

ISE 417: Nonlinear Optimization
FINAL PROJECT REPORT

Xi He

A survey on some nonlinear programming algorithms

Department of Industrial and Systems Engineering
Lehigh University
Spring 2015

Contents

Table of Contents	ii
List of Tables	iii
List of Figures	iv
1 Introduction	1
2 Fundamentals of Unconstrained Optimization	3
3 Algorithm Descriptions I: Flexible Step Method	6
3.1 Steepest Descent and Newton method	6
3.2 Line Search Methods	7
3.2.1 Wolfe Conditions	7
3.2.2 Backtracking Line Search	9
3.3 Quasi-newton method	10
4 Algorithm Descriptions II: Restricted Step Methods.	12
4.1 Trust Region Method	12
4.2 Conjugate Gradient Method	13
4.3 Trust Region Subproblem with CG Method	14
5 Numerical Results	16
6 Conclusion and Experience	20
Bibliography	22

List of Tables

5.1	Default Parameter Set	16
5.2	Rosenbrock, with initial $0\mathbf{e}$	17
5.3	Genhumps, with initial \mathbf{e}	17
5.4	Quadratic, with initial $0\mathbf{e}$	18
5.5	Leastsquares, with initial $(0, 0, 0, 1)^T$	19

List of Figures

2.1	Global minimum and Local minimum	4
3.1	Sufficient decrease condition	8
3.2	Wolfe curvature condition	8

Chapter 1

Introduction

This project concerns the practical implementation of several classical optimization algorithms in the context of unconstrained nonlinear problems. By choosing proper parameter set and delicate implementation, we intend to achieve economy of computation or the solution of such problems. With respect to several test problems, we compare the various performance of each algorithms with different parameters setup. It also might be a reasonable instruction on deciding which algorithm should be applied when faced a new problem.

At the first part of this project, we study several algorithms on solving unconstrained nonlinear optimization problems, such as steepest descent method, newton method and BFGS method with two kinds of linear search strategy: backtrack line search method and wolfe line search method. Trust region method is also argued with two similar approaches to solve trust region subproblem: conjugate gradient method and conjugate gradient method with SR1 Hessian matrix update method¹. Throughout the introduction of each algorithm, we also list convergence and complexity issues, however, one should note that we state some correct conclusions without rigorous proof.

In the latter part of this project, some detailed implementation concerns are discussed with respect to different problems and algorithms, including practical parameter set choosing, tricks on attaining economy computation and also some difficulties with respect to implementation. Results on numerical experiment are shown to clearly compare the performance of each algorithm with different parameters on each problem. Also, some analysis of the numerical result are presented.

The structure of this project report is as follows. Chapter 2 includes relevant background on unconstrained optimization problems which forms the basis of the discussions in later chapters. In Chapter 3, after showing the very basic two descent directions we used in implementation, we discuss two kinds of linear search strategy. And then we present two kinds of quasi-newton method and show the global convergence results when applying the two line search method mentioned before. We introduce in Chapter 4 about trust region algorithms and conjugate gradient method with is proposed by solving the trust region subproblems. SR1 update which is stated in Chapter 3 will be reconsidered as a approach to solving trust region subproblem when combining with conjugate gradient method. Finally, in Chapter 5, we present, analyze and provide numerical results with respect to the algorithms we discuss in above chapters on some test problems. Brief summary of this project and some general conclusion will be stated in the Chapter 6.

In this report, we use the following notation. Let $f(x)$ be original nonlinear, smooth and differentiable objective function we want to minimize and x be decision variables. We also denote gradient function of $f(x)$ by $g(x) = \nabla f(x)$ and denote $H(x) = \nabla^2 f(x)$ and $B(x)$ by the Hessian

¹Most of figures and algorithms pseudocode come from [1].

and approximation Hessian matrix of $f(x)$ at point x . With respect to algorithms, we use $d(x)$ to be an acceptable descent directions at point x and $\alpha(x)$ to be an acceptable step-size along $d(x)$. At the last, $H \succ (\succeq) 0$ represents that the matrix is (semi) positive definite.

Chapter 2

Fundamentals of Unconstrained Optimization

We frame this report in the context of the unconstrained optimization problem

$$\min_x f(x), \tag{2.1}$$

where $x \in \mathbb{R}^n$ is a real vector with $n \geq 1$ components $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a smooth function.

Generally, we intend to find out a global minimizer of f , a point where the function attains its least value in whole space. We state a formal definition as

Definition 2.1. *A point x^* is a global minimizer if $f(x^*) \leq f(x)$ for all x .*

Note that it can be very difficult to find the global minimizer, since our knowledge of f is usually only local. Actually, most algorithms are able to find only a local minimizer, a point that achieves the smallest value of f in its neighborhood.

