**Numerical Optimization**

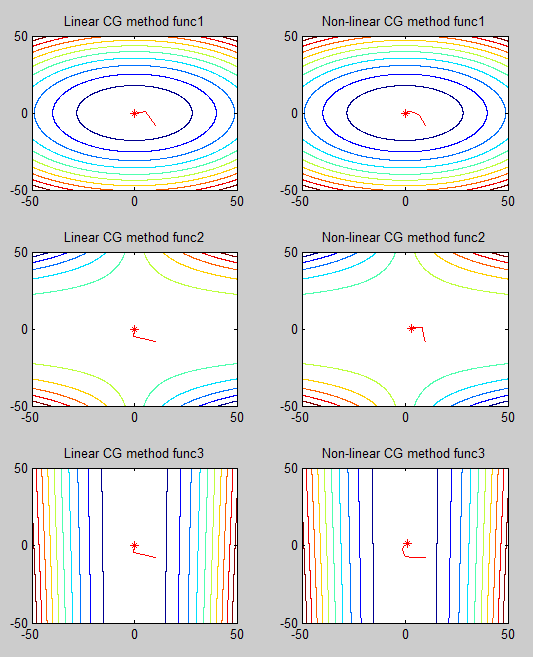
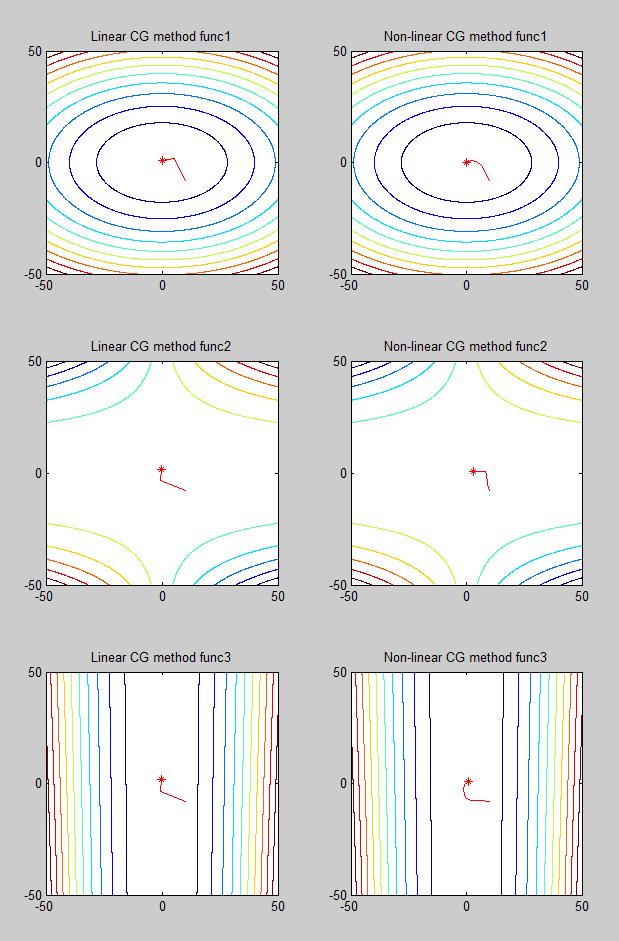
**Programming #5**

**Electrical Engineering & Computer Science**

**20171014 Young-Chul Yoon**

1. **Introduction  
   1) Goal of the assignment**- Implement linear/nonlinear Conjugate Gradient methods  
   - discuss their performance

**2) Description about implementation and setting**I was little confused about definition of A and b. There is no clear explanation, how to define A and b. I just found that matrix A has to be a positive symmetric matrix.  
I defined A the Hessian matrix as mentioned in lecture note. And if it is not satisfy the positive condition, A is substituted by pre-defined matrix [10 1;1 5] instead of Hessian matrix.  
b could be derived by solving . But, it is only applicable to problem 1. Other problems have over 2th power. So, it couldn’t be calculated by the equation above. I used 2 kind of b, one was derived from problem 1, and the other was chosen by myself.  
**1) Global setting**  
- starting point : (10, -8)  
- termination criteria : norm(r)<1.0e-3, norm(grad)<1.0e-3  
- time limit : 10 seconds **2) Linear CG method**  
- A : Hessian(f(x0))   
(If A is not positive matrix, A = [10 1;1 5])  
- b : [0 0]’(derived from problem 1), [0 8]’  
**< 3 problems >**  
1)   
2)   
3)

