CSC 753 Nonlinear Optimization Midterm

Shuowen Wei

1.

Proof:

Let set and , thus .

For any two points and in , we have . Since and is a convex set, for any , the point is also in . Plus is a convex function, thus

Since is the minimal value of in , then , then the point is also a minimizer of , thus . Because the two points and are arbitrary in , hence is a convex set. QED

2.

(a).

Since , we can rewrite as:

Thus, the gradient of is:

And the Hessian of is:

(b).

Yes, since , has full column rank and , thus is strictly (strong) convex, which is, for all in the domain.

(c).

By the conclusion from question (b) that is strongly convex, thus only has one global minimizer, which can be solved by setting the gradient of to be a zero vector.

Then,

Hence, solving for the minimizer is equivalent to a linear least squares problem. QED.

(d).

clc;clear

H=hilb(15);

A=H(:,1:13);

b=A\*ones(13,1);

>> cond(H)

ans =

2.5699e+017

>> cond(A)

ans =

6.1648e+016

As you see, the condition number of matrix is very large, thus is ill-conditioned, it is not wise to use the *inv* command in Matlab to compute , we use slash ‘\’ instead, and also computer the relative error:

x=(A'\*A)\(A'\*b)

x =

0.9988

1.0600

0.2748

4.4685

-6.9167

12.2136

-18.4075

37.8339

-39.9372

27.7935

-21.7479

22.4648

-7.0998

x0=ones(13,1);

>> norm(x-x0)/norm(x0)

ans =

20.3279

The relative error is very large, thus we are thinking that by choosing a good might be possible to get a good solution, we first choose to be from 0 to 0.99 with step=0.01:

relative\_err=[];

for lamda=0:0.01:0.99

x=(A'\*A+lamda\*eye(13))\(A'\*b);

abs\_err=norm(x-x0);

new\_rel\_err=abs\_err/norm(x0);

relative\_err=[relative\_err new\_rel\_err];

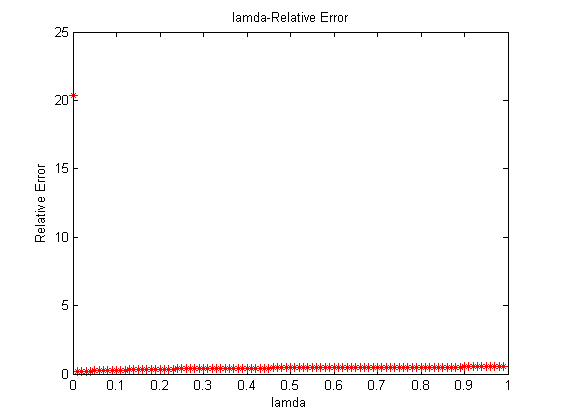
end

lamda=0:0.01:0.99;

plot(lamda,relative\_err,'r\*');

title('lamda-Relative Error');

xlabel('lamda');ylabel('Relative Error')



>> position=find(relative\_err==min(relative\_err))

position =

2

>> lamda(position)

ans =

0.0100

>> relative\_err(position)

ans =

0.1603

This relative error 16.03% is still very large, but the good thing we know from the graph is that a better might be chosen from 0 to 0.01, then we will try , codes and results are as follows:

format compact

format long

index=1:15;

lamda\_index=10.^(-index);

relative\_err=[];

for i=1:length(lamda\_index)

lamda=lamda\_index(i);

x=(A'\*A+lamda\*eye(13))\(A'\*b);

abs\_err=norm(x-x0);

new\_rel\_err=abs\_err/norm(x0);

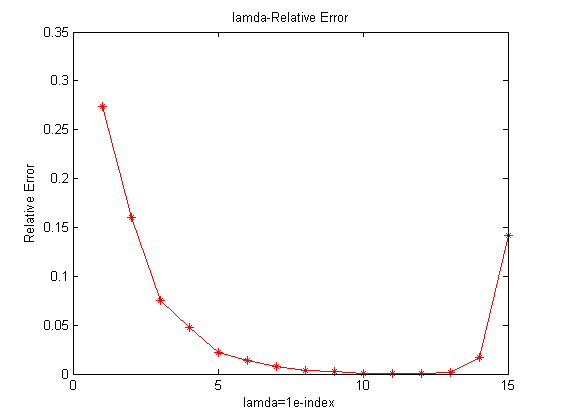
relative\_err=[relative\_err new\_rel\_err];

end

plot(index,relative\_err,'r-\*');

title('lamda-Relative Error');

xlabel('lamda=1e-index)');ylabel('Relative Error')



