Math 560: Nonlinear Optimization
FINAL PROJECT REPORT

VRAJ PATEL
(261022581)

Department of Mathematics and Statistics
McGill University
Winter 2022

Contents

List of Tables

Chapter 1

Introduction

This project consist of two types of algorithms first one the line search which are **Steepest Descent**, **Newton**, **BFGS(Quasi Newton)**, second type is **Trust Region**.
The goal is to find the optimum of the given functions which are,

1. rosenbrock
2. quadratic
3. genhumps
4. leassquares

The goal to either reach at the value of the gradient at the given i.opttol or to run the algoritms to given i.maxiter whichever terminates first.

I've simplified my iterate updates into simple form.

$$x_{k+1} = x + \alpha d \tag{1.1}$$

I have divided my optsolver into two parts,

1. Step size acceptance
2. Search direction

1. Step size acceptance
    - Wolfe line search
    - Backtracking
    - Trust Region
    - SR1 trust Region
2. Search Direction
    - Steepest Descent
    - Newton
    - CG
    - BFGS(Quasi Newton)

Chapter 2

Algorithm Descriptions

2.1 Line Search Methods

(a) Steepest Descent
Steepest Descent algorithm is one of the most simple algorithm among all other algorithm. In the steepest descent we take descent direction of the function by going in the negative of the gradient of that function. The biggest drawback of the steepest descent is that it is very slow algorithm and takes too long to reach at the optimum.
In the steepest descent the direction is given by the below, which is the negative of the gradient.

$$d = -g$$

After finding the direction we update our iterate by,

$$x_{k+1} = x_k + \alpha d$$

Here, α is the stepsize which either calculated by backtracking or wolfe algorithm.
The backtracking algorithm is to find the optimum step size which is given by the Algorithm 3.1 in the book. Steepestbacktracking is the simplest algorithm compared to other algorithms that we'll dicuss.

(b) Newton
Newton's method is the approximation of the given function by creating an affine model for the function and solve till the gradient of that function is approximately equal to zero.
Affine model is given by,

$$m_k(x) = F(x_k) + F'(x_k)(x - x_k)$$

We call at x_{k+1} the solution to $m_k(x) = 0$ and find out the next iterate,

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)}$$

Here,
$F(x_k) = \nabla f(x_k)$ and $F'(x_k) = \nabla^2 f(x_k)$

$$\nabla F(x_k)d = -F(x_k)$$
$$H(x_k)d = -\nabla f(x_k)$$
$$x_{k+1} = x_k + \alpha d$$

We calculate the value of stepsize either by backtracking or by wolfe algorithm.
Newton's algorithm is very fast compared to other algorithms but there are some drawbacks too, the first draw back is that we've to start at a neighbourhood of the optimum otherwise our algorithm wouldn't converge to the optimum point. The second draw back is that there is one singularity which is, our hessian of the function has to be positive definite otherwise it won't converge.

(c) BFGS
The BFGS is also called Quasi-Newton method. BFGS overcomes the drawbacks of the Newton algorithm by making sure that each and every iteration our hessian remains positive definite. In the Newton algorithm our hessian is calculated by simple hessian calculation but in the BFGS we use iterative method which uses values of iterates and gradients.
Hessians are calculated by,

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

where,
$s_k = x_{k+1} + x_k = \alpha_k d_k$
$y_k = \nabla f(x_{x+1}) - \nabla f(x_k)$
The only difference between Newton and Quasi-Newton is the Hessian update other than that all the things remains the same.
Direction is given by,

$$H(x_k)d = -\nabla f(x_k)$$

Iterate update is done by below equation,

$$x_{k+1} = x_k + \alpha d$$

Where value of stepsize is calculated either by wolfe or backtracking.

(d) Backtracking
Backtracking approach ensures either that the selected step length α is some fixed value or else that it is short enough to satisfy the sufficient decrease condition but not too short. The letter claims holds because the accepted value α is within a factor ρ of the previous trial value, α/ρ, which was rejected for violating the sufficient decrease, that is for being too long.
For backtracking line search I followed the algorithm 3.1 from the textbook.

