An Investigation into Numerical Solutions of Nonlinear Systems - Algorithm Analysis of BFGS and SR1 Quasi-Newton Methods

Jordan Smart and Sophie Redd

November 18, 2021

1 Abstract

In this investigation, we seek to analyze modern algorithm extensions and updates to classic Quasi-Newton methods. Primarily, we will investigate extensions and updates to Broyden method. Broyden method employs a rank one update to the Jacobian, which prevents the cost of inversion and reduces the calculations of derivatives.

2 Introduction

Classic Newtons method of root finding employs the use of a tangent line to predict the next guess and find the root within a given tolerance. The problem with this method of root finding is that when there is a system of nonlinear equations, the cost of inverting the Jacobian is $O(n^3)$. This also requires the first derivative of the functions over the entire interval. Thus, this is why Broyden was developed. Broyden uses a rank one update to the Jabobian which reduces the cost of inversion from $O(n^3)$ to $O(n^2)$ when the Jacobian of the first step is a sparse matrix. This is a significant improvement to Newton and relieves the cost of knowing the derivative at every point. However, because Broyden depends on the initial Jacobian being sparse, this reduces the cases where it is faster. Broyden can be costly, and lacks some accuracy in certain cases. For this reason, we will invsetigate extension of Broyden such as BFGS and SR1.

2.1 Hessian Matrix

In each algorithm, we apply a Hessian matrix. It is well known that the Hessian matrix is a square matrix composed of partial second derivatives, named after Ludwig Otto Hesse, and takes the following form,

$$\mathrm{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

In this discussion, we will investigate application of approximations of this matrix. This matrix is distinct from the Jacobian matrix that is classically used in newton and quasi-newton methods of solutions to non linear systems.

We will name the approximation to the Hessian as B_k where B_0 is equal to the Hessian as applied to the function at the initial guess and B_k is the Hessian approximated at the next step. In the algorithm section of this paper we will define the update for each algorithm.

The Hessian is going to be more demanding on the first step because it requires the function to have a continuous first derivative.

2.2 BFGS

The BFGS uses a symmetric rank two update to be more accurate. It Preserves qualities of the hessian $O(n^2)$ which is less costly than the Broyden for a dense matrix ($O(n^3)$). This uses a lot of data so there is another algorithm called L-BFGS which uses less memory for lower computation applications.

The BFGS algorithm applies a rank 2 update to the Hessian at each step, the general for of which is as follows,

$$B_k = B_{k-1} + auu^T + bvv^T \tag{1}$$

where a and b are scalar multiples, u and v are updated approximations to the matrix. We then apply secant method $B_k s = y$ to this and arrive at the following

$$y - Bs = (au^T s)u + (bv^T s) \tag{2}$$

From this, we apply the generalization of Sherman Morrison, the Woodbury formula to invert the updated Hessian approximation as follows

$$C^T = (I - \frac{sy^T}{y^T s})C_{k-1}(I - \frac{ys^T}{y^T s}) + \frac{ss^T}{y^T s} \tag{3}$$

BFGS is used in high level computing to compute solutions to systems quickly and efficiently.

2.3 SR1 (Symmetric Rank-One)

This is very similar to BFGS except it updates the Hessian every once in a while and is more efficient overall.

Where BFGS does a rank one and rank two update to the Jacobian, the SR1 employs a rank one update on the Hessian which significantly simplifies the algorithm.

When it is impossible to impose curvature conditions, Hessian approximations are desirable.

