A Comparison between Optimization Methods using Hessian Matrix and Approximation of Hessian matrix

Ji Cheol, Kang
Department of Data Sciences,
Seoul National University of Science and Technology

Introduction

In optimization problem, Newton's method calculates second derivative term called Hessian matrix for quadratic approximation and utilize it in update step. But because calculating Hessian matrix is quite expensive in terms of computation, alternative methods have been proposed which use approximation term of Hessian matrix. These include The Symmetric Rank 1 (SR1) method, Broyden–Fletcher–Goldfarb–Shanno(BGFS) method and they are also called Quasi-Newton methods in general. In this report, comparison between original Newton's method and some Quasi-Newton methods like BFGS, SR1 will be implemented and conducted first. After comparison, results are analyzed on diverse settings. With this analysis, we try to figure out which algorithm is effective for each optimization setting and what should be considered in practical implementation process.

Background

Newton's method

Different from root finding problem, Newton's method in optimization problem need to find root of derivative of original function and not root of original function itself. Because of this, Newton's method requires calculating Hessian matrix of original function and its update formula resulted as:

$$x^{(k)} = x^{(k-1)} - \left(\nabla^2 f(x^{(k-1)})\right)^{-1} \nabla f(x^{(k-1)}), \qquad k = 1, 2, 3,$$

BFGS method

BFGS takes advantage from approximating Hessian using secant equation and weighted Frobenius norm. It changes problem from original form to update matrix B which satisfying secant equation. L-BFGS-B is another form of BFGS which is known to be more useful in problems with constraints. The update formula of BFGS is represented as:

$$B_{k+1} = B_k + \frac{\mathbf{y}_k \mathbf{y}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}} \mathbf{s}_k} - \frac{B_k \mathbf{s}_k \mathbf{s}_k^{\mathrm{T}} B_k^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}} B_k \mathbf{s}_k}.$$

SR1 method

SR1 method updates B with rank-1 symmetric matrix iteratively so that B could remain

symmetric and satisfy secant equation. But it does not guarantee positive definiteness of B or H. The update representation of SR1 method is:

$$B_{k+1} = B_k + \frac{(y_k - B_k \Delta x_k)(y_k - B_k \Delta x_k)^T}{(y_k - B_k \Delta x_k)^T \Delta x_k},$$

Experiments Setting

For optimization test, three well-known test functions are used called Rosenbrock, Booth, Himmelblau's function, respectively.

Rosenbrock function:

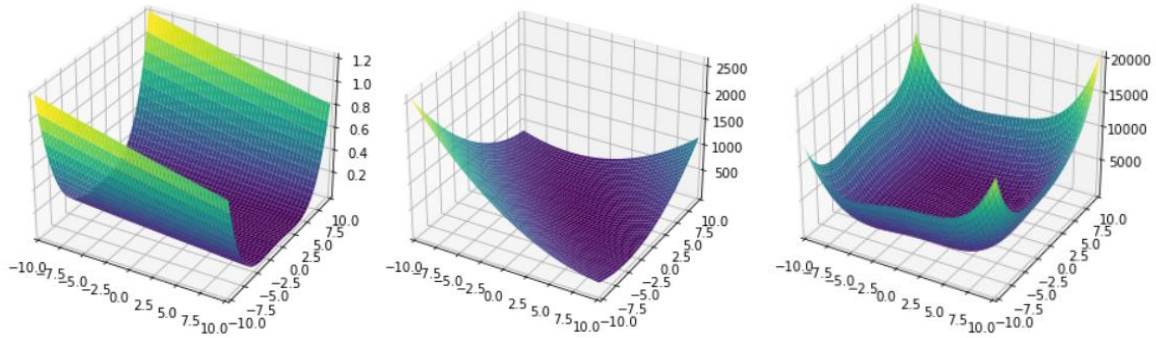
$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

Booth function:

$$f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2 \quad -10 \leq x, y \leq 10$$

Himmelblau's function:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2. \quad -5 \leq x, y \leq 5$$



<**Fig. 1**. The $3d$ Plot of Rosenbrock(left), Booth(center), Himmelblau's(right) functions.>

The global minimum point of Rosenbrock functions is $f(1, 1) = 0$ while $f(1, 3) = 0$ for Booth function. Himmelblau's function is minimum at multiple points: $f(3.0, 2.0) = f(-2.805118, 3.131312) = f(-3.3779310, -3.283186) = f(3.584428, -1.848126) = 0$.

