

ME6103 – Engineering Design Optimization :
Homework #3

Submitted by:
Nathan Rolander

Prepared for:
Dr. Bert Bras

Georgia Institute of Technology

Tuesday, November 09, 2004

PROBLEM

PROBLEM DESCRIPTION

In this exercise, the objective is to obtain first-hand experience with coding, using and comparing a few techniques for simple unconstrained optimization algorithms. The focus of the homework is on solution algorithms, rather than on modeling.

The assignment is to test and validate our algorithms, using different starting points both close and far away from the known minimum. Our write-up should include:

- The results obtained
- A comparison between the three algorithms in terms of accuracy, efficiency, ease of use, robustness, etc.
- Which algorithm we would recommend to others and why
- What we have learned from this exercise
- The source code of our implementation

Methods to be Investigated

The following methods are to be investigated:

1. Hooke and Jeeves Pattern Search
2. Cauchy's Method of Steepest Decent
 - a. Brent's Quadratic Fit Lines Search using Golden Section
3. Pure Newton's Method

All of these codes and the various sub functions they call are compiled in the source code section at the end of this assignment. These codes are well documented and commented, I recommend referring to them while reading this report.

Function Investigation

To this end, the problem to solve is to find the minimum value for Fenton and Easton's Function, which is stated below:

$$f(x) = \left\{ 12 + x_1^2 + \frac{1 + x_2^2}{x_1^2} + \frac{x_1^2 x_2^2 + 100}{(x_1 x_2)^4} \right\} \left(\frac{1}{10} \right) \quad (1)$$

The results in the following contour plot shown in Figure 1, which is identical to that given in the assignment sheet.

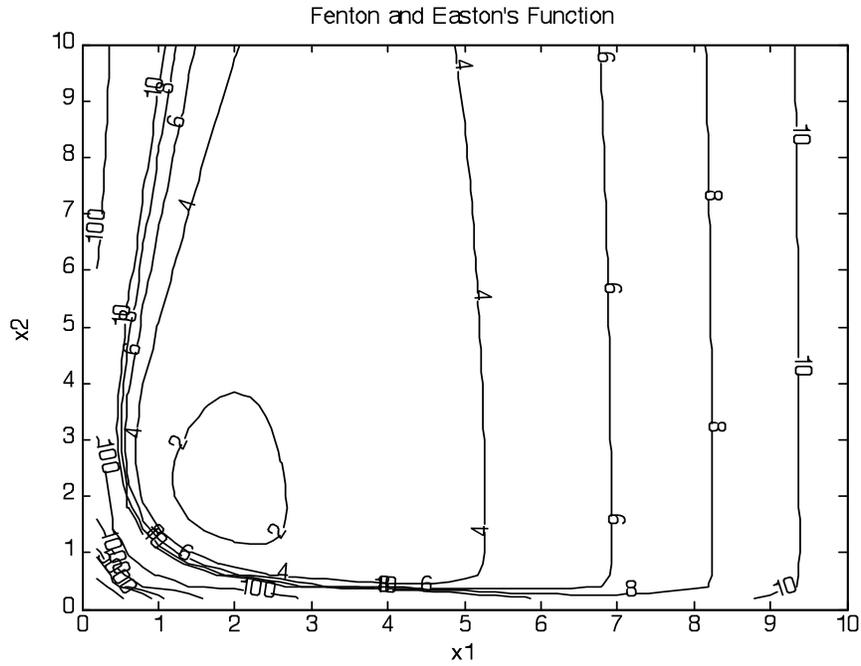


Figure 1 – Contour Plot of Fenton and Easton's Function

To better view the gradient of the function, the following surface plot is presented. Because of the singularity at $x, y = 0$ where the function approaches infinity, a logarithmic z scale is used for easier viewing, as shown in Figure 2. This logarithmic z scale is used throughout this assignment.

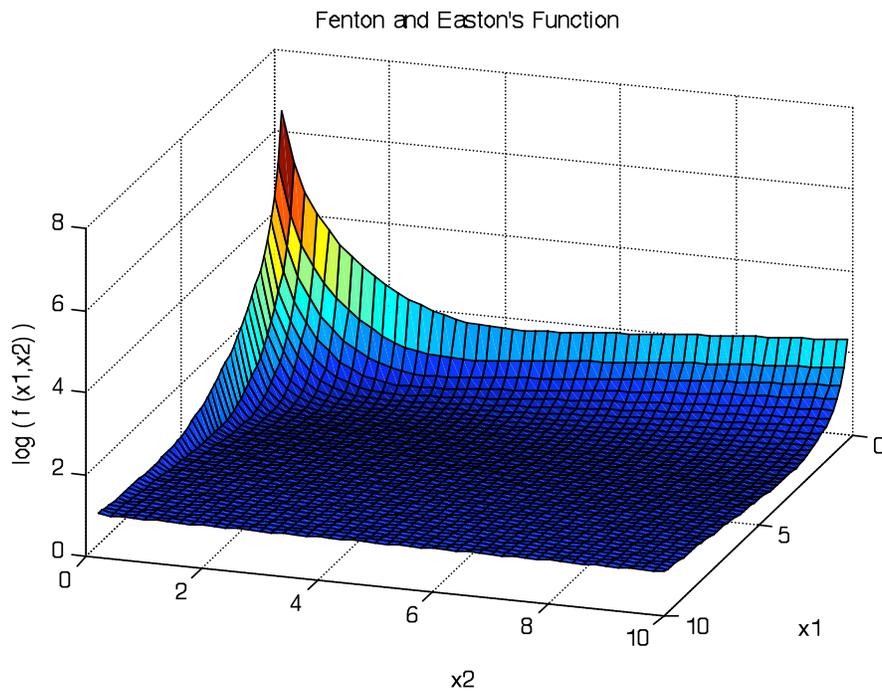


Figure 2 – Surface Plot of Fenton and Easton's Function

Local Minima

The Fenton and Easton's function is mirrored in both the x and y axes, creating four local minima of equal value. Although I will only be investigating the positive quadrant in this assignment, I must be aware of this during the convergence and testing of my algorithms. I have also investigated the extent of the Fenton and Easton's function, to determine if a gradient that slopes away from the minima exists at a distant region, which it does not. The slope in each quadrant always points towards the minimum point.

