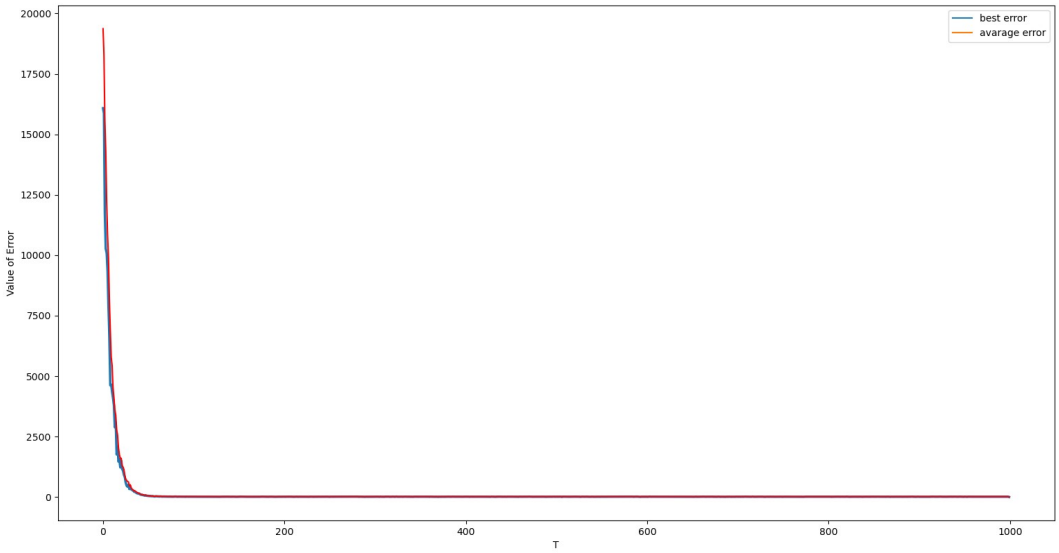
Отчет по второму заданию «Генетические алгоритмы»
выполнил Женин Максим Николаевич, 523 группа

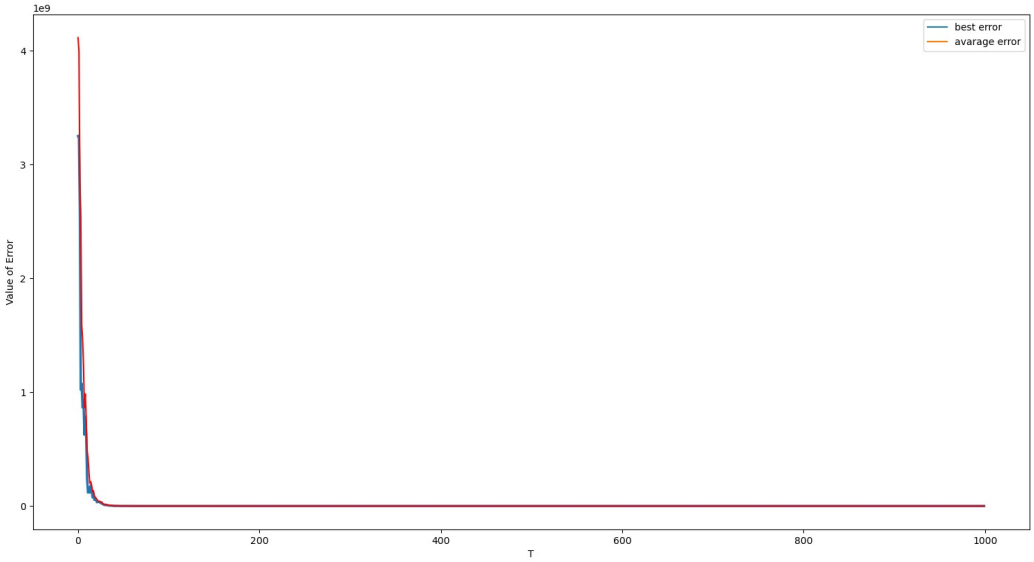
Тестирование проводилось на своей(локальной машине) 4 ядра.
В программе зафиксировал кол-во миграции, которое равно 0.25 процентов от общего кол-во решений на одном острове.
Параметры запуска программы: n — 20, где n размер одной популяции, m — 6000, где m размер популяции, T — 1000, где T кол-во итераций, Mig — 500, где Mig частота миграции
Пример запуска: mpirun -n 4 ./main 20 6000 1000 500

