Unconstrained Optimization on Scalar Functions and Functions with Vector Arguments

Abstract

Purpose*:* The aim of this assignment was to investigate methods of optimization, specifically three, which act as the building blocks to more complex optimization algorithms. Furthermore the computation and characterization of stationary points for multivariate functions were explored. These experiments help highlight the faults of these optimization methods and the pieces that are used in later methods.

Methods: To test the first point in question the mathematical programming software MATLAB was used to code three algorithms which are Quadratic Approximation, Backtracking line search with one step, and Backtracking with multiple steps until convergence of a certain condition.

The second concept was tested with functions taking a vector argument and the gradient, defined in lecture. After solving a system of equations, the Hessian Matrix was then computed leading to a last computation that gave the sufficient condition needed to indicate whether a point is a minimum or a point of other importance.

Results: Through the script written in MATLAB, it was observable these algorithms implemented for scalar functions do provide a way to compute or estimate the minima values. From the outputted data though it can be seen these methods are not entirely accurate or fault proof and may not always return points within a desirable range to the true minimum.

The analysis of stationary points for functions taking vector arguments, well-demonstrated the goal that was set out. This success is largely due to this idea being tested by hand-done calculations involving simple functions. The output of these computations is displayed in tables in the corresponding section.

Add something about plots supporting both questions is also found in results

Conclusions: By running algorithms in MATLAB, it was successfully proven that unconstrained optimization can be used as a way of finding local minima of scalar functions in a bounded region. From the results obtained it is also shown that backtracking line search with multiple steps can provide the most accurate interpolation given a suitable starting point.

By applying the same theories with a vector input argument to multivariate functions it becomes clear that these results for computing and characterizing maxima and minima specifically extend to n-dimensions albeit with additional work.

Introduction

The goal of these experiments is to analyze three methods of optimization and their ability to interpolate a local minimizer for three different functions. Secondly stationary points of simple functions with vector arguments were explored.

Optimization is a branch of mathematics that is found in all fields. It involves efficiently finding some input which either acts as a maximum or minimum output of the function. In this course the focus is on finding the minimizer or local minimizer in a certain region. On the graph of a function this point will resemble the bottom of a “valley”. Optimization requires a blend of both calculus and linear algebra and so knowledge in these two topics is highly helpful. The three methods of optimization studied in this assignment begin as exercises in calculus but utilize special matrices and techniques drawn from linear algebra.

Scalar functions are quite simple to optimize in general and difficulty arises when we move to N-dimensions. In this course a few common terms pertaining to calculus and multivariate functions are redefined to allow for an easier understanding of the theory. These ideas are then tried with simple functions taking vector arguments.

To test the first part, code in MATLAB was written implementing the algorithms in question, returning results that allow comparisons between the methods to be made. This was completed by following pseudocode in the lecture notes and theories studied. The latter idea was tested with computation done on paper and cross-checked using MATLAB.

Methods

The methods for question one will first be described, then followed by question two for simplicity of the reader.

Prior to describing the optimization algorithms it is necessary to compute the gradient and second derivatives of each objective function. These differentiated functions are found in the associated MATLAB script. These are then stored in a combined variable by the deal command which creates an indexed object where each function passed is accessible by its respective index.

The first method of optimization that was tested is quadratic approximation using a single point, and the first and second derivative at that point. The starting point supplied to our quadratic approximation function is the end point of our boundaries with the larger evaluated value. Let this starting point be t_1. The function begins by initializing a special matrix called the Vandermonde matrix which is as shown…

$$V = \begin{bmatrix} {t_1}^2 & t_1 & 1 \\ 2t_1 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix} \qquad (1)$$

This gives us an easily solvable equation commonly found in linear algebra.

$$V\vec{x} = \vec{y} \qquad (2)$$

Where $\vec{x}$ is the vector of coefficients and $\vec{y}$ is the vector containing the function, gradient and second derivative evaluated at t_1. By inverting the Vandermonde matrix $\vec{x}$ is acquired.

$$\vec{x} = inv(V)\vec{y} \qquad (3)$$

The estimated value of the minimizer t^* is then returned by calculating the following.

$$t^* = -\frac{x_2}{2x_1} \qquad (4)$$

The second and third method are near the same aside from the difference in one parameter which is the number of steps to be taken. The second method only uses one step whereas the third takes k steps and converges based on a different inequality in the while loop. For these two methods only the objective function and gradient are needed. In MATLAB we implement backtracking line search in the steepline function. Once again, the function is passed a starting point, but now takes a step size that is $\frac{endpoint_2 - endpoint_1}{10}$, two parameters alpha and beta which control the comparison and reduction ratio respectively. Lastly are two optional parameters imax_in which dictates the maximum number of iterations to be performed and eps_in which gives a limit to the gradient norm.