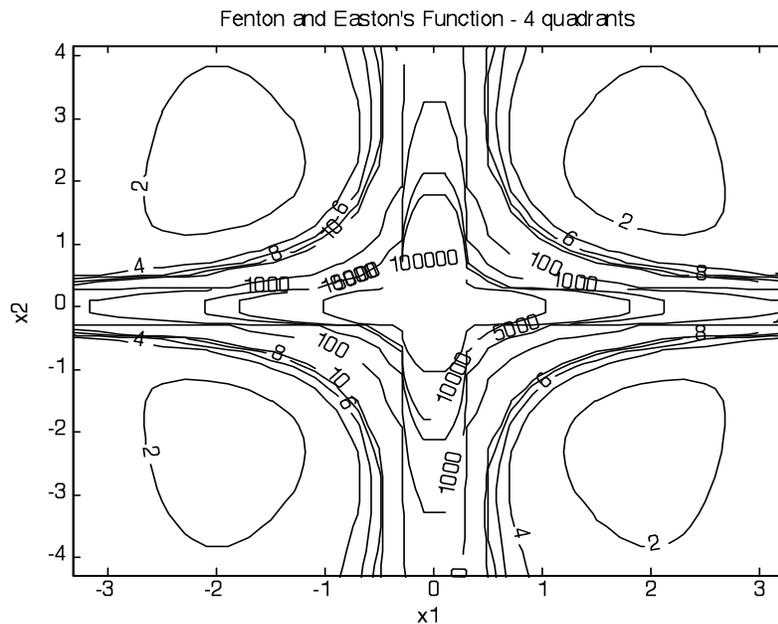


Figure 3 – Contour Plot of Fenton and Easton's Function

Setting The Benchmark

In order to determine the accuracy of the different algorithms we need a benchmark value for comparison. I used the *fminsearch* unconstrained minimization solver in MATLAB, which used a pseudo Newton method, to solve for the minimum to a tolerance of 6 decimal places. For all accuracy comparisons, I have only used the first four decimal places, therefore the benchmark minimum is:

Variable	Value
x_1	1.7435
x_2	2.0297
$f(x_1, x_2)$	1.7442

THE METHODS

CODING & IMPLEMENTATION

All of my algorithms were coded and executed in MATLAB. No complex internal functions were called such as *min* or *max* to enable the codes to run on any version or computer. All of the codes were compiled as functions, to be called in the command window. This allowed for the easy re-use of functions and general neatness and readability of the code.

Something new (and pretty neat) that I implemented was the use of the MathWorks commenting scheme. This means that by typing 'help function_name' the description and inputs and outputs are described to the user as a help file. I also implemented variable number of input arguments in many functions, allowing the user to use the default values for input variables or supply their own. I feel that this adds a touch of professionalism to my codes I'm proud of.

PATTERN SEARCH

The pattern search method I implemented was the Hooke and Jeeves method as described in the text from ME6103. This method involves finding a search direction, and then moving in that direction until the evaluation of the objective function reveals a non-improvement, then a new search direction is established.

For stopping criteria, I simply used a minimum size for the search area, when the search radius went below this value, the algorithm was considered converged. I found a value of $1e-5$ was sufficient to obtain the 4 decimal places of accuracy I required. I did not add a stopping condition for functional evaluation because I found this condition alone to be sufficient. However, I did add an iteration limit to stop the algorithm if it diverged, and warned the user of this result.

Convergence

I found this algorithm to be robust, despite the number of parameters that affect the convergence and accuracy. The algorithm would only converge to one of the negative quadrant minima if the initial step size was taken to be large enough to step over the asymptotic axis boundaries. The algorithm will converge from any point within the 10x10 region, except for strictly on the x-y axes. An example of the steps taken during convergence is shown next in Figures 4 & 5, starting with $x_0 = [6 \ 5]$. However, the algorithm will also converge from further away, I have tested as far as starting at [100 100].