(e) Wolfe
Wolfe algorithm is used to calculate the optimum stepsize for the given function. When our current point is very far from the optimum point, wolfe algorithm says that take bigger steps to reach optimum faster, and as we go nearer to optimum we may want to take smaller steps. So, we introduce two conditions in wolfe, first one is sufficient decrease which takes the smaller steps so that we can reach to optimum, and the second one is called curvature condition to overcome the effect of the short steps which are unacceptable.
Wolfe conditions needs to satisfy the below inequalities,

$$f(x_k + \alpha_k d) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T d$$
$$\nabla f(x_k + \alpha_k d)^T d \geq c_2 \nabla f_k^T d$$
$$0 < c_1 < c_2 < 1$$

2.2 Trust Region Methods

In the trust region method we approximate our objective function with some quadratic model and try to find direction by solving for that quadratic with some constraint on it and then find a radius which can perfectly fit our objective function which is quadratic in our case.
In here we assume below quadratic model,

$$m_k(d) = f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T H_K d$$
$$\text{s.t.} \parallel d \parallel \leq \Delta_k$$

Where, Δ_k is the radius of the trust region.
Here, the above problem is called the sub problem of the trust region which is calculated by the CG algorithm.

(f) Conjugate Gradient (CG)
The main objective of the CG algorithm is to solve the quadratic model and find the direction of the quadratic model with some constraints.
It solves the simple equation, which is,

$$Ax = b$$

Here,
$A = H(x_k)$ (Hessian of the given function)
$x = d$(descent direction) which is our objective to solve.
$b = -\nabla f(x_k)$ (negative of the gradient)
We pass this direction d value to the trust region algorithm (which I followed from the lecture slides) and will decide that the given radius of the trust region is able to approximate our objective function or not, if the radius is approximating the objective function properly then we take stepsize $\alpha = 1$ and update our iterate as,

$$x_{k+1} = x_k + d$$

If the given trust region is not approximating the objective function properly then we reduce the trust region size and do not update our iterate. Here in this we change our trust region size and once again solve for the direction and see that it is approximating the function or not. We'll keep repeating this till we find the proper trust region radius size then only we update our iterate.

(g) SR1 Trust Region
SR1 Trust Region is same as the trust region algorithm in which we approximate our objective function in some radius and try to solve for the direction in that region by CG and update the iterate.
The difference here is the update of Hessian in the SR1 algorithm, in the SR1 we update our hessian by SR1 iterate equation which is given by below equation,

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

where,
$s_k = x_{k+1} + x_k = \alpha_k d_k$
$y_k = \nabla f(x_{x+1}) - \nabla f(x_k)$
This update of Hessian in SR1 make sure that the hessian would always be positive definite for the all the iterations.

Chapter 3

Numerical Results

Below is the table for the default input parameters which are consistent for all the problem, these parameters are best for my algorithms.

Algorithm	Parameter values
wolfe	C1 = 0.30 C2 = 0.70
Trust region	C1 = 0.10 C2 = 0.90
CG	maximum iteration = 10^3 tolerance = 10^{-6}
SR1	update tolerance = 10^{-6}
BFGS	BFGS damping tolerance = 10^{-6}

Table 3.1: Default parameter values

Algorithm	Starting point	No. of iterations	Optimum value	CPU time
Steepestbacktrack	[1.2 1.2]	1001	[1.0 1.0]	1.849
Steepestwolfe	[1.2 1.2]	1001	[1.0 1.0]	1.432
Newtonbacktrack	[1.2 1.2]	35	[1.0 1.0]	1.5035
Newtonwolfe	[1.2 1.2]	11	[1.0 1.0]	2.44
Trustregioncg	[1.2 1.2]	11	[1.0 1.0]	1.79
SR1Trustregioncg	[1.2 1.2]	34	[1.0 1.0]	1.4821
BFGSbacktrack	[1.2 1.2]	40	[1.0 1.0]	1.369
BFGSwolfe	[1.2 1.2]	15	[1.0 1.0]	1.4796

