ECE580 Funwork #4

Yukun An

4/10/2015

1. Construct matrices A_1 and A_2 such that

$$(A_1 A_2)^\dagger \neq A_2^\dagger A_1^\dagger$$

Consider matrices:

$$A_1 = \begin{bmatrix} 0.5 & 1 \\ 0 & 1 \\ 2 & 0.5 \end{bmatrix} \text{ and } A_2 = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}$$

Calculate the pseudo-inverse of A_1, A_2 and $A_1 A_2$ gives:

$$(A_1 A_2)^\dagger = \begin{bmatrix} 0.0433 & 0.0346 & 0.0519 \\ 0.0433 & 0.0346 & 0.0519 \\ 0.0433 & 0.0346 & 0.0519 \end{bmatrix}$$

$$A_1^\dagger = \begin{bmatrix} -0.0513 & -0.2051 & 0.5128 \\ 0.4786 & 0.5812 & -0.1197 \end{bmatrix} \text{ and } A_2^\dagger = \begin{bmatrix} 0.0667 & 0.1333 \\ 0.0667 & 0.1333 \\ 0.0667 & 0.1333 \end{bmatrix}$$

Therefore,

$$A_2^\dagger A_1^\dagger = \begin{bmatrix} 0.0604 & 0.0638 & 0.0182 \\ 0.0604 & 0.0638 & 0.0182 \\ 0.0604 & 0.0638 & 0.0182 \end{bmatrix}$$

And so, given these A_1 and A_2,

$$(A_1 A_2)^\dagger \neq A_2^\dagger A_1^\dagger$$

2. Minimize the 2D Griewank function over the search area

$$[-5, 5] \times [-5, 5]$$

using the PSO algorithms and produce plots of the best, average, and the worst objective function values in the population for every generation.

The Griewank function in 2D has 2 variables, x_1 and x_2. So variable number is 2. The swarm population was set to 100 and the number of iterations was set to 200.

The constriction-factor version of PSO was used with $c_1 = 2$ and $c_2 = 2.1$. The initial population $x_i^{(0)}$ was generated by $-5 + 10 * uniform[0, 1]$. The initial velocity $v_i^{(0)}$ was set to 0.

The MATLAB function $'sort'$ was used to update $gbest$. At each iteration, the smallest objective function value, the first entry in the ordered set, is compare to the objective function value at $gbest$. If a better value is obtained by $x_i^{(k+1)}$, then $gbest = x_i^{(k+1)}$ where $i = index(1)$ gives the population index i that gives $gbest$.

Five different possible gbest were obtained with my settings of population size and iteration number. At these gbest values, the objective function values are all very close to 0. The gbests are:

$$gbest = \begin{bmatrix} -3.1400 \\ -4.4384 \end{bmatrix}, \begin{bmatrix} -3.1400 \\ 4.4384 \end{bmatrix}, \begin{bmatrix} 3.1400 \\ -4.4384 \end{bmatrix}, \begin{bmatrix} 3.1400 \\ 4.4384 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The folowing are two plots of possible best, average, and worst objective function value for every generation. The best value is very close to 0 for every generation. The average value converges to the best value as generations passes. The worst value is scattered with a trend of moving closer to the best value.

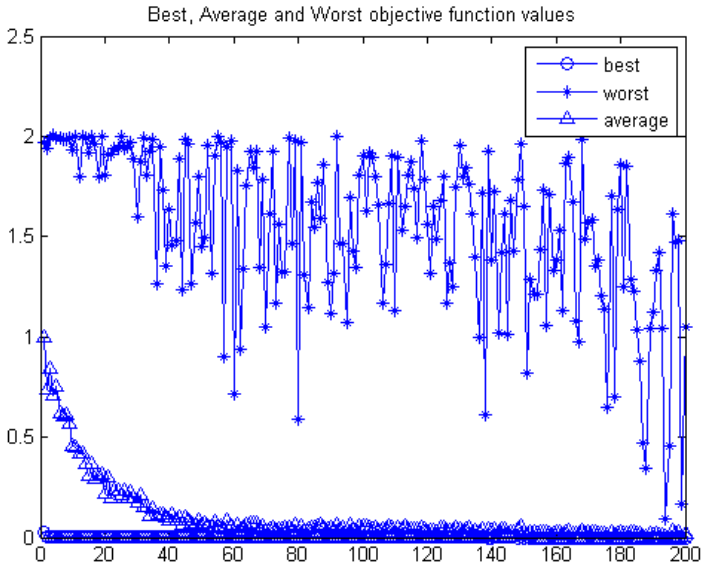

Figure 1: Best, average, worst opjective function values, First trial.

