

Unconstrained Optimization: Exercise 3

Beatrice Macchia, Riccardo Bosio

Politecnico di Torino

Student ids: s245844, s291299

s245844@studenti.polito.it, s291299@studenti.polito.it

I. PROBLEM OVERVIEW

The problem consists of implementing the Inexact Newton method with line search to solve $\min_{x \in \mathbb{R}^n} \sum_{i=1}^n (\frac{1}{4}x_i^4 + \frac{1}{2}x_i^2 + x_i)$ with $n = 10^4$ and $n = 10^5$. We use the method with several choices of the forcing terms η_k , testing choices which guarantee linear, superlinear and quadratic convergence. Then we make comparisons between the behaviors of these three cases and the pure Newton method.

II. PROPOSED APPROACH

To solve this unconstrained optimization problem we first create *function_to_be_optimized.m* to introduce the function that we have to optimize. This function is of class C^2 .

We build the gradient function in *grad.m* using the exact derivatives, while we define the hessian matrix in *U3.m* exploiting her sparsity. As a matter of fact, the hessian matrix is a diagonal matrix in which the i -th component of the diagonal corresponds to $3 \cdot x_i^2 + 1$. Moreover, the hessian matrix is symmetric positive definite, which is a necessary condition in order to get a descent direction.

We implement the pure Newton method in *newton_method.m* with backtracking, using the Armijo condition. We compare it with the Inexact Newton method, which we create in *inexact_newton_method.m*. Here we put the input variable 'type_fterms', which is a string that specifies the type of forcing terms: 'l' stands for 'linear', 's' for 'superlinear' and 'q' for 'quadratic'. Then we use the forcing terms to define the tolerance in the pcg solver. It must be used to compute the approximated descent direction. Also, we implement the backtracking strategy using the Armijo condition.

Lastly we compare the two behaviors in *U3.m*.

III. RESULTS

We tested the methods with the following initialization of the parameters:

Initialization	
Parameter	Value
alpha0	1
kmax	1000
tolgrad	1e-12
c1	1e-4
rho	0.8
btmax	50
max_pcgiterations	50

In the following two tables we analyse the behavior of Pure Newton method and Inexact Newton method, with the three types of forcing terms, for $n = 10^4$.

All the methods reach the same solution: $xk = (-0.6823, \dots, -0.6823)^T$. The corresponding value of the function is $f(xk) = -3.9535e + 03$.

Comparisons are made in terms of computing time, while iterations, backtracking iterations and, in the case of Inexact Newton method, pcg iterations.

$n = 10^4$			
Pure Newton method			
Computing time	0,100093		
while iterations	6		
backtracking iterations	(0, 0, 0, 0, 0, 0)		
$n = 10^4$			
Inexact Newton method			
	Linear	Superlinear	Quadratic
Computing time	0,172089	0,083273	0,086158
pcg iterations	(1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 1, 2)
while iterations	6	6	6
backtracking iterations	(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0)

Now we repeat the comparison with $n = 10^5$. Once again all the methods give the same solution $xk = (-0.6823, \dots, -0.6823)^T$ with $f(xk) = -3.9535e + 04$.

$n = 10^5$			
Pure Newton method			
Computing time	0,555737		
while iterations	6		
backtracking iterations	(0, 0, 0, 0, 0, 0)		

$n = 10^5$			
Inexact Newton method			
	Linear	Superlinear	Quadratic
Computing time	0,825968	0,929477	0,845908
pcg iterations	(1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 1, 1)	(1, 1, 1, 1, 1, 2)
while iterations	6	6	6
backtracking iterations	(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0)

These results are obtained starting from $x_0 = (0, \dots, 0)^T$. We tried for other values of x_0 : even if the values of the parameters we chose to compare change, the general behavior of the methods remains the same. For example for $x_0 = (-1, \dots, -1)^T$ all methods reach the same solution with 5 while iterations and no backtracking. Computing times are similar and pcg iterations are the same as for $x_0 = (0, \dots, 0)^T$.

