Derivative based Multi-Variable Optimization - Conjugate Gradient Methods

Srikrishnan Sengottai Kasi

November 9, 2021

Abstract

This report focuses on implementing the Fletcher-Reeves conjugate gradient method in Matlab. The conjugate gradient method helps us to identify the direction in which the minima is found for a Multi-variable testing function with reduced storage. In this paper I will try to run the implementation in quadratic and non-quadratic test functions and interpret the results.

1 Introduction

Solving large system of linear equations can be done by the conjugate gradient methods. When I try to minimize a non-quadratic objective function over a large dimension with limited knowledge of the parameters, I try to find the direction of the minima using the conjugate gradient methods. Also, conjugate gradient method does not require quadratic dimension storage. In this report I have tried two things. i) Implemented the Fletcher reeves conjugate gradient method in Matlab. ii) Analyzed the results of running my implementation over two different type of multi-dimensional test functions.

2 Background

Before going into the implementation, the technical background to understand the experiment is needed. The two test function that is used to carry this experiment are the Ostermeier Ellipsoid and the Rosenbrock function.

The Ostermeier Ellipsoid is,

$$f(x) = x^T AX, where A = diag(\alpha, \alpha^{(n-2/n-1)}, .., \alpha^{(1/n-1)}, 1)$$

Few characteristics of Ostermeier ellipsoid is it generates an ill-conditioned Hessian matrix. So, it will be good to test the performance of the algorithm on it. If the algorithm is able to achieve a linear convergence graph then it is clearly possible for algorithm to find the global minimizer of the Ostermeier ellipsoid. Since it is a quadratic test function the global minimizer should be found by the algorithm at the iteration which is equal to the input normally distributed vector's x_0 dimension.

The Rosenbrock function is,

$$f(x) = \sum_{i=1}^{n-1}(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2)$$

Rosenbrock function is a clear example of non-quadratic multi dimensional test function. Ideally, the algorithm should find the global minimizer of the non-quadratic test function which can be verified if the plot of function values are linearly convergent. When the Rosenbrock function is our test function we will try to restart the algorithm after few iterations and see the difference of the linear convergence.

In the quasi-newton approaches, the direction of the minimizer is given by the second-order derivative. This algorithm gives the direction without computing the second-order derivative. So it wont generate a symmetrical hessian matrix to find the direction of the global minimizer.

The Fletcher-Reeves method finds the direction with the help of the below formulation.

$$p_0 = -\Delta(f(x));$$
$$p_k = -\Delta f(x_k) + (\Delta f(x_k)^T \Delta f(x_k)/\Delta f(x_{k-1}^T \Delta f(x_{k-1})) * p_{k-1}$$

3 Experiment

I will be considering a 100-dimensional vector as my initialization sample point and run my experiment. Below is my algorithm implementation in matlab.

```
function [recn, recfx] = conjugategradsearch(testFunc, x)
    dim = 100;
    iterations = 100;
    if nargin < 2
        x = randn(dim, 1);
    end
    recn = zeros(iterations, 1);
    k = 1;
    recfx = zeros(iterations, 1);
    [fx, p0] = testFunc(x);
    recfx(k, :) = fx;
    p0 = -p0;
    while true
        if k == iterations
            break;
        end
        gf = @(g) testFunc(x + g*p0);
        [gamma, n, ~] = ags(gf, 0, 1, 1.0e-40);
        recn(k, :) = n;
        newx = x + (gamma*p0);
        [~, dk] = testFunc(newx);
        numerator = dk.'*dk;
        [~, dkm1] = testFunc(x);
```

```
24              denominator = dkm1.'*dkm1;
25              pn = -dk + ((numerator/denominator)*(p0));
26              recfx(k, :) = testFunc(newx);
27              p0 = pn;
28              x = newx;
29              k = k + 1;
30        end
31  end
```

The lines 1 to 13 initializes the algorithm by taking a normally distributed sample points and function is being applied to those points. The first derivative gives the value of p_0. This p_0 gives the initial direction of where to look for the global minimizer in the next step. The iteration count and the dimension count is set to be equal for my implementation of the algorithm to test the efficiency for the quadratic function. Line number 9 and 11 records the function value and the point x reached at the end of each iteration which are later required to plot the convergence graphs.

The line 13 to 30 starts a while loop and the line 14 to 16 states the termination criteria for the while loop implemented which is till the end of reaching the nth dimension in steps of 1 which is written in line number 29.

