Nonlinear Unconstrained Optimization Methods

Ahmed H. Abdelrazik *University of Sciece and Technology in Zewail City*

Friday, January 1, 2021

As most of optimization problems are restricted by constraints, unconstrained optimization techniques are useful in constrained optimization problems. More clearly, constrained problems can be turned into unconstrained problems by invoking the objective function and the constraints altogether in order to get another objective function that inherits the constraints within itself. Then they are solved using unconstrained optimization techniques. In this report, we are to implement 1-D minimization techniques that are later used in optimizing the step length. Moreover, we will implement three nonlinear unconstrained optimisation methods (Quasi-Newton, Marquardt, Fletcher-Reeves). Then, those algorithms are used to solve Rosenbrock's parabolic valley function, afterwards a comparison is conducted between those methods in terms of the number of iterations, CPU time, and the convergence to the optimal solution.

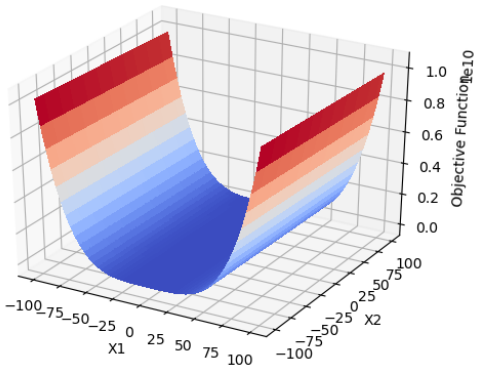

Figure 1: *Rosenbrock's parabolic valley function*

Introduction

Rosenbrock's parabolic valley function is a benchmark problem that is used in order to determine the performance of an optimization algorithm.

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (1)$$

It's somehow challenging because the global optimum is located on plateau surface in a valley as shown in Figure 1. Hence, we can get close to the optimum point so quickly, but once we approached the optimum, the convergence becomes so slow and very little improvements are made at each step close to the optimum. In this report, a combinations of 1-D minimization techniques alongside with nonlinear unconstrained algorithms are used in order to solve this Rosenbrock's parabolic valley function.

Fletcher-Reeves Method

Fletcher-Reeves method is firstly used to solve the given problem. This method utilizes the conjugate directions rather than the steepest descent directions in order to achieve faster convergence. it's said to have a quadratic convergence.

Algorithm

Given an initial point X_0

Step 1 first direction is set to be the negative gradient of the current point.

$$s_0 = -\nabla f_0 \quad (2)$$

Step 2 update the x value

$$x_1 = x_0 + \lambda^* s_0 \quad (3)$$

Step 3 Use Cubic interpolation in order to get the optimum step length

Step 4 Then enter a loop:

- get the gradient at the new point
- Calculate beta:

$$B_i = \frac{||\nabla f_i||_2^2}{||\nabla f_{i-1}||_2^2} \quad (4)$$

- update the searching direction:

$$s_i = \nabla f_i + B_i s_{i-1} \quad (5)$$

- update the x value:

$$x_i = x_{i-1} + \lambda^* s_i \quad (6)$$

- get lambda star using Cubic interpolation.
- get the gradient at the current point. If the gradient norm is less than epsilon exit the loop, else repeat.

Results

As shown in Figure 2 the algorithm converged in 30 iteration with CPU time = 3.867 seconds Figure 3

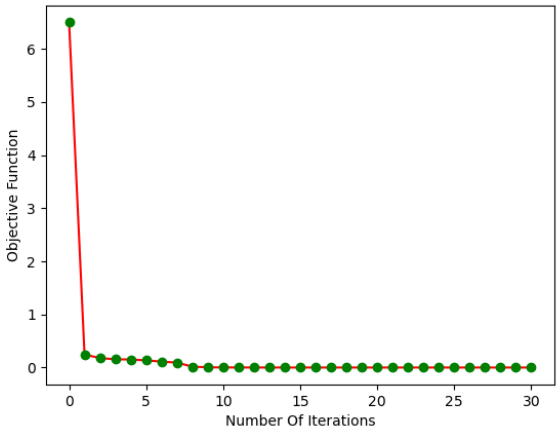

Figure 2: *Number of iterations vs Objective function using Fletcher-Reeves*

shows the improvements of the objective function vs the current points in the runtime.

