 \sigma_p = \frac{\boldsymbol{\Sigma}}{\sigma_p} - \frac{(\boldsymbol{\Sigma}\mathbf{w})(\boldsymbol{\Sigma}\mathbf{w})^T}{\sigma_p^3} [3]. \tag{21}$$

Using these, the Hessian is computed as:

$$H = \frac{2(\boldsymbol{\Sigma}\mathbf{w})(\boldsymbol{\Sigma}\mathbf{w})^T}{\sigma_p^3} - \frac{\boldsymbol{\Sigma}}{\sigma_p}. \tag{22}$$

Newton's Method efficiently converges to the optimal portfolio weights by exploiting the curvature information provided by the Hessian matrix.

2.3 Numerical Experiments

In this section, we conduct numerical experiments to evaluate the performance of Newton's Method for portfolio optimization. Using historical stock data, we compute the optimal portfolio weights, analyze the convergence behavior of the algorithm, and assess the portfolio's expected performance in terms of annual returns and the Sharpe ratio. These experiments highlight the efficiency and applicability of Newton's Method in solving real-world financial optimization problems [6].

Experiment Setup

1. **Data Collection:**
   - We selected five stocks: NVIDIA (NVDA), Chevron (CVX), Eli Lilly (LLY), McDonald's (MCD), and Costco (COST).
   - Historical adjusted closing prices were obtained using the 'yfinance' library for January 2023.
   - From the price data, we computed:
     - Daily returns: Percentage change in adjusted closing prices.
     - Mean returns (μ): Average of daily returns for each stock.
     - Covariance matrix (Σ): Covariance of daily returns, representing the risk structure.
2. **Objective:**
   - Maximize the Sharpe ratio of the portfolio using Newton's Method under the following constraints:
     - Minimum weight ($w_i \geq 0.1$) for each stock.
     - Fully invested portfolio ($\sum_i w_i = 1$).
3. **Algorithm Parameters:**
   - Initial portfolio weights: Equal allocation ($w_i = 1/5$).
   - Tolerance (tol $= 10^{-4}$) for gradient norm.
   - Maximum iterations ($N_{\max} = 1000$).
4. **Performance Metrics:**
   - Convergence rate: Measured using the gradient norm ($||\nabla f(\mathbf{w}_k)||$) at each iteration.
   - Optimal weights: Final weights after convergence.
   - Sharpe ratio and annualized portfolio returns.

Results and Analysis

1. **Optimal Portfolio Weights:**

   The algorithm converged to the following portfolio weights:

   $$\mathbf{w}_{optimal} = [0.3463, 0.1522, 0.0998, 0.0998, 0.3015],$$

   where two weights reached the minimum constraint ($w_i = 0.1$), indicating these assets contributed minimally to the Sharpe ratio.

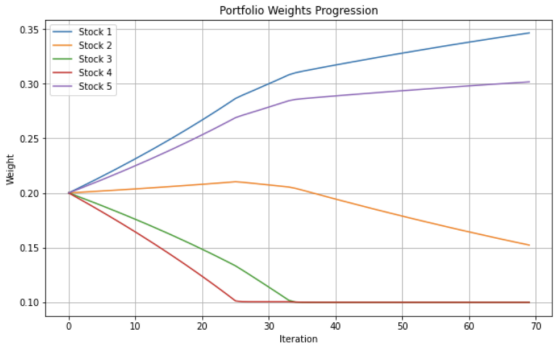


Figure 1: Graph of Portfolio Weights

The weights initially start equally distributed at 0.2 but shift as the optimization process progresses. NVIDIA becomes the highest weighted stock in the portfolio, which tells us that it is providing the most favorable risk-adjusted return. This makes intuitive sense because NVIDIA had over a 200% annual return over the year 2023.

2. **Convergence Behavior:**

   - The algorithm exhibits rapid convergence, achieving the stopping criterion ($\|\nabla f(\mathbf{w}_k)\| < 10^{-4}$) in X iterations.
   - The convergence rate is linear which is consistent with the behavior of Newton's Method for moderately-well conditioned problems. Figure 2 confirms this trend.