In order to find the minimizer in the above computed direction we apply the below formulation,

$$f(x_{k+1}) = f(x_k + \delta_k * p_k)$$

Since we know that above formulation holds good. We have to choose a δ_k for which the value of $f(x_{k+1})$ is minimum. In order to find that we apply the golden section search algorithm which is a line search algorithm. It searches for the minimizer in the direction computed earlier. Line number 17 and 18 implements the above. Since I am not sure about the closing interval for the golden section search I try to initialize it as 1. And then the below piece of code tries to determine the closing interval till the base conditions are satisfied.

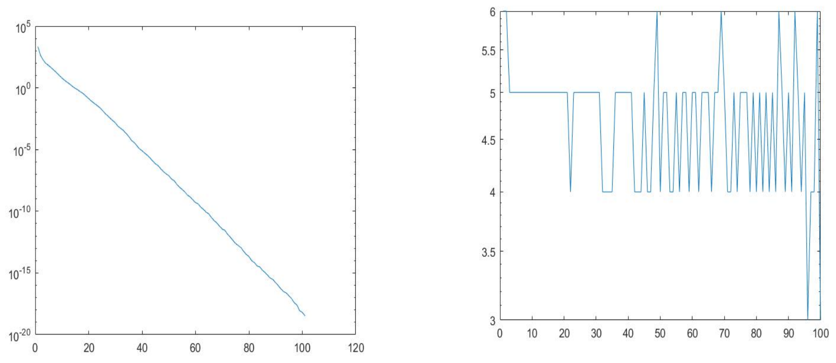
```
1   findClosingInterval(f, a, c, epsilon)
2           while(min(fb, fx) >= min(f(a), f(c)))
3               if(f(c) > f(a))
4                   c = c/2;
5                   fb = f(a+(c-a)*w);
6                   fx = f(a+(c-a)*(1-w));
7               end
8               if(f(c) <= f(a))
9                   c = c*2;
10                  fb = f(a+(c-a)*w);
11                  fx = f(a+(c-a)*(1-w));
12              end
13              n = n + 1;
14          end
15  end
```

The line 2 in the above modification specifies the condition which is necessary to find the minimum within the interval a and c for the golden section search. So till the condition is failed we are trying to shrink or grow the interval based on values obtained after applying the function to point c and a. The total number of computations required to obtain the closing interval is being recorded by n in line number 13. So if the c is going behind a then we have to increase the c. This is done in line number 8 to 11. While, if the function value at c is higher than a then we reduce the c by 2. This implementation is done between lines 3 and 6.

So going back to the implementation of conjugate gradient method, once the minimizer in that direction is found, we are finding the next direction in which the minimizer is found with the help of the Fletcher-reeves formulation earlier described in the background section. This implementation happens between line number 20 to 25. This completes the explanation of the implementation adapted for the conjugate gradient method in matlab. As mentioned earlier the two test functions as discussed earlier are now passed to the code and the results are interpreted briefly in the observation.

4 Observation

First, lets observe how the implementation worked on Ostermeier ellipsoid. The figure is plotted based on function values over the number of iterations which in our case is the dimension.



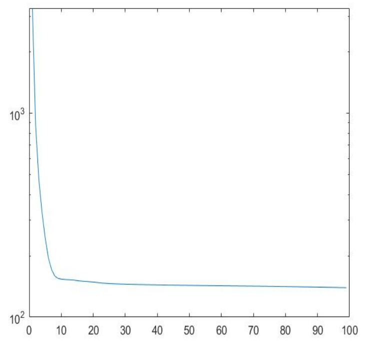
(a) Convergence graph

(b) Iterations to find the 'c' in golden section search

Figure 1: Observation of running Fletcher-Reeves algorithm on Ostermeier Ellipsoid

As we can see from the Figure 1(a) that the linear convergence is being observed on the quadratic function when the algorithm is ran. The value of c in Figure 1(b) seems to pique at certain iterations and those piques are closely related to steep descendants in some part of the convergence graph shown in Figure 1(a). So finding c includes more computation cost than other approaches to find the minimizer of Ostermeier ellipsoid. We are saving space but increasing the computational complexity here. Similarly, the algorithm is ran over the Rosenbrock test function which is non-quadratic.