>> min(relative\_err)

ans =

4.469976609828409e-004

>> position=find(relative\_err==min(relative\_err))

position =

12

>> lamda\_index(position)

ans =

1.000000000000000e-012

>> x\_best=(A'\*A+lamda\_index(position)\*eye(13))\(A'\*b)

x\_best =

0.999996755944332

1.000077855548524

0.999615198136232

1.000450080586117

1.000337665282783

0.999714365065206

0.999554583346823

0.999768237245273

0.999810433749668

1.000324283018898

1.000938648884424

1.000249534296386

0.999158196141711

Now, we should be satisfied since we have successfully made the relative error down to 0.0447%, which is very small, when we choose .

3.

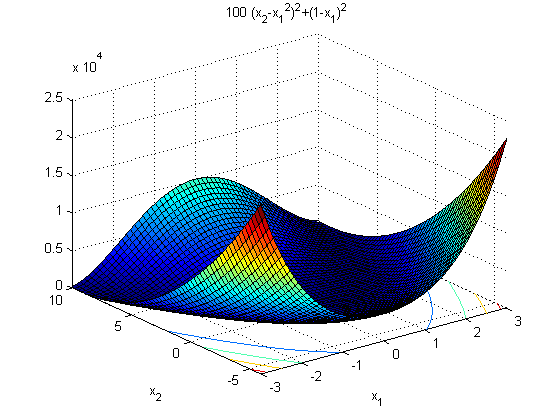
(a).

First of all, let’s see the graph of the Rosenbrock (banana) function:

clc;clear

f = @(x1,x2) 100.\*(x2-x1.^2).^2+(1-x1).^2;

ezsurfc(f,[-3,3],[-6,10]);



Clearly, the Rosenbrock function is not a convex function, there is a concave region around the point (0,10).

We choose some starting vectors e.t.c and apply the *fminunc* command in Matlab:

fun = @(x) f(x(1),x(2));

options = optimset('LargeScale','off','Display','iter','TolFun',1e-8);

>> x0 = [0 2];

>> [x, fval, exitflag, output, grad, hessian] = fminunc(fun,x0,options)

>> [x, fval, exitflag, output, grad, hessian] = fminunc(fun,x0,options)

First-order

Iteration Func-count f(x) Step-size optimality

0 3 401 400

1 6 100.985 0.0025 200

2 9 0.922191 1 1.89

3 15 0.750194 10 1.69

4 24 0.626276 0.153308 4.21

5 27 0.546491 1 6.11

6 30 0.379839 1 1.99

7 33 0.322538 1 6.29

8 36 0.223113 1 0.55

9 39 0.152471 1 1.63

10 45 0.123688 0.499711 3.92

11 48 0.081469 1 2.11

12 51 0.0480674 1 2.59

13 54 0.0277131 1 2.69

14 60 0.00851779 0.484821 1.21

15 63 0.00614371 1 1.6

16 66 0.00101847 1 0.128

17 69 0.000335803 1 0.624

18 72 2.23066e-005 1 0.036

19 75 3.47877e-007 1 0.0174

First-order

Iteration Func-count f(x) Step-size optimality

20 78 1.04696e-009 1 0.000198

21 81 1.26242e-011 1 4.29e-005

Local minimum possible.

fminunc stopped because it cannot decrease the objective function

along the current search direction.

<stopping criteria details>

Computing finite-difference Hessian using user-supplied objective function.

x =

1.0000 1.0000

fval =

1.2624e-011

exitflag =

5

output =

iterations: 22

funcCount: 96

stepsize: 1

firstorderopt: 4.2854e-005

algorithm: 'medium-scale: Quasi-Newton line search'

message: [1x362 char]

grad =

1.0e-004 \*

0.4285

-0.2033

hessian =

802.2876 -400.0231

-400.0231 200.0000

>> det(hessian)

ans =

439.0747

The next starting point is [0 1]:

>> x0 = [0 1];

>> [x, fval, exitflag, output, grad, hessian] = fminunc(fun,x0,options)