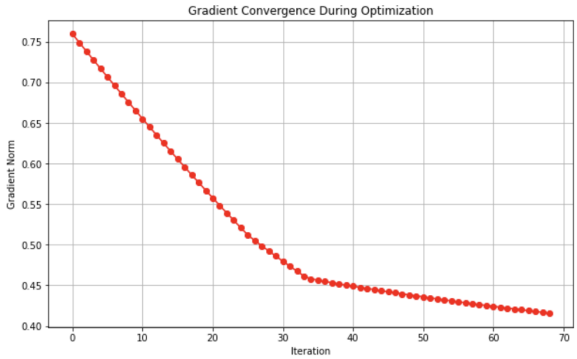


Figure 2: Gradient Convergence

3. **Performance of the Optimal Portfolio:**

   - Expected Annual Return: Using the annualized returns ($\mu_{annual} = \mu \times 252$), the portfolio's expected annual return was:

$$R_{portfolio} = \mathbf{w}^T \mu_{annual} = 176.4\%.$$

   - Sharpe Ratio: The Sharpe ratio of the optimal portfolio was computed as:

$$\text{Sharpe Ratio} = \frac{R_{portfolio} - R_f}{\sigma_p} = 0.3901.$$

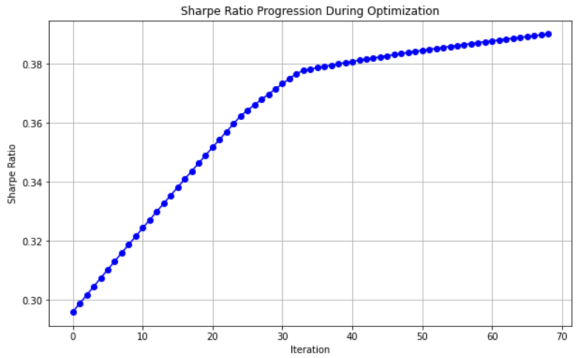


Figure 3: Sharpe Ratio Progression

Figure 3 showcases the progression of the sharp ratio over 70 iterations. The constant increase implies that our algorithm is working properly as it is achieving our goal of finding the highest possible sharp ratio.

4. **Computational Efficiency:**

   - The algorithm completed in 0.0064 seconds, demonstrating its efficiency even with the added complexity of constrained optimization.

Discussion

The numerical experiments demonstrate the effectiveness of Newton's Method in optimizing portfolio weights for maximizing the Sharpe ratio. While the method converges quickly, the results also highlight the influence of constraints on portfolio allocation, with several stocks receiving minimal weights due to their low contribution to the risk-adjusted return. These findings showcase the importance of careful constraint selection to balance portfolio diversification and optimization goals.

2.4 Conclusions

In this section, we successfully applied Newton's Method to solve the portfolio optimization problem, specifically maximizing the Sharpe ratio under realistic constraints. By using the method's rapid convergence, we efficiently determined the optimal portfolio weights using historical stock data. The results highlight Newton's Method as a powerful tool for non-linear and constrained optimization problems in finance.

Through numerical experiments, we observed that the algorithm effectively balances risk and return, allocating higher weights to assets with favorable risk-return profiles while minimizing exposure to less favorable assets. The impact of constraints, such as minimum weight thresholds and full investment requirements, was seen in the final allocation, emphasizing the trade-offs between diversification and optimization.

This extension of Newton's Method to portfolio optimization shows the versatility of the technique, making it suitable for a wide range of financial applications. Future work could explore incorporating additional constraints, such as transaction costs or specific investment mandates, or extending the method to dynamic portfolio rebalancing scenarios.

References

[1] Amit Agarwal. Algorithms for portfolio management based on the newton method. *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 9–16, 2006.

[2] Edwin J. Elton, Martin J. Gruber, Stephen J. Brown, and William N. Goetzmann. *Modern Portfolio Theory and Investment Analysis*. Wiley, 2009.

[3] Frank J. Fabozzi, Petter N. Kolm, Dorina Pachamanova, and Sergio M. Focardi. *Robust Portfolio Optimization and Management*. Wiley Finance, 2007.

[4] Roger Fletcher. *Practical Methods of Optimization (2nd ed.)*. Wiley, 2013. [Chapter 3].

[5] Stephen J. Wright Jorge Nocedal. *Numerical Optimization*. Springer, 2006. [Chapter 3.3].

[6] Michael Kearns. Thoughts on hypothesis boosting. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 335–342, 1988.

[7] Adrian Lam. Bfgs in a nutshell: An introduction to quasi-newton methods, 2020. Accessed: 2024-12-12.

[8] Adrian S. Lewis and Michael L. Overton. Behavior of bfgs with exact line search on nonsmooth examples, 2013. Accessed: 2024-12-12.

[9] Andrew W. Lo. The statistics of sharpe ratios. *Financial Analysts Journal*, 58(4):36–52, 2002.

[10] José Mario Martınez. Practical quasi-newton methods for solving nonlinear systems. *Journal of Computational and Applied Mathematics*, 124(1-2):97–121, 2000.

[11] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006. [Chapter 3.3].

[12] Boris T. Polyak. Newton's method and its use in optimization. *European Journal of Operational Research*, 181(8):1086–1096, 2007.