IV. DISCUSSION

Pure Newton method seems to perform better in terms of computing time. Among forcing terms, the linear type is the slowest for $n = 10^4$, while for $n = 10^5$ the three types behave quite the same.

For both values of n and both starting points x_0 that we tried, we get that pcg solver always needs 1 iteration, unless with forcing terms that guarantee quadratic convergence. In this case pcg always does 1 iteration except for the last while iteration, where it does 2 iterations.

V. APPENDIX

A. *function_to_be_optimized.m*

```
1 %Create the function to be optimized
2 function y = function_to_be_optimized(x,n)
3 %
4 %INPUTS:
5 %x=column vector of length n;
6 %n=number of dimensions of x;
7 %OUTPUTS:
8 %y=real scalar value equal to f(x)
9 %
10 y=0;
11 for i=1:n
12     y=y+(1/4*x(i)^4+1/2*x(i)^2+x(i));
13 end
14 end
```

B. *grad.m*

```
1 %Create gradient function
2 function gradfx=grad(x,n)
3 %
4 %INPUTS:
5 %x=column vector of size n;
6 %n=number of dimensions;
7 %
8 %OUTPUTS:
9 %gradf=column vector where i-th component is ...
10 %the partial derivative of f
11 %with respect to the i-th component of x.
12 gradfx=zeros(n,1);
13 for i=1:n
14     gradfx(i)=x(i)^3+x(i)+1;
15 end
16 end
```

C. *newton_method.m*

```
1 %Newton method
2
3 function [xk,fk,gradfk_norm,k,xseq, ...
4     btseq]=newton_method(x0,f,gradf,...
5     Hess_f,alpha0,c1,kmax,tolgrad,rho,btmax)
6 %
7 %[xk,fk,gradfk_norm,k,xseq, ...
8     btseq]=newton_method(x0,f,gradf,...
9     Hess_f,alpha0,c1,kmax,tolgrad,rho,btmax)
10 %
11 %INPUTS:
12 %x0=column vector of dimension n;
13 %f=function R^n->R;
14 %gradf=gradient of f;
15 %Hess_f=Hessian of f;
16 %alpha0=the initial factor that multiplies ...
17 %the descent direction at each
18 %iteration;
19 %c1=factor of the Armijo condition that must ...
20 %be a scalar in (0,1);
21 %kmax=maximum number of iterations;
22 %tolgrad=value used as stopping criterion ...
23 %with respect to the norm of the
24 %gradient;
25 %rho=fixed factor, lesser than 1, used for ...
26 %reducing alpha0;
27 %btmax=maximum number of steps for updating ...
28 %alpha during the backtracking
29 %strategy.
30 %
31 %OUTPUTS:
32 %xk=the last x computed by the method;
33 %fk=f(xk);
34 %gradfk_norm=norm of gradf(xk);
35 %k=last iteration performed;
36 %xseq=n-by-k matrix where the columns are ...
37 %the xk computed during the
38 %iterations.
39 %btseq=1-by-k vector with the number of ...
40 %backtracking iterations at each
41 %optimization step.
42 %
43 %Initializations
44 xseq=zeros(length(x0), kmax);
45 btseq=zeros(1, kmax);
46 xk=x0;
47 fk=f(xk);
48 k=0;
49 gradfk_norm=norm(gradf(xk));
50
51 % Function handle for the armijo condition
52 f_armijo=@(fk, alpha, xk, pk) ...
53     fk+c1*alpha*gradf(xk) '*pk;
54
55 while k<kmax && gradfk_norm>tolgrad
56     %Compute the descent direction as ...
57     %solution of
58     %Hess_f(xk) p = - graf(xk)
59     pk=-Hess_f(xk)\gradf(xk);
60     %Reset the value of alpha
61     alpha=alpha0;
62     %Compute the candidate new xk
63     xnew=xk+alpha*pk;
64     %Compute the value of f in the candidate ...
65     %new xk
66     fnew=f(xnew);
67
68     bt = 0;
69     %Backtracking:
```

```

59     while bt<btmax && fnew>f_armijo(fk, ...
        alpha, xk, pk)
60         %Reduce the value of alpha
61         alpha=rho*alpha;
62         %Update xnew and fnew
63         xnew=xk+alpha*pk;
64         fnew=f(xnew);
65
66         bt = bt + 1;
67     end
68
69     xk=xnew;
70     fk=fnew;
71     gradfk_norm=norm(gradf(xk));
72
73     %Increase the step by one
74     k=k+1;
75
76     %Store current xk in xseq
77     xseq(:,k)=xk;
78     %Store bt iterations in btseq
79     btseq(k)=bt;
80 end
81
82 %"Cut" xseq and btseq to the correct size
83 xseq=xseq(:,1:k);
84 btseq=btseq(1:k);
85
86 end