Definition 2.2. *A point x^* is a local minimizer if there is a neighborhood \mathcal{N} of x^* such that $f(x^*) \leq f(x)$ for $x \in \mathcal{N}$.*

As it is shown in Figure 2.1, in general case, a function f may have a lot of local minimizers but just one global minimizer. Most algorithms is sensitive on the initial point we chose, which means, by choosing different initial point to run those algorithms, we may get different local minimizer.

Now, we state sufficient conditions, which are conditions on the derivatives of f at the point x^* that guarantee that x^* is a local minimizer.

Theorem 2.7. (*Second-Order Sufficient Conditions*) Suppose that H is continuous in an open neighborhood of x^* and that $g(x^*) = 0$ and $H(x^*)$ is positive definite. Then x^* is a strict local minimizer of f .

Chapter 3

Algorithm Descriptions I: Flexible Step Method

In flexible step method, general framework of algorithms is as following

Algorithm 1 General Algorithm of Flexible Step Method

Require: Initial point x_0 and $k := 0$.

- 1: **repeat**
 - 2: Derive descent direction d_k satisfying $g(x_k)^T d_k < 0$
 - 3: Compute step-size α_k along descent direction d_k by line search method
 - 4: Update $x_{k+1} = x_k + \alpha_k d_k$, set $k = k + 1$
 - 5: **until** Termination condition satisfied.
-

3.1 Steepest Descent and Newton method

Steepest descent direction and newton descent direction are two natural ways to get descent direction. In terms of steepest descent direction, due to $\|g(x)\| > 0$ when x is not a stationary point, we can simply choose

$$d = -g(x), \quad (3.1)$$

where we have $g(x)^T d = -\|g(x)\|^2 < 0$.

For newton descent method, first we note that if $H(x)$ is a positive definite matrix, we have $H(x)^{-1}$ is positive definite and furthermore $d^T H(x)^{-1} d > 0$, whenever $d \neq 0$. By considering that since $g(x) \neq 0$ when x is not a stationary point, we have

$$g(x)^T H(x)^{-1} g(x) > 0, \quad (3.2)$$

therefore, we can select

$$d = -H(x)^{-1} g(x) \quad (3.3)$$

to be one possible descent direction.

Besides, when $H(x)$ is not positive definite, we can perturb it to be positive definite by add positive number to its diagonal. Actually, we use the following procedure to implement this perturbation.

Algorithm 2 Perturbation for non-positive definite matrix

Require: A non-positive definite matrix P and a same-size identity matrix I

```

1: if  $\min \text{eig}(P) < 0.1$  then
2:    $P = P + (1 - \min \text{eig}(P))I$ 
3: end if
4: Output a positive definite matrix  $P$ 

```

Note that by using this approach, we guarantee the perturbed matrix P to be positive definite and its minimal eigenvalue is at least greater than 0.9, which suppose to be well-defined matrix when solving the linear system (3.3).

After we pick up a descent direction, we need find a proper step-size, which is what we called line search method in the next section.

3.2 Line Search Methods

Always, step-size are chosen to give a sufficient reduction of f , but at the same time, we prefer to spend less time to choose a proper step-size. At current point x_k with a descent direction d_k , we let

$$\phi(\alpha) = f(x_k + \alpha d_k), \alpha > 0, \quad (3.4)$$

we may say the ideal choice is to find a global minimizer α^* of the one-dimensional function $\phi(\alpha)$. However, it is too costly to find the global minimizer. Therefore, we apply some more practical strategies that calculating step in an inexact approach. In this project, we discuss two kinds of line search method as following.

3.2.1 Wolfe Conditions

It contain two parts when find a proper step-size by wolfe conditions. Firstly, we should choose a step-size that provide sufficient reduction. We use the following condition to guarantee it

$$f(x_k + \alpha d_k) \leq f(x_k) + c_1 \alpha \nabla f_k^T d_k, \quad (3.5)$$

which $c_1 \in (0, 1)$ is a parameter. By noting that $\nabla f_k^T d_k < 0$ since d_k is a descent direction, we say that (3.5) is sufficient decrease condition. We show geometry explanation in Figure 3.1.

