**Numerical Optimization**

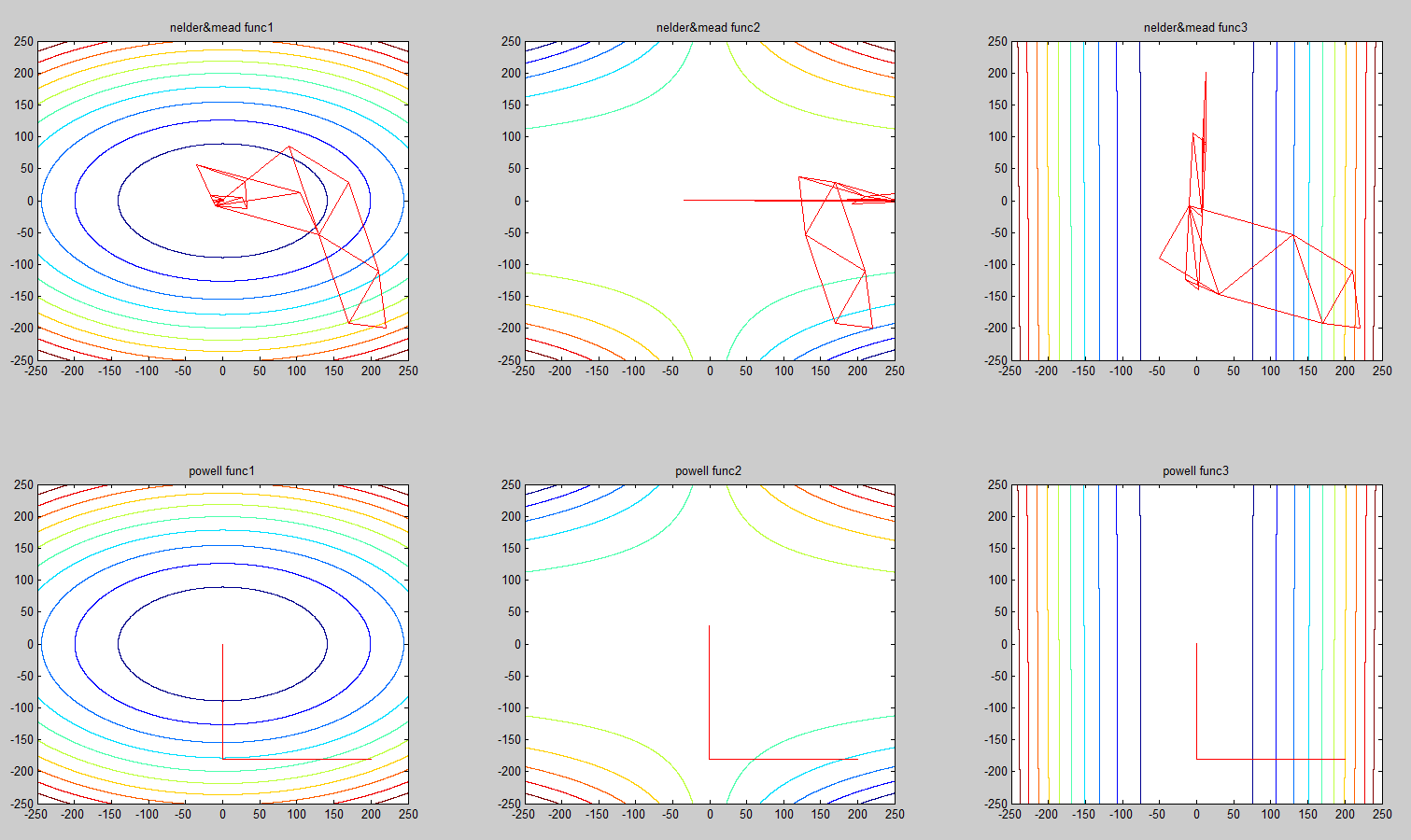
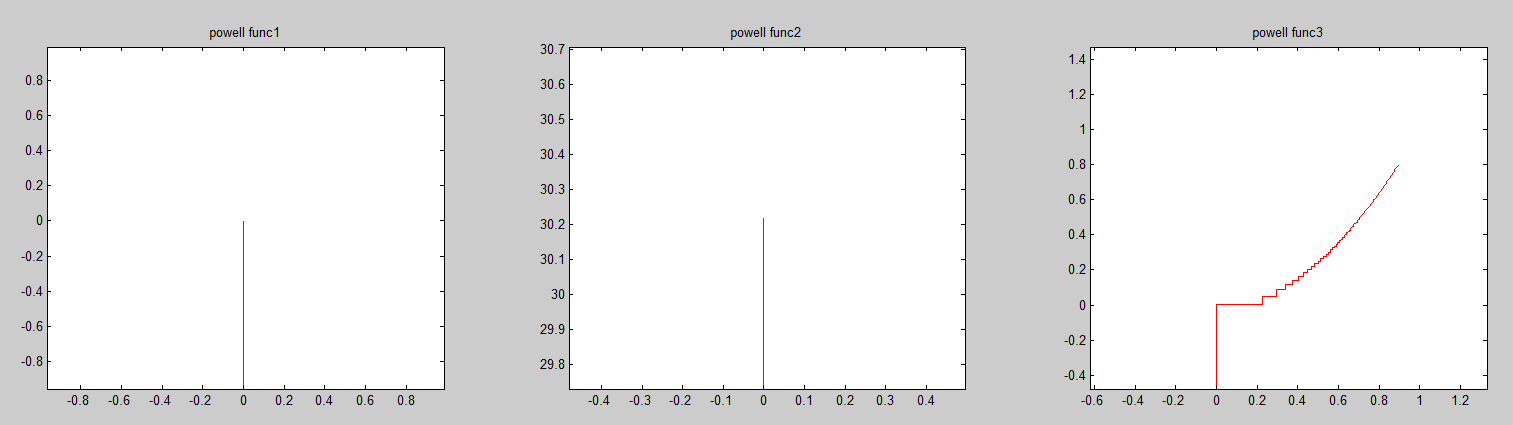
**Programming #3**