```

D. inexact_newton_method.m

```

1  %Inexact Newton method
2  function ...
    [xk,fk,gradfk_norm,k,xseq,btseq,pcg_iter]=...
3      inexact_newton_method(x0,f,gradf,Hess_f,...
4      alpha0,kmax,tolgrad,c1,rho,btmax,...
5      type_fterms,max_pcgiterations)
6  %
7  %function [xk,fk,gradfk_norm,k,xseq,btseq]=...
8  %     inexact_newton_method(x0,f,gradf,Hess_f,...
9  %     alpha0,kmax,tolgrad,c1,rho,btmax,...
10 %     fterms,maxpcgiterations)
11 %
12 %INPUTS:
13 %x0=column vector of dimension n;
14 %f=function R^n->R;
15 %gradf=gradient of f;
16 %Hess_f=Hessian of f;
17 %alpha0=initial factor that multiplies the ...
    descent direction at each
18 %iteration;
19 %kmax=maximum number of iterations;
20 %tolgrad=value used as stopping criterion;
21 %c1=scalar factor of the Armijo condition, ...
    it must be in (0,1);
22 %rho=fixed factor used to reduce alpha, ...
    lesser than 1;
23 %btmax=maximum number of steps of ...
    backtracking strategy;
24 %type_fterms='l','s'or'q', string that ...
    specifies the type of forcing terms;
25 %max_pcgiterations=maximum number of iterations ...
    for the pcg solver to compute
26 %pk.
27 %
28 %OUTPUTS:
29 %xk=the last x computed by the method;
30 %fk=f(xk);
31 %gradfk_norm=norm of gradf(xk);
32 %k=last iteration;

```

```

33 %xseq=n-by-k matrix where the columns are ...
    the xk;
34 %btseq=1-by-k vector where elements are the ...
    number of backtracking
35 %iterations at each optimization step.
36 %pcg_iter=number of iterations of pcg
37 %
38
39 %Initializations
40 xseq=zeros(length(x0),kmax);
41 btseq=zeros(1,kmax);
42 pcg_iter=zeros(1,kmax);
43 xk=x0;
44 fk=f(xk);
45 k=0;
46 gradfk_norm=norm(gradf(xk));
47
48 %Function handle for the Armijo condition
49 f_armijo=@(fk,alpha,xk,pk) ...
    fk+c1*alpha*gradf(xk)*pk;
50
51 while k<kmax && gradfk_norm>tolgrad
52     %Compute the descent direction as ...
        solution of
53     %Hessf(xk) p = - graf(xk)
54     eta_k=0;
55     switch type_fterms
56         case 'l'
57             eta_k=0.5;
58         case 's'
59             eta_k=min(0.5,sqrt(norm(gradf(xk))));
60         case 'q'
61             eta_k=min(0.5,norm(gradf(xk)));
62         otherwise
63             eta_k=0.5;
64     end
65
66     %Tolerance varying with respect to ...
        forcing terms
67     epsilon_k=eta_k*norm(gradf(xk));
68     [pk,~,~,iter]=pcg(Hess_f(xk),...
        -gradf(xk),epsilon_k,max_pcgiterations);
69     pcg_iter(k+1)=iter;
70
71
72     %Reset the value of alpha
73     alpha=alpha0;
74
75     %Compute the candidate new xk
76     xnew=xk+alpha*pk;
77
78     %Compute f in the candidate new xk
79     fnew=f(xnew);
80
81     %Use backtracking strategy
82     bt=0;
83     while bt<btmax && ...
        fnew>f_armijo(fk,alpha,xk,pk)
84         %Reduce alpha
85         alpha=rho*alpha;
86
87         %Update xnew and fnew
88         xnew=xk+alpha*pk;
89         fnew=f(xnew);
90
91         bt=bt+1;
92     end
93
94     %Update xk,fk,gradfk_norm
95     xk=xnew;
96     fk=fnew;
97     gradfk_norm=norm(gradf(xk));
98
99     k=k+1;
100