The first few lines in the steepline function use conditionals to properly assign the optional parameters. Proceeding is another chunk of lines used for assigning values to variables for an initial estimation. From there we use a while loop to continuously recompute…

$$t^* = t^* + sd \qquad (5)$$

Where t^* is the location of the minimizer, s is the step size and d is the direction which we take the step in.

$$d = -\frac{f\prime}{|f\prime|} \qquad (6)$$

In between loops we re-evaluate the step size as necessary using another nested while loop. The nested loop ends, and a new step size is decided upon when the following condition is met.

$$f(t_1 + sd) \geq f(t_1) + \alpha s|f'|d \qquad (7)$$

For each loop on the inner while loop s is re-evaluated with the following calculation.

$$s = s * \beta \qquad (8)$$

More in-depth details of the steepline function are available in the MATLAB script. As mentioned above the difference between the call to this function for the second and third method is the number of iterations the outer loop is informed to do. For the third method where the loop performs k steps, the loop will terminate when the norm of the gradient drops below the value 10^{-6}.

$$norm(f') < 10^{-6} \qquad (9)$$

Once the loop is finished the value that is assigned to t^* is the estimated location of the local minimizer.

The second question does not involve programming in MATLAB. The general process behind performing these computations is explained in this section.

When calculating stationary points the gradient needs to be found. For a function taking a vector argument this will resemble…

$$\underline{\nabla} f = [\frac{df}{dw_1} \dots \frac{df}{dw_n}] \qquad (10)$$

For this question each function, the vector arguments consist of two elements thus $n = 2$. Once the gradient is found we set the partial derivatives equal to zero creating a system of equations that can be solved for values of $w_1$and w_2 that provide the coordinates for a stationary point.

$$\frac{df}{dw_1} = 0 \qquad (11)$$

$$\frac{df}{dw_2} = 0$$

Once values ${w_1}^*$ and ${w_2}^*$ are found further calculations are required to characterize the point as a minimizer or something else. To do this we require another special matrix called the Hessian matrix. This matrix is defined as…

$$\underline{\nabla}[\underline{\nabla} f^T] = \underline{\nabla}^2 f = \begin{bmatrix} \frac{d^2f}{dw_1w_1} & \cdots & \frac{d^2f}{dw_1w_n} \\ \vdots & \ddots & \vdots \\ \frac{d^2f}{dw_mw_1} & \cdots & \frac{d^2f}{dw_mw_n} \end{bmatrix} \qquad (12)$$

In this case though, the Hessian matrix will be a 2 x 2 matrix. Since in this course the emphasis is on minimizer's only the conditions for such points will be noted. The possible vectors $\vec{w}^*$ consist of the values w_1 and w_2 that were solutions to the above system of equations.

$$(\vec{w}^*)^T \cdot \underline{\nabla}^2 f \cdot \vec{w}^* > 0 \qquad (13)$$

The above condition implies that the point $\vec{w}^*$ is a minimizer of the function f.

Results

Table 1: Numerical results obtained by a single step using quadratic approximation, a single step using backtracking line search, and iteration with k steps using backtracking line search.

Method	Value			Function		
	$t_2,\ t_2, t^*$			f_1	f_2	f_3
Quadratic	0.6790	9.8296	-0.7500	8.9526	4.7756	-0.14
Backtracking	0.9	9.8025	0.6283	15.7062	4.2628	-1.7557
Backtracking	0.2408	8.6239	31415.92	5.1483	1.2053	-94247.77