1. **Result in program  
   (star symbol means last point)  
   < b = [0, 0] >**  
    **< b = [0, 8] >**
2. **Analysis**

**< b = [0, 0] >**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function** | **Method** | **Point** | **Solution** | **Time** | **iter** |
|  | Linear | (0, -0) | 0 | 0.18 | 3 |
| Non-linear | (0.000114, -0.00006) | 0 | 0.299 | 14 |
|  | Linear | (-0, 0) | 14.203 | 0.026 | 3 |
| Non-linear | (2.97, 0.58) | 0.19 | 10.012 | 116 |
|  | Linear | (-0, 0) | 3 | 0.026 | 3 |
| Non-linear | (0.9998, 0.9996) | 0 | 1.79 | 46 |

**< b = [0, 8] >**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function** | **Method** | **Point** | **Solution** | **Time** | **iter** |
|  | Linear | (0, 0.8) | 3.2 | 0.1786 | 3 |
| Non-linear | - | - | - | - |
|  | Linear | (-0.163, 1.633) | 10.18 | 0.026 | 3 |
| Non-linear | - | - | - | - |
|  | Linear | (-0.163, 1.633) | 261.98 | 0.024 | 3 |
| Non-linear | - | - | - | - |

Non-linear method works well in every problem. It takes little more time than Linear method, but it is much simple and robust normally. Relatively high time consumption than Linear method is because of its process of finding step-length alpha.  
 In simple problem, like first problem, Linear method perform better than Non-linear method. When max order of polynomial equation is equal or less than 2, ‘b’ could be calculated. It is clear that b = [0, 0] works well on Linear method solving first problem. This is because that b = [0, 0] is derived from first problem. But, it doesn’t perform well in other problem. Other problem would have its own ‘b’, but, it is hard to derive because of their high order.  
In the second table(b = [0,8]), the linear CG method, applied on second problem, perform better than the case when it used b = [0, 0]. But this is also not an best ‘b’ for it.  
 In conclusion, Linear CG method could work well on low order polynomials. But, when solving high order polynomials, that Linear CG method couldn’t be applied, it is better to use Non-linear CG method.

1. **Code Implementation**

**< Setting and Main function >**

|  |
| --- |
| %% Copyright (C) 2017 Young-Chul Yoon  %% All rights reserved.  %% Conjugate Gradient methods  clc;  clear all  base = [pwd, '\'];  addpath(genpath(base));  syms x y;  %% 3 functions  f = 2\*x^2+5\*y^2;  g = (1.5-x+x\*y)^2 + (2.25-x+x\*(y^2))^2 + (2.625-x+x\*(y^3))^2;  h = 100\*(y-x^2)^2 + 3\*(1-x)^2;  f\_set = {f, g, h};  %% Global setting  start\_pt = [10 -8]';  %% 3 functions for contour plot  x = -5\*abs(max(start\_pt)):0.1:5\*abs(max(start\_pt));  y = -5\*abs(max(start\_pt)):0.1:5\*abs(max(start\_pt));  [X, Y] = meshgrid(x,y);  Z1 = 2\*X.^2+5\*Y.^2;  Z2 = (1.5-X+X.\*Y).^2 + (2.25-X+X.\*(Y.^2))^2 + (2.625-X+X.\*(Y.^3))^2;  Z3 = 100\*(Y-X.^2)^2 + 3\*(1-X).^2;  cont\_set = {Z1, Z2, Z3};  for i = 1:length(f\_set)  %% Linear CG method  subplot(3,2,2\*i-1);contour(X,Y,cont\_set{i}, 10);  title(['Linear CG method func',num2str(i)]);  [solution, t, iter] = linear\_cg(f\_set{i}, start\_pt);  fprintf('< Linear CG method >\n');  x = solution(1);  y = solution(2);  fprintf('Point : (%f, %f), Solution : %f\n', x, y, (eval(f\_set{i})));  fprintf('Time : %f\n', t);  fprintf('Iteration : %d\n', iter);  %% Non-linear CG method  subplot(3, 2 ,2\*i);contour(X,Y,cont\_set{i}, 10);  title(['Non-linear CG method func',num2str(i)]);  [solution, t, iter] = nonlinear\_cg(f\_set{i}, start\_pt);  fprintf('< Non-linear CG method >\n');  x = solution(1);  y = solution(2);  fprintf('Point : (%f, %f), Solution : %f\n', x, y, (eval(f\_set{i})));  fprintf('Time : %f\n', t);  fprintf('Iteration : %d\n', iter);  end |