First-order

Iteration Func-count f(x) Step-size optimality

0 3 101 200

1 6 0.980101 0.005 1.98

2 12 0.807974 10 2.37

3 18 0.728199 0.348315 6.33

4 21 0.622166 1 3.97

5 24 0.47105 1 5.95

6 27 0.327299 1 2.26

7 33 0.232498 0.657725 2.85

8 36 0.201878 1 4.17

9 39 0.120646 1 1.5

10 42 0.0876488 1 4.36

11 45 0.0440962 1 0.379

12 51 0.0282506 0.659123 2.58

13 54 0.0159084 1 1.79

14 57 0.00643211 1 0.0781

15 63 0.00350151 0.399096 1.3

16 66 0.00123136 1 0.681

17 69 0.000193982 1 0.164

18 72 3.13952e-005 1 0.211

19 75 1.82196e-005 1 0.17

First-order

Iteration Func-count f(x) Step-size optimality

20 78 2.48751e-010 1 6.01e-005

21 81 2.6868e-011 1 6.14e-006

22 84 2.00565e-011 1 3.81e-009

Local minimum found.

Optimization completed because the size of the gradient is less than

the selected value of the function tolerance.

<stopping criteria details>

Computing finite-difference Hessian using user-supplied objective function.

x =

1.0000 1.0000

fval =

2.0057e-011

exitflag =

1

output =

iterations: 22

funcCount: 84

stepsize: 1

firstorderopt: 3.8098e-009

algorithm: 'medium-scale: Quasi-Newton line search'

message: [1x440 char]

grad =

1.0e-008 \*

-0.3810

0.1839

hessian =

802.2858 -400.0226

-400.0226 200.0000

>> det(hessian)

ans =

439.0665

Next starting point is [2 2]:

>> x0 = [2 2];

options = optimset('LargeScale','off','GradObj','on','GradObj','on','Display','iter','TolFun',1e-8);

[x, fval, exitflag, output, grad, hessian] = fminunc(@fun\_3b,x0,options)

First-order

Iteration Func-count f(x) Step-size optimality

0 1 401 802

Local minimum possible.

fminunc stopped because it cannot decrease the objective function

along the current search direction.

<stopping criteria details>

x =

2 2

fval =

401

exitflag =

5

output =

iterations: 1

funcCount: 15

stepsize: []

firstorderopt: 802

algorithm: 'medium-scale: Quasi-Newton line search'

message: [1x362 char]

grad =

-802

-400

hessian =

-402.0000 -800.0000

-800.0000 200.0000

>> det(hessian)

ans =

-7.2040e+005

As you can see that, it is really hard to approximate the minimizer (1,1), or it even can’t reach the minimizer when start from some starting points, like [0 2] and [2 2], their iteration stopped because the *fminunc* cannot decrease the objective function along the current search direction. You can easily find that the values from the *“**First-order optimality”* (highlighted above) are not always decreasing even in some iteration that found the minimizer in the end. The reason for this is that the Rosenbrock (banana) function is not convex, then when the iteration points are getting close to the local minimizer, they will fall into the concave part of the function, where makes the *“First-order optimality”* not decreasing along the current search direction, thus the iterations either stopped (like starting point [0 2], its hessian value is negative) or slowly but will finally reach the minimizer (like [0 1]).

(b).

Define the function and its gradient in a m-file as follows:

function [f,g] = fun\_3b(x)

f = 100\*(x(2)-x(1)^2)^2+(1-x(1))^2;

if (nargout>1)

g(1) = -400\*x(1)\*(x(2)-x(1)^2)-2+2\*x(1);

g(2) = 200\*(x(2)-x(1)^2);

end

Then run *fminuns* using the gradient, the results are as follows:

>> x0 = [0 2];

>> options = optimset('LargeScale','off','GradObj','on','GradObj','on','Display','iter','TolFun',1e-8);

[x, fval, exitflag, output, grad, hessian] = fminunc(@fun\_3b,x0,options)

First-order

Iteration Func-count f(x) Step-size optimality

0 1 401 400

1 2 100.985 0.0025 200

2 3 0.922191 1 1.89

3 5 0.750194 10 1.69

4 8 0.626276 0.153308 4.21

5 9 0.546492 1 6.11

6 10 0.37984 1 1.99

7 11 0.322538 1 6.29

8 12 0.223112 1 0.55

9 13 0.152471 1 1.63

10 15 0.123687 0.49972 3.92

11 16 0.0814676 1 2.11

12 17 0.0480665 1 2.59

13 18 0.0277122 1 2.69

14 20 0.00851736 0.484801 1.21

15 21 0.00614343 1 1.6

16 22 0.0010184 1 0.128

17 23 0.000335696 1 0.624

18 24 2.22358e-005 1 0.0358

19 25 3.44133e-007 1 0.0175

First-order

Iteration Func-count f(x) Step-size optimality

20 26 7.78422e-010 1 0.000201

21 27 2.22716e-012 1 4.27e-005

22 28 1.08115e-016 1 4.1e-007

Local minimum found.

