Code Part

### Bump2Q.m (Bump function for question 2)

function result = bump2Q(x)

result = 0;

leng = length(x);

prod = 1;

for i = 1:leng

if (x(i) < 0 || x(i) > 10)

return;

end

prod = prod \* x(i);

end

if prod < 0.75

return;

end

num1 = 0;

num2 = 2;

denum = 0;

for i = 1:leng

num1 = num1 + (cos(x(i)))^4;

num2 = num2 \* (cos(x(i)))^2;

denum = denum + i \* (x(i))^2;

end

result = abs((num1-num2)/sqrt(denum));

### Bump3Q.m(Bump function for question 3)

function [result, violation] = bump3Q(x)

leng = length(x);

prod = 1;

for i = 1:leng

if (x(i) < 0 || x(i) > 10)

result = calcViolation(x);

violation = true;

return;

end

prod = prod \* x(i);

end

if prod < 0.75

result = calcViolation(x);

violation = true;

return;

end

num1 = 0;

num2 = 2;

denum = 0;

for i = 1:leng

num1 = num1 + (cos(x(i)))^4;

num2 = num2 \* (cos(x(i)))^2;

denum = denum + i \* (x(i))^2;

end

result = abs((num1-num2)/sqrt(denum));

violation = false;

function violationMeasure = calcViolation(x)

leng = length(x);

prod = 1;

violationMeasure = 0;

%TODO: need to normalize...how?

for i = 1:leng

ineq = 10-x(i);

if ineq < 0

violationMeasure = violationMeasure - ineq;

end

ineq = x(i);

if ineq < 0

violationMeasure = violationMeasure - ineq;

end

prod = prod \* x(i);

end

ineq = prod - 0.75;

if ineq < 0

violationMeasure = violationMeasure - ineq;

end

### crossover.m

function children = crossover(parents1, parents2, pCrossover)

children = zeros(2 \* size(parents1, 1), size(parents1, 2));

for i = 1:size(parents1,1)

% If crossover is to be performed

if (rand() < pCrossover)

crossoverPoint = floor(rand()\*(size(parents1,2)))+1;

for j = 1:size(parents1,2)

if (j <= crossoverPoint)

children(i\*2-1,j) = parents1(i,j);

children(i\*2,j) = parents2(i,j);

else

children(i\*2-1,j) = parents2(i,j);

children(i\*2,j) = parents1(i,j);

end

end

else

children(i\*2-1,:) = parents1(i, :);

children(i\*2,:) = parents2(i, :);

end

end

### fitness2Q.m(Fitness for question 2)

function result = fitness2Q(x)

result = bump2Q(x);

### fitness3Q.m(Fitness for question 3)

function [result, violation] = fitness3Q(x, fmin)

[bumpres, violation] = bump3Q(x);

if violation

result = fmin - bumpres;

else

result = bumpres;

end

### GAReal2Q.m(GA for question 2)

function eliteSln= GAReal2Q(Xinitial, popsize, maxGen, pCrossover, pMutation, V)

population = Xinitial;

fitness\_pop = zeros(popsize, 1);

fitness\_children = zeros(popsize,1);

eliteSln = zeros(1, maxGen);

for iter=1:maxGen

[parents1, parents2] = selection\_T2Q(population, popsize/2);

children = crossover(parents1, parents2, pCrossover);

children = mutation(children, pMutation, V);

% Calculate fitness for each children

for i=1:popsize

fitness\_children(i) = fitness2Q(children(i,:));

end

% Calculate fitness for every sample

for i=1:popsize

fitness\_pop(i) = fitness2Q(population(i,:));

end

[maxpVal, maxpIndex] = max(fitness\_pop);

[dummy, mincIndex] = min(fitness\_children);

% Put the answer into eliteSln

eliteSln(iter) = maxpVal;

% Pick the best ones to stay

children(mincIndex,:) = population(maxpIndex,:);

population = children;

end

### GARealQ3.m(GA for question 3, largely similar to GARealQ2, difference is only the fitness function and selection calls)

function eliteSln= GAReal3Q(Xinitial, popsize, maxGen, pCrossover, pMutation, V)

population = Xinitial;

fitness\_pop = zeros(popsize, 1);

fitness\_children = zeros(popsize,1);

eliteSln = zeros(1, maxGen);

fmin = 0.5;

for iter=1:maxGen

[parents1, parents2] = selection\_T3Q(population, popsize/2, fmin);

children = crossover(parents1, parents2, pCrossover);

children = mutation(children, pMutation, V);

