Numerical Optimization
Week 5 Assignment

CHM564

1 Introduction

In this report, we will take the trust-region method from last week and turn it into a quasi-newton method. We will discrible the performance of algorithm and disscuss the parameters and initial hessian-approximation B_0. Also plots will be showed to certify the programs can work on benchmark functions.

2 Theory analysis

2.1 Theoretical assignment

We want to show when f is strongly convex, $s_k^T y_k > 0$ holds for any vectors x_k and x_{k+1}. Firstly, f is strongly convex, so all eigenvalues of $\bigtriangledown^2 f(x)$ are positive and bounded away from zero. It means there exists $\sigma > 0$ such that

$$p^T \bigtriangledown f(x) p \geq \sigma \|p\|^2 \tag{1}$$

Then Taylor's theorem, if $x_{k+1} = x_k + \alpha_k p_k$,

$$\bigtriangledown f(x_{k+1}) = \bigtriangledown f(x_k) + \int_0^1 [\bigtriangledown^2 f(x_k + z\alpha_k p_k)\alpha_k p_k] \, dz$$

By (1) we have

$$\begin{aligned} \alpha_k p_k^T y_k &= \alpha_k p_k^T [\bigtriangledown f(x_{k+1}) - \bigtriangledown f(x_k)] \\ &= \alpha_k^2 \int_0^1 [p_k^T \bigtriangledown^2 f(x_k + z\alpha_k p_k) p_k] \, dz \\ &\geq \sigma \|p_k\|^2 \alpha_k^2 > 0 \end{aligned}$$

By notes , we can get $s_k = \alpha_k p_k$, so it is proved.

2.2 Description of algorithm

Quasi-Newton method does not involve Hessian matrix, so it is superlinear convergence. Now, we store a hessian-approximation B_k which will replace Hessian on last week's algorithms.
The update formula for B_k :

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

From the formula, we find:
If $y_k \neq B_k s_k$ and $(y_k - B_k s_k)^T s_k = 0$: the denominator in the formula is vanish.
It is obvious that the denominator in the formula may be small, resulting in numerical instability, even when the objective function is a convex quadratic. It means SR1 can't satisfy all the conditions.

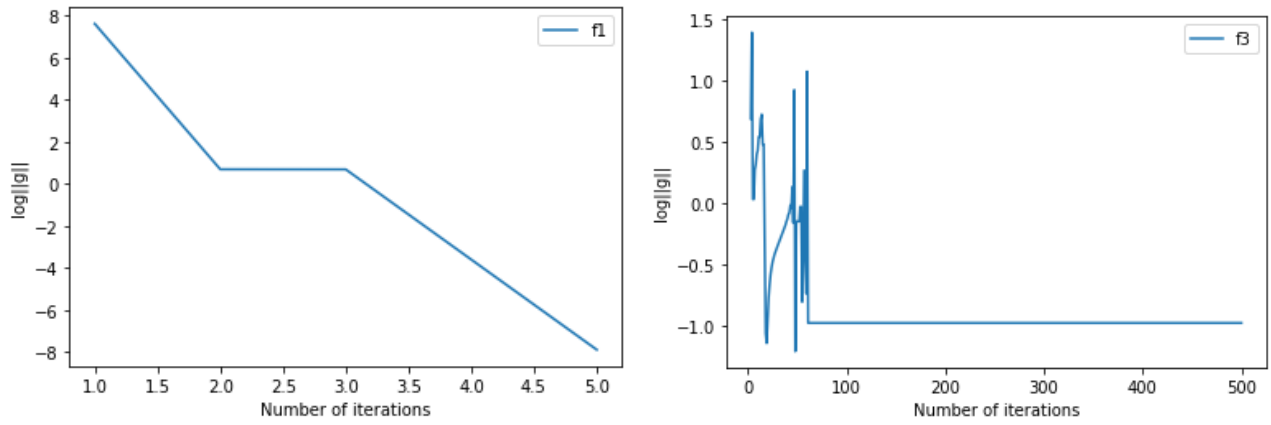
To avoid this case, we use a safeguard to adequately prevent the break down of the method and the occurrence of numerical instabilities.
B_k's update is applied only if

$$|s_k^T (y_k - B_k s_k)| \geq r \|s_k\| \|y_k - B_k s_k\|$$

where $r \in (0, 1)$ is a small number. And if this equation does not hold, we set $B_{k+1} = B_k$.