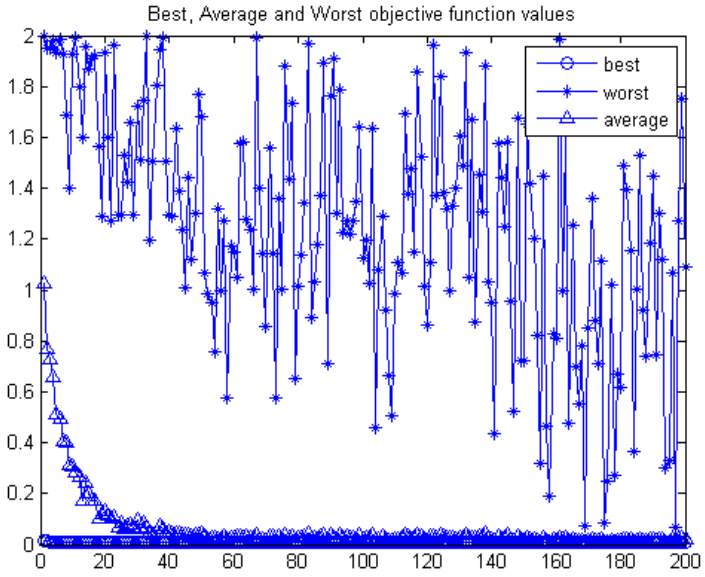

Figure 2: Best, average, worst opjective function values, Second trial.

The MATLAB program used:

```
clear all;
```

```
c1 = 2;
c2 = 2.1;
k = 0.729;
range_max = 5;  %The minimum range
range_min = -5; %The maximum range
numVar = 2; %The number of variables
numInd = 100; %Swarm Size
numIter = 200; %number of iterations
OFV_best = zeros(1,numIter);
OFV_avg = zeros(1,numIter);
OFV_worst = zeros(1,numIter);

position = range_min + (range_max-range_min).*...
rand(numInd,numVar); % Initial swarm

velocity = zeros(numInd, numVar);
pbest = position;
iter = 1;   %iteration

for n = 1:numInd
       F(n) = Griewank(position(n,1), position(n,2));
       %F at iter = 1
end
[F_ord, index] = sort(F);       %Ordered set ascending order
gbest = position(index(1),:);   %gbest = min f(x)
OFV_best(iter) = F_ord(1);
OFV_worst(iter) = F_ord(numInd);
OFV_avg(iter) = mean(F_ord);

iter = iter + 1;
for lcv = iter:numIter
   for n = 1:numInd
       r = rand(1,2);
       s = rand(1,2);
       %randomly generate r and s
       velocity(n,:) = k*(velocity(n,:) +...
       c1*r.*(pbest(n,:) - position(n,:)) + ...
       c2*s.*(gbest - position(n,:)));
```

```
        %update velocity
        position(n,:) = position(n,:) + velocity(n,:);
        %update position
        if position(n,1) > 5
            position(n,1) = 5;
        end
        if position (n,1) < -5
            position(n,1) = -5;
        end
        if position(n,2) > 5
            position(n,2) = 5;
        end
        if position(n,2) < -5
            position(n,2) = -5;
        end
        %restricting position to within range
    end
    for m = 1:numInd
        F_prev = Griewank(pbest(m,1), pbest(m,2));
        F(m) = Griewank(position(m,1), position(m,2));
        if F(m) < F_prev
            pbest(m,:) = position(m,:);
        end
    end
    %%Generating next pbest
    [F_ord, index] = sort(F);
    OFV_best(Icv) = F_ord(1);   %best OFV
    OFV_worst(Icv) = F_ord(numInd); %worst OFV
    OFV_avg(Icv) = mean(F_ord); %average OFV
    if F_ord(1) < Griewank(gbest(1),gbest(2))
        gbest = position(index(1), :);
    end
    %generate next gbest
end
disp(gbest);
n = 1:numIter;
figure(1)
plot(n, OFV_best,'-o');
```

```
hold on
plot(n, OFV_worst,'-*');
hold on
plot(n, OFV_avg,'-^');
hold off
legend best worst average
title('Best, Average and Worst objective function values')
%%%%
% Global best values:
% [-3.1400 -4.4384]
% [-3.1400  4.4384]
% [ 3.1400 -4.4384]
% [ 3.1400  4.4384]
% [ 0.0000  0.0000]
%
```

3. Maximize the 2D Griewank function over the search area

$$[-5, 5] \times [-5, 5]$$

using the PSO algorithms and produce plots of the best, average, and the worst objective function values in the population for every generation.