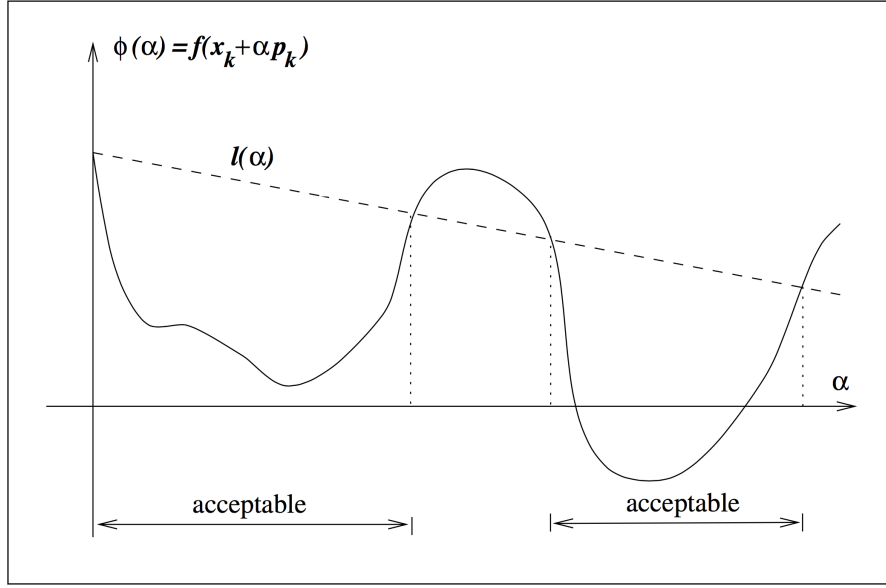


Figure 3.1: Sufficient decrease condition

Except this condition, we make other condition called curvature condition to get rid of unacceptably short steps

$$\nabla f(x_k + \alpha_k p_k)^T \geq c_2 \nabla f_k^T p_k, \quad (3.6)$$

where $c_2 \in (c_1, 1)$ is another parameter. Geometry illustration is shown in Figure 3.2.

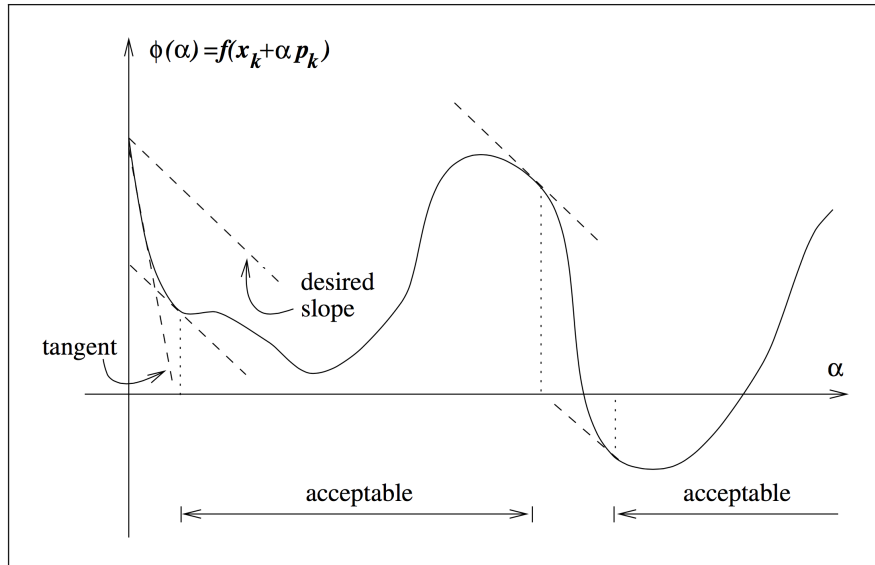


Figure 3.2: Wolfe curvature condition

Practically, we implement wolfe condition as following

Algorithm 3 Wolfe Conditions

Require: Initial step-size $\alpha_0 > 0$, $\alpha_1 > 0$ and α_{\max} . Set $i := 1$.

```

1:  $\alpha = \bar{\alpha}$ 
2: repeat
3:   if  $\phi(\alpha_i) > \phi(0) + c_1\alpha_i\phi'(0)$  or  $[\phi(\alpha_i) \geq \phi(\alpha_{i-1})$  and  $i > 1]$  then
4:      $\alpha_* \leftarrow \mathbf{zoom}(\alpha_{i-1}, \alpha_i)$  and stop;
5:   end if
6:   if  $|\phi'(\alpha_i)| \leq -c_2\phi'(0)$  then
7:     set  $\alpha_* \leftarrow \alpha_i$  and stop;
8:   end if
9:   if  $\phi'(\alpha_i) \geq 0$  then
10:    set  $\alpha_* \leftarrow \mathbf{zoom}(\alpha_i, \alpha_{i-1})$  and stop;
11:   end if
12:   Choose  $\alpha_{i+1} \in (\alpha_i, \alpha_{\max})$ 
13:    $i \leftarrow i + 1$ 
14: until
15: Terminate with  $\alpha_k = \alpha$ 

```

Here zoom function is proposed to find a proper step-size in an interval.