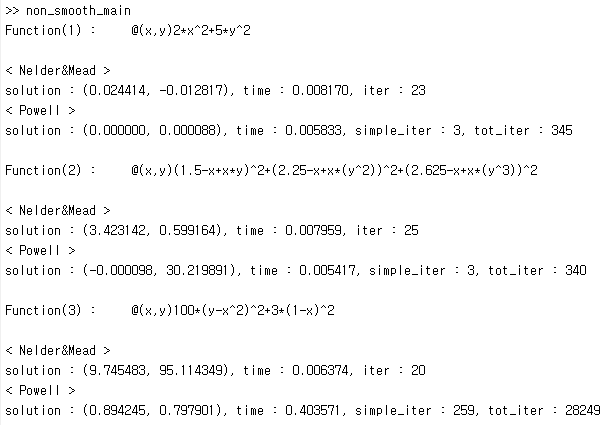
**Electrical Engineering & Computer Science**

**20171014 Young-Chul Yoon**

1. **Introduction  
   1) Goal of the assignment  
   -** Implement Nelder&Mead and Powell’s method  
   - Apply these methods to solve 3 given problems  
   - Show how the best point is moving on the contour plot of f(x,y)

**2) Description about implementation and setting**I implemented my code using MATLAB. I implemented 2 functions separately for better readability. The specific settings and implementation details are provided below. (I tried to match the initial points of two methods, even if the one is triangular points and the other is just a point)  
**1) Nelder and Mead method**  
- initial points : (210, -110), (170, -192), (220, -200)  
- alpha : 1, beta : 2, gamma : 0.5  
- stopping criterion : (area of triangle) < 1.0e-3  
**2) Powell’s method**  
- N : 2  
- Initial point : (200, -180)  
- stopping criterion : (distance of two recent points) < 1.0e-3  
- method for finding minimum step length : seeking bound & golden section  
- Calculated two kind of iteration for powell’s method. Because it has its own iteration and also internal iteration of method for finding minimum step length(Ex. Iteration of golden section). ‘simple iter’ include only powell’s iteration and **‘total iter’ include powell + internal iteration.**  
**< 3 problems >**  
1)   
2)   
3)

1. **Result in program  
   1) contour plot with converging process**  
   (Up : Nelder&Mead, Down : Powell, Horizon(left to right) : func1, func2, func3) **2) magnified plot of Powell’s method near local min-point** **3) result in number**