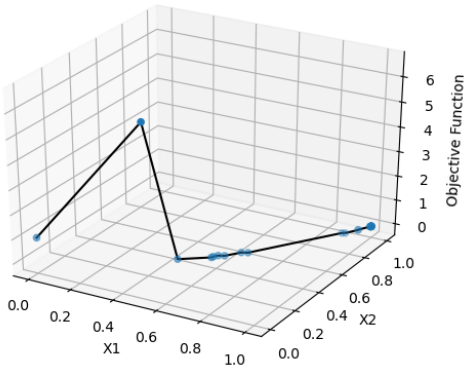

Figure 3: *X1,X2 values vs Objective function improvements during the runtime*

Marquardt Method

Basically Marquardt method is a hybrid method between the gradient descent and Newton algorithms. This method utilizes the privilege of using the steepest descent method when it's away from the optimum point and as it get closer to the optimum it takes the advantage that the newton direction converges faster in the vicinity of the optimum solution.

Algorithm

Given an initial point X_0, constant α, c_1, and c_2. where $1 > c_1 > 0$, $c_2 > 1$, and alpha of order 10^4

Step 1 Calculate the gradient at the initial point ∇f_i

Step 2 get the current searching direction using the hessian matrix B_i

$$s_i = -[B_i + \alpha * I]^{-1} \nabla f_i \tag{7}$$

Step 2 update the x value

$$x_i = x_{i-1} + \lambda^* s_i \tag{8}$$

Step 3 Use Cubic interpolation in order to get the optimum step length.

Step 4 Get the objective function value at x_i.

- if $OFnext < OFprev$:

$$alpha = c_1(alpha) \tag{9}$$

else

$$alpha = c_2(alpha) \tag{10}$$

- if norm of the gradient at the current point < epsilon, exit the loop, else go to step 2:

Results

As shown in Figure 4 the algorithm converged in 15 iteration with CPU time = 2.418 seconds

Figure 5 shows the surface of Rosenbrock's function as the algorithm moves through a banana like shape in order to reach the global optimum.

Figure 6 shows the improvements of the objective function vs the current points in the runtime.

The results obtained from Marquardt algorithm ensures that Marquardt method has higher performance that Fletcher-Reeves algorithm, as it has less number of iteration and CPU time.