% Calculate fitness for each children

for i=1:popsize

[fitness\_children(i), violation] = fitness3Q(children(i,:), fmin);

if ~violation && fitness\_children(i) < fmin

fmin = fitness\_children(i);

end

end

% Calculate fitness for every sample

for i=1:popsize

fitness\_pop(i) = fitness3Q(population(i,:), fmin);

end

[maxpVal, maxpIndex] = max(fitness\_pop);

[~, mincIndex] = min(fitness\_children);

% Put the answer into eliteSln

eliteSln(iter) = maxpVal;

% Pick the best ones to stay

children(mincIndex,:) = population(maxpIndex,:);

population = children;

end

### generatePopulation.m(Used to generate initial population randomly)

function population = generatePopulation(populationSize, n)

%Generate values from the uniform distribution on the interval [0, 10].

population = 10.\*rand(populationSize,n);

### mutation.m(for mutation functionality)

function children = mutation(children, pMutation, V)

numChildren = size(children,1);

numVar = size(children,2);

% For each child

for i = 1:numChildren

if rand() < pMutation

found = false;

while ~found

varIndex = randi(numVar,1);

tempsln = children(i, varIndex) + randn \* sqrt(V);

found = true;

%see if it's within bound

leng = length(tempsln);

prod = 1;

for ii = 1:leng

if (tempsln(ii) < 0 || tempsln(ii) > 10)

found = false;

break;

end

prod = prod \* tempsln(ii);

end

if prod < 0.75

found = false;

end

end

children(i, varIndex) = tempsln;

end

end

plotBump.m(Script to plot the bump function in 3D and its contour)

figure

resolution = 300;

xvec = linspace(-1,11,resolution);

yvec = linspace(-1,11,resolution);

[x, y] = meshgrid(xvec, yvec);

costmap = zeros(resolution,resolution);

for i = 1:resolution

for j = 1:resolution

costmap(i, j) = bump2Q([xvec(i), yvec(j)]);

end

end

surf(x, y, costmap);

figure

contour(x,y,costmap);

### q2iii.m(Used to solve question 2.iii, where it determines the best parameters)

%%% Determine the parameters.

% Constants

K = 1;

POP\_SIZE = 50;

ITERATION = 200;

% Arbitrarily define reference points to test other parameters

P\_MUTATION\_REF = 0.7;

P\_CROSSOVER\_REF = 0.2;

V\_REF = 3;

pMutationRes = zeros(11,10);

pCrossoverRes = zeros(11,10);

pVarRes = zeros(11,10);

pRandRes = zeros(11,10);

for temp = 1:10

% Generate initial population. n=?

InitPop = generatePopulation(POP\_SIZE, 20);

% Determine the pMutation

for r = 1:11

ip = generatePopulation(POP\_SIZE \* ITERATION, 20);

res = zeros(POP\_SIZE \* ITERATION, 1);

for rr = 1: POP\_SIZE \* ITERATION

res(rr) = fitness2Q(ip(rr, :));

end

pRandRes(r, temp) = max(res);

end

% Determine the pMutation

for pM = 0:0.1:1

result = GAReal2Q(InitPop, POP\_SIZE, ITERATION, P\_CROSSOVER\_REF, pM, V\_REF);

pMutationRes(floor(pM\*10)+1, temp) = result(ITERATION);

end

% Determine the pCrossover

for pC = 0:0.1:1

result = GAReal2Q(InitPop, POP\_SIZE, ITERATION, pC, P\_MUTATION\_REF, V\_REF);

pCrossoverRes(floor(pC\*10)+1, temp) = result(ITERATION);

end

% Determine the variance

for pV = 0:1:10

result = GAReal2Q(InitPop, POP\_SIZE, ITERATION, P\_CROSSOVER\_REF, P\_MUTATION\_REF, pV);

pVarRes(floor(pV)+1, temp) = result(ITERATION);

end

end

figure;

hold on

plot(1:11, mean(pRandRes, 2));

plot(1:11, max(pRandRes, [], 2));

title('Random algorithm');

xlabel('Random Point');

ylabel('Elite Solution');

figure;

hold on

plot(0:0.1:1, mean(pMutationRes, 2));

plot(0:0.1:1, max(pMutationRes, [], 2));

title('Mutation Rate vs. performance');

xlabel('Mutation Rate');

ylabel('Elite Solution');

figure;

hold on

plot(0:0.1:1, mean(pCrossoverRes, 2));

plot(0:0.1:1, max(pCrossoverRes, [], 2));

title('Crossover Rate vs. performance');

xlabel('Crossover Rate');

ylabel('Elite Solution');

figure;

hold on

plot(0:1:10, mean(pVarRes, 2));

plot(0:1:10, max(pVarRes, [], 2));

title('Variance vs. performance');

xlabel('Variance');

ylabel('Elite Solution');