**< Linear CG method>**

|  |
| --- |
| function [solution, t\_accum, iter] = linear\_cg(f, start\_pt)  %% Copyright (C) 2017 Young-Chul Yoon  %% All rights reserved.  %% Linear conjugate gradient  st = tic;  thresh = 1.0e-3;  pt = start\_pt;  %% Define A as hessian matrix  H\_val = my\_hessian(f, pt);  %% When hessian is not positive  if H\_val == 0 | check\_positive(H\_val) == false  A = [10 1;1 5];  else  A = H\_val;  end  b = [0, 8]';  r = A\*pt - b;  p = -r;  iter = 1;  pt\_set = [];  while true  pt\_set = [pt\_set, pt];  %% Stop when norm of residue is smaller than threshold or time is over  if norm(r) < thresh | toc(st) > 10  solution = pt;  t\_accum = toc(st);  break;  end  alpha = (r'\*r)/(p'\*A\*p);  pt = pt + alpha\*p;  prev\_r = r;  r = r + alpha\*A\*p;  beta = (r'\*r)/(prev\_r'\*prev\_r);  p = -r + beta\*p;  iter = iter + 1;  end  %% Draw converging process  hold on  plot(pt\_set(1, :), pt\_set(2, :), 'Color', 'r');  plot(pt\_set(1, end), pt\_set(2, end),'r\*')  hold off  end |

**< Non-linear CG method>**

|  |
| --- |
| function [solution, t\_accum, iter] = nonlinear\_cg(f, start\_pt)  %% Copyright (C) 2017 Young-Chul Yoon  %% All rights reserved.  %% Non-linear conjugate gradient  st = tic;  thresh = 1.0e-3;  pt = start\_pt;  der\_x = diff(f, 'x');  der\_y = diff(f, 'y');  grad = [der\_x, der\_y]';  x = pt(1);  y = pt(2);  p = -eval(grad);  cur\_grad = eval(grad);  iter = 1;  pt\_set = [];  while true  pt\_set = [pt\_set, pt];  %% Stop when gradient is smaller than threshold or time is over  if norm(cur\_grad) < thresh | toc(st) > 10  solution = pt;  t\_accum = toc(st);  break;  end  %% Compute alpha  alpha = find\_step\_length(f, pt, grad, p);  pt = pt + alpha\*p;  x = pt(1);  y = pt(2);  pr\_grad = cur\_grad;  %% Compute gradient  cur\_grad = eval(grad);  beta = (cur\_grad'\*cur\_grad)/(pr\_grad'\*pr\_grad);  p = -cur\_grad + beta\*p;  iter = iter + 1;  end  %% Draw converging process  hold on  plot(pt\_set(1, :), pt\_set(2, :), 'Color', 'r');  plot(pt\_set(1, end), pt\_set(2, end),'r\*')  hold off  end |

**< find step length algorithm >**

|  |
| --- |
| function [ step\_length ] = find\_step\_length(f, pt, grad, p)  %% Copyright (C) 2017 Young-Chul Yoon  %% All rights reserved.  alpha = 10;  c = 0.5;  rho = 0.5;  t\_accum = 0;  while true  temp\_st = tic;  x = pt(1);  y = pt(2);  right = eval(f) + c\*alpha\*eval(grad')\*p;  x = pt(1) + alpha\*p(1);  y = pt(2) + alpha\*p(2);  left = eval(f);    if left <= right | t\_accum > 0.5  step\_length = alpha;  return;  else  alpha = alpha\*rho;  end  t\_accum = t\_accum + toc(temp\_st);  end |