Графики

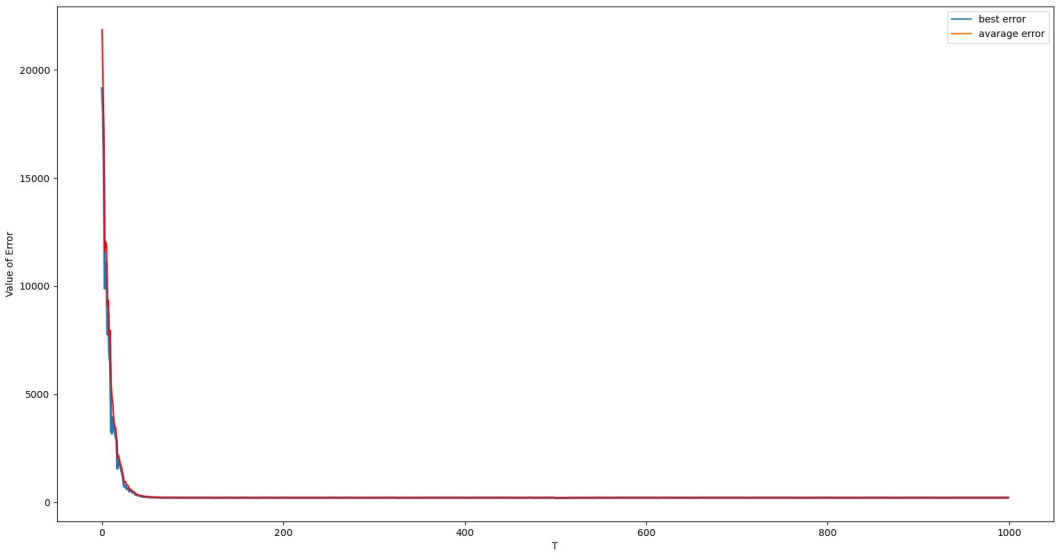
Сферическая функция



функция Розенброка



функция Растригина



Код:

```
#include <iostream>
#include <cmath>
#include <fstream>
#include <mpi.h>
#include <random>

using namespace std;
int Nprocs = 1;
int Rank;
```

```
double frand() // ������������ ������ ������ [0,1)
{
return double(rand())/RAND_MAX;
}

int gen(const double start , const double end ) {
static random_device rd ;
static mt19937 gen (rd()) ;
uniform_real_distribution <> dis (start,end) ;
return dis(gen) ;
}

double sphere(double *a, int n, int offset) {
double sum = 0;
for(int i = 0 ; i < n; ++i) {
sum += pow(a[n * offset + i],2);
}
return sum;
}

double Rozenbrok(double *a, int n, int offset) {
double sum = 0;
for(int i = 0 ; i < n - 1; ++i) {
sum += 100 * pow(pow(a[n * offset + i],2) - a[offset * n + i + 1],2) + pow(a[offset * n + i]
- 1,2);
}
return sum;
}

double Rastrigin(double *a, int n, int offset) {
double sum = 0;
for(int i = 0 ; i < n; ++i) {
sum += pow(a[n * offset + i],2) - 10 * cos(2 * 180 * i) + 10;
}
return sum;
}


void init(double* P, int m, int n)
{
for( int k=0; k<m; k++ )
for( int i=0; i<n; i++ )
P[k*n+i] = gen(-100,100);
}

void shuffle(double* P, int m, int n)
{
for( int k=0; k<m; k++ )
{
int l = rand()%m;
```

```
for( int i=0; i<n; i++ )
swap(P[k*n+i],P[l*n+i]);
}
}

void select(double* P, int m, int n)
{
double pwin = 0.9;
shuffle(P, m, n);
for( int k=0; k<m/2; k++ )
{
int a = 2*k;
int b = 2*k+1;
double fa = sphere(P, n, a);
double fb = sphere(P, n, b);
double p = frand();
if( (fa<fb && p<pwin ) || (fa>fb && p>pwin) )
for( int i=0; i<n; i++ )
P[b*n+i] = P[a*n+i];
else
for( int i=0; i<n; i++ )
P[a*n+i] = P[b*n+i];
}
}

void crossover(double* P, int m, int n)
{
shuffle(P, m, n);
for( int k=0; k<m/2; k++ )
{
int a = 2*k;
int b = 2*k+1;
int j = rand()%n;
for( int i=j; i<n; i++ )
swap(P[a*n+i],P[b*n+i]);
}
}

void mutate(double* P, int m, int n)
{
double pmut = 0.1;
for( int k=0; k<m; k++ )
for( int i=0; i<n; i++ )
if( frand() < pmut )
P[k*n+i] += gen(-5,5);
}

double printthebest(double* P, int m, int n)