The SR1 algorithm applies a, guess what, symmetric rank 1 update to the Hessian, this is much cheaper than the rank 2 update in BFGS but as we will investigate further, is much less stable under certain conditions. We will recall the rank one updates is as follows

$$B_k = B_{k-1} + auu^T \tag{4}$$

When we then apply secant method, we find

$$(au^T s)u = y - Bs \tag{5}$$

We observe here, that is only holds if u is a multiple of y, so we then apply $u = y - Bs$ and find

$$B_k = B_{k-1} + \frac{(y - Bs)(y - Bs)^T}{(y - Bs)^T s} \tag{6}$$

To then find the inverse of the updated Hessian approximation, we simply apply the Sherman Morrison formula and get

$$C_k = C_{k-1} + \frac{(s - Cy)(s - Cy)^T}{(s - Cy)^T y} \tag{7}$$

2.3.1 Differences Between SR1 and BFGS

BFGS guaranties a positive definite for B_{k+1} when B_k is positive definite, SR1 does not include this safegaurd. SR1 can be positive definite but because it is only a rank 1 update, it does not guarantee positive definiteness. Positive definiteness means that we preserve the space that we are in. Always positive real numbers.

The SR1 has a very good approximation for the Hessian inverse matrix then the BFGS.

SR1 has built in safeguards to prevent breakdowns

For small n, we expect BFGS to be more accurate.

2.3.2 Similarities Between SR1 and BFGs

Both BFGS and SR1 take $O(n^2)$ time vs Newtons $O(n^3)$

Both are used to solve LP and other similar applications.

3 Algorithm Derivations

3.1 BFGS

The BFGS algorithm, like many other newtons algorithms, starts with an initial guess at x_0 and continues to iterate getting closer to a more accurate answer. Starting with x_0 as our initial guess and B_0 as our initial hessian approximation, we run the following algorithm at x_k and B_k

$$B_k p_k = -\nabla f(x_k) \tag{8}$$

where we want to solve for p_k, and our $\nabla f(x_k)$ is the gradient function at the given x_k. we then want to find the step size α_k by doing

$$\alpha_k = argmin f(x_k + \gamma p_k) \tag{9}$$

where > 0. we set $s_k = \alpha_k p_k$ and update $x_{k+1} = x_k + x_k$. we want $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ so we can update the hessian matrix as such.

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{s_k^T (y_k - B_k s_k)} \tag{10}$$

3.2 SR1

The SR1 algorithm follows very similar steps to the BFGS, except at each step we also approximate the inverse hessian matrix instead of inverting the hessian matrix each time (i.e $H_k = B_k^{-1}$)

$$H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{y_k^T (s_k - H_k y_k)} \tag{11}$$

4 Numerical Tests and Results

4.1 Testcases

To test BFGS and SR1, we will want to test it on a strongly convex system due to its positive definiteness of BFGS which implies near perfect convergence for a positive definite system.

Further, we will want to test against something that is solvable using newtons method to compare. Thus, we want something with at least a continuous first derivative on the interval. For these reasons, we will apply the LP barrier problem. We observe that the standard LP barrier problem takes a form as follows,

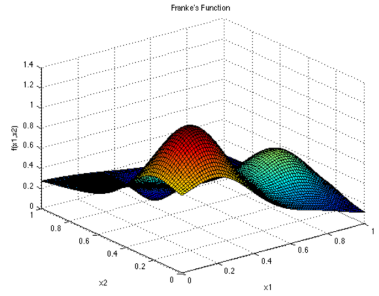
$$\min_x : C^T x - \Sigma_{i=1}^m \log\left(b_i - a_i^T x\right) \tag{12}$$

as applied to $Ax = b$.

We observe that this problem is strongly convex and will exploit the positive definiteness of BFGS. SR1 will then also be able to converge and we will be able to compare the algorithms on a classic minimization problem

To break BFGS, we will want to find a condition on the Hessian that negates the existence of an inverse. This same condition will also break SR1.

For robustness, we may test on the Franke function, which is a function that has multiple local minima. Our inspiration for the Franke function is as follows,



$$f(\mathbf{x}) = 0.75\exp\left(-\frac{(9x_1-2)^2}{4} - \frac{(9x_2-2)^2}{4}\right) + 0.75\exp\left(-\frac{(9x_1+1)^2}{49} - \frac{9x_2+1}{10}\right)$$
$$+ 0.5\exp\left(-\frac{(9x_1-7)^2}{4} - \frac{(9x_2-3)^2}{4}\right) - 0.2\exp\left(-(9x_1-4)^2 - (9x_2-7)^2\right)$$

Figure 1: This shows the plot and equation of the Franke funtion. [6]

4.2 Results

This section will include our numerical resutls which will include solutions from newtons, BFGS, and SR1 for each test case, as well as the absolute error. We will also compare the wall clock time for each algorithm.