Algorithm 4 Wolfe Conditions (zoom function)

Require: Initial step-size $\alpha_{low} > 0$ and $\alpha_{high} > 0$

```

1: repeat
2:   Choose  $\alpha_j$  between  $\alpha_{low}$  and  $\alpha_{high}$ .
3:   if  $\phi(\alpha_j) > \phi(0) + c_1\alpha_j\phi'(0)$  or  $\phi(\alpha_j) \geq \phi(\alpha_{low})$  then
4:      $\alpha_{high} \leftarrow \alpha_j$ 
5:   else
6:     if  $|\phi'(\alpha_j)| \leq -c_2\phi'(0)$  then
7:       set  $\alpha_* \leftarrow \alpha_j$  and stop
8:     end if
9:     if  $\phi'(\alpha_j)(\alpha_{high} - \alpha_{low}) \geq 0$  then
10:      set  $\alpha_{high} \leftarrow \alpha_{low}$ 
11:    end if
12:     $\alpha_{low} \leftarrow \alpha_j$ 
13:   end if
14: until

```

3.2.2 Backtracking Line Search

Another popular approach is to drop curvature condition (3.6) by appropriately choosing candidate step lengths, which we called backtracking strategy.

Algorithm 5 Backtracking Line Search**Require:** Initial step-size $\bar{\alpha} > 0$ and shrinking parameter $\rho \in (0, 1)$

-
- 1: $\alpha = \bar{\alpha}$
 - 2: **repeat**
 - 3: $\alpha = \rho\alpha$
 - 4: **until** $f(x_k + \alpha p_k) \leq f(x_k) + c\alpha \nabla f_k^T p_k$
 - 5: Terminate with $\alpha_k = \alpha$
-

Global convergence result for line search methods with respect to steepest descent method is proved under some mild assumption.

3.3 Quasi-newton method

Another approach to derive descent direction is to use Quasi-newton method, which only depends on only first derivatives. Actually, what we do in Quasi-newton method is to approximate Hessian matrix by first derivatives instead of computing accurate Hessian matrix directly, which is sometimes computational expensive.

In this project, we use BFGS method to update Hessian matrix with flexible step, which is one type of Quasi-newton method.

Let B_k be BFGS approximate Hessian matrix at point current point x_k . And then we can derive descent direction from

$$B_k d_k = -\nabla f(x_k). \quad (3.7)$$

Furthermore, we have $x_{k+1} = x_k + a_k d_k$, where a_k is a proper step-size derived from line search method in section 3.2. By defining

$$s_k = x_{k+1} - x_k \text{ and } y_k = \nabla f_{k+1} - \nabla f_k, \quad (3.8)$$

we have

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}, \quad (3.9)$$

where $-\frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$ is a rank two matrix which only depends on first derivatives. Notice that to derive descent direction by (3.7), we need guarantee B_k to be positive definite, luckily, as long as $B_k \succ 0$ and the curvature condition

$$s_k^T y_k > 0 \quad (3.10)$$

is satisfied, then we have B_{k+1} is positive definite.

Practically, we use damped BFGS updating to overcome some shortage of BFGS updating, such as to avoid $y_k^T s_k \approx 0$ or poorly conditioned Hessian approximation. We state damped BFGS updating as following

Algorithm 6 Damped BFGS updating

Require: H_k , $s_k = x_{k+1} - x_k$ and $r_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ at current point x_k . A parameter θ .

1: Compute

$$\theta_k = \begin{cases} 1, & \text{if } r_k^T s_k \geq \theta s_k^T H_k s_k \\ \frac{(1-\theta)s_k^T H_k s_k}{s_k^T H_k s_k - r_k^T s_k}, & \text{otherwise} \end{cases} \quad (3.11)$$

2: Set $y_k = \theta_k r_k + (1 - \theta_k) H_k s_k$

By this procedure, we guarantee that

$$y_k^T s_k \geq \theta s_k^T H_k s_k > 0, \quad (3.12)$$

which leads to a well-defined BFGS updating.

It is proved that BFGS method with Wolfe line search is global convergent and shares a superlinear convergence rate.

Chapter 4

Algorithm Descriptions II: Restricted Step Methods.

In restricted step method, general framework of algorithms is as following

Algorithm 7 General Algorithm of Restricted Step Method

Require: Initial point x_0 and $k := 0$.