The objective function now became the negative of the Griewank function. Therefore all objective function value calculations were negated so the problem became finding the minimum of the negative 2D Griewank function. And in the end, the value of the best, average and the worst objective function values were negated again to take the positive values.

The settings are the same as in Problem 2 with variable number equals to 2, the swarm population equals to 100 and the number of iterations equals to 200. And the constriction-factor version of PSO was used with $c_1 = 2$ and $c_2 = 2.1$.

The constriction-factor version of PSO was used with $c_1 = 2$ and $c_2 = 2.1$. The initial population $x_i^{(0)}$ was generated by $-5 + 10 * uniform[0, 1]$. The initial velocity $v_i^{(0)}$ was set to 0.

The MATLAB function $'sort'$ was used to update $gbest$ as in Problem 2. And four different possible gbest were obtained with my settings of population size and iteration number. At these gbest values, the objective function values are all very close to 2. The gbests are:

$$gbest = \begin{bmatrix} -3.1432 \\ 0 \end{bmatrix}, \begin{bmatrix} 3.1400 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -4.4473 \end{bmatrix}, \begin{bmatrix} 0 \\ -4.4473 \end{bmatrix}$$

The folowing are two plots of possible best, average, and worst objective function value for every generation. The best value is very close to 2 for every generation. The average value converges to the best value as generations passes. The worst value is scattered with a trend of moving closer to the best value.

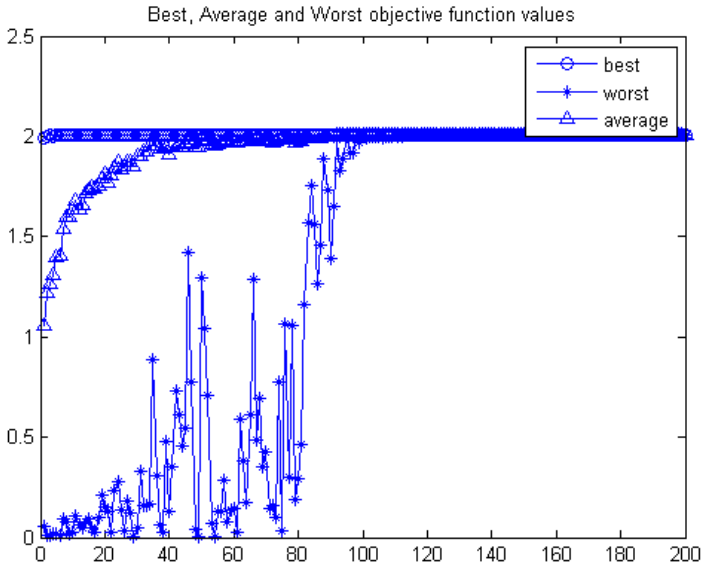

Figure 3: Best, average, worst opjective function values, First trial.

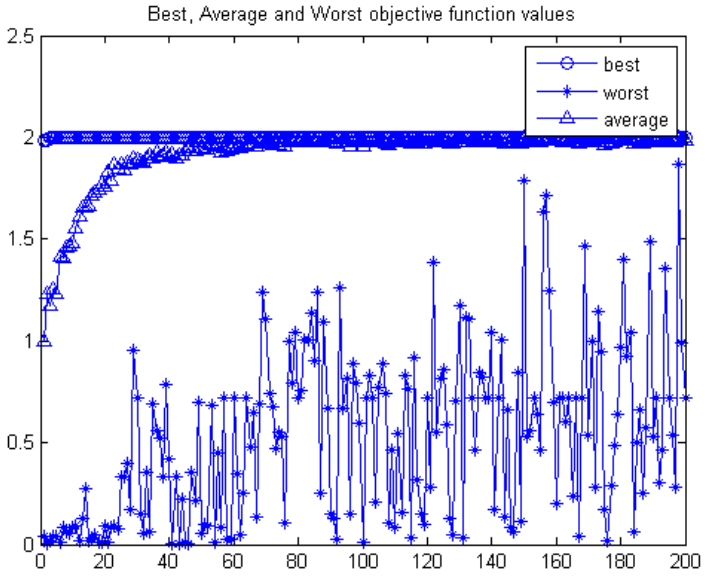

Figure 4: Best, average, worst opjective function values, Second trial.

