The equation represented by the Matlab ‘peaks’ function was recreated to include a Goldberg transformation seen below as a +7 offset in the z-axis. Because we are trying to maximize the profit function, this transformation was necessary to remove the negative values present in the default ‘peaks’ function. If this is not done, the algorithm will create negative fitness values leading it to prematurely exit.

% Tim Cardenuto – EN.525.770 – Module 9

clc

clear

rand('state',0); % edit this to get different results every time

x=-3:0.188:3; % the resolution here doesn't seem to make a difference

y=-3:0.188:3; % used 32 values to match the resolution of 'peaks'

z = zeros(2,length(x));

for i=1:length(x)

for j=1:length(y)

z(j,i) = 3\*((1-x(i))^2)\*exp((-x(i)^2)-((y(j)+1)^2)) - 10\*((x(i)/5)-(x(i)^3)-(y(j)^5))\*exp((-x(i)^2)-(y(j)^2)) - (1/3)\*exp((-(x(i)+1)^2)-(y(j)^2)) + 7;

end

end

mesh(x,y,z)

xlabel('x');

ylabel('y');

zlabel('z');

title('Fitness Function to Maximize');

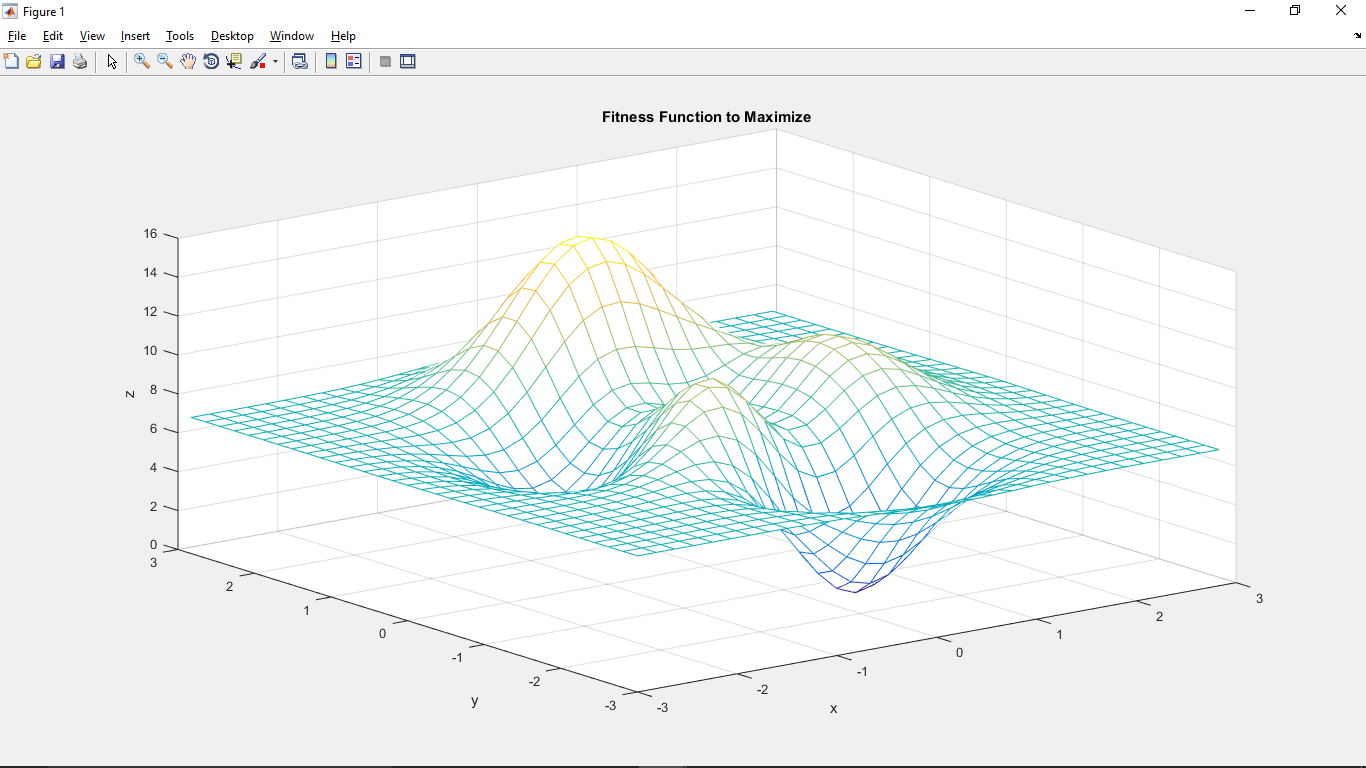
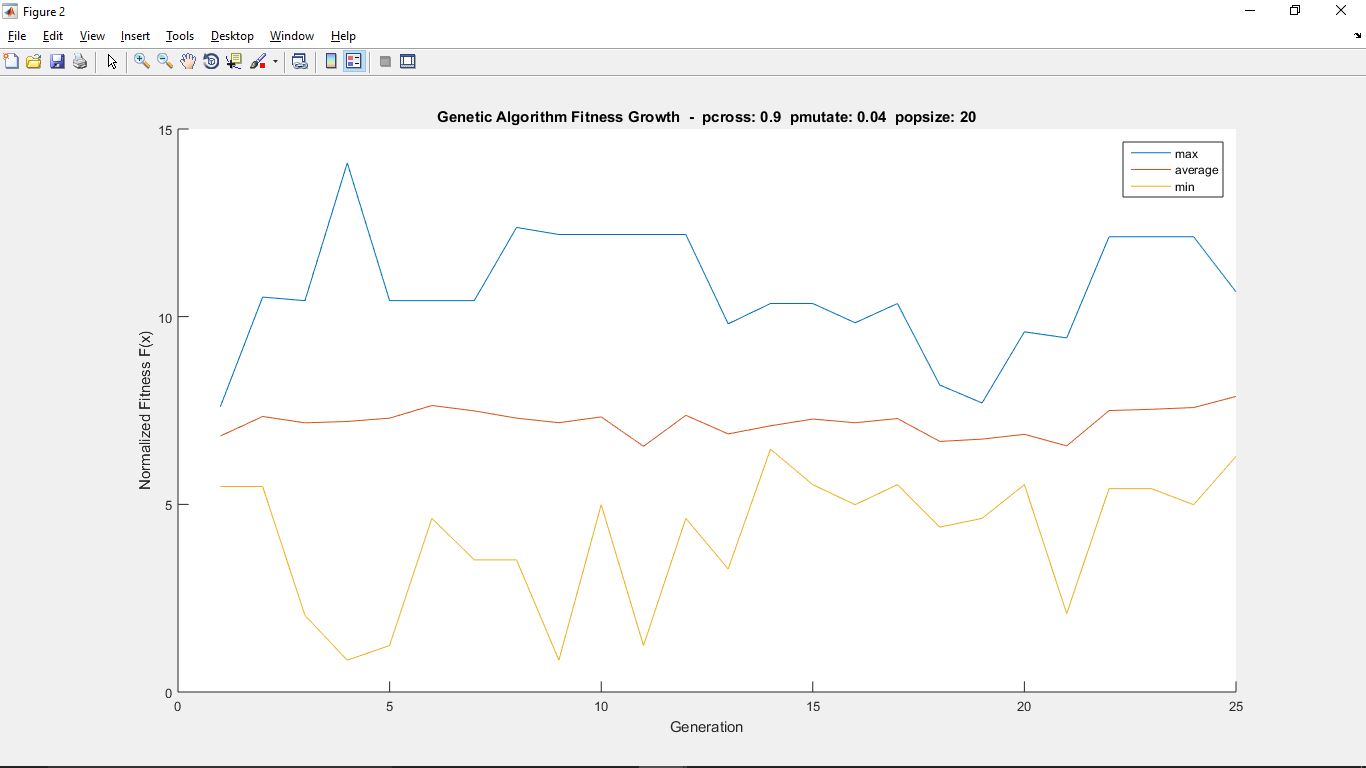
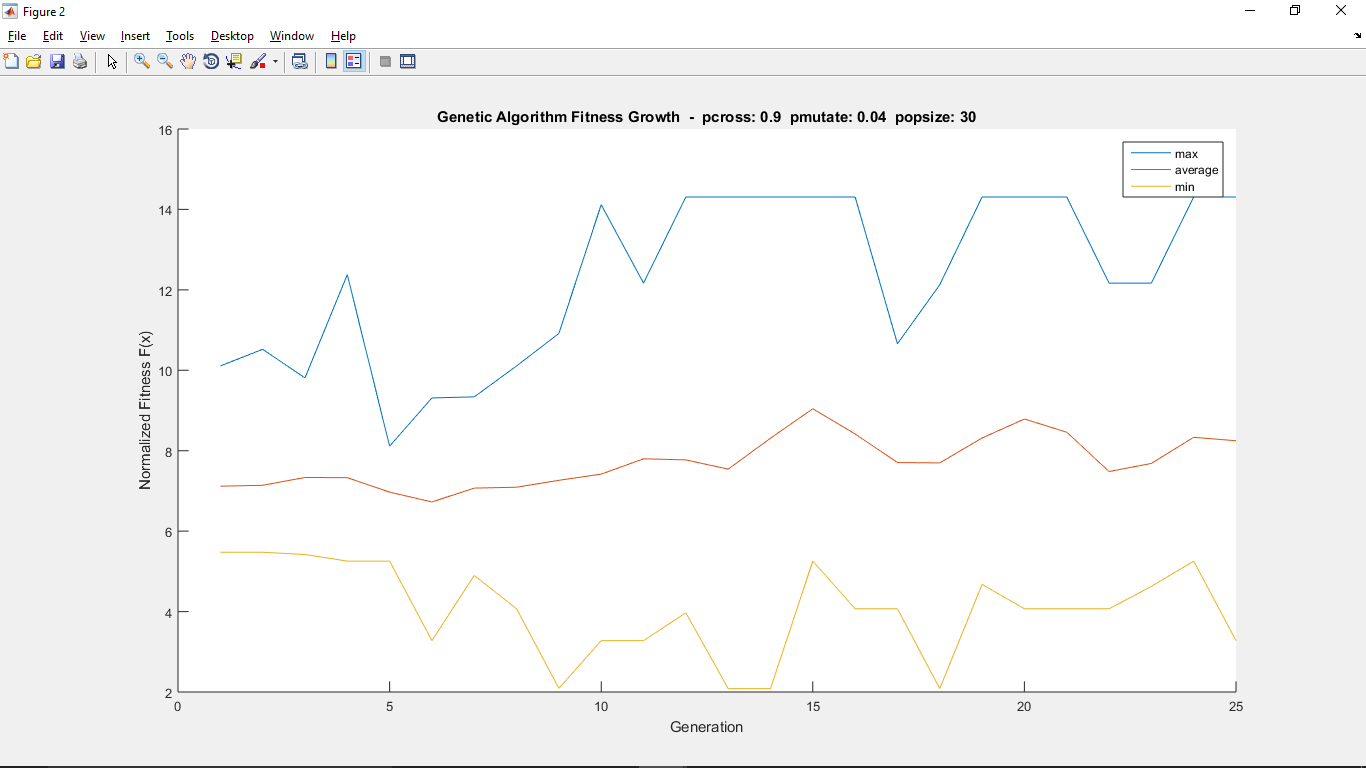