3 Practical part

3.1 Implement the function



(a) Test on convex functuon

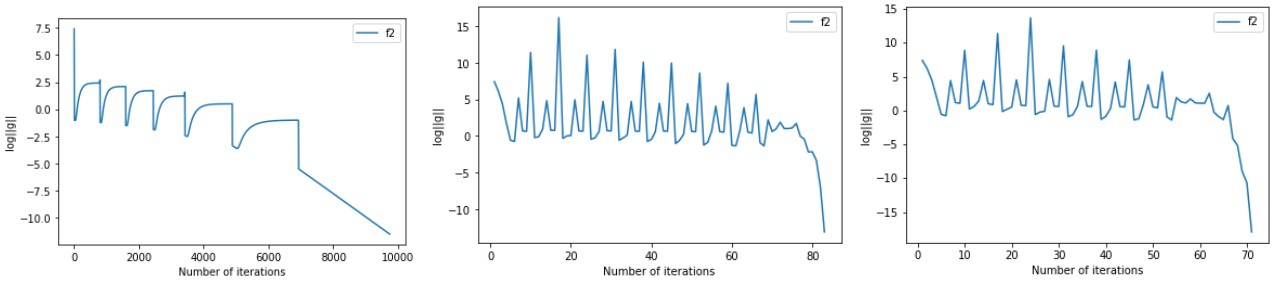
(b) Test on non-convex functuon

Figure 1: Results of different types

We can see that the algorithm can work on non-convex functions. But algorithm can't work on some B_0. The reason is that if B_k become small, then p_k will be small. This will cause ρ can't be big enough to update x_k. The specific analysis of B will be show on next subsection. And we all know r should be small, so I choose $r = 10^{-8}$.

3.2 Different initial B

We test different initial B_0 on function2. Function2 is sensitive to B_0. It easily to show $infty$ or nan on the result. There we choose three different B_0.



(a) (6000,200; 200,1000) (b) (6000,0; 0,1000) (c) (6000,-200; -200,1000)

Figure 2: Results of different initial B

plot	x_k	number of iterations
(a)	(1.00001117,1.00002239)	9741
(b)	(1.00000077,1.00000134)	83
(c)	(1,1))	71

Table 1: nformation about Figure 2

It is easily to see that fig.(c) is the best.

We assum $B = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. From note, we know that

$$\begin{aligned} p(\lambda) &= -(B + I\lambda)^{-1} g \\ &= -\begin{bmatrix} a+\lambda & b \\ c & d+\lambda \end{bmatrix}^{-1} g \\ &= \begin{bmatrix} \frac{-d}{(a+\lambda)(d+\lambda)-bc} & \frac{b}{(a+\lambda)(d+\lambda)-bc} \\ \frac{c}{(a+\lambda)(d+\lambda)-bc} & \frac{-a}{(a+\lambda)(d+\lambda)-bc} \end{bmatrix} \begin{bmatrix} -2(1-x_1) - 400x_1(x_2 - x_1^2) \\ 200(x_2 - x_1^2) \end{bmatrix} \end{aligned}$$

By (2,2) as initial guess. We can get

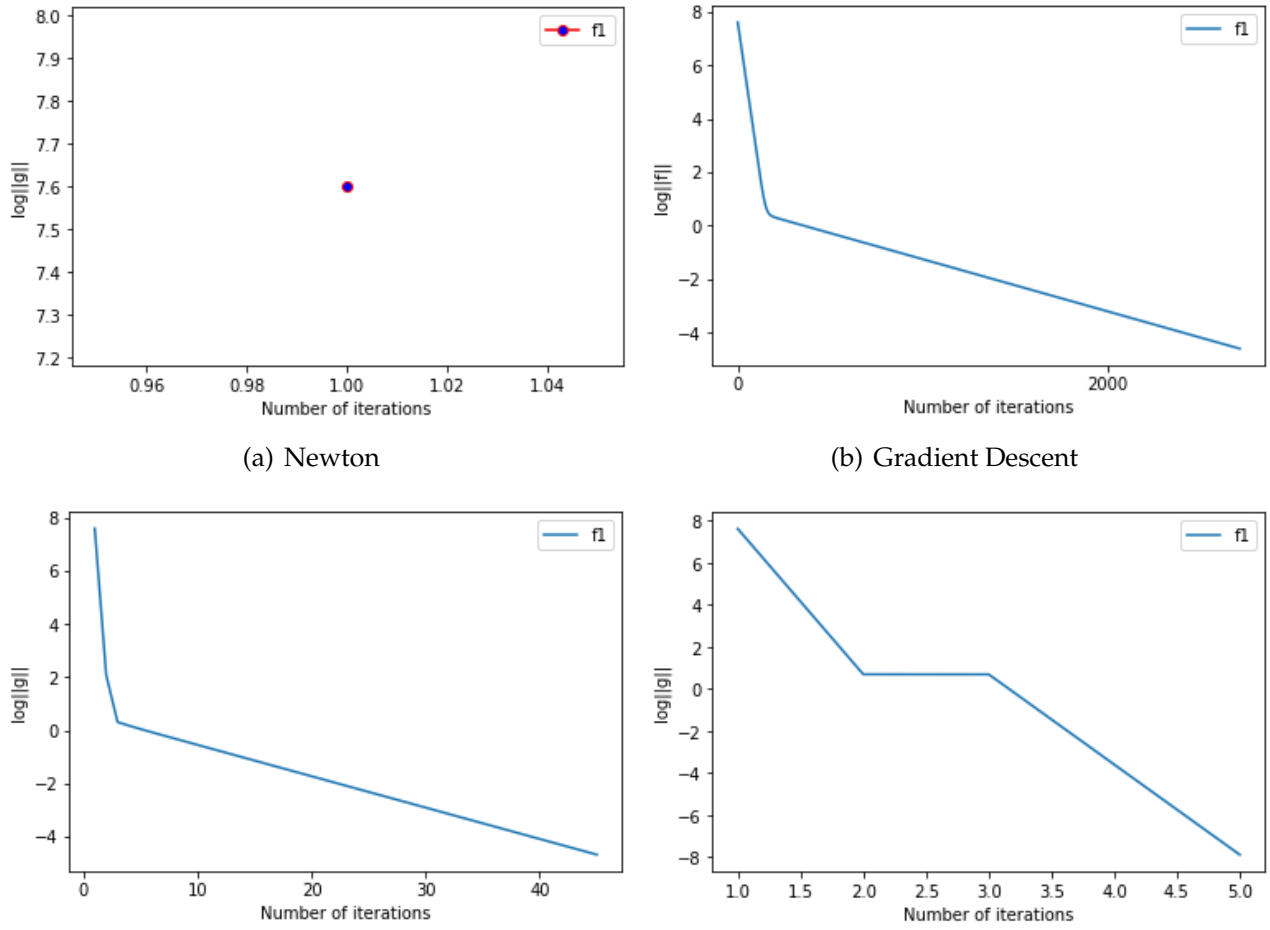
$$p(\lambda) = \begin{bmatrix} \frac{-d}{(a+\lambda)(d+\lambda)-bc} & \frac{b}{(a+\lambda)(d+\lambda)-bc} \\ \frac{c}{(a+\lambda)(d+\lambda)-bc} & \frac{-a}{(a+\lambda)(d+\lambda)-bc} \end{bmatrix} \begin{bmatrix} 1602 \\ -400 \end{bmatrix}$$

