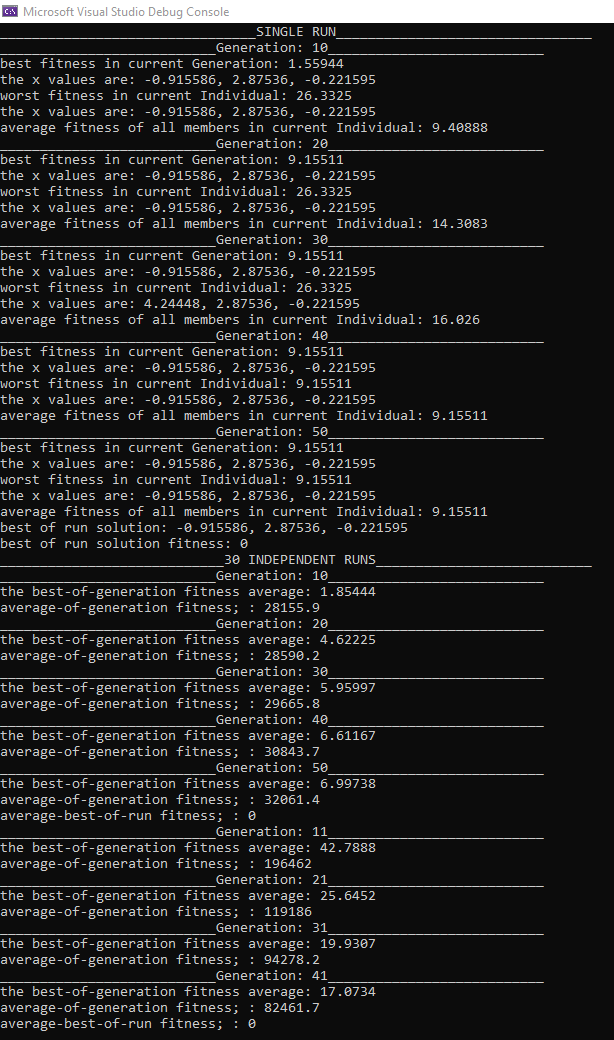
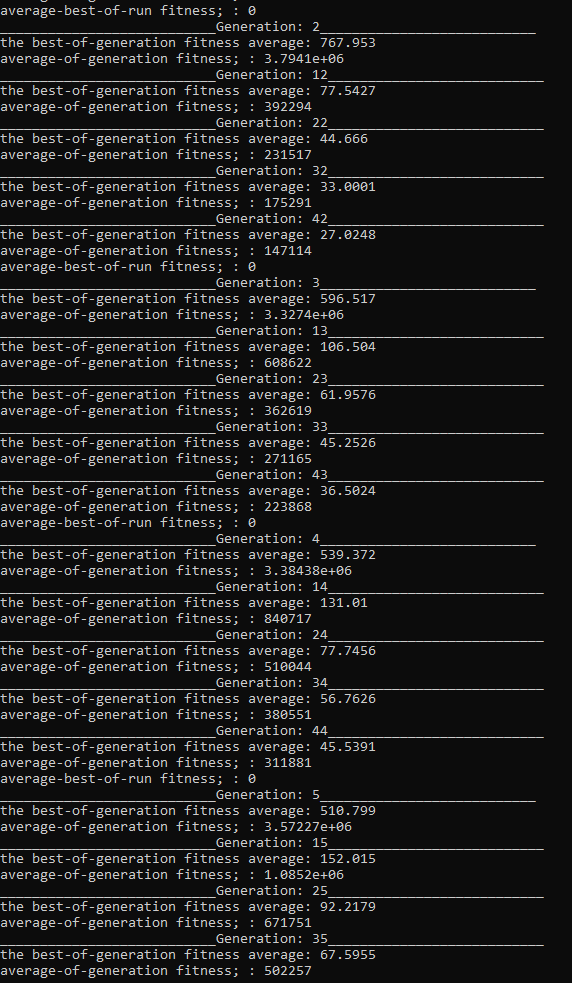
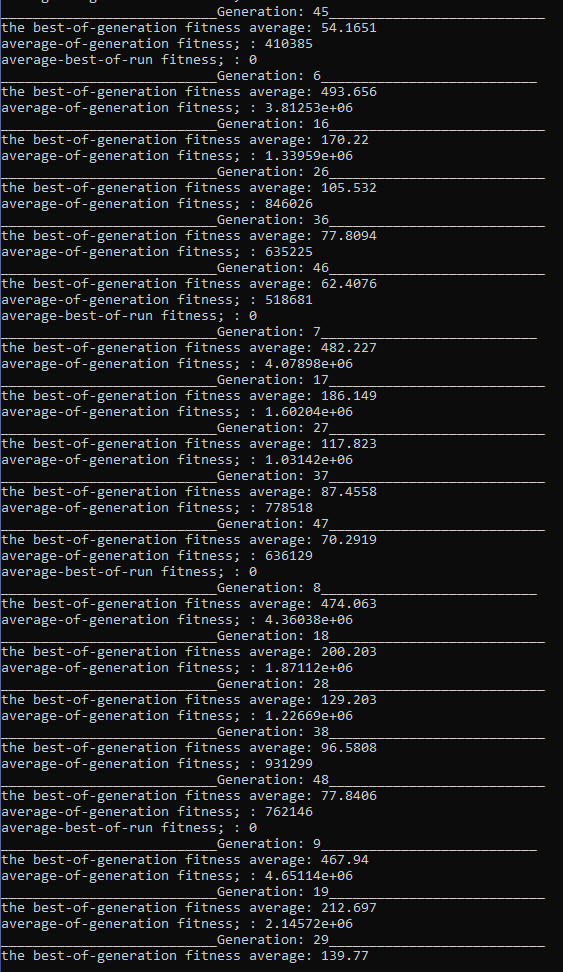
**Notes**