Figure 1 – Fitness function to maximize with Goldberg transformation

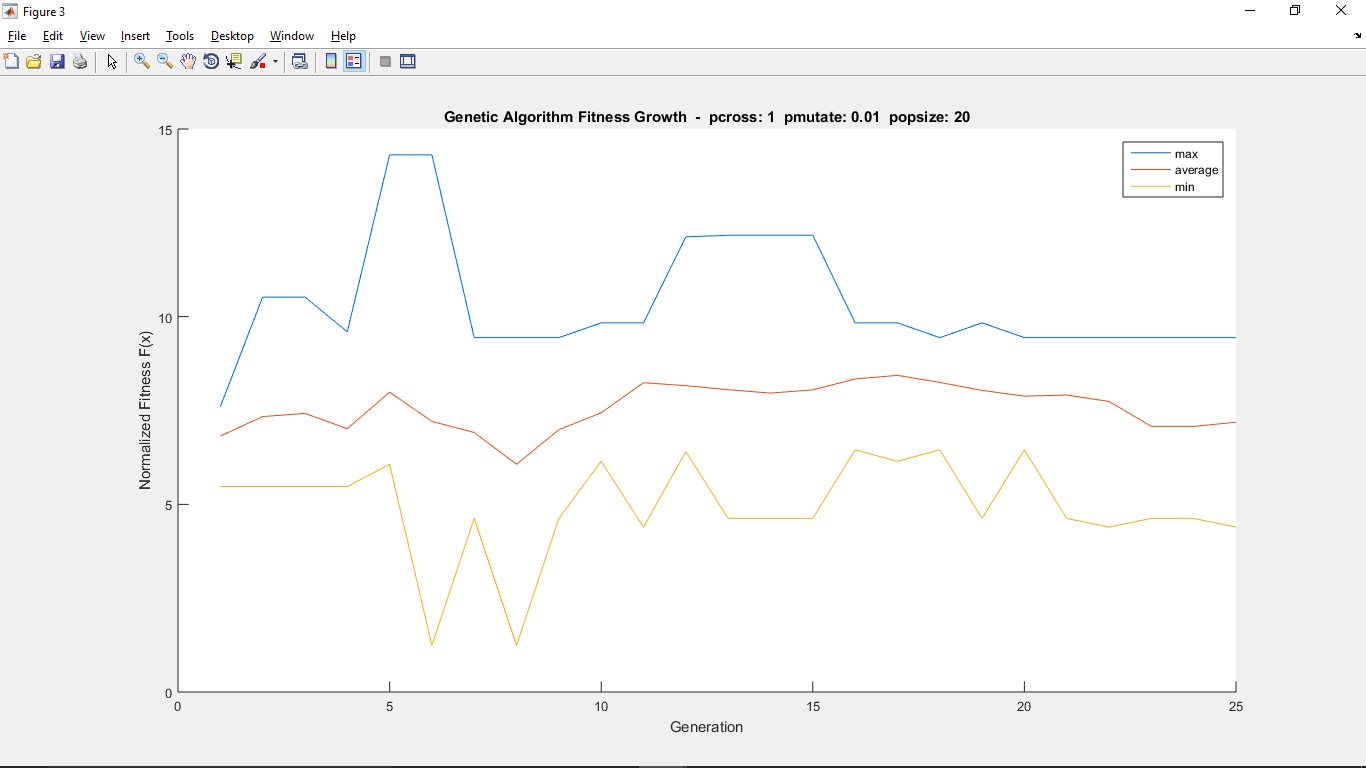
The following charts show the max, average, and min fitness timelines for given pcross, pmutate, and population size shown in the chart title. The total overall max, average, and min for each scenario is also provided. Note that the fitness values in the chart still incorporate the +7 transformation (an average of ~7 is actually an average ~0). This transform was removed from the table values.