It is easy to get that, the case(c) will make p bigger. And the iterations will become less.

4 Comparison of different methods

In the last 4 weeks, we implemented Line-Search method:Newton, Gradient Descent; Trust-Region method. Now we want to compare these four methods.

As I said on last week assignment, line-search may be easier and simple. However, trust-region is much more accuary than line-search. For line-search, if one step gets mistake, it will be difficult to correct. But turst-region can modify in its region , even stop updating the points until the next suitable step.



(a) Newton

(b) Gradient Descent

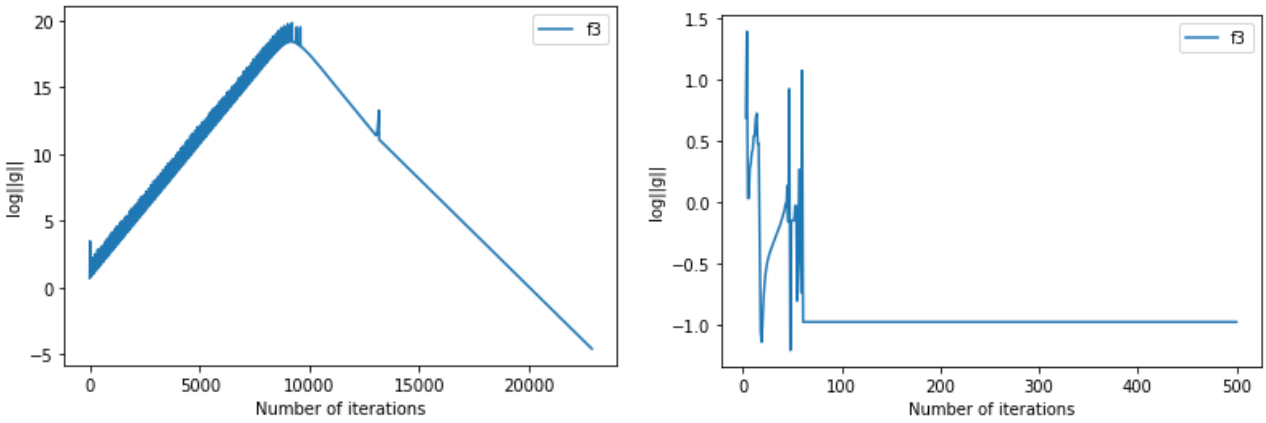
(c) Trust Region

(d) Trust Region SR1

Figure 3: Different methods test on convex function

Now, we use figure to compare these methods. We choose $(1,1)$ as initial guess and $10^(-4)$ as stopping critiria.
For a convex function, like function1: Newton[1] always shows best. But gradient descent took much more time whatever the function is. The trust region shows better than GD, but not as good as Newton. SR1 is better than TR, because we use hessian-approximation B, so it will faster than Newton trust region.

[1]Newton iterate 2 times and the second time $\log \| \bigtriangledown (f) \| \to -\infty$ in the Figure (a).



(a) Test on convex functuon

(b) Test on non-convex functuon

Figure 4: Results of different types

For non-convex function: Because of non-positive hessian definite, Newton can't always worked. So, we compare GD, SR1. Obviously, SR1 is better.

We can conclude that for simple convex function, Newton will be fast and accuary. But for non-convex function, trust region method, like SR1, will be more suitablbe.

5 Conclusion

In this report, we described the SR1 trust region, and implement on different functions, include non-convex function. Then we analyzed the initial B_0. Finally we discussed the comparison of 4 methods, and we get the Newton could be used on convex function and for non-convex, trust-region SR1 is better.