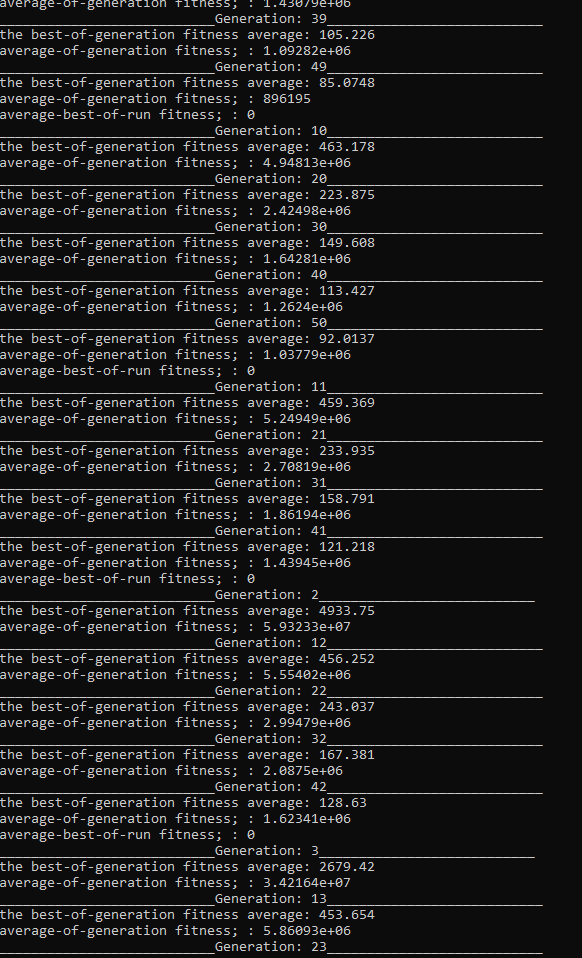
Unfortunately, I ran out of time. The singular run works fine but when I try to do 30 runs it gets messed up. I thought I had a good understanding of the process, but then in your email you said ” the u value is to be compared against the cumulative p\_i's (partial sums of p\_i's), not the p\_i's.”. So I might not be sure anymore. Here is what I thought about the project