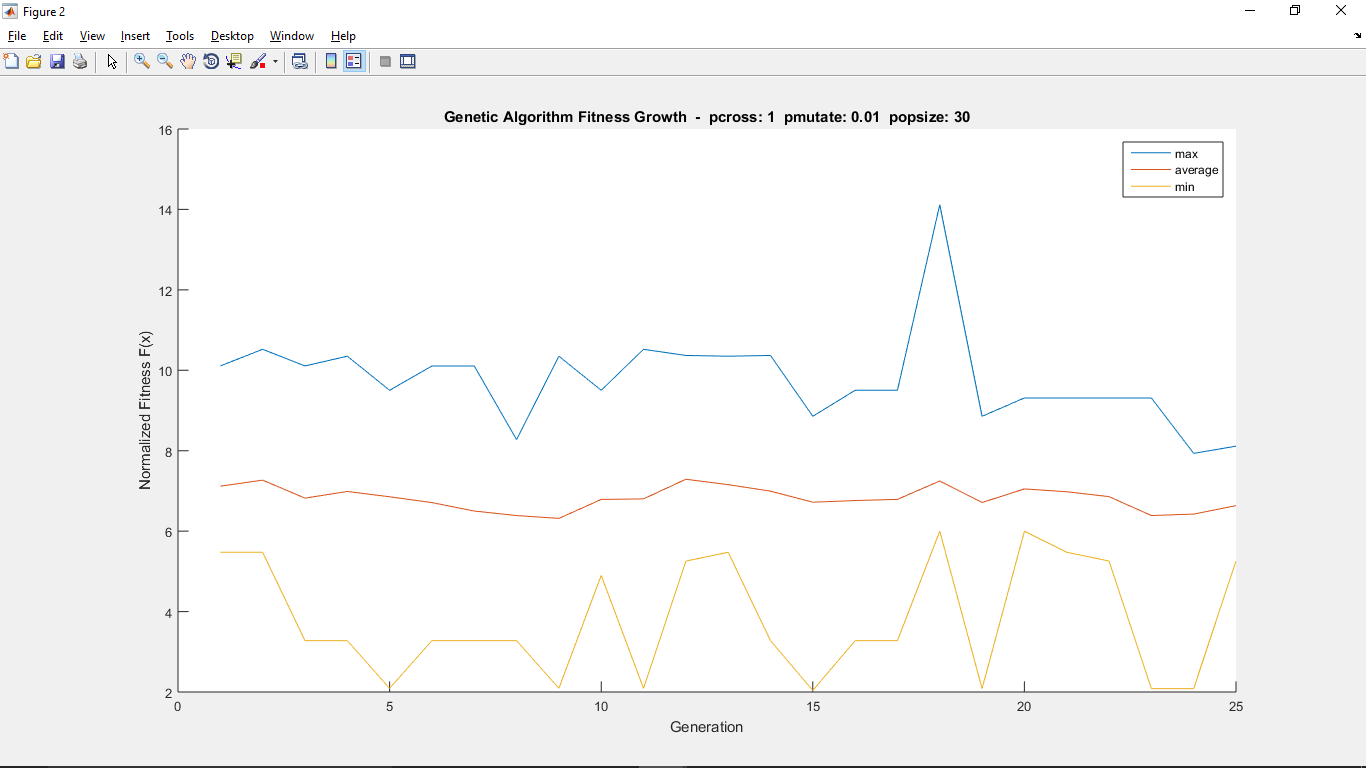
|  |  |  |
| --- | --- | --- |
| **maximum** | **average** | **minimum** |
| 7.0957 | 0.1879 | -6.1525 |



|  |  |  |
| --- | --- | --- |
| **maximum** | **average** | **minimum** |
| 7.3103 | 0.7225 | -4.9164 |



|  |  |  |
| --- | --- | --- |
| **maximum** | **average** | **minimum** |
| 7.3103 | 0.5842 | -5.7611 |



|  |  |  |
| --- | --- | --- |
| **maximum** | **average** | **minimum** |
| 7.1169 | -0.1776 | -4.9612 |

Continued from page 1 code.

%% Genetic Algorithm

max\_generation = 25;

length\_chromosome = 8;

population\_size = 20;

pcross = 1.0;

pmutate = 0.01;

% set initial population &

pop = zeros(population\_size, length\_chromosome);

x\_decode = zeros(1, population\_size);

y\_decode = zeros(1, population\_size);

x\_mapped = zeros(1, population\_size);

y\_mapped = zeros(1, population\_size);

fitness = zeros(1, population\_size);

for ii=1:population\_size,

for jj=1:length\_chromosome,

pop(ii,jj) = (rand <= 0.5); % generate chromosome bit

end;