{
int idx_best = -1;
double f0 = 1000000000000;
```

```
for(int i = 0 ; i < m; ++i) {
double f = sphere(P, n, i);
if (f < f0) {
f0 = f;
idx_best = i;
}
}
/*cout << f0 << ": ";
for( int i=0; i<n; i++ )
cout << P[k0*n+i] << " ";*/
return f0;
}

void migration(double *P, int m, int n,int left,int right) {
MPI_Status status;
double *toLeft = new double[int(m * n * 0.25)];
double *toRight = new double[int(m * n * 0.25)];
for(int i = 0 ; i < int(m * n * 0.25); ++i) {
toLeft[i] = P[i];
toRight[i] = P[int(m * n * 0.25) + i];
}
MPI_Sendrecv(toLeft, int(m * n * 0.25), MPI_INT, left, 0, P + n * m - int(m * n * 0.25) - 1,
int(m * n * 0.25), MPI_INT, right, 0, MPI_COMM_WORLD, &status);
MPI_Sendrecv(toRight, int(m * n * 0.25), MPI_INT, right, 0, P,
int(m * n * 0.25), MPI_INT, left, 0, MPI_COMM_WORLD, &status);
}

void runGA(int n, int m, int T, int mig)
{
double* P = new double[n*m];
init(P, m, n);
double best = 0.;
double sum = 0.;
double totalsum = 0.;
double totalbest = 0.;
ofstream Answer("outout.txt");
for( int t = 1; t <= T; t++ )
{
select(P, m, n);
crossover(P, m, n);
mutate(P, m, n);
if (t % mig == 0) {
int left = (Rank + Nprocs - 1) % Nprocs;
int right = (Rank + 1) % Nprocs;
migration(P, m, n,left, right);
}
best = printthebest(P, m, n);
sum = best;
MPI_Allreduce(&sum, &totalsum, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
MPI_Allreduce(&best, &totalbest, 1, MPI_DOUBLE, MPI_MIN, MPI_COMM_WORLD);
if (Rank == 0) {
```

```
Answer << "T = " << t << " best error = " << totalbest << " average error = " <<
totalsum / Nprocs << "\n";
}
}

delete[] P;
}

int main(int argc, char** argv)
{
if (MPI_Init(&argc, &argv) != MPI_SUCCESS) {
fprintf(stderr, "failed to init MPI\n");
exit(1);
}
if (MPI_Comm_size(MPI_COMM_WORLD, &Nprocs) != MPI_SUCCESS ||
MPI_Comm_rank(MPI_COMM_WORLD, &Rank) != MPI_SUCCESS) {
fprintf(stderr, "failed to get communicator size or rank\n");
exit(1);
}
srand(time(NULL)+ Rank);
int n = atoi(argv[1]);
int m = atoi(argv[2]);
int T = atoi(argv[3]);
int mig = atoi(argv[4]);

runGA(n, m, T, mig);
MPI_Finalize();
return 0;
}
```