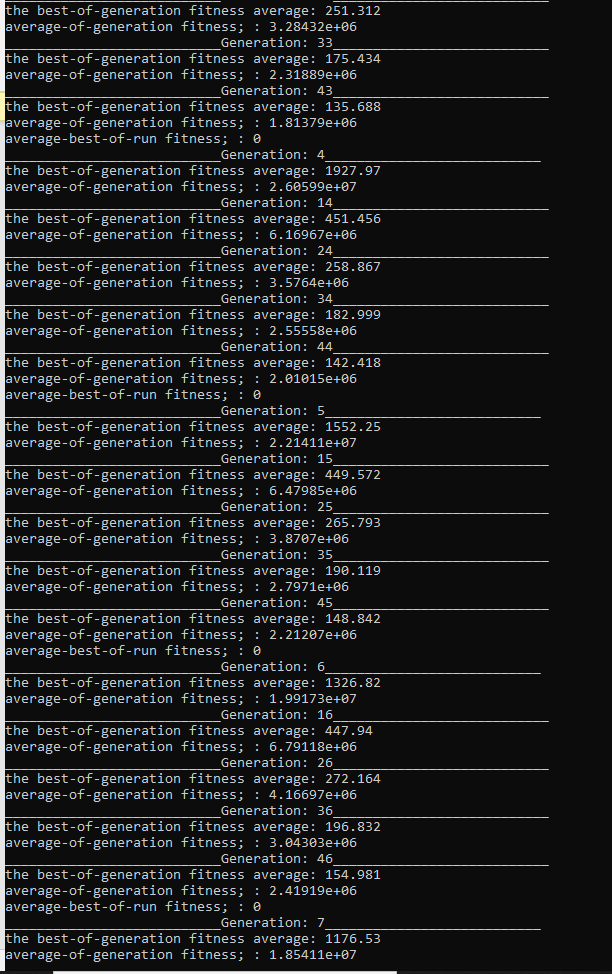
First a sample population is created. In this case 30 individuals are created. Each individual has 3 member variables which are x1, x2, x3. Every member variable is given a random number from -1.0 to 5.0. After filling the sample population, you then get the f value for every individual. In my case I stored it into an array which corresponds to the index of the individual. Then Pi values are calculated by getting the total sum of f’s in the generation which is also stored in an array corresponding to the individual. At this point proportional selection happens. A random number is generated from 0 to 1, I call this U. U decides whether an individual is selected. To do this, all pi values are compared to U. If a pi has a lower value than U, then the individual is selected. In my case when an individual Is selected it is stored in an array. The selection population has to be 30 so any individuals that didn’t get selected will have another chance. After going through the sample population and not having enough for the selection population, the selection process is repeated again until the selection population is filled. After it’s been filled, single-point crossover is utilized using the sample population. I created 2 children from 2 parents. There are only 2 crossover points because there are only 3 members to an individual. For every 2 parents I roll a number 1-2 and it decides where the crossover points are. I also roll a number from 1-10 to indicate the 80% of a crossover. If it is from 1-8 I then switch member values based on the crossover points and if it’s not then I do nothing and move on to the next two parents. Then I move on to gene-wise mutation which has a 10% chance to mutate one of the member variables. I generate a random number from 1-10 and if it is a 1 then it will mutate the member variable(gene). I do this to all member variables. If a gene needs to be mutated, a number from 1-2 is generated and that determines whether I mutate the gene by adding the delta or subtracting the delta. After all this, the new population is the new generation. And I keep doing this until the maximum amount of generations which is 50. Hopefully this summary and the comments I put in the project help you understand what I’m trying to do. I included my whole project because my .cpp file didn’t paste well in the word document.