****

1. **Analysis**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Problem** | **Initial point** | **Method** | **Solution** | **Time(sec)** | **Iteration** |
|  | (210, -110), (170, -192), (220, -200) | Nelder and Mead | (0.024414, -0.012817) | 0.00817 | 23 |
| [200, -180] | Powell | (0.0, 0.000088) | 0.005833 | 3 |
|  | (210, -110), (170, -192), (220, -200) | Nelder and Mead | (-3.423142, 0.599164) | 0.007959 | 25 |
| [200, -180] | Powell | (0.000098, 30.219891 | 0.005417 | 3 |
|  | (210, -110), (170, -192), (220, -200) | Nelder and Mead | (9.745483, 95.114349) | 0.006374 | 20 |
| [200, -180] | Powell | (0.894245, 0.797901) | 0.403571 | 259 |

Despite, it is hard to say which one is better simply, we can compare them according to several aspects. First, Nelder and Mead method is easier to implement than Powell’s method. Because it just takes several steps iteratively without any internal optimization process. But Powell’s method need internal optimization step(in this case minimization).

In aspect of performance, Powell’s method is normally better than Nelder and Mead method. Nelder and Mead method need manually defined 3 control parameters. It’s performance is depedent on those parameters. But, Powell’s method doesn’t need it. Instead, using univariate optimization method like Golden section search, it automatically find proper parameters during process.

Powell’s time consumption and iteration is better than Nelder&Mead’s in most cases(func1, func2). But, in some cases, which converges slowly near solution(func3), Powell’s could take much time than Nelder and Mead’s. It could be found on magnified plot above, that it converges really slow near solution in func3.

1. **Code Implementation**

**< Setting and Main function >**

|  |
| --- |
| %% Copyright (C) 2017 Young-Chul Yoon  %% All rights reserved.  %% Multivariate Optimization  %% Nelder-Mead, Powell's method  %% 3 functions  f1 = @(x, y) 2\*x^2+5\*y^2;  f2 = @(x, y) (1.5-x+x\*y)^2 + (2.25-x+x\*(y^2))^2 + (2.625-x+x\*(y^3))^2;  f3 = @(x, y) 100\*(y-x^2)^2 + 3\*(1-x)^2;  f\_set = {f1, f2, f3};  %% 3 functions for contour plot  x = -250:0.1:250;  y = -250:0.1:250;  [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)  fprintf('Function(%d) : ', i);  disp(f\_set{i});  %% Nelder and Mead algorithm  subplot(2,3,i);contour(X,Y,cont\_set{i}, 10);  title(['nelder&mead func',num2str(i)]);  [sol,t\_sol,iter] = nelder\_mead(f\_set{i});  fprintf('< Nelder&Mead >\n');  fprintf('solution : (%f, %f), time : %f, iter : %d\n', sol, t\_sol, iter);  subplot(2,3,i+length(f\_set));contour(X,Y,cont\_set{i}, 10);  title(['powell func',num2str(i)]);  %% Powell's method  [sol, t\_sol, simp\_it, tot\_it] = powell(f\_set{i});  fprintf('< Powell >\n');  fprintf('solution : (%f, %f), time : %f, simple\_iter : %d, tot\_iter : %d\n\n', sol, t\_sol, simp\_it, tot\_it);  end |

**< Nelder and Mead method>**

|  |
| --- |
| function [ min\_point, t\_sol, sol\_iter ] = nelder\_mead(f)  %% Copyright (C) 2017 Young-Chul Yoon  %% All rights reserved.  %% Nelder and Mead Method  %% Initial setting  X = {{210, -110}, {170,-192}, {220,-200}};  alpha = 1;  gamma = 0.5;  beta = 2;  iter = 1;  t\_sol = 0;  while true  %% Draw triangle (Excluded from time checking)  hold on  draw\_triangle(f, X);  hold off  temp\_st = tic;  %% Stopping criterion : area of triangle  if tri\_heron(X) < 1.0e-3  sol\_iter = iter-1;  %% Traingle is sufficiently small and just return one of its point  min\_point = cell2mat(X{1});  break;  end  %% Reflection  X = sort\_func(f, X);  c = [0,0];  for i=1:length(X)-1  c = c + cell2mat(X{i});  end  c = c/(length(X)-1);  x\_r = num2cell(c + alpha\*(c - cell2mat(X{end})));  %% Continue  if f(x\_r{:}) <= f(X{end-1}{:}) & f(x\_r{:}) >= f(X{1}{:})  X{end} = x\_r;  continue;  %% Expansion  elseif f(x\_r{:}) <= f(X{1}{:})  x\_e = num2cell(c + beta\*(cell2mat(x\_r) - c));  if f(x\_e{:}) <= f(x\_r{:})  X{end} = x\_e;  else  X{end} = x\_r;  end  %% Contraction  elseif f(x\_r{:}) >= f(X{end-1}{:})  if f(x\_r{:}) < f(X{end}{:})  x\_c = num2cell(c + gamma\*(cell2mat(x\_r) - c));  else  x\_c = num2cell(c + gamma\*(cell2mat(X{end})));  end  if f(x\_c{:}) < min(f(x\_r{:}), f(X{end}{:}))  X{end} = x\_c;  else  for i=2:length(X)  X{i} = num2cell((cell2mat(X{i})+cell2mat(X{1}))/2);  end  end  end  iter = iter+1;  t\_sol = t\_sol + toc(temp\_st);  end  end |