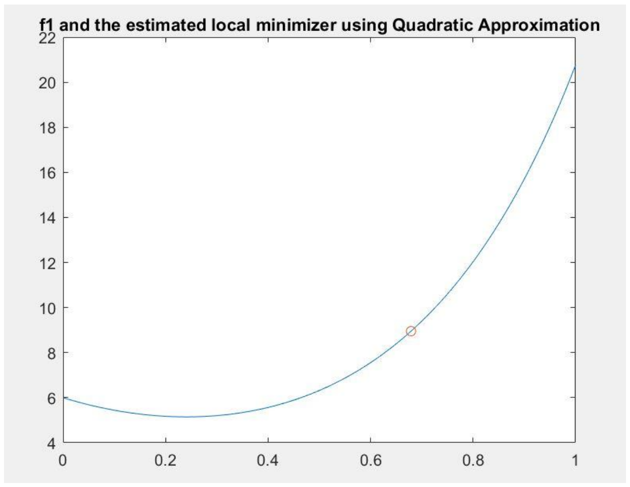

Figure 1: Graphs objective function f1, versus the estimate local minimizer called tstat1 in the MATLAB script. This estimation was obtained by Quadratic Approximation.

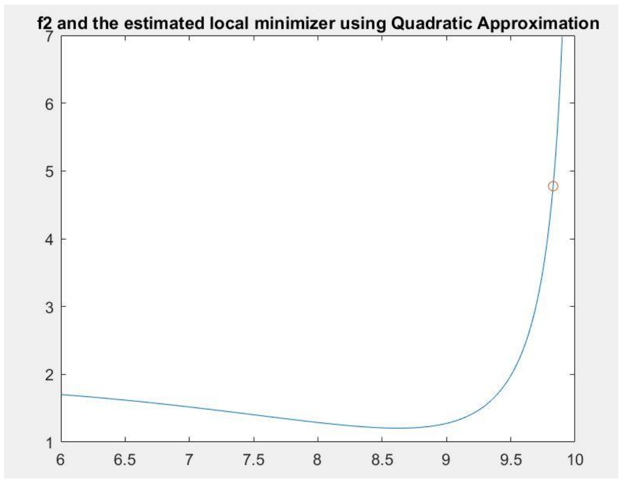

Figure 2: Graphs objective function f2, versus the estimate local minimizer called tstat2 in the MATLAB script. This estimation was obtained by Quadratic Approximation.

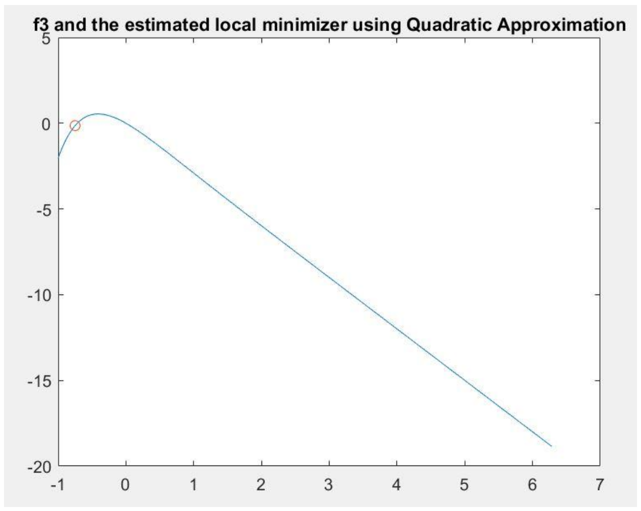

Figure 3: Graphs objective function f3, versus the estimate local minimizer called tstat3 in the MATLAB script. This estimation was obtained by Quadratic Approximation.

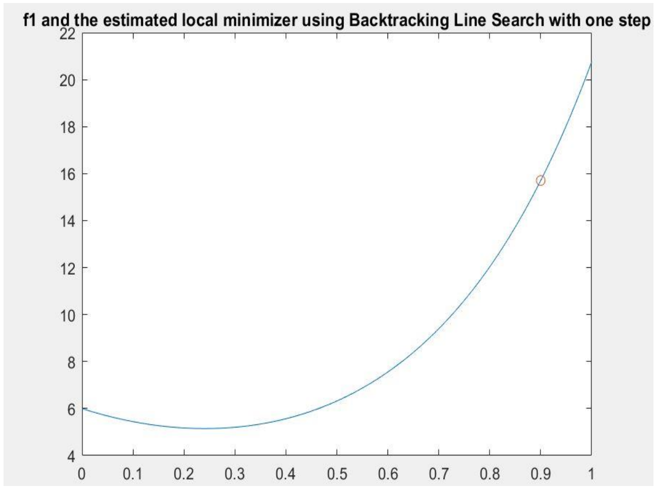

Figure 4: Graphs objective function f1, versus the estimate local minimizer called tmins1 in the MATLAB script. This estimation was obtained by Backtracking Line Search with only one step.