- 1: **repeat**
 - 2: Generate a proper subproblem $G(d)$ at x_k
 - 3: Solve subproblem to derive descent direction d_k
 - 4: Update $x_{k+1} = x_k + d_k$, set $k = k + 1$
 - 5: **until** Termination condition satisfied.
-

4.1 Trust Region Method

Instead of calculate step size at each iteration, trust region method is one other approach to solve nonlinear optimization problems. In a sense, trust region is a restricted step method as the direction and step size are uniquely decided at each iteration. The main idea of trust region method is to generate a subproblem on a neighborhood of current point x_k . Normally, we set the subproblem as

$$\min_{d_k} \quad m_k(d_k) := f(x_k) + \nabla f(x_k)^T d_k + \frac{1}{2} d_k^T H_k d_k \quad (4.1)$$

$$s.t. \quad \|d_k\| \leq \Delta_k. \quad (4.2)$$

Note that $m_k(d_k)$ is a approximate function of f at point x_k and with respect to H_k , we often set $H_k \succ 0$, but not necessarily, for example, we can choose H_k to be the SR1 approximate Hessian matrix.

Therefore, based on the framework 7, we have the following algorithm

Algorithm 8 Trust Region Method

Require: Current point x_k , trust region radius Δ_k , subproblem $m_k(d_k)$ and two threshold $0 < c_1 < c_2 < 1$.

```

1: repeat
2:   Solve subproblem  $m_k(d_k)$ 
3:   Compute the ratio

$$\rho_k(d_k) := \frac{f(x_k) - f(x_k + d_k)}{m_k(0) - m_k(d_k)}. \quad (4.3)$$

4:   if  $\rho_k(d_k) \geq c_2$  then
5:     Update  $x_{k+1} = x_k + d_k$  and expand trust region  $\Delta_{k+1} > \Delta_k$ 
6:   else
7:     if  $\rho_k(d_k) \in (c_1, c_2)$  then
8:       Update  $x_{k+1} = x_k + d_k$  and keep trust region  $\Delta_{k+1} = \Delta_k$ 
9:     end if
10:  else
11:    if  $\rho_k(d_k) \leq c_1$  then
12:      Keep  $x_{k+1} = x_k$  and shrink trust region  $\Delta_{k+1} < \Delta_k$ 
13:    end if
14:  end if
15: until Meet termination condition

```

4.2 Conjugate Gradient Method

By deriving a set of n vectors which is all conjugated for each other, we build a cheaper way to compute descent direction for quadratic problem

$$\min_x \phi(x) := \frac{1}{2}x^T Ax - b^T x. \quad (4.4)$$

Actually, if we have a set of conjugated direction $\{p_0, \dots, p_{n-1}\}$, we can solve the problem in at most n steps by repeating the following procedure

- Compute the current residual

$$r_k = Ax_k - b \quad (4.5)$$

- Compute a steplength to minimize $\phi(x)$ along $x_k + \alpha p_k$

$$\alpha_k = -\frac{r_k^T p_k}{p_k^T A p_k} \quad (4.6)$$

- Update $x_{k+1} = x_k + \alpha_k p_k$

Actually, by letting the first conjugated direction $p_0 = -r_0$, we can derive others conjugated direction step by step by the following algorithm

Algorithm 9 Conjugate Direction Method**Require:** Initial point x_0 , set $r_0 = Ax_0 - b$ and $p_0 = -r_0$

```

1: repeat
2:    $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$ 
3:    $x_{k+1} = x_k + \alpha_k p_k$ 
4:    $r_{k+1} = r_k + \alpha_k A p_k$ 
5:    $\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$ 
6:    $p_{k+1} = -r_{k+1} + \beta_{k+1} p_k$ 
7: until Meet termination condition

```

4.3 Trust Region Subproblem with CG Method

Here we state a nice approach to solve the trust region subproblem (4.1).

Algorithm 10 Trust Region Subproblem with CG Method**Require:** Initial point x_0 , trust region radius Δ_k and we set $r_0 = Ax_0 - b$ and $p_0 = -r_0$

```

1: repeat
2:   if  $p_k^T A p_k < 0$  then
3:     Set  $\alpha_k$  to be a positive value such that  $\|x_k + \alpha_k p_k\|_2 = \Delta_k$ 
4:     Update  $x_{k+1} = x_k + \alpha_k p_k$  and stop
5:   else
6:      $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$ 
7:   end if
8:   if  $\|x_k + \alpha_k p_k\|_2 > \Delta_k$  then
9:     Set  $\alpha_k$  to be a positive value such that  $\|x_k + \alpha_k p_k\|_2 = \Delta_k$ 
10:    Update  $x_{k+1} = x_k + \alpha_k p_k$  and stop
11:  else
12:     $x_{k+1} = x_k + \alpha_k p_k$  and  $r_{k+1} = r_k + \alpha_k A p_k$ 
13:  end if
14:  if  $\|r_{k+1}\|_2 \approx 0$  then
15:    Update  $x_{k+1} = x_k + \alpha_k p_k$  and stop
16:  else
17:     $\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$  and  $p_{k+1} = -r_{k+1} + \beta_{k+1} p_k$ 
18:  end if
19: until Meet termination condition

```

Note that in this algorithm, A can be chosen directly as Hessian matrix at current point x_k or SR1 approximate Hessian matrix. With respect to SR1 update, which means to update $A(H_k)$ by the following formula

$$H_{k+1} = H_k + \frac{(y_k - H_k s_k)(y_k - H_k s_k)^T}{(y_k - H_k s_k)^T s_k}, \quad (4.7)$$

where y_k and s_k are defined same as BFGS method in Section 3.3, and $\frac{(y_k - H_k s_k)(y_k - H_k s_k)^T}{(y_k - H_k s_k)^T s_k}$ is a rank-one matrix.