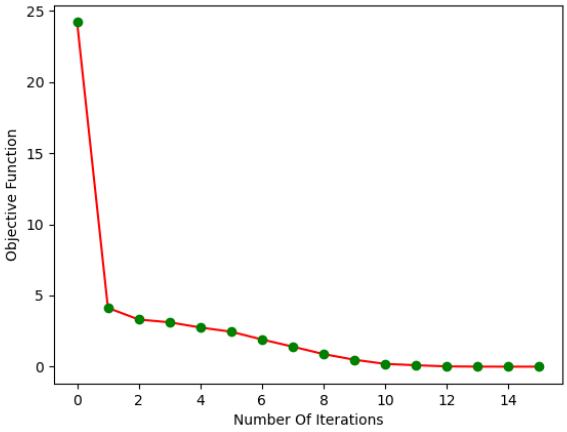

Figure 4: *X1,X2 values vs Objective function improvements during the runtime*

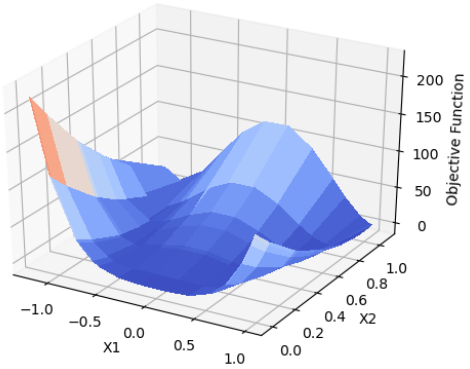

Figure 5: *Number of iterations vs Objective function using Marquardt Algorithm*

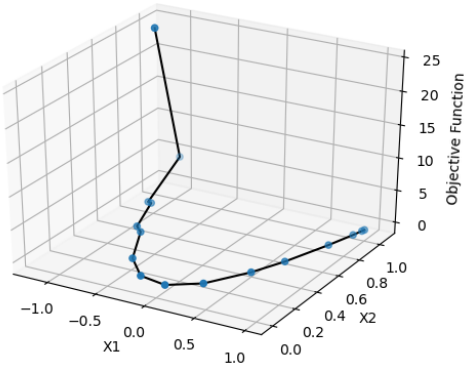

Figure 6: *X1,X2 values vs Objective function improvements during the runtime*

Quasi-Newton Method

Newton method requires the calculation of the hessian at each step which is considered a huge overhead on the algorithm in terms of flops and CPU time. Also, some problems it's not possible to obtain the Hessian matrix as in Navier-Stokes equation in fluid dynamics. Hence Quasi-Newton method tries to approximate the Hessian matrix, and this approximation is being updated at each step. Theere are multiple updating techniques like Rank 1, Rank 2 updates (Davidon, Fletcher and Powell (DFP), Broyden–Fletcher–Goldfarb–Shanno (BFGS)). In the algorithm DFP rank 2 update will be used in order to update the approximated Hessian matrix.

Algorithm

Given an initial point X_0.

Step 1 Initialize the approximated Hessian matrix by the identity B_i.

Step 2 Calculate the gradient of the initial point. Then get the searching direction.

$$s_0 = -[B_i]^{-1}\nabla f_0 \tag{11}$$

Step 2 3 update the x value

$$x_1 = x_0 + \lambda^* s_0 \tag{12}$$

Step 4 Use Cubic interpolation subroutine in order to get the optimum step length.

Step 5 Enter a loop:

- get the gradient and update the inverse Hessian matrix using DFP.
- Calculate the searching direction.
- get λ^* by Cubic interpolation subroutine.
- Update the x value.
- if the norm of the gradient at the current point $<$ epsilon, exit the loop, else repeat.

Results

As shown in Figure 7 the algorithm converged in around 25 iteration with CPU time = 3.269 seconds .Figure 8 shows the improvements of the objective function vs the current points in the runtime.

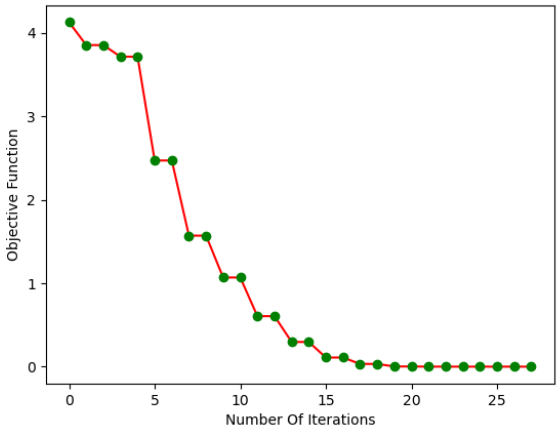

Figure 7: *X1,X2 values vs Objective function improvements during the runtime*

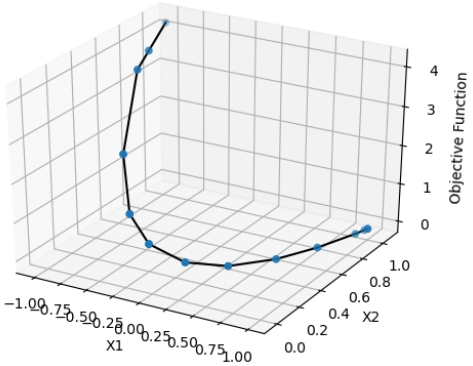

Figure 8: *Number of iterations vs Objective function using Quasi-Newton Algorithm*

Conclusion

In conclusion, Marquardt algorithm has faster convergence than Quasi-Newton algorithm and Fletcher-Reeves in solving the Rosenbrock's parabolic valley function in terms of the CPU time and the number of iterations.

References

[1] Rao, Singiresu S *Engineering optimization*, John-Wiley, 2009.

[2] Luenberger, David G and Ye, Yinyu, *Linear and nonlinear programming*, Springer, CA, 2016.