Table 3.2: Results of the Rosenbrock function

Algorithm	Starting point	No. of iterations	Optimum value	CPU time
Steepestbacktrack	[1 1 1 1 1]	105	[0.00 0.14 0.00 0.14 0.00]	1.5
Steepestwolfe	[1 1 1 1 1]	58	[0.21 0.00 0.00 0.00 0.21]	1.43
Newtonbacktrack	[1 1 1 1 1]	39	[-0.03 0.11 -0.07 0.11 -0.03]	1.45
Newtonwolfe	[1 1 1 1 1]	N/A	N/A	N/A
Trustregioncg	[1 1 1 1 1]	18	[0.05 -0.07 0.11 -0.07 0.05]	1.44
SR1Trustregioncg	[1 1 1 1 1]	52	[-0.05 0.05 0.00 0.06 -0.10]	1.35
BFGSbacktrack	[1 1 1 1 1]	88	[0.00 0.12 0.00 0.12 0.00]	1.75
BFGSwolfe	[1 1 1 1 1]	N/A	N/A	N/A

Table 3.3: Results of the Genhumps function

Algorithm	Starting point	No. of iterations	Optimum value	CPU time
Steepestbacktrack	[1 1 1 1]	1001	[2.36 0.84 1.59 0.29]	2.12
Steepestwolfe	[1 1 1 1]	741	[2.36 0.81 1.65 0.25]	1.4
Newtonbacktrack	[1 1 1 1]	42	[2.36 37.24 -0.09 0.00]	1.4
Newtonwolfe	[1 1 1 1]	N/A	N/A	N/A
Trustregioncg	[1 1 1 1]	7	[2.36 0.68 1.94 0.10]	1.54
SR1Trustregioncg	[1 1 1 1]	7	[2.36 0.09 1.08 0.05]	1.49
BFGSbacktrack	[1 1 1 1]	N/A	N/A	N/A
BFGSwolfe	[1 1 1 1]	N/A	N/A	N/A

Table 3.4: Results of the Least Squares function

Chapter 4

Conclusion

I used the same parameters for every problems. I tuned my parameters as per Rosenbrock problem and used the same parameters for all the other problem.

4.1 Rosenbrock

All the eight algorithm converged for this problem. I started with the initial point of [1.2 1.2] and all algorithms reached the correct optimum that is [1 1]. Although, steepestbacktrack and steepestwolfe did reach the maximum iterations and gave the output which as very closer to the actual optimum.
For this problem, Newtonwolfe and Trust Region CG take the least number of steps but in the newton algorithm we need to start from very near to optimum point otherwise we can diverge from the optimum. For the Trust Region with CG I used the initial trust region radius as 2.

4.2 Genhumps

This problem has multiple optimum points so in this problem our starting point matters the most. For this problem I've choose [1 1 1 1 1] as the starting point. In this problem Newton wolfe and BFGS wolfe fails to converge to optiumum given that while finding the best value for the stepsize it is not able to find the value of the stepsize and our algorithm fails to converge.
For this problem Trust Region with CG performs the best with least number of iterations and also it has CPU time of 1.44 seconds which is very low compared to the other algorithms it has a optimum value of [0.05 -0.07 0.11 -0.07 0.05].

4.3 Least Squares

For this problem Newton wolfe, BFGS backtrack and BFGS wolfe fails to converge, the main reason was that hessian was not positive definite and our problem would become singular. Other algorithm converged but steepest descent reached the maximum iterations and also it took the highest CPU time which was 2.12 seconds. For this problem also Trust Region algorithms performs really fast compared to other line search algorithms and converge faster.

4.4 Quadraric

None of the algorithms could find the solution for the quadratic problem.

After analyzing the results for different algorithms I concluded that Trust Region with CG as the sub problem solver comes out to be winner. The main reason is that it converge so faster compared to other algorithms and it converged for three problems. There is a reason why this algorithm works better compared to other algorithms and converged faster. The reason is that we approximate our objective function with some quadratic model and try to minimize that quadratic model since, quadratic models are easy to minimize because they are convex in nature.
As I share my experience with coding the algorithms the hardest algorithm to code was wolfe conditions. There were lots of parameters to consider while code this algorithm and also there was zoom function to code. Other that this algorithm, Trust Region with CG was hard to understand conceptually and also in coding.

Bibliography