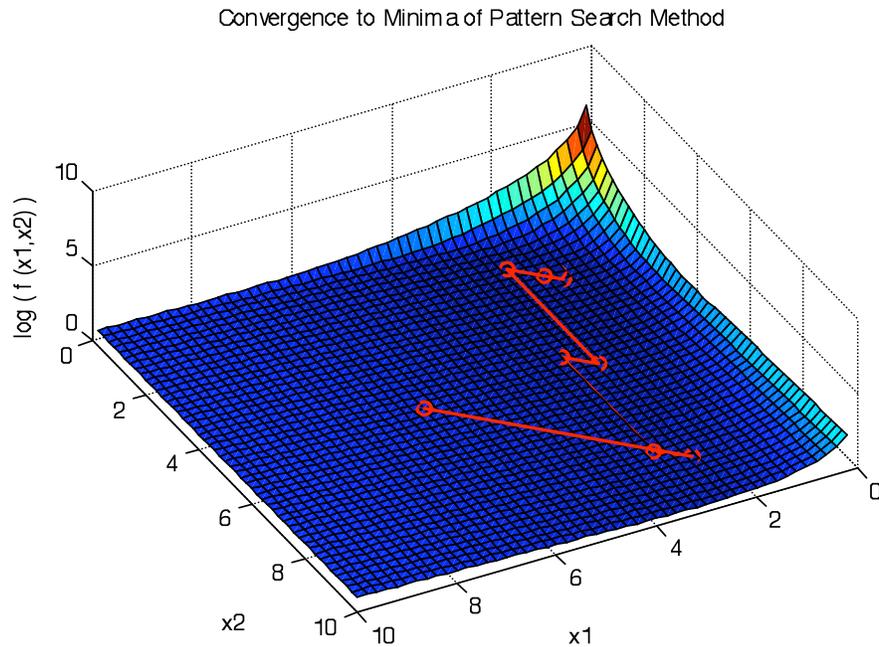


Figure 4 – Convergence to minima of Pattern Search Method

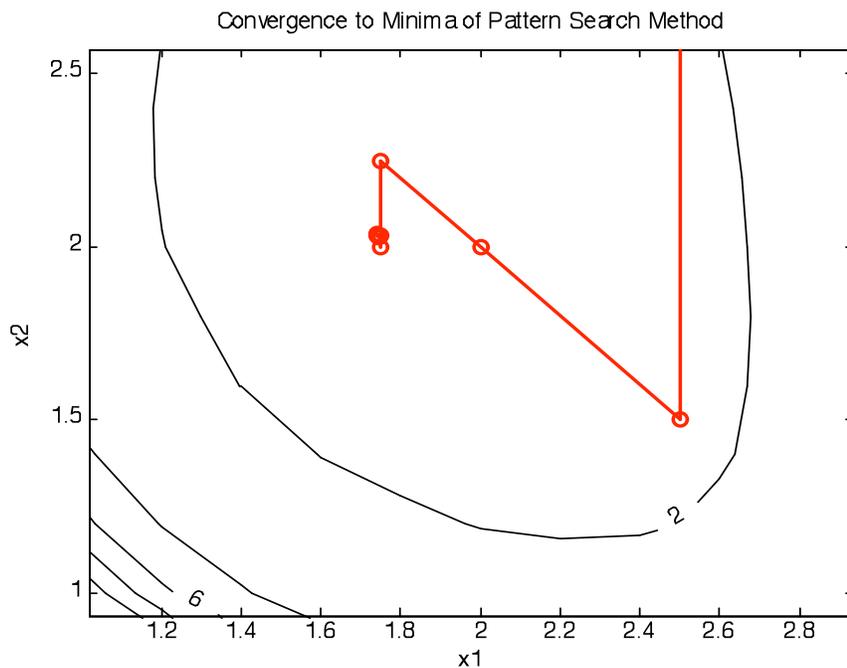


Figure 5 – Convergence to minima of Pattern Search Method

This convergence pattern requires many steps as it is forced to follow the pattern, a direction in 45 degree increments. This creates the visible saw tooth pattern. The convergence also slows around the point of the minima, as the movement is restricted and it jiggles around to find the minima.

Parametric Investigation

There are many different parameters that can be set during the pattern search. These parameters are: initial step size 's', the step extension parameter to move in once a pattern is found 'a', and the step size reduction ratio, the amount the search area is reduced by when a pattern is not found 'r'. By changing these parameters, different rates of convergence will occur. These are problem specific, but I felt it pertinent to investigate these effects. The results are shown below in Figures 6-8.

Pattern Search Parametric Study, $a = 2$

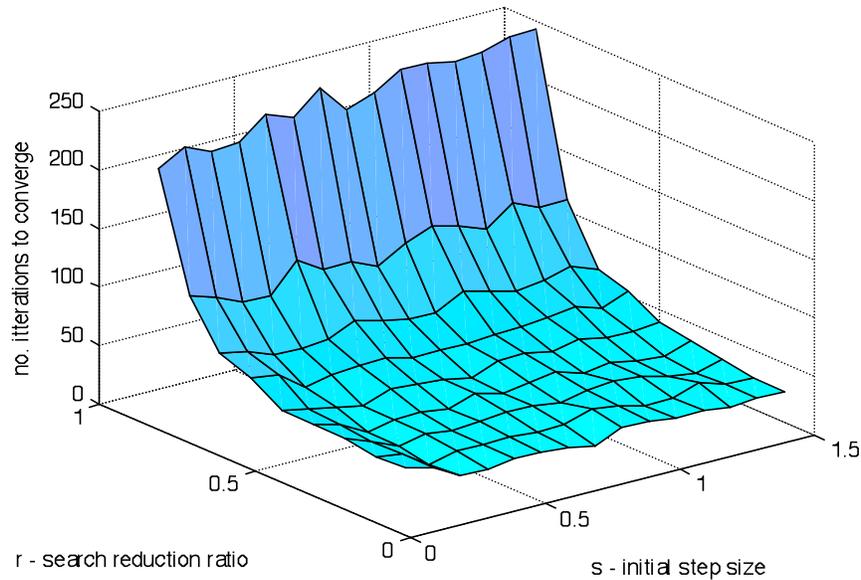


Figure 6 – Pattern Search Parametric Study, $a = 2$

Pattern Search Parametric Study, $a = 1.5$

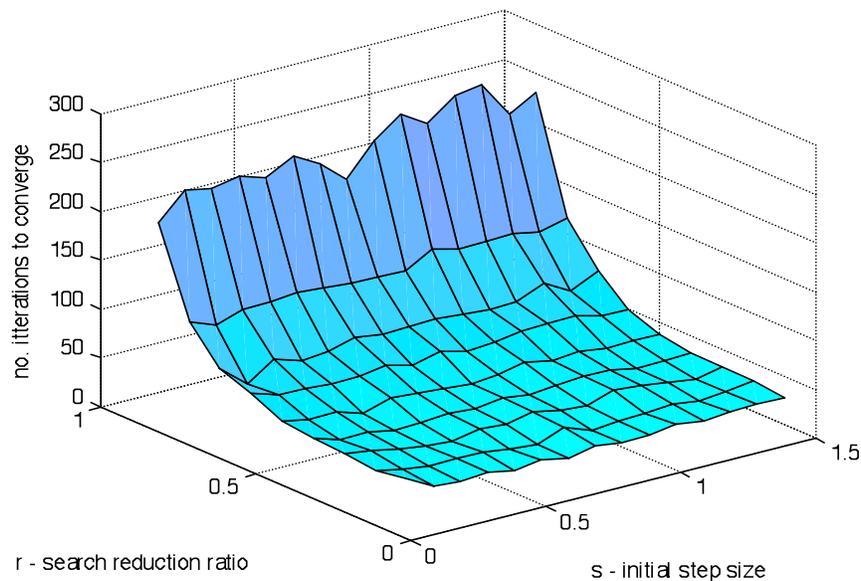


Figure 7 - Pattern Search Parametric Study, $a = 1.5$

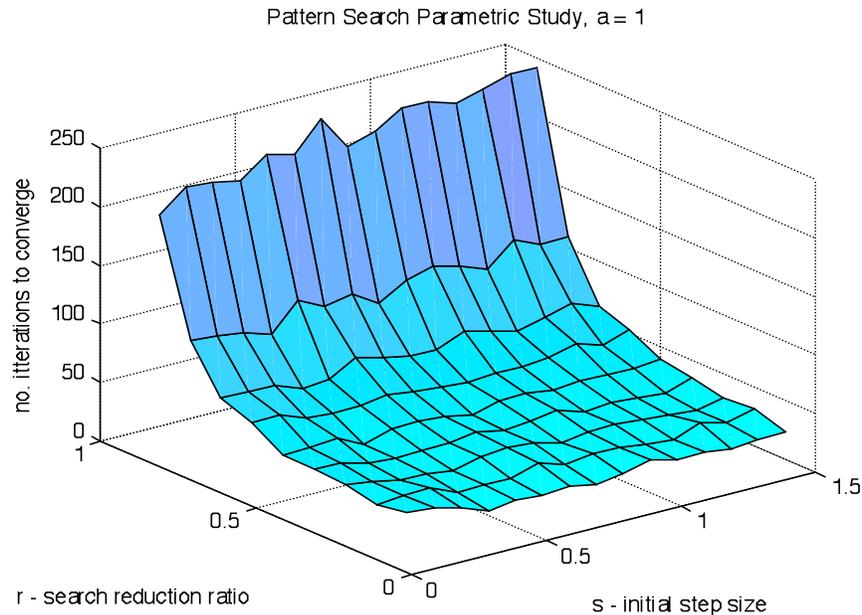


Figure 8 - Pattern Search Parametric Study, $a = 1$

Viewing the figures above, it is evident that the dominant parameter is r , the search radius reduction ratio. The initial step size, s , only plays a minor role and a , the step size reduction ratio does not have any noticeable effect. All of these tests were conducted from the same starting point $x_0 = [6 \ 5]$, however I also tested the effects from several starting points and the results were the same. Using these results I believe that my default value of step size reduction ratio of 0.5 is appropriate, as it allows for very good speed while maintaining the 4 decimal place accuracy I require.