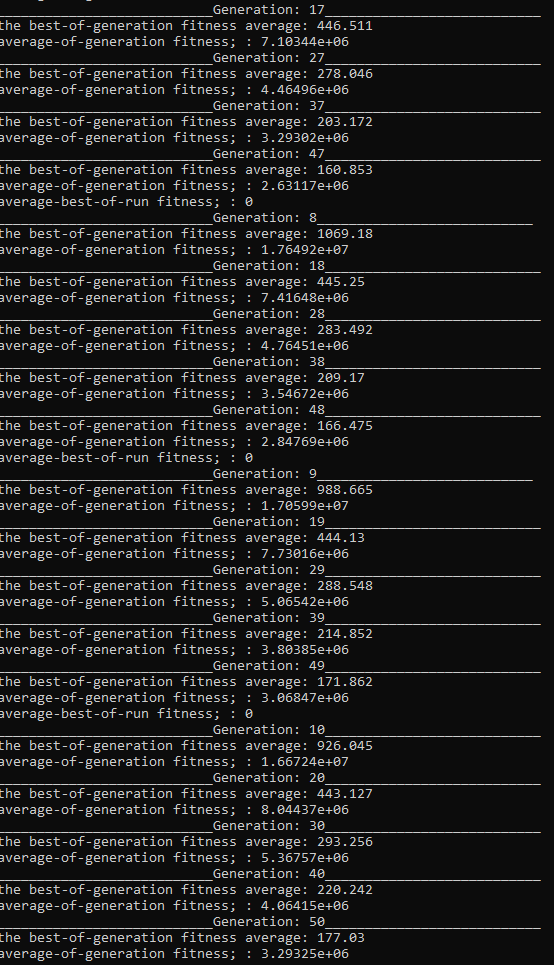


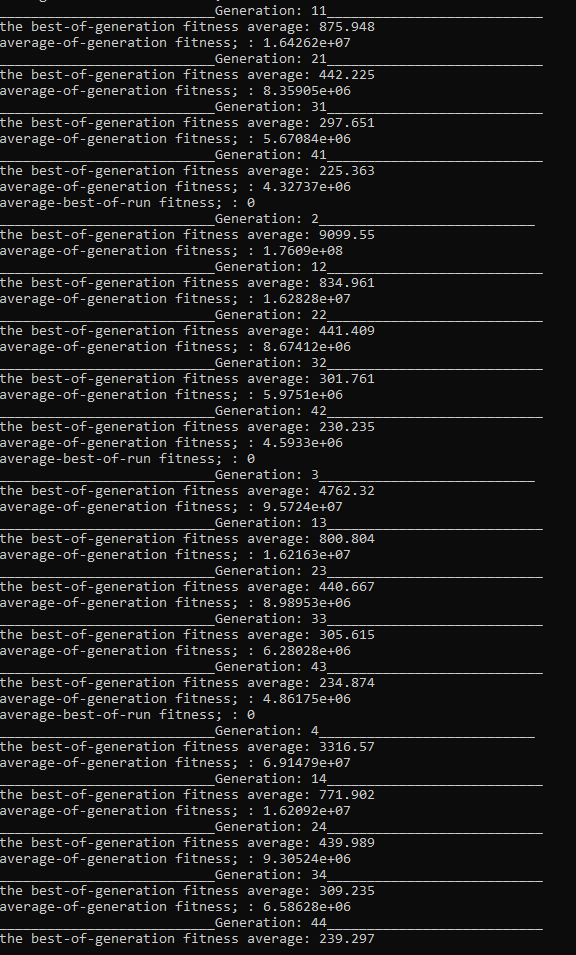


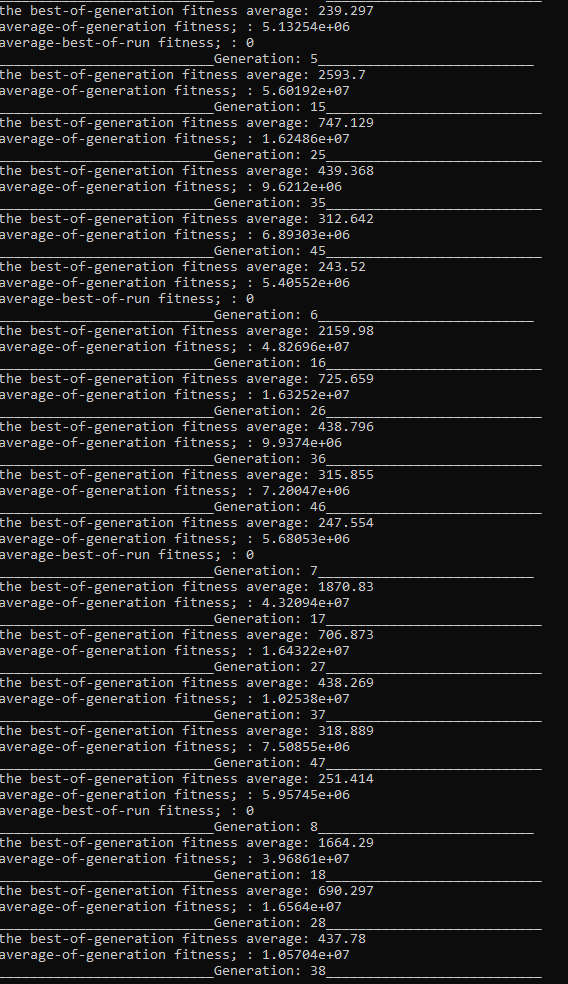


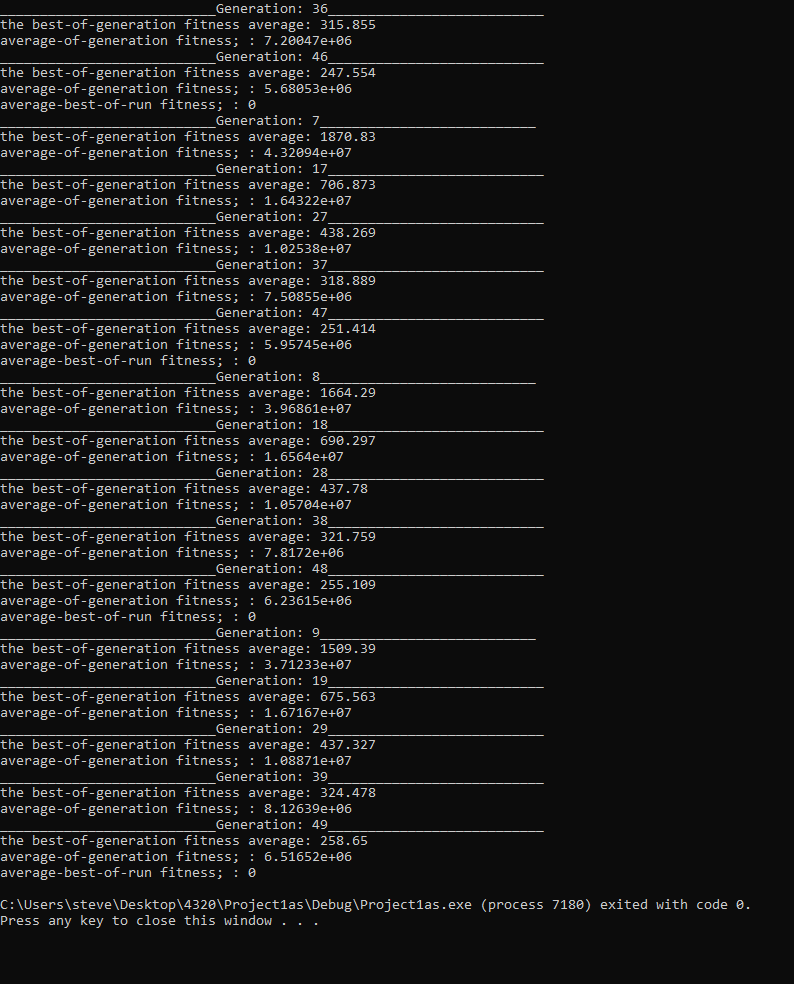












#pragma once

//steven guo

//4320

//2/14/21

//this is the header file called header.h

const int n = 30;// constant for population size

const double pc = 0.8;//percentage for single point crossover

const double pm = 0.1;//percentage for gene mutation

const int maxGen = 50;//maximum amount of generations

const double delta = 6 / 1000;//delta for gene mutation

int best\_index=0;//index of the best fitness in a generation

int worst\_index=0;//index of the worst fitness in a generation

int solution\_index=0;//index of the best of run solution

double solution\_fitness = 0;//best of run solution fitness

int generation\_count = 0;

double sum\_of\_generation\_fitness=0;//accumulate the sum of the total fitness in a generation

double sum\_of\_best\_of\_generation\_fitness=0;// accumulate the sum of the best of generation fitness

double sum\_of\_best\_of\_run\_fitness=0;

//individual

typedef struct

{

double x1;

double x2;

double x3;

} Individual;