Note that to make this update well-defined, we need the extra judgment that

$$|(y_k - H_k s_k)^T s_k| \geq c \|y_k - H_k s_k\| \|s_k\|, \quad (4.8)$$

where $c \in (0, 1)$ is a user-specified constant. Also there is no guarantee that this update $H_{k+1} \succ 0$ even if $H_k \succ 0$. However, it is not necessary to restrict H_k to be positive definite, since the trust region subproblem are a constraint optimization problems.

Chapter 5

Numerical Results

This chapter contains numerical results with respect to those algorithms we discussed in Chapter 3 and 4.

Note that all algorithms are implemented by Matlab under Intel Core i5 2.6 GHz processor and 8 GB memory. Throughout this chapter, with respect to each algorithm, we use *Iter.* to represent the total number of Iterations it takes to meet termination conditions. If the actual *Iter.* is greater than defined *maxiter*, we mark *Iter.* by a slash. *Cputime* stands for total time of a algorithms spend to find the minimizer and when the corresponding *Iter.* is marked as a slash, *Cputime* means time cost up to *maxiter* iteration. We denote *xNorm* by the norm of difference between ending point of an algorithm and the accurate minimizer. *gNorm* represents for the norm of gradient at the ending point.

We begin by providing some general comments that may be useful for achieve a economy computation.

We set the default parameter value as following

Table 5.1: Default Parameter Set

i	maxiter	opttol	(c1ls,c2ls)	(c1tr,c2tr)	
value	1000	10^{-6}	$(10^{-4}, 0.9)$	$(0.3, 0.9)$	
i	cgopttol	cgmaxiter	sr1updatetol	bfgsupdatetol	initialradius
value	10^{-6}	50	10^{-6}	0.2	0.25
i	perturbHession	shrinkradius	expandradius	shrinkbacktrack	residuetol
value	10^{-4}	0.25	2	0.25	10^{-6}
i	wolfemax	posdeftol	mineigtol		
value	10	0.1	1.0		

By this default set, we state numerical results for all four test problems, respectively.

(1) Rosenbrock

Objective function:

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2. \quad (5.1)$$

Accurate minimizer:

$$(x_1, x_2) = (1, 1). \quad (5.2)$$

Numerical results with initial point $0\mathbf{e}$:

Table 5.2: Rosenbrock, with initial $0\mathbf{e}$

ROSENBROCK	gNorm	Iter.	Cputime	feval	geval	Hevel	Linsolved
steepestbacktrack	$1.000e-01$	—	0.3856	6236	1002	0	0
steepestwolfe	$1.010e-01$	—	0.8075	18629	2071	0	0
newtonbacktrack	$7.406e-07$	13	0.0544	30	14	13	13
newtonwolfe	$3.652e-10$	14	0.0500	49	19	14	14
trustregioncg	$1.019e-06$	54	0.0784	55	32	32	0
sr1trustregioncg	$8.581e-07$	68	0.0642	69	42	0	0
bfgsbacktrack	$3.416e-08$	23	0.0530	55	24	0	23
bfgswolfe	$6.357e-07$	21	0.0580	76	27	0	21

Rosenbrock function is supposed to be create to illustrate the inefficiency of steepest descent method. Especially, when current point lies in some area nearby the optimal point $(1, 1)$, the convergent rate decrease dramatically. By checking its contour, which we call banana shape, it shows that in such a long and narrow arrow, steepest descent direction are roughly orthogonal with each other. This fact lead to low convergence result. However, with respect to other algorithms, their performance are somehow similar. They all lead to a minimizer in roughly same iterations.