STEEPEST DESCENT

I implemented Cauchy's method of steepest descent, using a line search algorithm to find three points spanning a minima and then apply Brent's Quadratic Fit method of finding a minimum using the Golden Section, as described in the ME6103 text. This method involves finding the direction of steepest descent, using the gradient of the objective function, and then searching along that line to find the minimum, and then repeating.

For stopping criteria, I used several conditions. First was an absolute condition, if the gradient was below a certain value, the search was stopped. Second was to check if the change in functional value changed less than a certain percentage and under a certain value within two consecutive iterations. I found values of $1e-5$, $1e-6$ and $1e-4$ resulted in the 4 decimal places of accuracy required from any starting point.

Sub Functions

The method of steepest descent uses several sub functions. The first of which is the determination of the gradient. I analytically computed the gradient by hand and checked it using the MAPLE engine in MATLAB. This ensured that I was correct. This

analytical gradient allowed for fast computation as the objective function did not have to be called to obtain a numerical derivative.

To then determine how far to move a line search was implemented. The first stage was to find three points spanning the minimum, with the center point at the lowest evaluation. This was a simple process using the golden section approach. Then, the search is continued using Brent's Quadratic Golden Section algorithm. This method uses a normal Golden Section approach, and then tries to fit a quadratic to the points to find a minimum, so speed the convergence. I implemented a robust approach that considered many problematic scenarios, which makes it the bulkiest code I have written in this assignment. I believe that this results in the loss of computational speed as found later.

Convergence

The final parameter, initial step size, I investigated in the same manner as the parameters for Pattern Search, however I found no difference in convergence rates. However, this step size does play an important role in the convergence location of the method. If the initial step size is too large, when starting from a far away point, the method can jump to a negative quadrant minima, as shown below in Figure 9. The solution to this is to use a smaller step size, as shown in Figure 10 & 11, using $x_0 = [5 \ 6]$. However, if the initial point is very far away, such as around 50 in either dimension, the solution will converge to a different quadrant.