Optimization completed because the size of the gradient is less than

the selected value of the function tolerance.

<stopping criteria details>

x =

1.0000 1.0000

fval =

1.0812e-016

exitflag =

1

output =

iterations: 22

funcCount: 28

stepsize: 1

firstorderopt: 4.1006e-007

algorithm: 'medium-scale: Quasi-Newton line search'

message: [1x440 char]

grad =

1.0e-006 \*

-0.4101

0.2063

hessian =

802.0000 -400.0000

-400.0000 200.0000

>> det(hessian)

ans =

400.0023

Here, the reason that the *fminunc* is able to find the minimizer from starting point [0 2] is that we supply *fminunc* with the gradient of the Rosenbrock function and set the *GradObj* parameter in the *options* is set to ‘*on’* using *optimset.* Thus, *fminunc* could approximate the minimizer more precisely and closely since the default *‘off’* causes *fminunc* to estimate gradients using finite differences. According to the *doc fminunc*:for Medium-Scale Optimization (setting *LargeScale* parameter to *‘off’*),the *fminunc* uses the default line search algorithm, i.e., when the *LineSearchType* parameter is set to *'quadcubic'*, is a safeguarded mixed quadratic and cubic polynomial interpolation and extrapolation method. This second method generally requires fewer function evaluations but more gradient evaluations. Thus, if gradients are being supplied and can be calculated inexpensively, the cubic polynomial line search method is preferable.

(c).

Now, we are going the use *fminsearch* in Matlab to solve this problem again, codes and results are showed below:

clc;clear

f = @(x1,x2) 100.\*(x2-x1.^2).^2+(1-x1).^2;

fun = @(x) f(x(1),x(2));

x0 = [0 2];

options = optimset('Display','iter','TolFun',1e-8);

[x fval,exitflag,output] = fminsearch(fun,x0,options)