(2) Genhumps

Objective function:

$$f(x) = \sum_{i=1}^4 (\sin(2x_i)^2 \sin(2x_{i+1})^2 + 0.05(x_i^2 + x_{i+1}^2)). \quad (5.3)$$

Accurate minimizer:

$$x = 0\mathbf{e}. \quad (5.4)$$

Numerical results with initial point \mathbf{e} :

Table 5.3: Genhumps, with initial \mathbf{e}

GENHUMPS	gNorm	Iter.	Cputime	feval	geval	Hevel	Linsolved
steepestbacktrack	$3.589e-06$	102	0.1166	261	103	0	0
steepestwolfe	$3.884e-06$	145	0.1051	572	197	0	0
newtonbacktrack	$2.716e-06$	18	0.0491	38	19	18	18
newtonwolfe	$1.742e-07$	12	0.0591	43	16	12	12
trustregioncg	$1.841e-06$	64	0.0778	65	40	40	0
sr1trustregioncg	$1.414e-06$	67	0.0670	68	43	0	0
bfgsbacktrack	$3.330e-06$	29	0.0613	63	30	0	29
bfgswolfe	$2.921e-06$	19	0.1469	66	23	0	19

Genhumps function is much nicer than Rosenbrock. By looking at the formula (5.3), in the neighbor of accurate minimizer, $\sin(x) \sim x$, the function is suppose to have good convexity.

Actually, it can be shown in the numerical experiment, all algorithms show consistent performance. Compared with all those algorithms, newtonbacktrack and newtonwolfe are much better, based on number of iteration and number of function evaluation.

(3) Quadratic

Objective function:

$$f(x) = g^T x + \frac{1}{2} x^T H x. \quad (5.5)$$

where $g \in \mathbb{R}^{10}$ and $H \in \mathbb{R}^{10 \times 10}$.

Accurate minimizer:

$$x = (H^T H)^\dagger H^T g. \quad (5.6)$$

Numerical results with initial point $0\mathbf{e}$:

Table 5.4: Quadratic, with initial $0\mathbf{e}$

QUADRATIC	gNorm	Iter.	Cputime	feval	geval	Hevel	Linsolved
steepestbacktrack	$1.537e-06$	18	0.1983	37	19	0	0
steepestwolfe	$1.537e-06$	18	0.0968	55	19	0	0
newtonbacktrack	$4.591e-16$	1	0.0665	3	2	1	1
newtonwolfe	$4.591e-16$	1	0.0483	4	2	1	1
trustregioncg	$4.949e-11$	9	0.0737	10	7	7	0
sr1trustregioncg	$6.036e-07$	15	0.0681	16	10	0	0
bfgsbacktrack	$2.952e-06$	10	0.0969	21	11	0	10
bfgswolfe	$2.952e-06$	10	0.0923	31	11	0	10

Quadratic function turns to be the most well-defined function. All algorithm have good performance when solving this problem, and no surprisingly, by using newton method, it always give the accurate result in one iteration with one Hessian evaluation.

(4) Leastsquares

Objective function:

$$f(x) = \frac{1}{2} \|x_1 \mathbf{e} + x_2 e^{-\frac{t+x_3 \mathbf{e}}{x_4}} - y\|^2. \quad (5.7)$$

where $y = z_1 \mathbf{e} + z_2 e^{-\frac{t+z_3 \mathbf{e}}{z_4}} + \epsilon \in \mathbb{R}^{100}$, $z = (2, 1, -5, 4)^T$ and $\epsilon \in \mathbb{R}^{100}$ is a perturbation.

Accurate minimizer:

$$x = (189.9 - 187.2 - 4.948 - 4862)^T. \quad (5.8)$$

Numerical results with initial point $(0, 0, 0, 1)^T$:

Table 5.5: Leastsquares, with initial $(0, 0, 0, 1)^T$

LEASTSQUARES	gNorm	Iter.	Cputime	feval	geval	Hevel	Linsolved
steepestbacktrack	$4.093e-03$	—	0.5644	4791	1002	0	0
steepestwolfe	$1.185e-02$	—	1.3112	13179	2054	0	0
newtonbacktrack	$2.227e-04$	49	0.0764	99	50	49	49
newtonwolfe	$2.366e-04$	46	0.0710	143	49	46	46
trustregioncg	$2.089e-04$	41	0.0676	42	42	42	0
sr1trustregioncg	$1.833e-04$	37	0.0593	38	24	0	0
bfgsbacktrack	$1.981e-05$	86	0.0848	180	87	0	86
bfgswolfe	$2.227e-05$	4	0.0481	28	8	0	4

Leastsquares suppose to be a not well-conditioned function, by some points, my programming can not work well, even go divergence. The reason why it happens should be a huge gradient Lipschitz constant during some area. The gradient value changes dramatically at some nearby point. The other reason might be a ill-conditioned Hessian matrix at some point.

Beyond those four experiments, I also test a lot of starting points to test all of these eight algorithms with various parameters. I'll leave conclusion to the next chapter and say few words about the number of function evaluation (function value, gradient value and Hessian). Actually, by proper implementation, in backtracking linear search related method, the minimum number of gradient value evaluation should be exactly greater than iteration by 1. For trust region method, since we use cg method to solve subproblem, there is no need to compute gradient value in solving subproblem. And if we reject the direction, there is no update on x , we can simply keep the function information at current point. It will decrease the number of gradient value evaluation. Thus, it's reasonable to see the gradient value evaluation are less than iteration. However, since we always need calculate function value at next point to decide whether should we reject or accept the direction. It supposed to be one more time of function value evaluation than iteration on trust region type method.