Individual selection\_array[n];//selection array

Individual solution;//stores an individual which is the best individual in a run

double rng();

void proportional\_selection(Individual gen\_array[], double pi\_array[]);

void single\_point\_crossover(Individual sel\_array[]);

void gene\_wise\_mutation(Individual sel\_array[]);

double average\_fitness(double f\_array[]);

double best\_fitness(double array[]);

double worst\_fitness(double array[]);

double get\_pi(double x, double f);

double get\_f(Individual gen);

#include<stdio.h>

#include<string>

#include<stdlib.h>

#include<iostream>

#include <cstdlib>

#include "header.h"

#include <time.h>

#include <cmath>

//steven guo

//project1 for 4320

//2/14/21

using namespace std;

int main(int argc, char\* argv[])

{

srand(time(NULL));

Individual population\_array[n];//generate array of 30 members

double pi\_array[n];//array to hold the pi values of each member

double f\_array[n];//array to hold the f values of each member

double total\_f\_values=0;//holds the sum of all f values

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SINGLE RUN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

for (int i = 0; i < n; i++)//fill array for population

{

population\_array[i].x1 = rng();//random number from -1.0 to 5.0

population\_array[i].x2 = rng();

population\_array[i].x3 = rng();

f\_array[i] = get\_f(population\_array[i]);//calculates f value of individual and puts it into an array of fitnesses

if (f\_array[i] < solution\_fitness)//keeps track of the solution, i think the smallest fitness is the best solution so if it is smaller than the best solution we got then replace it with the better one

{

solution\_index = i;//store index

solution.x1 = population\_array[i].x1;//store values

solution.x2 = population\_array[i].x2;

solution.x3 = population\_array[i].x3;

solution\_fitness = f\_array[i];//store fitness

}

total\_f\_values += f\_array[i];//sums up the total fitness in the generation

}

for (int i = 0; i < n; i++)//finds pi values of the individuals

{

pi\_array[i] = get\_pi(f\_array[i], total\_f\_values);//puts pi values into array corresponding to individual

}

proportional\_selection(population\_array, pi\_array);

single\_point\_crossover(selection\_array);

gene\_wise\_mutation(selection\_array);

int count = 1;//counter for intervals of 10 generations

for (int i = 1; i < maxGen; i++)

{

for (int i = 0; i < n; i++)

{

f\_array[i] = get\_f(selection\_array[i]);//calculates f value of individuals and puts it into an array of fitnesses

if (f\_array[i] < solution\_fitness)//keeps track of the solution, i think the smallest fitness is the best solution so if it is smaller than the best solution we got then replace it with the better one

{

solution\_index = i;//store index

solution.x1 = selection\_array[i].x1;//store values

solution.x2 = selection\_array[i].x2;

solution.x3 = selection\_array[i].x3;

solution\_fitness = f\_array[i];//store fitness

}

total\_f\_values += f\_array[i];//sums up the total fitness in the generation

}

for (int i = 0; i < n; i++)//finds pi values of the individuals

{

pi\_array[i] = get\_pi(f\_array[i], total\_f\_values);//puts pi values into array corresponding to individual

}

proportional\_selection(selection\_array, pi\_array);

single\_point\_crossover(selection\_array);

gene\_wise\_mutation(selection\_array);

count++;

if (count == 10)//outputs information

{

count = 0;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Generation: "<< i+1<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "best fitness in current Generation: "<< best\_fitness(f\_array) << endl;

cout << "the x values are: " << selection\_array[best\_index].x1 << ", " << selection\_array[best\_index].x2 << ", " << selection\_array[best\_index].x3 << endl;

cout << "worst fitness in current Individual: "<< worst\_fitness(f\_array) << endl;

cout << "the x values are: " << selection\_array[worst\_index].x1<< ", " << selection\_array[worst\_index].x2 << ", " << selection\_array[worst\_index].x3 << endl;

cout << "average fitness of all members in current Individual: " << average\_fitness(f\_array) << endl;

}

}

cout << "best of run solution: " << selection\_array[solution\_index].x1 << ", " << selection\_array[solution\_index].x2 << ", " << selection\_array[solution\_index].x3 <<endl;

cout << "best of run solution fitness: " << solution\_fitness<< endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_30 INDEPENDENT RUNS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

for (int i = 0; i < n; i++)//fill array for population

{

population\_array[i].x1 = rng();//random number from -1.0 to 5.0

population\_array[i].x2 = rng();

population\_array[i].x3 = rng();

f\_array[i] = get\_f(population\_array[i]);//calculates f value of individual and puts it into an array of fitnesses

if (f\_array[i] < solution\_fitness)//keeps track of the solution, i think the smallest fitness is the best solution so if it is smaller than the best solution we got then replace it with the better one