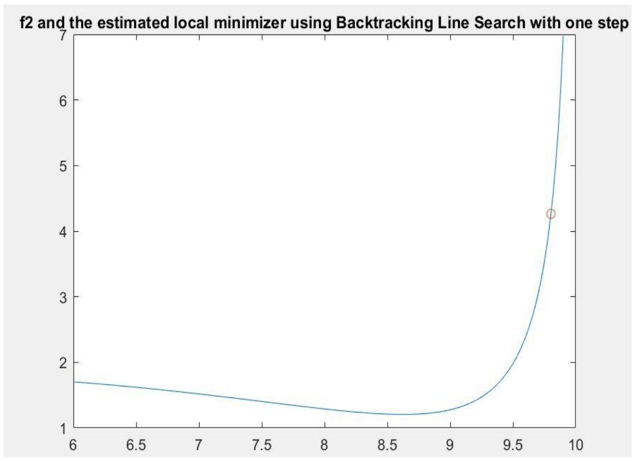

Figure 5: Graphs objective function f2, versus the estimate local minimizer called tmins2 in the MATLAB script. This estimation was obtained by Backtracking Line Search with only one step.

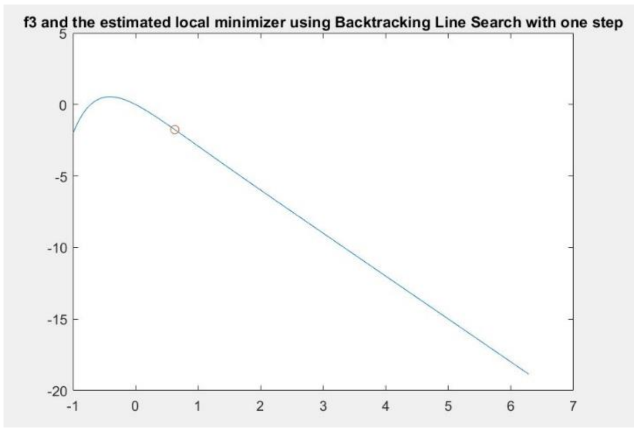

Figure 6: Graphs objective function f3, versus the estimate local minimizer called tmins3 in the MATLAB script. This estimation was obtained by Backtracking Line Search with only one step.

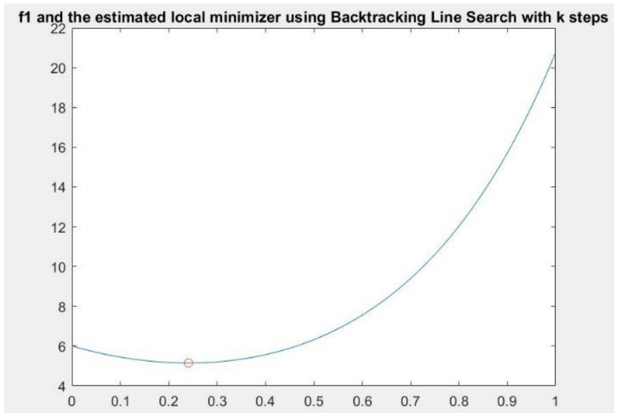

Figure 7: Graphs objective function f1, versus the estimate local minimizer called tminm1 in the MATLAB script. This estimation was obtained by Backtracking Line Search with k steps.

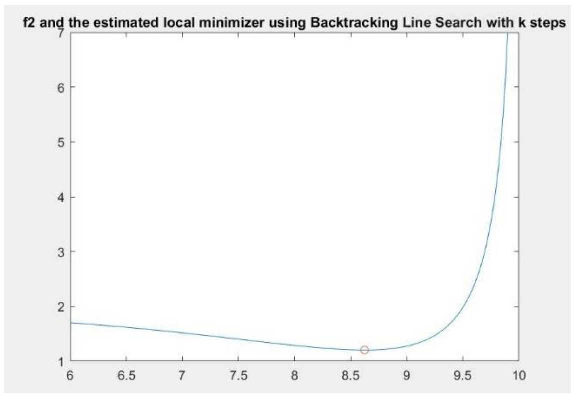

Figure 8: Graphs objective function f2, versus the estimate local minimizer called tminm2 in the MATLAB script. This estimation was obtained by Backtracking Line Search with k steps.