Talking about Hessian evaluation, we need calculate Hessian matrix only in newton method and trust region with cg method. For linear systems solve, it only happened in bfgs method and newton method, which can be found in those four tableau, where the number of linear system solve is exactly same with the number of iteration.

Chapter 6

Conclusion and Experience

In this project, I implement eight algorithms to solve several test problems. To conduct the conclusion, first I'll say as expected there is no algorithm can always solve problems efficiently. In general, specific algorithm which works pretty well in a problem will possibly failed when solve other problems. Even when solve the same problems at different initial points, the final results may be totally different. One interesting thing to mention is the parameters chosen, in general, are not so crucial. For a roughly stable algorithm, by small change of parameters, numerical results are not affect too much. Thus, the efficiency of an algorithm are more related to the structure of the problems. Meanwhile, with respect to different algorithms, there are different corresponding ill-conditional and well-conditional problems. Some detailed discussion is stated as follows.

Numerical performance

- As I claimed in the first paragraph, there is no algorithm that can always beat others in general.
- I may say that steepest descent method with line search method is weaker than others in majority test problems. To perform better, a well-conditional problem should be provided. However, steepest descent is tend to be the most stable algorithm, not enough to obtain the accurate result, but still much powerful to handle all kinds of problems.
- With respect to newton direction type method, it shows great performance when in the case that all others can somehow derive the correct results. To have excellent performance, it should expect a good Hessian conditions. And as we known, practically, the performance of this method depends on the initial point choosing.
- In terms of BFGS method, the performance is relatively stable, since it is not so much dependent on Hessian matrix. Under same condition, BFGS method is slight weaker that newton method.
- The trust region method is relatively sensitive on parameters selecting. And it performance also depends much on problem. For problems which are changing dramatically, it shows bad performance, since in this case, the radius may decrease to be very close to 0 which make the ration (real function value reduction over estimated function value reduction) even larger, which is unstable when concern at numerical issue.
- About line search method, wolfe type method is more stable than backtracking strategy, since for backtracking, the strategy is to reduce initial step-size till it satisfies sufficient

condition, while in wolfe condition, it is also necessary to satisfies curvature condition which will reject some too small step-size.

Coding complexity

- With respect to coding complexity, it's obvious that steepest descent method is easiest among all the algorithms. This might be reasonable when turns to its performance, since there are less control on the algorithm.
- Talking about newton type method, the implementation is also simple, while we need special concern about the Hessian matrix since it effect it performance a lot. Different strategy can be used to modify a ill-conditional Hessian matrix.
- For BFGS method, more concern happen. Except for regular implementation, we need also pay attention on decrease the number of gradient evaluation to make it run cheaply. Also damped BFGS is applied on making sure the positive definiteness of the BFGS method.
- In term of trust region method, which is the most complex implementation in this project, we need more concern since not only to develop code to solve subproblem but also need tricks to save computation.
- About line search method, wolfe type method is more complex than backtracking strategy, since for backtracking, we can only evaluate gradient value once and using a simply loop to decide a proper step-size. While when turns to wolf condition, not only we need verify two conditions and have more amount of evaluation, also we need choose a proper zoom function to find the step-size and more parameters to decide.

Summery

- Obvious result is that the more sophisticated designation of the algorithm it shows, the more stable performance it has. However, I can't say the same observation with respect to their performance.
- For all coder, I'll recommend newton method or steepest method with backtracking line search strategy, since they are cheaper to implement and in general, it will obtain good result, either stable or fast.
- To recommend to an expert coder, I prefer to suggest to use trust region method and BFGS method with wolfe line search strategy, since such algorithms are more flexible to accept user set-up. Also variety choices can be made to adjust the efficiency of algorithms.
- If someone concern more on the expense of evaluation of Hessian matrix, I'll suggest them to use BFGS method and trust region method with SR1 approximate matrix cg method. Especially, when it's expensive to obtain Hessian matrix.

Begin with passion, stuck in debugging and meanwhile find that there are always something can be done to improve this program even it nicely give me a correct result. This should be a real feeling of finishing this project. Sometimes, when I felt that my code runs well and suddenly some others came to me and said: Hey, buddy, just try this initial point. Occasionally, it made me sad because sometimes the program crashed and I had to go back to check if anything wrong somewhere. But, fortunately, it finally finished and seems show some reasonable results always. I'm glad to see this and look forward to test your secret function!

Bibliography

- [1] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, New York, NY, USA, 2nd edition, 2006.