% check that chromosome can be decomposed into substrings

if (mod(length\_chromosome,2)),

disp('error, cant evenly derive substrings from chromosome')

break;

end;

% decode chromosome substring x into decimal number

x\_substring = pop(ii,1:length\_chromosome/2);

accum = 0;

powerof2 = 1;

for jj=length(x\_substring):-1:1,

if ( x\_substring(jj) ),

accum = accum + powerof2;

end

powerof2 = 2\*powerof2;

end

x\_decode(ii) = accum;

% decode chromosome substring y into decimal number

y\_substring = pop(ii,length\_chromosome/2+1:length\_chromosome);

accum = 0;

powerof2 = 1;

for jj=length(y\_substring):-1:1,

if ( y\_substring(jj) ),

accum = accum + powerof2;

end

powerof2 = 2\*powerof2;

end

y\_decode(ii) = accum;

% map decoded substring x to range [-3,3]

x\_mapped(ii) = ((3-(-3)) / ((2^4)-1)) \* x\_decode(ii) + (-3);

if (x\_mapped(ii) < -3 | x\_mapped(ii) > 3)

disp('error, x\_mapped out of bounds')

end

% map decoded substring y to range [-3,3]

y\_mapped(ii) = ((3-(-3)) / ((2^4)-1)) \* y\_decode(ii) + (-3);

if (y\_mapped(ii) < -3 | y\_mapped(ii) > 3)

disp('error, y\_mapped out of bounds')

end

% determine fitness from objective function

fitness(ii) = objectivefunc(x\_mapped(ii),y\_mapped(ii));

end;

sum\_fitness = sum(fitness);

average\_fitness = sum\_fitness/population\_size;

max\_fitness = max(fitness);

min\_fitness = min(fitness);

% ga optimization loop

for kk=2:max\_generation

% select mating pool M(i) from population for offspring reproduction

parents = zeros(1, population\_size);

ii = 1;

while (ii <= population\_size),

% select 1st parent

jj = 0;

partial\_sum = 0;

random\_variable = rand \* sum\_fitness;

while ((partial\_sum <= random\_variable) & (jj < population\_size)),

jj = jj + 1;

partial\_sum = partial\_sum + fitness(jj);

end;

parents(ii) = jj;

% select 2nd parent

jj = 0;

partial\_sum = 0;

random\_variable = rand \* sum\_fitness;

while ((partial\_sum <= random\_variable) & (jj < population\_size)),

jj = jj + 1;

partial\_sum = partial\_sum + fitness(jj);

end;

parents(ii+1) = jj;

% doing all these 2 children at a time inside population\_size loop

if (rand <= pcross)

cross\_point = randi([1,(length\_chromosome-1)],1,1);

else

cross\_point = length\_chromosome;

end

% M(i) mutate - perturb the mated population stochastically

for bb=1:cross\_point

if (rand <= pmutate)

child(ii,bb) = ~pop(parents(ii),bb);

else

child(ii,bb) = pop(parents(ii),bb);

end

if (rand <= pmutate)

child(ii+1,bb) = ~pop(parents(ii+1),bb);

else

child(ii+1,bb) = pop(parents(ii+1),bb);

end

end

% M(i) crossover - recombine genes of selected parents

for jj=cross\_point+1:length\_chromosome

if (rand <= pmutate)

child(ii,jj) = ~pop(parents(ii+1),jj);

else

child(ii,jj) = pop(parents(ii+1),jj);

end

if (rand <= pmutate)

child(ii+1,jj) = ~pop(parents(ii),jj);

else

child(ii+1,jj) = pop(parents(ii),jj);

end

end

% decode child 1 chromosome x substring into decimal number

x\_substring(1,:) = child(ii,1:length\_chromosome/2);

accum = 0;

powerof2 = 1;

for jj=length(x\_substring(1,:)):-1:1,

if ( x\_substring(1,jj) ),

accum = accum + powerof2;

end

powerof2 = 2\*powerof2;

end

x\_decode(ii) = accum;