Newton's method, BFGS and SR1 test is implemented with custom script while L-BFGS-B is conducted with SciPy package.

For BFGS and SR1, the learning rate(alpha, α) is set to 0.1, 0.01, 0.001 to find optimal rate. The max iteration is set to 1000 and some random points are selected to initial points considering distance from global minimum. The tolerance is set to 0.00001.

For BFGS, 'damp-update' algorithm is implemented to prevent violation of curvature condition which is adopted from SciPy package.

Results

Rosenbrock function	Initial points		
	[1.5, 2.5]	[-5, 4]	[-18, 12]
Optimization	Success	Success	Success
Step / Time(s)	4 / 0.4856	6 / 0.1376	5 / 0.1307

Booth function	Initial points		
	[1.5, 2.5]	[-5, 12]	[-10, 10]
Optimization	Success	Success	Success
Step / Time(s)	1 / 0.005	1 / 0.0060	1 / 0.0069

Himmelblau's function	Initial points		
	[-2, 5]	[-1, 2]	[-3.5, -2.1]
Optimization	Success	Failed	Success
Step / Time(s)	5 / 0.1795	-	4 / 0.1716

<**Table 1.** Newton's method optimization results.>

Rosenbrock function		Initial points		
		[1.5, 2.5]	[-5, 4]	[-12, 8]
$\alpha = 0.1$	Optimization	Failed	Error	Error
	Step / Time(s)	-	-	-
$\alpha = 0.01$	Optimization	Failed	Closed	Error
	Step / Time(s)	-	1000 / 0.0529	-
$\alpha = 0.001$	Optimization	Failed	Failed	Failed
	Step / Time(s)	-	-	-

Booth function		Initial points		
		[1.5, 2.5]	[-5, 12]	[-10, 10]
$\alpha = 0.1$	Optimization	Success	Success	Success
	Step / Time(s)	54 / 0.006	142 / 0.011	145 / 0.01
$\alpha = 0.01$	Optimization	Success	Success	Success
	Step / Time(s)	588 / 0.0329	1000 / 0.0499	1000 / 0.0479
$\alpha = 0.001$	Optimization	Success	Failed	Closed
	Step / Time(s)	1000 / 0.0469	1000 / 0.0489	1000 / 0.0479

Himmelblau's function		Initial points		
		[-2, 5]	[-1, 2]	[-3.5, -2.1]
$\alpha = 0.1$	Optimization	Error	Error	Error
	Step / Time(s)	-	-	-
$\alpha = 0.01$	Optimization	Failed	Success	Closed
	Step / Time(s)	-	1000 / 0.0609	1000 / 0.0518
$\alpha = 0.001$	Optimization	Failed	Closed	Success
	Step / Time(s)	-	1000 / 0.0589	1000 / 0.0588

<**Table 2.** BFGS method optimization results.>

Rosenbrock function		Initial points		
		[1.5, 2.5]	[-5, 4]	[-18, 12]
$\alpha = 0.1$	Optimization	Failed	Error	Error
	Step / Time(s)	-	-	-

$\alpha = 0.01$	Optimization Step / Time(s)	Failed -	Closed 1000 / 0.0389	Error -
$\alpha = 0.001$	Optimization Step / Time(s)	Failed -	Failed -	Failed -