The MATLAB program used:

```
clear all;
```

```
c1 = 2;
c2 = 2.1;
k = 0.729;
range_max = 5;  %The minimum range
range_min = -5; %The maximum range
numVar = 2; %The number of variables
numInd = 100; %Swarm Size
numIter = 200; %number of iterations
OFV_best = zeros(1,numIter);
OFV_avg = zeros(1,numIter);
OFV_worst = zeros(1,numIter);

position = range_min + (range_max-range_min).*...
rand(numInd,numVar); % Initial swarm

velocity = zeros(numInd, numVar);
pbest = position;
iter = 1;   %iteration

for n = 1:numInd
       F(n) = -Griewank(position(n,1), position(n,2));
       %F at iter = 1
end
[F_ord, index] = sort(F);       %Ordered set ascending order
gbest = position(index(1),:);   %gbest = min f(x)
OFV_best(iter) = -F_ord(1);     %best OFV
OFV_worst(iter) = -F_ord(numInd);%worst OFV
OFV_avg(iter) = -mean(F_ord);   %average OFV

iter = iter + 1;
for Icv = iter:numIter
   for n = 1:numInd
       r = rand(1,2);   %randomly generate r
       s = rand(1,2);   %randomly generate s

       velocity(n,:) = k*(velocity(n,:) + ...
       c1*r.*(pbest(n,:) - position(n,:)) + ...
       c2*s.*(gbest - position(n,:)));
```

```
        %update velocity

        position(n,:) = position(n,:) + velocity(n,:);
        %update position
        if position(n,1) > 5
            position(n,1) = 5;
        end
        if position (n,1) < -5
            position(n,1) = -5;
        end
        if position(n,2) > 5
            position(n,2) = 5;
        end
        if position(n,2) < -5
            position(n,2) = -5;
        end
        %restricting position to within range
    end
    for m = 1:numInd
        F_prev = -Griewank(pbest(m,1), pbest(m,2));
        F(m) = -Griewank(position(m,1), position(m,2));
        if F(m) < F_prev
            pbest(m,:) = position(m,:);
        end
    end
    %%Generating next pbest
    [F_ord, index] = sort(F);
    OFV_best(Icv) = -F_ord(1);   %best OFV
    OFV_worst(Icv) = -F_ord(numInd); %worst OFV
    OFV_avg(Icv) = -mean(F_ord); %average OFV
    if F_ord(1) < -Griewank(gbest(1),gbest(2))
        gbest = position(index(1), :);
    end
    %generate next gbest
end
disp(gbest);
n = 1:numIter;
figure(1)
```

```
plot(n, OFV_best,'-o');
hold on
plot(n, OFV_worst,'-*');
hold on
plot(n, OFV_avg,'-^');
hold off
legend best worst average
title('Best, Average and Worst objective function values')
%%%%
% Global best values:
% [-3.1432   0.0000]
% [ 3.1400   0.0000]
% [ 0.0000   4.4473]
% [ 0.0000  -4.4473]
```

4. Traveling Salesman Problem.
   The method described in the problem statement was used. Each city was assigned a index with $index = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$. This Index set also represents the order of the salesman's travel. The distance traveled under this index set was obtained to be compared with the best distance so far.
   Crossover in the index set was to randomly select 2 element and switch their order. For example, if the parent set was $parent = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, a possible offspring is $offspring = [1, 2, 7, 4, 5, 6, 3, 8, 9, 10]$.
   For 10 cities, there are $9! = 362880$ many possible paths. The shortest path is plotted in the plot below. The cities are indicated as a 'circle' and the line connecting the cities indicate the path of travel.

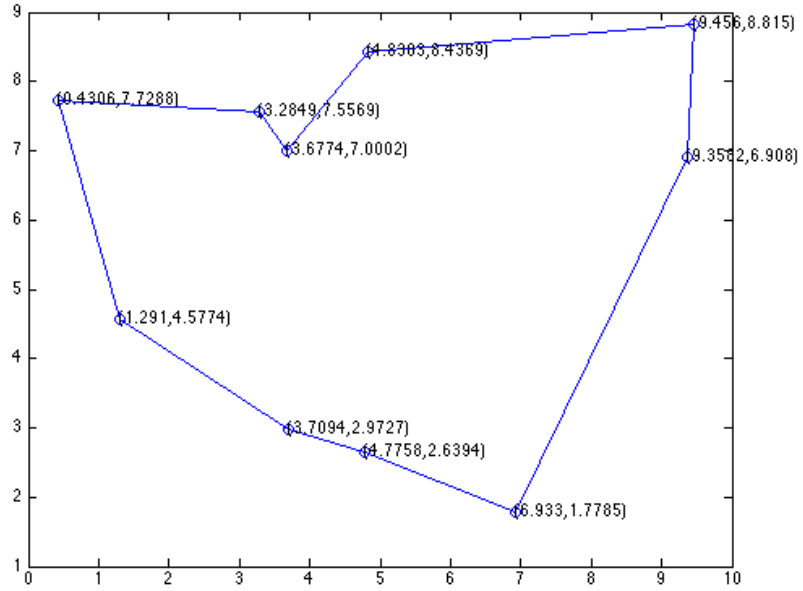

Figure 5: Shortest tour through these cities.