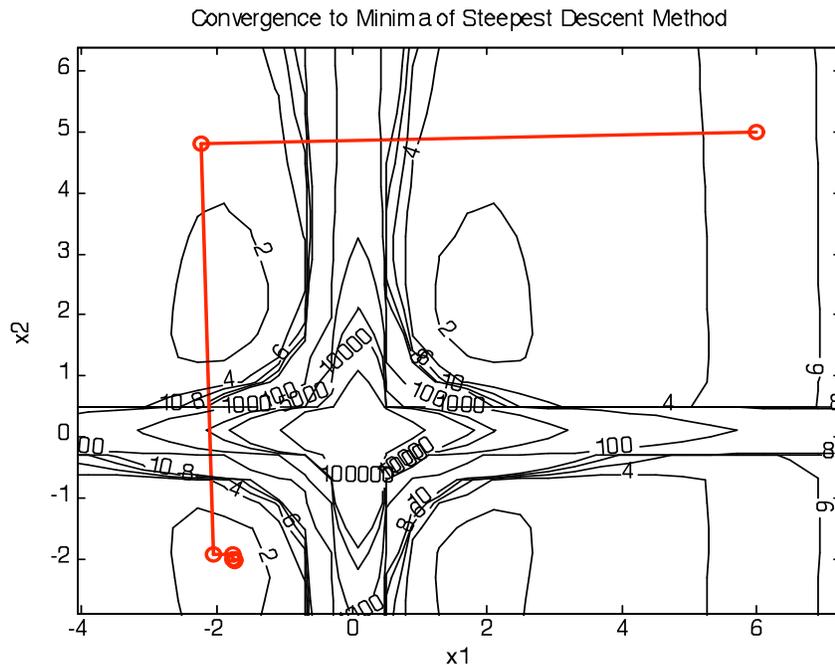


Figure 9 – Convergence to negative quadrant minima, using step = 100

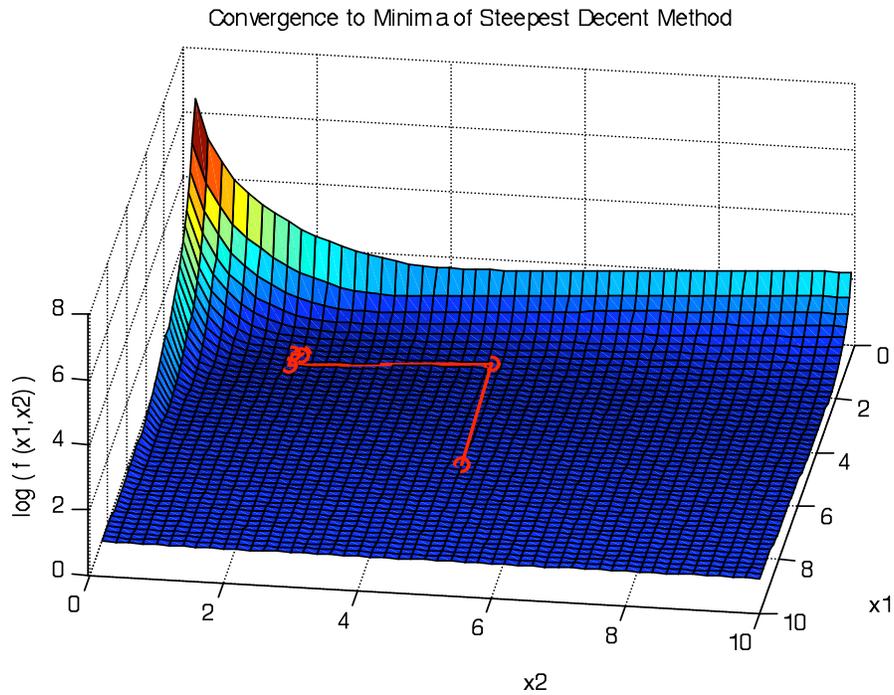


Figure 10 – Convergence of Steepest Descent Method, using step = 1

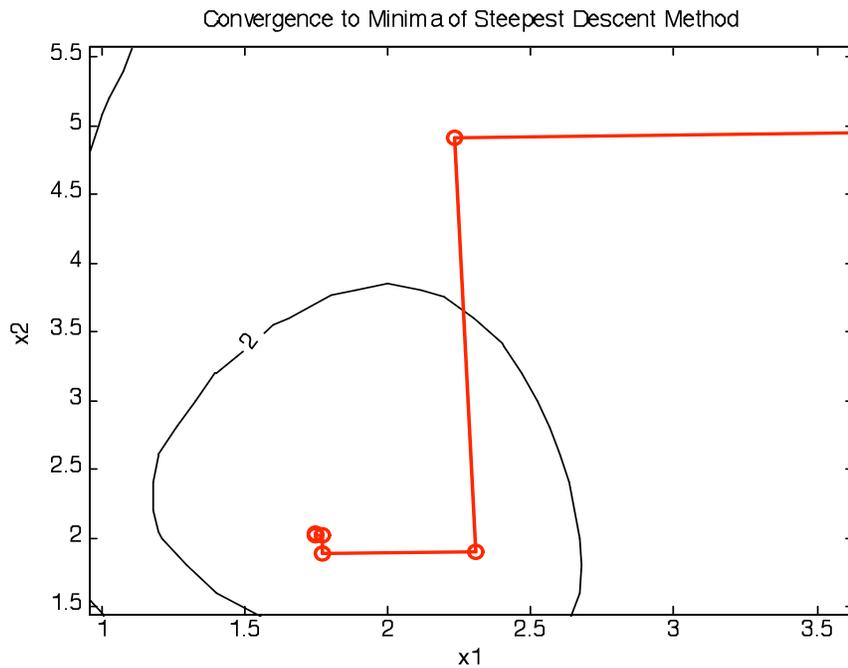


Figure 11 – Convergence of Steepest Descent Method, using step = 1

This convergence zigzags towards the minima following the gradient as expected. The convergence slows considerably reaching the minima, however, not many iterations are required. However, this is somewhat deceptive and during the line search, much iteration

is required which are not displayed on this plot. This is investigated further in the results section. An advantage of the Steepest Decent method is shown when the starting point is very far away. In this situation, the method makes very large steps towards the minimum and converges, requiring only one or so extra iterations, usually the number of iterations never exceeded ten. This makes it efficient if the starting position is completely random due to lack of knowledge of the functional domain.

PURE NEWTON'S METHOD

The Pure Newton's Method is a foundation for many unconstrained optimization methods, although in this form it has limitations which will be shown. It is a simple approach in which the second order gradients are considered using a Hessian matrix, which is used along with the gradient at a point to determine where and how far to move to the next point, and then iterate.

I again only implemented a gradient based stopping condition, and found that $1e-4$ was sufficient for 4 decimal places of accuracy, in the same matter as the Steepest Descent method stopping condition.

Sub Functions

The Pure Newton's Method uses the same gradient sub function as the steepest descent method as well as a Hessian sub function, which again was determined analytically using MAPLE through MATLAB and verified by hand. This allowed the function to run much faster than normal, as numerically derived second derivatives are computationally expensive.

Convergence

The convergence of the Newton's method only depends upon the starting point. From there depending upon the gradient and the Hessian, the method will either converge, converge to a negative quadrant, or diverge. This divergence is shown in Figures 12 & 13, and convergence in Figures 14 & 15.

The divergent solution takes progressively larger steps away from the minimum point, until the maximum iteration constraint is reached. The convergent solution takes an initial jump overshooting the minimum, and then almost slides down the slope, perpendicular to the isobars, directly to the minimum, with little or no zigzagging.