Iteration Func-count min f(x) Procedure

0 1 401

1 3 400.999 initial simplex

2 5 324.999 expand

3 7 289.998 expand

4 9 157.247 expand

5 11 69.0554 expand

6 13 9.25764 expand

7 14 9.25764 reflect

8 16 3.63193 contract inside

9 18 3.63193 contract outside

10 20 1.13235 contract inside

11 21 1.13235 reflect

12 23 1.13193 contract inside

13 25 1.13193 contract inside

14 27 1.10955 contract inside

15 29 1.02383 contract inside

16 31 0.99459 contract inside

17 33 0.99459 contract outside

18 35 0.99459 contract inside

19 37 0.992937 contract outside

20 39 0.991994 contract inside

21 41 0.991562 contract inside

22 43 0.991561 contract outside

23 45 0.991462 contract inside

24 46 0.991462 reflect

25 48 0.991443 contract inside

26 50 0.991423 reflect

27 52 0.991423 contract inside

28 54 0.99138 expand

29 56 0.991366 expand

30 58 0.991242 expand

31 59 0.991242 reflect

32 61 0.990999 expand

33 62 0.990999 reflect

34 64 0.990854 expand

35 66 0.990347 expand

36 67 0.990347 reflect

37 69 0.98926 expand

38 70 0.98926 reflect

39 72 0.987204 expand

40 74 0.986372 expand

41 76 0.981466 expand

42 77 0.981466 reflect

43 79 0.97178 expand

44 81 0.966916 expand

45 83 0.945083 expand

46 85 0.925423 expand

47 87 0.874422 expand

48 89 0.811599 expand

49 91 0.744992 expand

50 93 0.705966 reflect

51 95 0.705966 contract inside

52 97 0.705966 contract inside

53 99 0.656177 expand

54 100 0.656177 reflect

55 102 0.567215 expand

56 103 0.567215 reflect

57 105 0.45603 expand

58 107 0.45603 contract outside

59 109 0.45603 contract outside

60 111 0.382561 expand

61 112 0.382561 reflect

62 114 0.274754 expand

63 115 0.274754 reflect

64 116 0.274754 reflect

65 118 0.18629 expand

66 120 0.18629 contract inside

67 122 0.159194 reflect

68 124 0.107865 expand

69 125 0.107865 reflect

70 127 0.0468516 reflect

71 129 0.0468516 contract inside

72 131 0.0468516 contract inside

73 132 0.0468516 reflect

74 134 0.0384953 reflect

75 136 0.0370467 contract inside

76 138 0.017549 expand

77 139 0.017549 reflect

78 141 0.00717609 reflect

79 143 0.00717609 contract inside

80 144 0.00717609 reflect

81 146 0.00231194 reflect

82 148 0.00231194 contract inside

83 150 0.00231194 contract outside

84 152 0.00187996 reflect

85 154 1.63204e-005 expand

86 156 1.63204e-005 contract inside

87 158 1.63204e-005 contract outside

88 160 1.63204e-005 contract outside

89 162 1.63204e-005 contract inside

90 163 1.63204e-005 reflect

91 165 1.63204e-005 contract inside

92 167 1.13224e-005 contract inside

93 168 1.13224e-005 reflect

94 170 2.557e-006 contract inside

95 172 1.4293e-006 contract inside

96 174 1.33034e-006 contract inside

97 176 4.23026e-007 contract inside

98 178 1.85438e-007 contract inside

99 180 1.28204e-007 contract inside

100 182 8.27537e-008 contract inside

101 184 3.31438e-008 contract inside

102 186 5.8643e-009 contract inside

103 188 5.8643e-009 contract inside

104 189 5.8643e-009 reflect

105 191 3.36795e-009 contract inside

106 193 1.37165e-009 contract outside

107 195 7.23959e-010 contract inside

108 197 3.55656e-010 contract inside

Optimization terminated:

the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-004

and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-008

x =

1.0000 1.0000

fval =

3.5566e-010

exitflag =

1

output =

iterations: 108

funcCount: 197

algorithm: 'Nelder-Mead simplex direct search'

message: [1x196 char]

The algorithms that *fminsearch* uses,according to *doc fminsearch* in Matlab, is the simplex search method of Lagarias, which is a direct search method that does not use numerical or analytic gradients.

If *n* is the length of *x*, a simplex in *n*-dimensional space is characterized by the *n+1* distinct vectors that are its vertices. In two-space, a simplex is a triangle; in three-space, it is a pyramid. At each step of the search, a new point in or near the current simplex is generated. The function value at the new point is compared with the function's values at the vertices of the simplex and, usually, one of the vertices is replaced by the new point, giving a new simplex. This step is repeated until the diameter of the simplex is less than the specified tolerance.

Advantages and disadvantages?

Here, yes, it is advantageous to use *fminsearch*, since *fminsearch* does not use numerical or analytic gradients, so the concave part of the Rosenbrock function will not affect it in search. But there are also some other advantages and limitations in using *fminsearch* that we want to point out, according *doc fminsearch* in Matlab, *fminsearch* can often handle discontinuity, particularly if it does not occur near the solution. *fminsearch* may only give local solutions, and *fminsearch* only minimizes over the real numbers, that is, *x* must only consist of real numbers and *f*(*x*) must only return real numbers. When *x* has complex variables, they must be split into real and imaginary parts.

4.

Define the “Extended” Rosenbrock function and its gradient in a m-file as follows, choose :

function [f,g] = fun\_4(x)

k = 200;

f = 0;

for i=1:2\*k

f = f + (x(2\*i)-x(2\*i-1)^2)^2+(1-x(2\*i-1))^2;

end

g = ones(2\*2\*k,1);

if nargout>1

for i = 1:2\*k

g(2\*i-1) = -4\*x(2\*i-1)\*(x(2\*i)-x(2\*i-1)^2)+x(2\*i-1)-2;

g(2\*i) = 2\*(x(2\*i)-x(2\*i-1)^2);

end

end

Then use the *fminunc* in Matlab to solve this problem and results are as follows, except printing out the minimizer, since it’s too large:

clc;clear

x0=-ones(1,800);

options = optimset('LargeScale','on','GradObj','on','Display','iter','TolFun',1e-8);

[x, fval] = fminunc(@fun\_4,x0);

>> fval

fval =

1.1008e-008

According to *doc fminunc*, by default in Matlab, the *fminunc* chooses the large-scale algorithm if the gradient of the target function are provided and the *GradObj* parameter in the *options* is set to ‘*on’* using *optimset*. The algorithm is a subspace trust region method and is based on the interior-reflective Newton method. Each iteration involves the approximate solution of a large linear system using the method of preconditioned conjugate gradients (PCG).