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 $min_{x \in R^n} \sum_{i=1}^{n} (\frac{1}{4}x_i^4 + \frac{1}{2}x_i^2 + x_i)$ 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 derivates, 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 imput 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_pcgiters	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: $xk = (-0.6823, \ldots, -0.6823)^T$. 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 repeate the comparison with $n = 10^5$. Once again all the methods give the same solution $xk = (-0.6823, \ldots, -0.6823)^T$ 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 $x0 = (0,\ldots 0)^T$. We tried for other values of $x0$: even if the values of the parameters we chose to compare change, the general behavior of the methods remains the same. For example for $x0 = (-1,\cdots -1)^T$ 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 $x0 = (0,\ldots 0)^T$.

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 $x0$ 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*

```
%Create the function to be optimized
function y = function_to_be_optimized(x,n)
%
%INPUTS:
%x=column vector of length n;
%n=number of dimensions of x;
%OUTPUTS:
%y=real scalar value equal to f(x)
%
y=0;
for i=1:n
    y=y+(1/4*x(i)^4+1/2*x(i)^2+x(i));
end
end
```

B. *grad.m*

```
%Create gradient function
function gradfx=grad(x,n)
%
%INPUTS:
%x=column vector of size n;
%n=number of dimensions;
%
%OUTPUTS:
%gradf=column vector where i-th component is ...
    the partial derivative of f
%with respect to the i-th component of x.
%
gradfx=zeros(n,1);
for i=1:n
    gradfx(i)=x(i)^3+x(i)+1;
end
end
```

C. *newton_method.m*

```
%Newton method

function [xk,fk,gradfk_norm,k,xseq, ...
    btseq]=newton_method(x0,f,gradf,...
    Hess_f,alpha0,c1,kmax,tolgrad,rho,btmax)
%
%[xk,fk,gradfk_norm,k,xseq, ...
    btseq]=newton_method(x0,f,gradf,...
%Hess_f,alpha0,c1,kmax,tolgrad,rho,btmax)
%
%INPUTS:
%x0=column vector of dimension n;
%f=function R^n->R;
%gradf=gradient of f;
%Hess_f=Hessian of f;
%alpha0=the initial factor that multiplies ...
    the descent direction at each
%iteration;
%c1=factor of the Armijo condition that must ...
    be a scalar in (0,1);
%kmax=maximum number of iterations;
%tolgrad=value used as stopping criterion ...
    with respect to the norm of the
%gradient;
%rho=fixed factor, lesser than 1, used for ...
    reducing alpha0;
%btmax=maximum number of steps for updating ...
    alpha during the backtracking
%strategy.
%
%OUTPUTS:
%xk=the last x computed by the method;
%fk=f(xk);
%gradfk_norm=norm of gradf(xk);
%k=last iteration performed;
%xseq=n-by-k matrix where the columns are ...
    the xk computed during the
%iterations.
%btseq=1-by-k vector with the number of ...
    backtracking iterations at each
%optimization step.
%

%Initializations
xseq=zeros(length(x0), kmax);
btseq=zeros(1, kmax);
xk=x0;
fk=f(xk);
k=0;
gradfk_norm=norm(gradf(xk));

% Function handle for the armijo condition
f_armijo=@(fk, alpha, xk, pk) ...
    fk+c1*alpha*gradf(xk)'*pk;

while k<kmax && gradfk_norm≥tolgrad
    %Compute the descent direction as ...
        solution of
    %Hess_f(xk) p = - graf(xk)
    pk=-Hess_f(xk)\gradf(xk);
    %Reset the value of alpha
    alpha=alpha0;
    %Compute the candidate new xk
    xnew=xk+alpha*pk;
    %Compute the value of f in the candidate ...
        new xk
    fnew=f(xnew);

    bt = 0;
    %Backtracking:
```

```
    while bt<btmax && fnew>f_armijo(fk, ...
        alpha, xk, pk)
        %Reduce the value of alpha
        alpha=rho*alpha;
        %Update xnew and fnew
        xnew=xk+alpha*pk;
        fnew=f(xnew);

        bt = bt + 1;
    end

    xk=xnew;
    fk=fnew;
    gradfk_norm=norm(gradf(xk));

    %Increase the step by one
    k=k+1;

    %Store current xk in xseq
    xseq(:,k)=xk;
    %Store bt iterations in btseq
    btseq(k)=bt;
end

%"Cut" xseq and btseq to the correct size
xseq=xseq(:,1:k);
btseq=btseq(1:k);

end
```