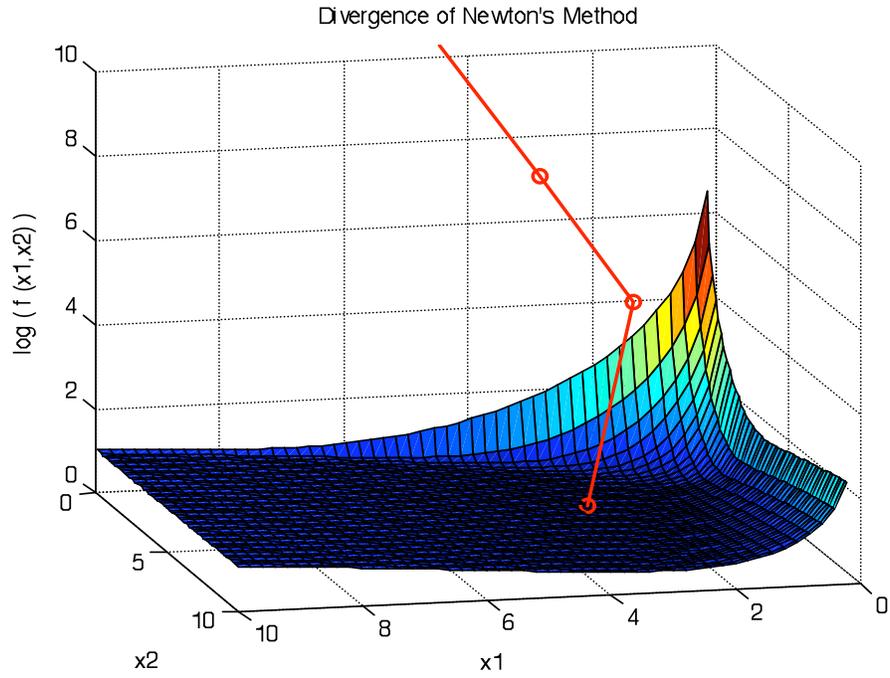


Figure 12 – Divergence of Newton's Method, $x_0 = [3, 4]$

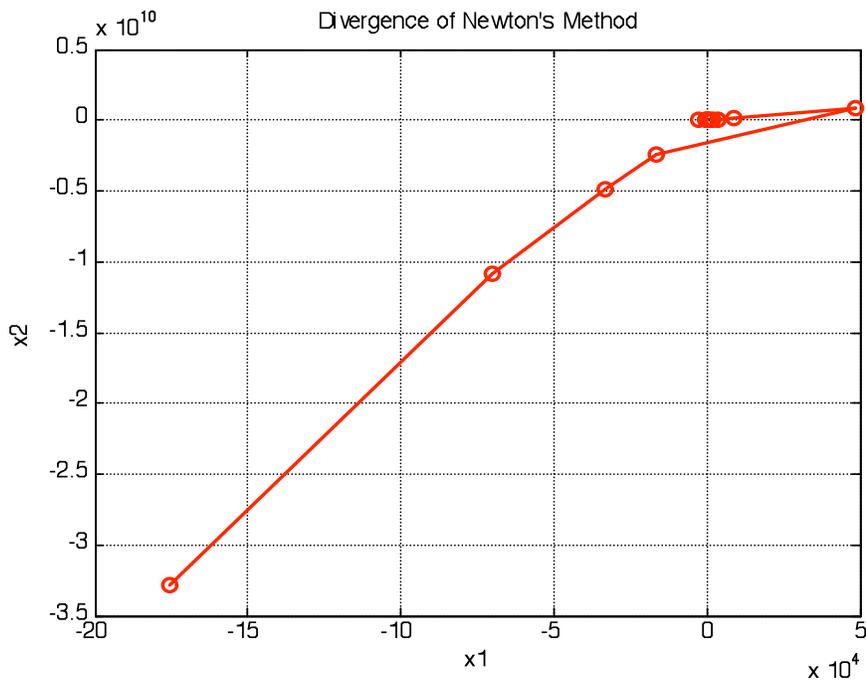


Figure 13 – Divergence of Newton's Method, $x_0 = [3, 4]$

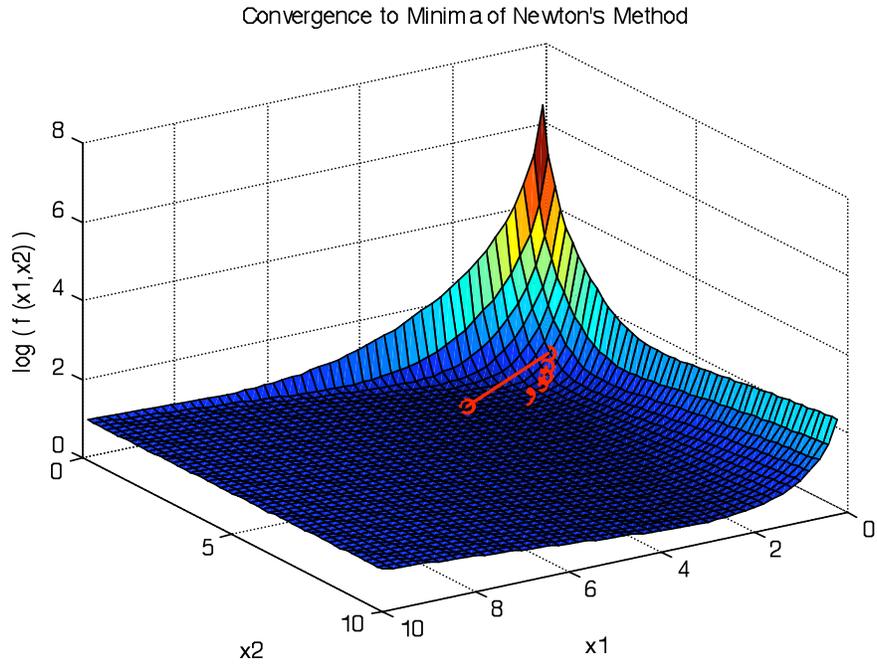


Figure 13 – Convergence of Newton's Method, $x_0 = [3 \ 2]$

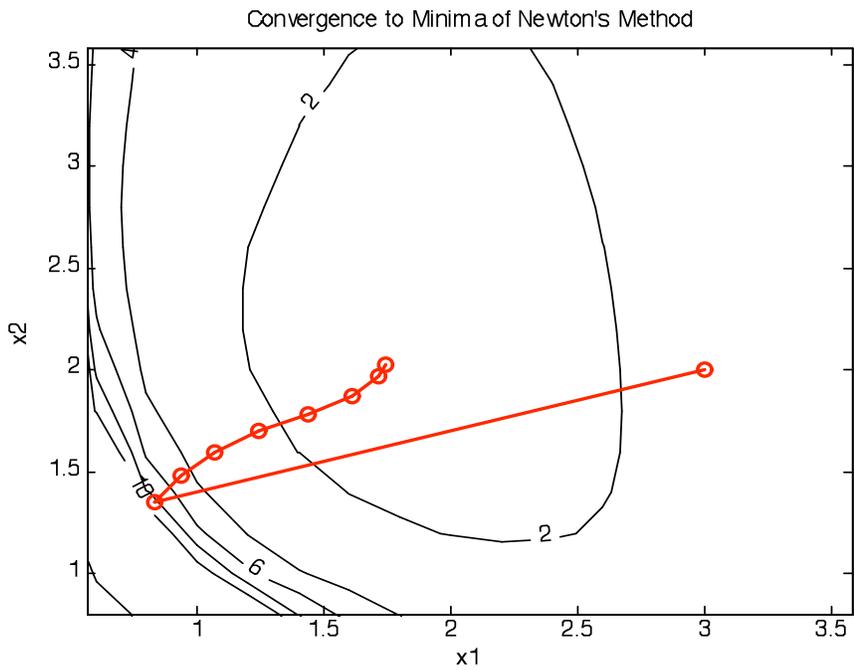


Figure 14 – Convergence of Newton's Method, $x_0 = [3 \ 2]$

Convergence Investigation

I further investigated this convergence dependence upon starting point by plotting convergence versus possible starting position. If the solution converged the point was colored green, if the solution converged to a negative quadrant, the point was colored red, and if the solution diverged, it was colored yellow. This color scheme was applied to the surface plot of the Fenton and Easton's function, and is displayed in Figure 15.