{

sum\_of\_best\_of\_run\_fitness += f\_array[i];

}

total\_f\_values += f\_array[i];//sums up the total fitness in the generation

}

for (int i = 0; i < n; i++)//finds pi values of the individuals

{

pi\_array[i] = get\_pi(f\_array[i], total\_f\_values);//puts pi values into array corresponding to individual

}

proportional\_selection(population\_array, pi\_array);

single\_point\_crossover(selection\_array);

gene\_wise\_mutation(selection\_array);

sum\_of\_best\_of\_generation\_fitness += best\_fitness(f\_array);// accumulate the sum of the best of generation fitness

sum\_of\_generation\_fitness +=total\_f\_values;//accumulate the sum of the total fitness in a generation

count = 1;//counter for intervals of 10 generations

for (int y = 0; y < n; y++)//this for loop represents each run

{

for (int i = 1; i < maxGen; i++)//this for loop represents each generation

{

for (int a = 0; a < n; a++)

{

f\_array[a] = get\_f(selection\_array[a]);//calculates f value of individuals and puts it into an array of fitnesses

if (f\_array[a] < solution\_fitness)//keeps track of the solution, i think the smallest fitness is the best solution so if it is smaller than the best solution we got then replace it with the better one

{

sum\_of\_best\_of\_run\_fitness += f\_array[a];//store fitness

}

total\_f\_values += f\_array[a];//sums up the total fitness in the generation

}

for (int b = 0; b < n; b++)//finds pi values of the individuals

{

pi\_array[b] = get\_pi(f\_array[b], total\_f\_values);//puts pi values into array corresponding to individual

}

proportional\_selection(selection\_array, pi\_array);

single\_point\_crossover(selection\_array);

gene\_wise\_mutation(selection\_array);

sum\_of\_best\_of\_generation\_fitness += best\_fitness(f\_array);// accumulate the sum of the best of generation fitness

sum\_of\_generation\_fitness += total\_f\_values;//accumulate the sum of the total fitness in a generation

count++;

if (count == 10)//outputs information

{

count = 0;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Generation: " << i+1 << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "the best-of-generation fitness average: " << sum\_of\_best\_of\_generation\_fitness / i << endl;

cout << "average-of-generation fitness; : " << sum\_of\_generation\_fitness / i << endl;

}

}

cout << "average-best-of-run fitness; : " << sum\_of\_best\_of\_run\_fitness/ 50 << endl;

}

return 0;

}

double rng()//return random number from -1.0 to 5.0

{

//-1.0 <= Xi <= 5.0

double f = (double)rand() / RAND\_MAX;

return -1.0 + f \* 6.0;

}

void proportional\_selection(Individual population\_array[], double pi\_array[])

{//this function compares the pi values from each individual and makes a check to be selected for the generation

double u;

int count = 0;//keeps track of how many individuals have been selected

int pop\_count = 0;//keeps track of population index

do

{

u = rand() / double(RAND\_MAX);//range from 0 to 1

if (u < pi\_array[count])//check if the individual is selected

{

selection\_array[count].x1 = population\_array[pop\_count].x1;

selection\_array[count].x2 = population\_array[pop\_count].x2;

selection\_array[count].x3 = population\_array[pop\_count].x3;

count++;

}

pop\_count++;

if (pop\_count > 29)//this is for when it has to cycle through the population again for selection process

{

pop\_count = 0;

}

} while (count < 30);//this will fill the selection array with 30 individuals

}

void single\_point\_crossover(Individual sel\_array[])

{

int coinflip = (rand() % 2) + 1;//decides the point of crossover. there is only 2 points so its a coinflip

double temp1,temp2,temp3;//temporary variables to exchange parts of parents

int randNum;//this

for (int i = 0; i < n; i += 2)//increment by 2 because i will be making 2 children from 2 parents