I can see that after iteration 20 the rate of convergence starts to slow down. There is still convergence but it is slow. While the iterations to determine the closing interval 'c' is also taken into account. Unlike its piques shown for a


(a) Convergence graph

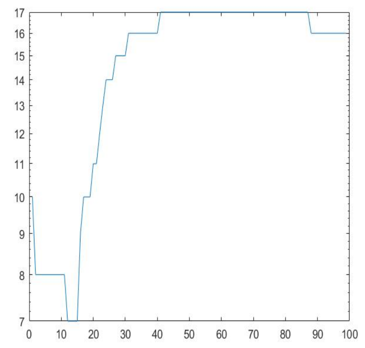
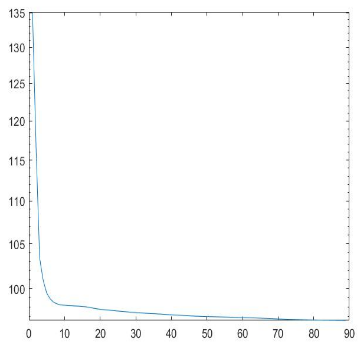

(b) Iterations to find the 'c' in golden section search

Figure 2: Observation of running Fletcher-Reeves algorithm on Rosenbrock function

quadratic function in figure 1(b), its pretty much constant in figure 2(b).

Finally, I tried to restart the implementation of the algorithm for the Rosenbrock function at various numbers. Below figures illustrates that.


(a) Till 10 iterations

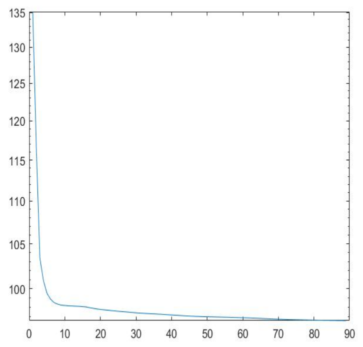
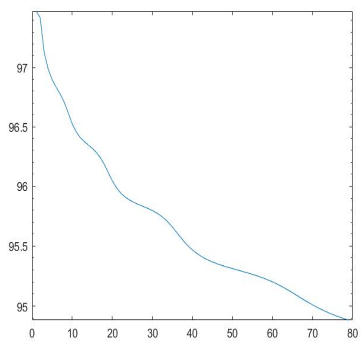

(b) After 10 iterations

Figure 3: Stopping and restarting the algorithm after 10 iterations

There is still not much of a change as I can see that the convergence slows down. There is still a stagnation. Hence I thought of restarting at 20 as we can see in the figure 2(a) that the rate drops at iteration after 20. So, we will try to stop it at 20 and then restart the algorithm after that for the Rosenbrock function. Below figure shows the findings.


(a) Running it till 20

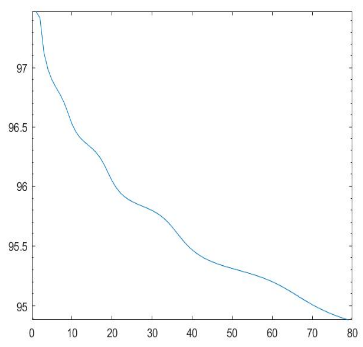
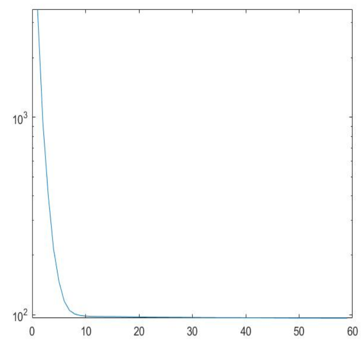

(b) After 20 iteration

Figure 4: Stopping and restarting the algorithm after 20 iterations

As there is a significant rate of increase in convergence is there. This supports my decision to restart the algorithm at the value 20. Now to make sure it works well at that point, I will try to do a restart after the 60 iteration. The result of it is as shown below. Though in this case from figure 5(b) we could see an


(a) Running till 60

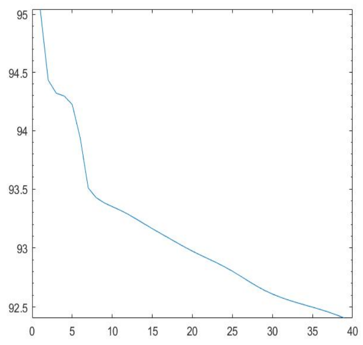

(b) Running after 60

Figure 5: Stopping and restarting the algorithm after 60 iterations

increased rate of convergence the first part figure 5(a) stagnates after 20. This further proves restarting at the iteration 20 is ideal thing to increase the rate of convergence for Rosenbrock function.

5 Conclusion

By doing this I have clearly distinguished the behavior of the Fletcher-Reeves conjugate gradient method on two Multi-dimensional test functions. Further the Polak-Ribiere can be implemented in the Matlab and applied on the same two functions. The results can be analyzed to see how the performance based on the restart metrics varies for the non-quadratic test function and how many iterations it takes to compute the 'c' for the quadratic test functions.