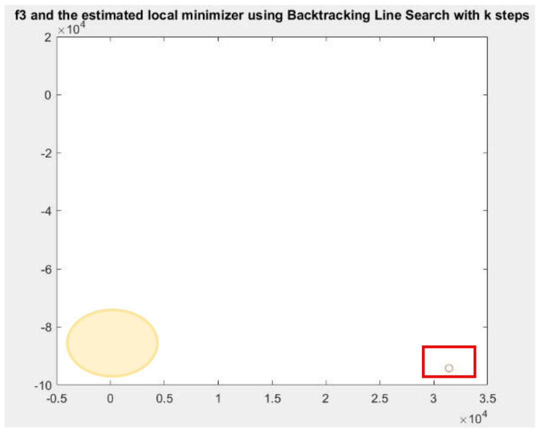

Figure 9: Graphs objective function f3, versus the estimate local minimizer called tminm3 in the MATLAB script. This estimation, note it is marked by the red box for clarity, was obtained by Backtracking Line Search with k steps. Due to the scaling of MATLAB's plotting we don't see the function but would be found if the yellow region was continuously zoomed in on.

Table 2: Numerical values of stationary points, numerical values of Hessian matrices, signs of the eigenvalues, and the kinds of stationary points for the three functions provided. Each stationary point is characterized as a local minimizer (min), a local maximizer (max), a saddle point (sdl), or as inconclusive using basic derivative conditions (inc). Values are empty for entries that do not have a distinct stationary point.

Stationary Info.	Function		
	f_4	f_5	f_6
$\vec{w}_1$	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$
$\underline{\nabla}^2 f(\vec{w}_1)$	$\begin{bmatrix}0 & 64\\64 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0\\0 & -12\end{bmatrix}$	$\begin{bmatrix}802 & 400\\400 & 200\end{bmatrix}$
At $\vec{w}_1$, λ_1 and λ_2:	$\lambda_1 > 0, \lambda_2 < 0$	$\lambda_1 = 0, \lambda_2 < 0$	$\lambda_1 > 0, \lambda_2 > 0$
At $\vec{w}_1$, kind is:	inc	inc	min
$\vec{w}_2$	$\begin{bmatrix}-1\\\frac{1}{2}\end{bmatrix}$	$\begin{bmatrix}0\\2\end{bmatrix}$	
$\underline{\nabla}^2 f(\vec{w}_2)$	$\begin{bmatrix}48 & 32\\32 & 192\end{bmatrix}$	$\begin{bmatrix}12 & 0\\0 & 12\end{bmatrix}$	
At $\vec{w}_2$, λ_1 and λ_2:	$\lambda_1 > 0, \lambda_2 > 0$	$\lambda_1 > 0, \lambda_2 > 0$	
At $\vec{w}_2$, kind is:	min	min	
$\vec{w}_3$	$\begin{bmatrix}1\\-\frac{1}{2}\end{bmatrix}$		
$\underline{\nabla}^2 f(\vec{w}_3)$	$\begin{bmatrix}48 & 32\\32 & 192\end{bmatrix}$		
At $\vec{w}_3$, λ_1 and λ_2:	$\lambda_1 > 0, \lambda_2 > 0$		
At $\vec{w}_3$, kind is:	min		

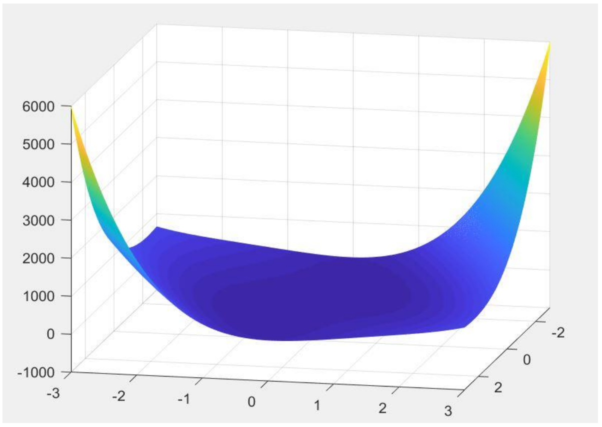

Figure 10: Graphs the simple function $f_4(\vec{w})$ which takes a vector input using the mesh function supplied in MATLAB.