Booth function		Initial points		
		[1.5, 2.5]	[-5, 12]	[-10, 10]
$\alpha = 0.1$	Optimization Step / Time(s)	Success 54 / 0.01	Failed -	Closed 1000 /0.0594
$\alpha = 0.01$	Optimization Step / Time(s)	Success 588 / 0.0289	Failed -	Failed -
$\alpha = 0.001$	Optimization Step / Time(s)	Success 1000 / 0.0289	Failed -	Failed -

Himmelblau's function		Initial points		
		[-2, 5]	[-1, 2]	[-3.5, -2.1]
$\alpha = 0.1$	Optimization Step / Time(s)	Error -	Error -	Error -
$\alpha = 0.01$	Optimization Step / Time(s)	Failed -	Closed 1000 / 0.0429	Closed 1000 / 0.0449
$\alpha = 0.001$	Optimization Step / Time(s)	Failed -	Closed 1000 / 0.0539	Closed 1000 / 0.0468

<**Table 3.** SR1 method optimization results.>

Rosenbrock function		Initial points		
		[1.5, 2.5]	[-5, 4]	[-12, 8]
BFSG $\alpha = 0.01$	Optimization Step / Time(s)	Failed -	Closed 1000 / 0.0529	Error -
L-BFGS-B	Optimization Step / Time(s)	Success 9 / 0.01125	Success 19 / 0.04712	Success 20 / 0.0556

Booth function		Initial points		
		[1.5, 2.5]	[-5, 12]	[-10, 10]
BFSG $\alpha = 0.01$	Optimization Step / Time(s)	Success 588 / 0.0329	Success 1000 / 0.0499	Success 1000 / 0.0479
L-BFGS-B	Optimization Step / Time(s)	Success 2 / 0.0008	Success 9 /0.0102	Success 8 / 0.0079

Himmelblau's function		Initial points		
		[-2, 5]	[-1, 2]	[-3.5, -2.1]
BFSG $\alpha = 0.01$	Optimization Step / Time(s)	Failed -	Closed 1000 / 0.0429	Closed 1000 / 0.0449
L-BFGS-B	Optimization Step / Time(s)	Success 8 /0.007	Success 8 /0.009	Success 8 / 0.007

<**Table 4.** BFGS & L-BFGS-B methods optimization results.>

In results, ‘Failed’ means optimal value is somewhat far from known optimal values or stuck in the wrong (or local) point. ‘Error’ means outputs showed null value or overflow error. It seems that these errors come from zero division error or singular matrix error.

With Newton's method, problems are optimized in few iterations compared to other algorithm. However, it takes more time in single iteration.

In BFGS, it caused many fail and error outputs especially in Rosenbrock function and Himmelblau's function. But adjustments in learning rate and initial point induced closely optimized results. Different from Newton's method, it needs more iterations while it takes less time in single iteration.

SR1 results is quite similar with BFGS results. It showed errors and fails in same condition compared to BFGS. Performance is also close to BFGS. What differ from BFGS is that it failed to optimization more in Booth function, which is quite simple function, than BFGS did.

In results L-BFGS-B algorithm with SciPy package, it succeeded optimization in all settings and showed high performance in computation compared to manually scripted BFGS algorithm.

Conclusion

Overall, Newton's method showed stable results with few errors and fail while BFGS and SR1 outperformed in computational cost aspect. Even though BFGS does not need Hessian matrix, it needs to satisfy curvature condition. So, we should implement adjustment term. In this report, 'damp-update' method is used to handle curvature condition adopted from SciPy package. Due to this complex implementation process, it might cause some minor errors like division error or overflow.

In comparison BFGS and L-BFGS-B, L-BFGS-B is generally better than BFGS. However, BFGS is manually implemented while L-BFGS-B is conducted using SciPy package. So, this result does not give much information but showed stability of package which is widely used.

In conclusion, Quasi-Newton is known to better than original Newton's method, but it should be considered that some Quasi-Newton methods need complex implementation which could cause obstacles to get optimization result.

But once algorithms are implemented in stable, Quasi-Newton methods can outperform original Newton method which is observed by a lot of references.