```

```

101     %Store xk in xseq
102     xseq(:,k)=xk;
103
104     %Store bt in btseq
105     btseq(k)=bt;
106 end
107
108 %Resize xseq and btseq
109 xseq=xseq(:,1:k);
110 btseq=btseq(1:k);
111 pcg_iter=pcg_iter(1:k);
112
113 end

```

E. U3.m

```

1
2 clear all;
3 clc;
4 %
5 %
6 %Unconstrained optimization
7 %3)Inexact Newton method
8 %
9 %
10
11 %Initializations
12 n=10^4; %We run U3 also with n=10^5
13 x0=zeros(n,1);
14 alpha=1;
15 kmax=100;
16 tolgrad=1e-12;
17 alpha0=alpha;
18 c1=1e-4;
19 rho=0.8;
20 btmax=50;
21 max_pcgiterations=50;
22
23 %Create the function f
24 f=@(x) function_to_be_optimized(x,n);
25
26 %Create the gradient function
27 gradf=@(x) grad(x,n);
28
29 %Create the hessian function
30 Hess_f=@(x) diag(sparse(3.*x(:,1).^2+1));
31
32 %Run newton method
33 tic
34 [xk,fk,gradfk_norm,k,xseq,btseq]=...
35     newton_method(x0,f,gradf,...
36         Hess_f,alpha,kmax,...
37         tolgrad,c1,rho,btmax);
38 toc
39 display('Number of iterations in the Newton ...
40         method:')
41 display(k)
42 display('Value of fk:')
43 display(fk)
44 display('Number of inner iterations:')
45 display(btseq)
46
47 %Run inexact newton method
48 %Linear convergence
49 tic
50 [xk,fk,gradfk_norm,k,xseq,btseq,pcg_iter]=...
51     inexact_newton_method(x0,f,...
52         gradf,Hess_f,alpha0,kmax,tolgrad,c1,...
53         rho,btmax,'l',max_pcgiterations);
54 toc

```

```

55 display('Number of iterations in the inexact ...
56         Newton method:')
57 display('with linear convergence of forcing ...
58         terms:')
59 display(k)
60 display('Value of fk:')
61 display(fk)
62 display('Number of inner iterations:')
63 display(btseq)
64 display(pcg_iter)
65
66 %Superlinear convergence
67 tic
68 [xk,fk,gradfk_norm,k,xseq,btseq,pcg_iter]=...
69     inexact_newton_method(x0,f,...
70         gradf,Hess_f,alpha0,kmax,tolgrad,c1,...
71         rho,btmax,'s',max_pcgiterations);
72 toc
73 display('Number of iterations in the inexact ...
74         Newton method:')
75 display('with superlinear convergence of ...
76         forcing terms:')
77 display(k)
78 display('Value of fk:')
79 display(fk)
80 display('Number of inner iterations:')
81 display(btseq)
82 display(pcg_iter)
83
84 %Quadratic convergence
85 tic
86 [xk,fk,gradfk_norm,k,xseq,btseq,pcg_iter]=...
87     inexact_newton_method(x0,f,...
88         gradf,Hess_f,alpha0,kmax,tolgrad,c1,...
89         rho,btmax,'q',max_pcgiterations);
90 toc
91 display('Number of iterations in the inexact ...
92         Newton method:')
93 display('with quadratic convergence of ...
94         forcing terms:')
95 display(k)
96 display('Value of fk:')
97 display(fk)
98 display('Number of inner iterations:')
99 display(btseq)
100 display(pcg_iter)

```