We have yet to implement the code, but we have run as simple test case using Newtons method to show what we expect to get as results from our quasi-newton codes. The test case is as follows, with an initial condition of $(3, 3, 3)$ and a tolerance of 10^{-3}

$$x = x - \log\left(10 - y * x\right)$$
$$y = y - \log\left(10 - y * z\right)$$
$$z = z - \log\left(10 - z * x\right)$$

Our results are as follows,

```
newtons:
8
x = 1.8703622328012195
y = 1.8703622328012195
z = 1.8703622328012195
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x7f59ca0b3b50>]
```

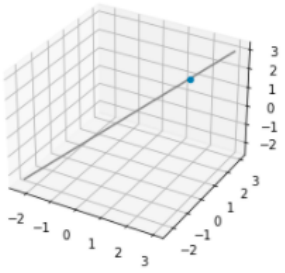


Figure 2: This shows the found root using Newtons method, the intersection found is plotted on the system.

4.3 Analysis

Here we will have an in depth analysis of what we observed from our test cases. We will determine the best cases for each algorithm and the time cost analysis.

5 Conclusion

Here we will finalize our finds, discuss and difficulties we had along the way and any further investigations that are needed or recommended.

5.1 Honorable mentions

The BHHH equation is a pseudo Newton's method, and is very similar to Newton's method. Equation can be found both on wikipedia and in reference [5]. The advantage of BHHH over Newton's method is that Bt is a faster equation to get (compared to Ht found in the refence), and that Bt is positive and will increase LL(B) in each iteration, meaning it has a very good convergence rate, i.e. is unlikely to diverge. The reference to these advantages can be found in the earlier reference, and it will show that LL(B) is the log-likelihood function that they use as an example throughout the document. A downside of BHHH is that some iterations are very small and slow, and this can be the case when far from the point of interest. This method is used for maximizing the log-likelihood function as mentioned earlier, over using Newton's method, taking the derivative, and finding zero.

This is like using Taylor expansions to calculate the derivative approximation instead of calculating the whole jacobian each time or even doing a rank one update (Broyden). This will be faster for certain cases but when there is no good convergence of the approximation then we slow down. This is why it is used for maximizing behavior. This will use then a jacobian and a hessian.

There is also an updated version. L-BFGS which is less costly in memory and is used in machine learning to quickly compute solutions without consuming too much memory

We will likely write down the equations and pseudo code for BFGS and focus more on that while mentioning how it came from DFP and leads to L-BFGS without as much detail there (i.e no equation just implementation).

References

[1] W. Press et al, Numerical Recipies in C, pp. 20-23 (1992).

[2] Marini, L., Morini, B. and Porcelli, M. Quasi-Newton methods for constrained nonlinear systems: complexity analysis and applications. Comput Optim Appl 71, 147–170 (2018). https://doi.org/10.1007/s10589-018-9980-7

[3] K.Zico, Quasi-Newton Methods. Carnagie Mellon University (2018).https://www.stat.cmu.edu/ ryantibs/convexopt-F18/lectures/quasi-newton.pdf

[4] Farzin Modarres, Malik Abu Hassan, Wah June Leong, Computers & Mathematics with Applications (2011). https://www.sciencedirect.com/science/article/pii/S0898122111004202

[5] Berkely (2002), https://eml.berkeley.edu/choice2/ch8.pdf

[6] Franke, R. (1979). A critical comparison of some methods for interpolation of scattered data (No. NPS53-79-003). NAVAL POSTGRADUATE SCHOOL MONTEREY CA, https://www.sfu.ca/ ssurjano/franke2d.html