{

if (coinflip == 1)//the crossover is between x1,x2

{

randNum = (rand() % 10) + 1;//rand Num is a number between 1-10

if (randNum < 9)//excludes 9 and 10 which means 80% of the time it will be true

{

temp2 = sel\_array[i].x2;//store parent1's x2 into temp variable

temp3 = sel\_array[i].x3;//store parent1's x3 into temp variable

sel\_array[i].x2 = sel\_array[i + 1].x2;//replace parent1's x2 with parent2's x2

sel\_array[i].x3 = sel\_array[i + 1].x3;//replace parent1's x3 with parent2's x3

sel\_array[i + 1].x2 = temp2;//replace parent2's x2 with temp variable

sel\_array[i + 1].x3 = temp3;//replace parent2's x2 with temp variable

}

}

else//the crossover is between x2,x3

{

randNum = (rand() % 10) + 1;

if (randNum < 9)

{

temp1 = sel\_array[i].x1;//store parent1's x1 into temp variable

temp2 = sel\_array[i].x2;//store parent1's x2 into temp variable

sel\_array[i].x1 = sel\_array[i + 1].x1;//replace parent1's x1 with parent2's x1

sel\_array[i].x2 = sel\_array[i + 1].x2;//replace parent1's x2 with parent2's x2

sel\_array[i + 1].x1 = temp1;//replace parent2's x1 with temp variable

sel\_array[i + 1].x2 = temp2;//replace parent2's x2 with temp variable

}

}

}

}

void gene\_wise\_mutation(Individual sel\_array[])

{

int randNum;

int coinFlip;//this determines if i subtract the delta or add it

for (int i = 0; i < n; i++)

{

//10% chance for gene mutation for x1

randNum = (rand() % 10) + 1;//random number between 1-10

if (randNum < 2)//excludes 1 therefore it is a 10% chance to be true

{

coinFlip = (rand() % 2) + 1;

if (coinFlip == 1)//subtract delta

{

sel\_array[i].x1 -= delta;

if (sel\_array[i].x1 < -1.0)

{

sel\_array[i].x1 = -1.0;

}

}

else

{

sel\_array[i].x1 += delta;

if (sel\_array[i].x1 > 5.0)

{

sel\_array[i].x1 = 5.0;

}

}

}

//10% chance for gene mutation for x2

randNum = (rand() % 10) + 1;

if (randNum < 2)

{

coinFlip = (rand() % 2) + 1;

if (coinFlip == 1)

{

sel\_array[i].x2 -= delta;

if (sel\_array[i].x1 < -1.0)

{

sel\_array[i].x1 = -1.0;

}

}

else

{

sel\_array[i].x2 += delta;

if (sel\_array[i].x2 > 5.0)

{

sel\_array[i].x2 = 5.0;

}

}

}

//10% chance for gene mutation for x3

randNum = (rand() % 10) + 1;

if (randNum < 2)

{

coinFlip = (rand() % 2) + 1;

if (coinFlip == 1)

{

sel\_array[i].x3 -= delta;

if (sel\_array[i].x3 < -1.0)

{

sel\_array[i].x3 = -1.0;

}

}

else

{

sel\_array[i].x3 += delta;

if (sel\_array[i].x3 > 5.0)

{

sel\_array[i].x3 = 5.0;

}

}

}

}

}

double average\_fitness(double array[])//returns the average fitness

{

double sum=0;

for (int i = 0; i < n; i++)

{

sum += array[i];

}

return sum / (double)n;

}

double best\_fitness(double array[])//returns the best fitness

{

double best = 1000.00;

int index = 0;

for (int i = 0; i < n; i++)

{

if (array[i] < best)

{

best = array[i];

best\_index = i;

}

}

return best;

}

double worst\_fitness(double array[])//returns the worst fitness

{

double worst=0.0 ;

for (int i = 0; i < n; i++)

{

if (array[i] > worst)

{

worst = array[i];

worst\_index = i;

}

}

return worst;

}

double get\_pi(double x, double f)//returns pi

{

return x / f;

}

double get\_f(Individual gen)//returns f

{

return ((gen.x1 \* gen.x1) + (gen.x2 \* gen.x2) + (gen.x3 \* gen.x3));

}