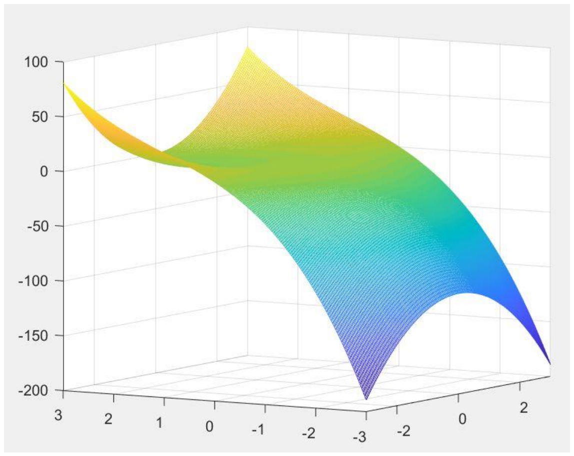

Figure 11: Graphs the simple function $f_5(\vec{w})$ which takes a vector input using the mesh function supplied in MATLAB.

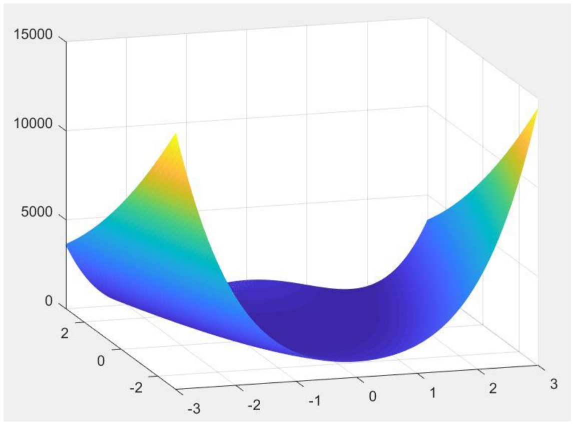

Figure 12: Graphs the simple function $f_6(\vec{w})$ which takes a vector input using the mesh function supplied in MATLAB.

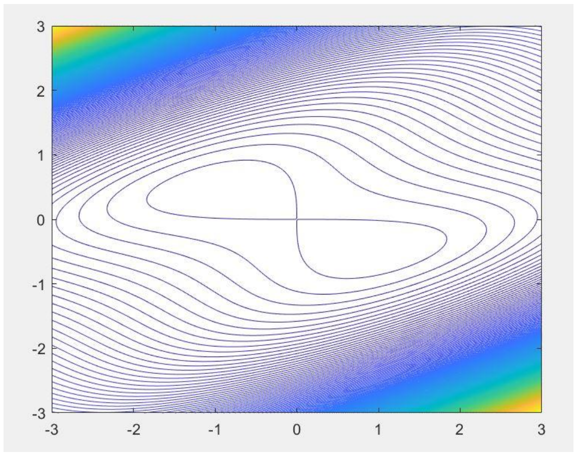

Figure 13: Shows the contours of $f_4(\vec{w})$, the same simple function taking vector arguments as before, using the contour function supplied in MATLAB.

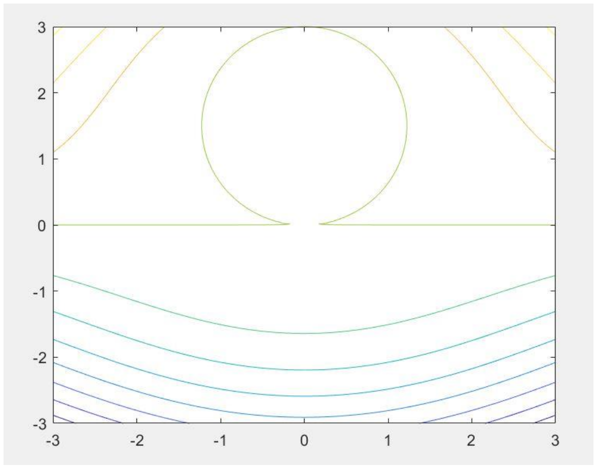

Figure 14: Shows the contours of $f_5(\vec{w})$, the same simple function taking vector arguments as before, using the contour function supplied in MATLAB.