The MATLAB program used:

```
clear;clc;close all;
% Location of Cities
x=[0.4306 3.7094 6.9330 9.3582 4.7758 ...
    1.2910 4.83831 9.4560 3.6774 3.2849];

y=[7.7288 2.9727 1.7785 6.9080 2.6394 ...
    4.5774 8.43692 8.8150 7.0002 7.5569];
index=[1 2 3 4 5 6 7 8 9 10];

%initial distance
CityDis=distance(index,x,y);
opti=index; %Best path
N = 1000; %number of iterations

for i=1:N
    p1=floor(1+10*rand(1,1));
    p2=floor(1+10*rand(1,1));
    temp=index(1,p1);
    %generate random location
    index(1,p1)=index(1,p2);
    index(1,p2)=temp;
    %switch random location
    if distance(index,x,y) < CityDis
        opti=index;
        CityDis=distance(index,x,y);
    else
        index=opti;
    end
    %update best path
end
% plot cityies
plot(x,y,'o');
hold on;
% Connect best route
for j=1:10
    if j<10
        plot([x(index(j));x(index(j+1))],...
```

```
            [y(index(j));y(index(j+1))]);

        text(x(index(j)),y(index(j)),...
            ['(' num2str(x(index(j))) ',' ...
            num2str(y(index(j))) ')']);

    elseif j == 10
        plot([x(index(j));x(index(1))],...
            [y(index(j));y(index(1))]);

        text(x(index(j)),y(index(j)),...
            ['(' num2str(x(index(j))) ',' ...
            num2str(y(index(j))) ')'];
    end
end


function dis = distance(index,x,y)
dis = 0;
for i=2:11
    if i<11
        dis=dis+sqrt((x(index(i))-x(index(i-1)))^2+...
        (y(index(i))-y(index(i-1)))^2);

    elseif i == 11
        dis=dis+sqrt((x(index(1))-x(index(10)))^2+...
        (y(index(1))-y(index(10)))^2);

    end
end
end
```

5. Maximize f subject to given constraints.

$$f(x_1, x_2, x_3, x_4) = 6x_1 + 4x_2 + 7x_3 + 5x_4$$

equivalent to minimizing

$$-f(x_1, x_2, x_3, x_4) = -6x_1 - 4x_2 - 7x_3 - 5x_4$$

subject to

$$x_1 + 2x_2 + x_3 + 2x_4 \leq 20$$
$$6x_1 + 5x_2 + 3x_3 + 2x_4 \leq 100$$
$$3x_1 + 4x_2 + 9x_3 + 12x_4 \leq 75$$

and positive production levels.

$$x_i \geq 0, i = 1, 2, 3, 4$$

The linprog function is used:

$$[x, fval, exitflag] = linprog(f, A, b, [], [], lb);$$

where inputs are:

$$f = \begin{bmatrix} -6 \\ -4 \\ -7 \\ -5 \end{bmatrix}, A = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 6 & 5 & 3 & 2 \\ 3 & 4 & 9 & 12 \end{bmatrix}, b = \begin{bmatrix} 20 \\ 100 \\ 75 \end{bmatrix}$$

The third and fourth input are empty because there is no equality constraints. The fifth input is the lower bound on the products x_1, x_2, x_3, x_4 which are all 0.

The output is the maximizer, x, the negative of the objective function value, fval, and a exit flag which will indicate whether a solution is found.

The maximizer x is found to be:

$$x = \begin{bmatrix} 15 \\ 0 \\ 3.3333 \\ 0 \end{bmatrix}$$

The value of the objective function at the maximizer is:

$$f = 113.3333$$

The MATLAB program used:

```
%%%
% Maximize f = 6x_1 + 4x_2 + 7x_3 + 5 x_4
% equavalent to minimize -f
f = [-6;-4;-7;-5]; %Minimize the negation of objective function
A = [1,2,1,2;6,5,3,2;3,4,9,12]; %inequality coefficients
b = [20;100;75];   %Input availability
lb = zeros(4,1);  %lower bound, x >=0
[x,fval,exitflag] = linprog(f,A,b,[],[],lb);
exitflag
x
f = fval
%%%
% exitflag =
%
%      1
% Function Converged to a solution x.
%
% x =
%
%    15.0000
%     0.0000
%     3.3333
%     0.0000
% Optimal solution to maximize f
%
% f =
%
%   113.3333
% Value of the maximum.
```