% decode child 1 chromosome y substring into decimal number

y\_substring(1,:) = child(ii,length\_chromosome/2+1:length\_chromosome);

accum = 0;

powerof2 = 1;

for jj=length(y\_substring(1,:)):-1:1,

if ( y\_substring(1,jj) ),

accum = accum + powerof2;

end

powerof2 = 2\*powerof2;

end

y\_decode(ii) = accum;

% decode child 2 chromosome x substring into decimal number

x\_substring(2,:) = child(ii+1,1:length\_chromosome/2);

accum = 0;

powerof2 = 1;

for jj=length(x\_substring(2,:)):-1:1,

if ( x\_substring(2,jj) ),

accum = accum + powerof2;

end

powerof2 = 2\*powerof2;

end

x\_decode(ii+1) = accum;

% decode child 2 chromosome y substring into decimal number

y\_substring(2,:) = child(ii+1,length\_chromosome/2+1:length\_chromosome);

accum = 0;

powerof2 = 1;

for jj=length(y\_substring(2,:)):-1:1,

if ( y\_substring(2,jj) ),

accum = accum + powerof2;

end

powerof2 = 2\*powerof2;

end

y\_decode(ii+1) = accum;

% map decoded substring x to range [-3,3]

x\_mapped(ii) = ((3-(-3)) / ((2^4)-1)) \* x\_decode(ii) + (-3); % child 1

if (x\_mapped(ii) < -3 | x\_mapped(ii) > 3)

disp('error, x\_mapped out of bounds')

end

x\_mapped(ii+1) = ((3-(-3)) / ((2^4)-1)) \* x\_decode(ii+1) + (-3); % child 2

if (x\_mapped(ii+1) < -3 | x\_mapped(ii+1) > 3)

disp('error, x\_mapped out of bounds')

end

% map decoded substring y to range [-3,3]

y\_mapped(ii) = ((3-(-3)) / ((2^4)-1)) \* y\_decode(ii) + (-3); % child 1

if (y\_mapped(ii) < -3 | y\_mapped(ii) > 3)

disp('error, y\_mapped out of bounds')

end

y\_mapped(ii+1) = ((3-(-3)) / ((2^4)-1)) \* y\_decode(ii+1) + (-3); % child 2

if (y\_mapped(ii+1) < -3 | y\_mapped(ii+1) > 3)

disp('error, y\_mapped out of bounds')

end

% determine fitness from objective function

fitness(ii) = objectivefunc(x\_mapped(ii),y\_mapped(ii));

fitness(ii+1) = objectivefunc(x\_mapped(ii+1),y\_mapped(ii+1));

ii = ii + 2; % create children in pairs

end;

pop = child;

sum\_fitness = sum(fitness);

max\_fitness = [max\_fitness max(fitness)];

average\_fitness = [average\_fitness sum\_fitness/population\_size];

min\_fitness = [min\_fitness min(fitness)];

end

%% Results

maximum = max(max\_fitness) - 7

average = sum(average\_fitness)/length(average\_fitness) - 7

minimum = min(min\_fitness) - 7

%% Plot stuff

figure()

hold on

plot(max\_fitness)

plot(average\_fitness)

plot(min\_fitness)

xlabel('Generation');

ylabel('Normalized Fitness F(x)');

title(['Genetic Algorithm Fitness Growth - pcross: ' num2str(pcross) ' pmutate: ' num2str(pmutate) ' popsize: ' num2str(population\_size)]);

legend('max','average','min');

Separate .m file with the fitness function.

function fitness = objectivefunc (x,y)

% single x,y value pair

% compute objective (fitness) function value

% + 7 at the end is really important, b/c the function has a large portion

% that is negative, if you don't raise the whole thing into the positive

% realm then you'll get negative fitness sums and the algorithm won't work

fitness = 3\*((1-x)^2)\*exp((-x^2)-((y+1)^2)) - 10\*((x/5)-(x^3)-(y^5))\*exp((-x^2)-(y^2)) - (1/3)\*exp((-(x+1)^2)-(y^2)) + 7;