### q2iv\_3ii(Used to solve question 2.iv and 3.ii):

% Constants

K = 1;

POP\_SIZE = 50;

ITERATION = 200;

P\_MUTATION = 0.7;

P\_CROSSOVER = 0.5;

V = 3;

resA = zeros(1, 20);

resultA = zeros(ITERATION, 20);

resB = zeros(1, 20);

resultB = zeros(ITERATION, 20);

for temp = 1:20

% Generate initial population. n=?

InitPop = generatePopulation(POP\_SIZE, 20);

resultA(:, temp) = GAReal2Q(InitPop, POP\_SIZE, ITERATION, P\_CROSSOVER, P\_MUTATION, V);

resA(temp) = resultA(ITERATION, temp);

resultB(:, temp) = GAReal3Q(InitPop, POP\_SIZE, ITERATION, P\_CROSSOVER, P\_MUTATION, V);

resB(temp) = resultB(ITERATION, temp);

end

fprintf('2iv:\n');

fprintf('%f is the average and %f is the standard deviation ', mean(resA), std(resA));

fprintf('of the fitness of the fittest member of the population (elite solution) over the 20 trials.\n');

fprintf('%f is the best and %f is the worst elite solution ', max(resA), min(resA));

fprintf('from the 20 trials after 200 generations.\n\n');

fprintf('3b:\n');

fprintf('%f is the average and %f is the standard deviation ', mean(resB), std(resB));

fprintf('of the fitness of the fittest member of the population (elite solution) over the 20 trials.\n');

fprintf('%f is the best and %f is the worst elite solution ', max(resA), min(resA));

fprintf('from the 20 trials after 200 generations.\n');

figure;

hold on

plot(2\*POP\_SIZE : 2\*POP\_SIZE : 2\*POP\_SIZE\*ITERATION, mean(resultA, 2), 'r');

plot(2\*POP\_SIZE : 2\*POP\_SIZE : 2\*POP\_SIZE\*ITERATION, mean(resultB, 2), 'g:');

title('Average elite solution vs. function evaluations');

xlabel('Function Evaluations');

ylabel('Average Elite Solution');

### selection\_T2Q.m(Tournament selection for question 2)

% selects numPairs pairs of parents from population using fitness

% proportional selection

% returns two parent matrix of size numPair x num\_of\_variables

function [parent1, parent2] = selection\_T2Q(population, numPairs)

sizePopulation = size(population, 1);

%cache fitness so we don't have to recalculate

fitnessCache = zeros(sizePopulation, 1);

for i = 1: sizePopulation

fitnessCache(i) = fitness2Q(population(i, :));

end

parent1 = zeros(numPairs, size(population, 2));

parent2 = zeros(numPairs, size(population, 2));

% for each pair

for i = 1 : numPairs

r = randperm(sizePopulation);

if fitnessCache(r(1)) < fitnessCache(r(2))

parent1(i,:) = population(r(2), :);

else

parent1(i,:) = population(r(1), :);

end

if fitnessCache(r(3)) < fitnessCache(r(4))

parent2(i,:) = population(r(4), :);

else

parent2(i,:) = population(r(3), :);

end

end

### selection\_T3Q.m(Tournament selection for question 3):

% selects numPairs pairs of parents from population using fitness

% proportional selection

% returns two parent matrix of size numPair x num\_of\_variables

function [parent1, parent2] = selection\_T3Q(population, numPairs, fmin)

sizePopulation = size(population, 1);

%cache fitness so we don't have to recalculate

fitnessCache = zeros(sizePopulation, 1);

for i = 1: sizePopulation

fitnessCache(i) = fitness3Q(population(i, :), fmin);

end

parent1 = zeros(numPairs, size(population, 2));

parent2 = zeros(numPairs, size(population, 2));

% for each pair

for i = 1 : numPairs

r = randperm(sizePopulation);

if fitnessCache(r(1)) < fitnessCache(r(2))

parent1(i,:) = population(r(2), :);

else

parent1(i,:) = population(r(1), :);

end

if fitnessCache(r(3)) < fitnessCache(r(4))

parent2(i,:) = population(r(4), :);

else

parent2(i,:) = population(r(3), :);

end

end