**< Sort func >**

|  |
| --- |
| function [X] = sort\_func(f, X)  %% Copyright (C) 2017 Young-Chul Yoon  %% All rights reserved.  for i=1:length(X)-1  for j = 1:length(X)-i  if f(X{j}{:}) > f(X{j+1}{:})  temp = X{j+1};  X{j+1} = X{j};  X{j} = temp;  end  end  end  end |

**< Draw triangle >**

|  |
| --- |
| function draw\_triangle(f, X)  %% Copyright (C) 2017 Young-Chul Yoon  %% All rights reserved.  %% Warning : Added z + 1 because of visibility  t1 = cell2mat(X{1});  %t1 = [t1, f(X{1}{:})];  t2 = cell2mat(X{2});  %t2 = [t2, f(X{2}{:})];  t3 = cell2mat(X{3});  %t3 = [t3, f(X{3}{:})];  tri = [t1', t2', t3', t1'];  plot(tri(1, :), tri(2, :), 'Color', 'r');  end |

**< Powell’s method >**

|  |
| --- |
| function [ solution, t\_sol, simp\_it, tot\_it ] = powell(f)  %% Copyright (C) 2017 Young-Chul Yoon  %% All rights reserved.  %% Powell's method  %% Initial point, setting  N = 2;  X = {[200,-180]};  U = [1 0;0 1];  iter = 1;  accum\_it = 0;  t\_sol = 0;  start = tic;  while true  %% Stopping criterion : Length of two recent point  if iter > 1 & sqrt((X{iter}(1)-X{iter-1}(1))^2 + (X{iter}(2)-X{iter-1}(2))^2) < 1.0e-3  solution = X{end};  simp\_it = iter-1;  tot\_it = iter-1 + accum\_it;  break;  end    P = {};  P = [P, X(iter)];  for n = 1:N  temp\_u = U(:,n)';  %% Univariate function of gamma  new\_f = @(gamma) f(P{n}(1) + gamma\*temp\_u(1), P{n}(2) + gamma\*temp\_u(2));  %% Seeking bound algorithm  [a, b] = bound\_seeking(new\_f);  %% Golden section search  [min\_gam, t\_gol, it\_gol] = golden\_section(new\_f, a, b);  accum\_it = accum\_it + it\_gol;  new\_p = P{n} + min\_gam\*temp\_u;  P = [P, {new\_p}];  end  for n = 1:N-1  U(:,n) = U(:, n+1);  end  %% Find the best directions  U(:,N) = (P{N}-P{1})';  temp\_u = U(:,N)';  %% Univariate function of gamma  new\_f = @(gamma) f(P{1}(1) + gamma\*temp\_u(1), P{1}(2) + gamma\*temp\_u(2));  %% Seeking bound algorithm  [a, b] = bound\_seeking(new\_f);  %% Golden section search  [min\_gam, t\_gol, it\_gol] = golden\_section(new\_f, a, b);  accum\_it = accum\_it + it\_gol;  new\_x = P{1} + min\_gam\*temp\_u;  X = [X, {new\_x}];  iter = iter+1;  end  t\_sol = toc(start);  %% Draw converging process  plot\_X = [];  for i = 1:length(X)  plot\_X = [plot\_X, X{i}'];  end  hold on  plot(plot\_X(1, :), plot\_X(2, :), 'Color', 'r');  hold off  end |

* I used ‘Seeking bound’ and ‘Golden section search’ to solve univariate optimization problem in Powell’s method. These functions were implemented in last programming assignment.
* If you need to check those code, please visit my personal Github below.

**https://github.com/yyc9268/Numerical\_optimization/tree/master/matlab/univariate\_undifferentiable**