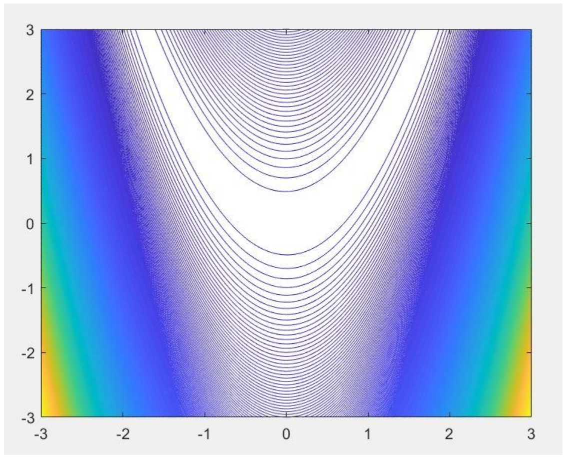

Figure 15: Shows the contours of $f_6(\vec{w})$, the same simple function taking vector arguments as before, using the contour function supplied in MATLAB.

Discussion

Referring to Table 1 in the results we are able to directly compare the results of different optimization methods. We see that quadratic approximation gives a good first estimate towards a local minimum whereas backtracking line search with a single step acts similarly but may not get us as close to the minimum. Clearly the best method of the three is iterative backtracking line search, if we plot the original function overlaid with the point of the estimated minimum, we see that t^* is near perfectly on the true minimum except for f_3, the reason for this is discussed below. Figures [1-9] show the estimated local minimizers on the original functions.

As stated, an important result to note from Table 1 is the location of the local minima for f_3. When we plot the function it is evident that this is indeed not a minimum but actually the opposite, a maximum. For quadratic approximation it recognizes a stationary point is in the locale but when it formulates the quadratic to estimate this point it uses a concave quadratic due to the gradient and second derivative being less than zero. Hence why quadratic approximation focuses on this maximum. The reason the latter two methods move away from the nearby maximum and tend towards positive infinity is because the condition for searching is to check the derivative at our current point, so the first iteration is the starting point $t_1 = 0$ which the gradient is negative. When the gradient is negative, backtracking line search wants to follow this path continuing to lower values of the objective function with the idea that the algorithm will eventually hit a local minimum. Therefore our MATLAB script hits the maximum number of steps we pass it as a parameter since the norm of the gradient never ends up being less than our epsilon parameter. Thus our MATLAB script returns a very large t^* value that when evaluated in the objective function outputs a very large negative value.

Table 2 is related to the results of Question 2 and displays information such as locations of stationary points $\vec{w}$, the Hessian matrix $\underline{\nabla}^2 f(\vec{w})$, the sign of the eigenvalues and what classification the stationary point is. These results are obtained following the computations detailed in the methods section. They demonstrate how optimization theory can be generalized to n-dimensions. This generalization once again uses the gradient and second derivatives to analyze local regions to determine where a plateau may exist. With multivariate functions we require more complex methods and ideas pulled from both calculus and linear algebra. The Hessian matrix and the solution vector $\vec{w}$ along with eq.(13) make it possible to determine whether the eigenvalues are both positive, negative or one of each. We care when both eigenvalues are negative as this indicates the point described by $\vec{w}$ is a minimum. We can tell that for f_4, $\vec{w}_2 = \begin{bmatrix} -1 \\ \frac{1}{2} \end{bmatrix}$ and $\vec{w}_3 = \begin{bmatrix} 1 \\ -\frac{1}{2} \end{bmatrix}$ are minimum of the function, for f_5, $\vec{w}_2 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$ is a minimum and for f_6, $\vec{w}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is a minimum of the function. To support these outcomes we can optionally create 3-D models, figures [10-12], or contour plots, figures [13-15], of the functions in MATLAB. These are included in the results section for a visual reference.

This assignment has acted as a learning guide for the basics of optimization with both scalar and multivariate functions. Using MATLAB we were able to implement three different optimization algorithms and see how they work on three different scalar functions, giving us comparable results as well. Following the outline for question 2 we saw how the fundamental ideas for optimization translate to multivariate functions and the sort of information one should find. Relating this information to the algorithms we know of and used for scalar functions we can begin to formulate how these algorithms will be applied to functions with vector arguments.

This exercise also solidified why we require an elegant balance between calculus and linear algebra to explore more complex and / or efficient theories found in the field of optimization.