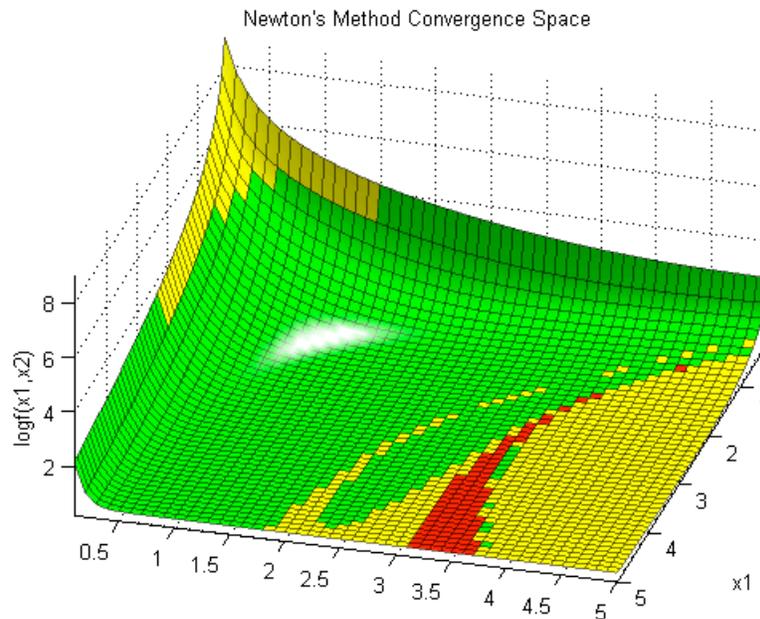


Figure 15 – Convergence Zones of Newton's Method

The above plot shows that the method only converges when it starts quite close to the minimum. However, there are also two bands where the solution diverges or is kicked into a negative quadrant. This convergence or divergence is based upon the first step made which is based on the computed Hessian and Gradient function of the point.

RESULTS & DISCUSSION

ACCURACY

All three of the methods can successfully converge to the minimum point at an accuracy of 4 decimal places, as compared to the benchmark. The default values for the various sufficient condition and other parameters, as given in the compiled codes and in this assignment, are adequate to achieve this accuracy or greater for any starting point (assuming convergence) within the 10 by 10 region investigated. The only major difference in accuracy is how many iterations are required to obtain the solution, where steepest descent method is the most accurate as it requires far less iterations than the other methods.

ROBUSTNESS

The Pattern Search and Steepest Descent methods are the most robust, converging to the positive quadrant minimum in almost every starting point within the 10 by 10 quadrant. I have noted the conditions and shown plots for each of these methods that can make the method converge to a negative quadrant minimum. In all of these cases this can be avoided through changing the step parameters. The Newton's method is the only method that can diverge; making it quite unsuitable for any problem where mapping the function space would require excessive computational time.

SPEED OF CONVERGENCE

In order to determine the speed of convergence of the different methods, I wrote a function to run each method from a random starting point within the 10 by 10 grid 1000 times. For each method the cumulative running time was recorded for each converged run, the divergent runs were discarded to not penalize this, and to only test the speed of the algorithm. By running the methods 1000 times more accurate results could be obtained as the short running times of the methods means that background computer processing has a large effect on the speed of the methods. During these runs the user interaction was suppressed, as such no plots were made or warnings displayed to the user. I ran this program several times, and each time the results were very similar, allowing for some discrepancy because of the random starting points. This again is why using many different starting points is important. The 1000 random starting points are shown below for reference in Figure 16.

The results for each method is shown in the bar chart in Figure 17. In this chart the length of each bar shows the average of how one run took to converge. The bar is subdivided into the average number of iterations needed per run. Therefore each segment of the bar shows how long each iteration of the method takes. The x-axis is in milliseconds.

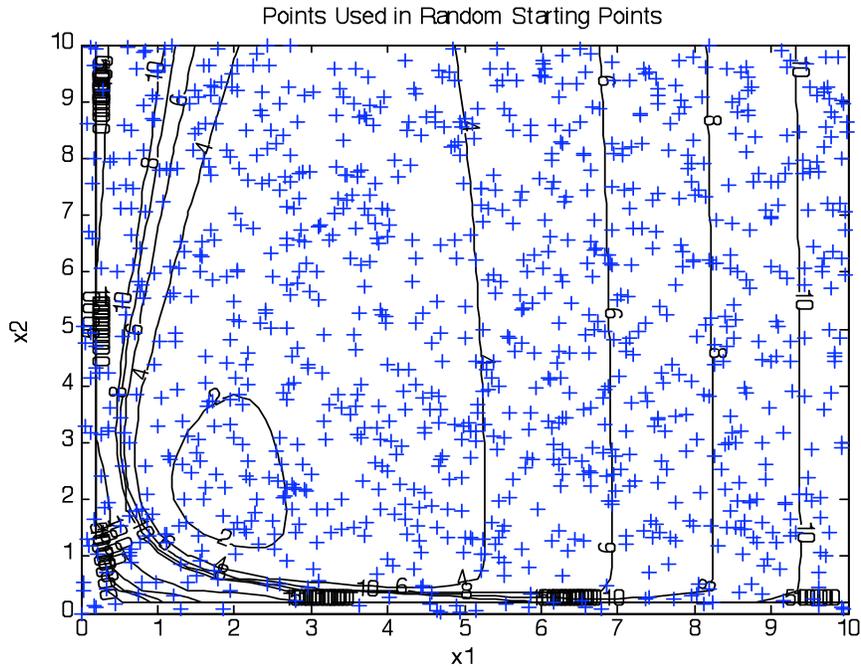


Figure 16 – 1000 Random x_0 locations: $x_1, x_2 \in [0,10]$

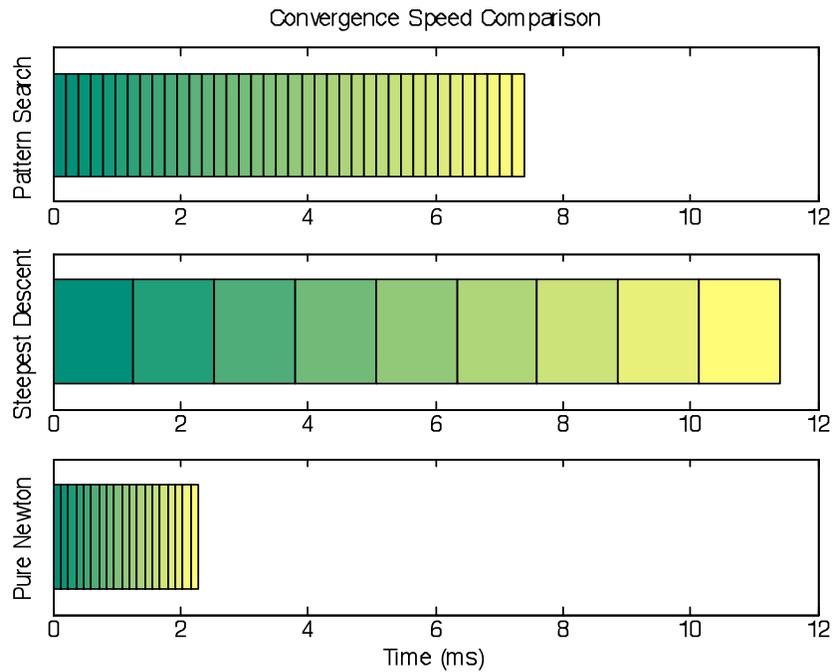


Figure 17 – Comparison of average time taken to converge

The results of Figure 17 are also tabulated below.