D. *inexact_newton_method.m*

```
%Inexact Newton method
function ...
    [xk,fk,gradfk_norm,k,xseq,btseq,pcg_iter]=...
    inexact_newton_method(x0,f,gradf,Hess_f,...
    alpha0,kmax,tolgrad,c1,rho,btmax,...
    type_fterms,max_pcgiters)
%
%function [xk,fk,grafk_norm,k,xseq,btseq]=...
%   inexact_newton_method(x0,f,gradf,Hess_f,...
%   alpha0,kmax,tolgrad,c1,rho,btmax,...
%   fterms,maxpcgiters)
%
%INPUTS:
%x0=column vector of dimension n;
%f=function R^n->R;
%gradf=gradient of f;
%Hess_f=Hessian of f;
%alpha0=initial factor that multiplies the ...
    descent direction at each
%iteration;
%kmax=maximum number of iterations;
%tolgrad=value used as stopping criterion;
%c1=scalar factor of the Armijo condition, ...
    it must be in (0,1);
%rho=fixed factor used to reduce alpha, ...
    lesser than 1;
%btmax=maximum number of steps of ...
    backtracking strategy;
%type_fterms='l','s'or'q', string that ...
    specifies the type of forcing terms;
%max_pcgiters=maximum number of iterations ...
    for the pcg solver to compute
%pk.
%
%OUTPUTS:
%xk=the last x computed by the method;
%fk=f(xk);
%grafk_norm=norm of gradf(xk);
%k=last iteration;
%xseq=n-by-k matrix where the columns are ...
    the xk;
%btseq=1-by-k vector where elements are the ...
    number of backtracking
%iterations at each optimization step.
%pcg_iter=number of iterations of pcg
%

%Initializations
xseq=zeros(length(x0),kmax);
btseq=zeros(1,kmax);
pcg_iter=zeros(1,kmax);
xk=x0;
fk=f(xk);
k=0;
gradfk_norm=norm(gradf(xk));

%Function handle for the Armijo condition
f_armijo=@(fk,alpha,xk,pk) ...
    fk+c1*alpha*gradf(xk)'*pk;

while k<kmax && gradfk_norm≥tolgrad
    %Compute the descent direction as ...
        solution of
    %Hessf(xk) p = - graf(xk)
    eta_k=0;
    switch type_fterms
        case 'l'
            eta_k=0.5;
        case 's'
            eta_k=min(0.5,sqrt(norm(gradf(xk))));
        case 'q'
            eta_k=min(0.5,norm(gradf(xk)));
        otherwise
            eta_k=0.5;
    end

    %Tolerance varying with respect to ...
        forcing terms
    epsilon_k=eta_k*norm(gradf(xk));
    [pk,¬,¬,iter]=pcg(Hess_f(xk),...
        -gradf(xk),epsilon_k,max_pcgiters);
    pcg_iter(k+1)=iter;

    %Reset the value of alpha
    alpha=alpha0;

    %Compute the candidate new xk
    xnew=xk+alpha*pk;

    %Compute f in the candidate new xk
    fnew=f(xnew);

    %Use backtracking strategy
    bt=0;
    while bt<btmax && ...
        fnew>f_armijo(fk,alpha,xk,pk)
        %Reduce alpha
        alpha=rho*alpha;

        %Update xnew and fnew
        xnew=xk+alpha*pk;
        fnew=f(xnew);

        bt=bt+1;
    end

    %Update xk,fk,gradfk_norm
    xk=xnew;
    fk=fnew;
    gradfk_norm=norm(gradf(xk));

    k=k+1;

```

```
    %Store xk in xseq
    xseq(:,k)=xk;

    %Store bt in btseq
    btseq(k)=bt;
end

%Resize xseq and btseq
xseq=xseq(:,1:k);
btseq=btseq(1:k);
pcg_iter=pcg_iter(1:k);

end
```

E. U3.m

```

clear all;
clc;
%
%
%Unconstrained optimization
%3)Inexact Newton method
%
%

%Initializations
n=10^4; %We run U3 also with n=10^5
x0=zeros(n,1);
alpha=1;
kmax=100;
tolgrad=1e-12;
alpha0=alpha;
c1=1e-4;
rho=0.8;
btmax=50;
max_pcgiters=50;

%Create the function f
f=@(x) function_to_be_optimized(x,n);

%Create the gradient function
gradf=@(x) grad(x,n);

%Create the hessian function
Hess_f=@(x) diag(sparse(3.*x(:,1).^2+1));

%Run newton method
tic
[xk,fk,gradfk_norm,k,xseq,btseq]=...
    newton_method(x0,f,gradf,...
    Hess_f,alpha,kmax,...
    tolgrad,c1,rho,btmax);
toc
display('Number of iterations in the Newton ...
    method:')
display(k)
display('Value of fk:')
display(fk)
display('Number of inner iterations:')
display(btseq)


%Run inexact newton method
%Linear convergence
tic
[xk,fk,gradfk_norm,k,xseq,btseq,pcg_iter]=...
    inexact_newton_method(x0,f,...
    gradf,Hess_f,alpha0,kmax,tolgrad,c1,...
    rho,btmax,'l',max_pcgiters);
toc
display('Number of iterations in the inexact ...
    Newton method:')
display('with linear convergence of forcing ...
    terms:')
display(k)
display('Value of fk:')
display(fk)
display('Number of inner iterations:')
display(btseq)
display(pcg_iter)

%Superinear convergence
tic
[xk,fk,gradfk_norm,k,xseq,btseq,pcg_iter]=...
    inexact_newton_method(x0,f,...
    gradf,Hess_f,alpha0,kmax,tolgrad,c1,...
    rho,btmax,'s',max_pcgiters);
toc
display('Number of iterations in the inexact ...
    Newton method:')
display('with superlinear convergence of ...
    forcing terms:')
display(k)
display('Value of fk:')
display(fk)
display('Number of inner iterations:')
display(btseq)
display(pcg_iter)

%Quadratic convergence
tic
[xk,fk,gradfk_norm,k,xseq,btseq,pcg_iter]=...
    inexact_newton_method(x0,f,...
    gradf,Hess_f,alpha0,kmax,tolgrad,c1,...
    rho,btmax,'q',max_pcgiters);
toc
display('Number of iterations in the inexact ...
    Newton method:')
display('with quadratic convergence of ...
    forcing terms:')
display(k)
display('Value of fk:')
display(fk)
display('Number of inner iterations:')
display(btseq)
display(pcg_iter)
```