<i>Method</i>	<i>Iterations</i>	<i>Total Time (ms)</i>	<i>Time/Iteration (ms)</i>
<i>Pattern Search</i>	38	7.4090	0.1950
<i>Steepest Descent</i>	9	11.3990	1.2666
<i>Pure Newton</i>	19	2.2810	0.1201
<i>Steepest Descent*</i>	117	11.3990	0.0974

Viewing the results above, it is clear that the Pure Newton's method is the fastest to converge, when it does converge. Pattern Search is the second fastest, despite requiring many iterations. Steepest Descent is the slowest, despite having few iterations. I will now discuss each of these methods results in turn.

Pattern Search

This method requires a lot of iterations, as is expected. However, because of the simplicity of this method, and the fast evaluation of the objective function, this allows for fast convergence. This is a simple and effective method for finding a minimum, and my coding of it is simple and efficient, with little or no regions of computational inefficiency.

Steepest Descent

This result surprised me. I had expected it to be quite fast, and require more iterations than it actually takes. This is surprising, because the gradient function is computed analytically which is significantly faster than a numerical approximation. However, upon further investigation, I realized the source of the slow speed. During the line search, the routine iterates an average of 13 times per iteration of the Steepest Descent algorithm. This results in closer to 117 iterations, and the recomputed values are shown in the last row of the table above.

To try to rectify this, I re-evaluated my coding for the line search to try to improve the efficiency. However, even with the modifications I made it still requires the longest time to converge. I believe that using a simpler, although possibly less robust, line search scheme would help this method converge faster. I believe that the endless if else statements required (as suggested by the text) slow this evaluation considerably. I also believe that the quadratic fit is almost a complete waste of time, supported by the results I have obtained using Newton's Method.

Pure Newton Method

I feel that this result is skewed because of this specific problem and the lack of a penalty for failing to converge. The method converges very fast because of the analytically computed gradient function and Hessian matrix. MATLAB is also very efficient at linear algebra, and the inversion of this simple 2 by 2 matrix would be far more complex with more variables. The high number of iterations is surprising, as I would have thought that this method would require very few iterations, but that each would take some time to compute.

EASE OF USE

I feel that this “ease of use/coding” is going to be skewed by the impression that I got from following the ME6103 text, with which there were aspects I was very unimpressed. However, my gut feeling is as follows.

The Pure Newton’s method was the easiest to code because implementing mathematics and mathematical formulae in MATLAB is very easy and straightforward, and there are very few steps. The Pattern Search method was the second most straightforward. The search for a direction is simple, as is the movement in that direction once it is found. The Steepest Descent method was a pain in the ass to code. Finding the gradient was very easy, and finding the three points spanning the minimum was also fairly simple. The line search with the quadratic fit was terrible. I feel that replacing this with a simple cleaner pure golden section search may be more effective, and certainly would be easier to code. However, this being said, it is also my most robust algorithm, and part of this complexity is the considerations of things such as machine error.

EFFICIENCY

I consider the efficiency of a method different than the number of iterations, but rather the number of function calls required. In this manner the Pure Newton’s method is most efficient, followed by the Steepest Descent and then Pattern Search. This correlates somewhat well to the speed results. However, I believe that if an analytical gradient function were not provided, then this efficiency of particularly the Newton and to a much lesser extent the Steepest Descent methods would decrease. This is because in order to calculate a first order derivative, two points are needed, and for a second derivative three are needed. This is then required in each dimension, and for the Hessian, all of the partial derivative combinations are required also. This would require an additional 24 function calls per iteration just for the 2D Hessian, which would massively reduce the computational efficiency of this method. In this way I feel that the Steepest Descent method is the most efficient.

RECOMMENDATIONS

Considering what I have learned through implementing these three methods for an analytical objective function, I would recommend different methods for different situations.

Pattern Search

I feel that this would be a very good default choice. It is simple to code, and runs quickly. It also does not depend on any analytical or numerical gradients. Its only flaw is determining good parameter values, however I believe that a reduction factor of 0.25 will almost always yield fast results. That leaves the initial step size, which is dependent upon the problem. However, being conservative with this means more functional evaluations, but not too many, and thus this is not critical. If I had to recommend only one method, this would be it.

Steepest Descent

This method requires the least functional evaluations of a method that is robust enough to use without a lot of knowledge a priori. Even with a numerical gradient, second order approximations of the slope only require two evaluations per dimension. The line search also only needs one evaluation per iteration, leaving it many times lower than the pattern search. This would be a good method to use if the objective function is complex and requires a long time to compute, such as a Finite Element Method or other numerical scheme. However, this method also took me the longest time to code, although with a simpler line search routine this could be cut down at the cost of some efficiency.

Pure Newton's Method

I really do not feel I can recommend the Pure Newton method because of its tendency to diverge. However, it is very efficient when using an analytical Hessian, and because of its low number of iterations is still acceptable when using a numerical Hessian. However, this method is really only applicable when a lot is known about the objective function space, in order to provide the method with a starting point from which it will converge to the minimum.

LEARNING

Through completing this assignment I feel I have learned the following key points:

- The if/else structure in a logical sequence is a pain to code and is also slow and inefficient, mathematics is much simpler and faster
- Quadratic fits are very finicky and can easily explode
- Our text book needs a few more revisions
- Starting point selection is key, as is using multiple starting points
- Knowing a little about the objective function space is also useful, as you can see if multiple minima or any asymptotic regions exist that may affect the optimization routine.
- How to use a variable number and type on input arguments into a MATLAB function, and write file headers that can be read by the help files
- With today's computational power available, the simpler methods such as pattern search are very viable methods

MATLAB CODE COMPILATION

I have compiled my MATLAB codes in the following pages. They are in order of main function, followed by a sub function that they call. This is outlined below, sub functions that are also called, but have already been compiled are shown in italics:

1. Pattern Search
 - a. FEfunction
 - b. Explore
 - c. SpacePlotter
2. SteepestDescent
 - a. *FEfunction*
 - b. FEgradient
 - c. ThreePoint
 - d. QuadFit
 - e. *SpacePlotter*
3. NewtonsMethod
 - a. *FEfunction*
 - b. *FEgradient*
 - c. FEHessian
 - d. *SpacePlotter*

The following codes are also compiled. These functions are used to gather performance information from the various methods.

4. NewtonTest
5. SpeedComparison
6. PParameters