Отчет по выполнению практического задания «Генетические алгоритмы» по курсу Естественные модели параллельных вычислений студента 523 группы Ухина Сергея Алексеевича.
ser191097@gmail.com

Графики зависимостей средней и наименьшей ошибки от номера итерации для трёх рассматриваемых функций. Размер популяции 40, размер одного члена популяции 20, частота миграций 5, размер миграции 5.

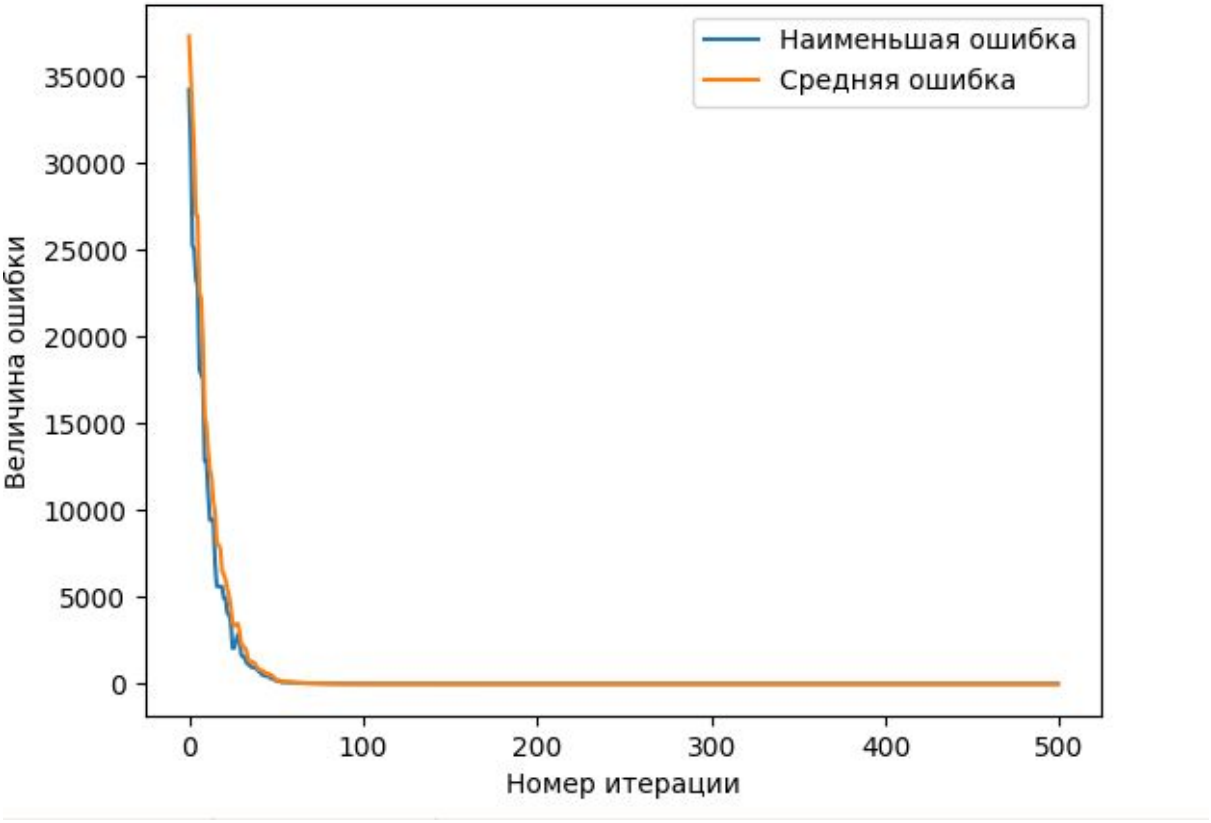

Рис. 1: Сферическая функция

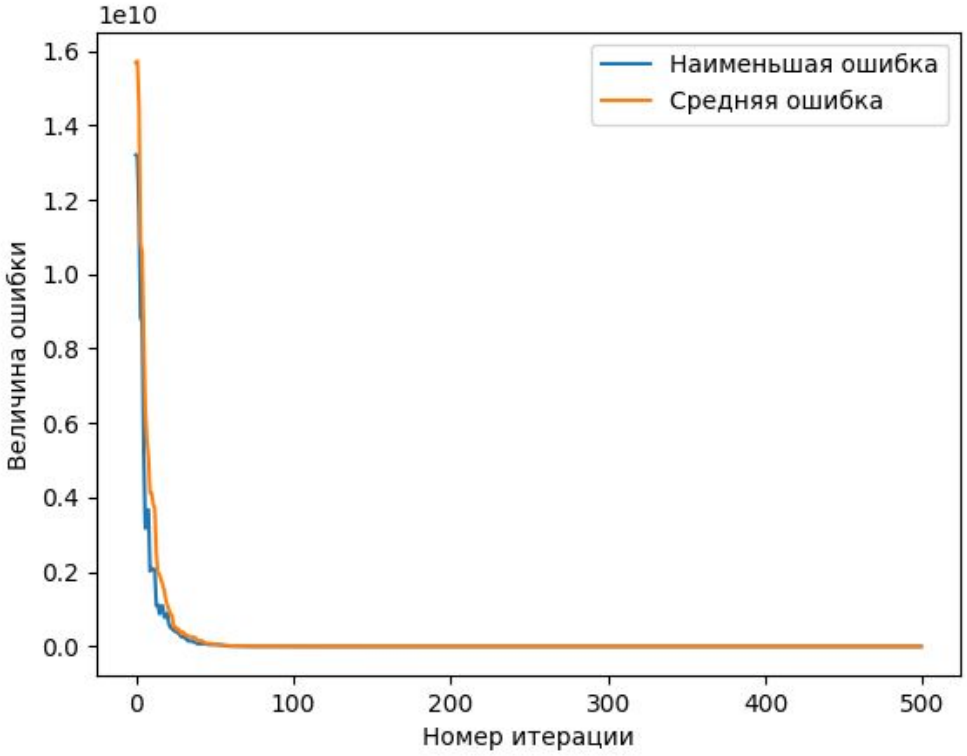

Рис. 2: Функция Розенброка

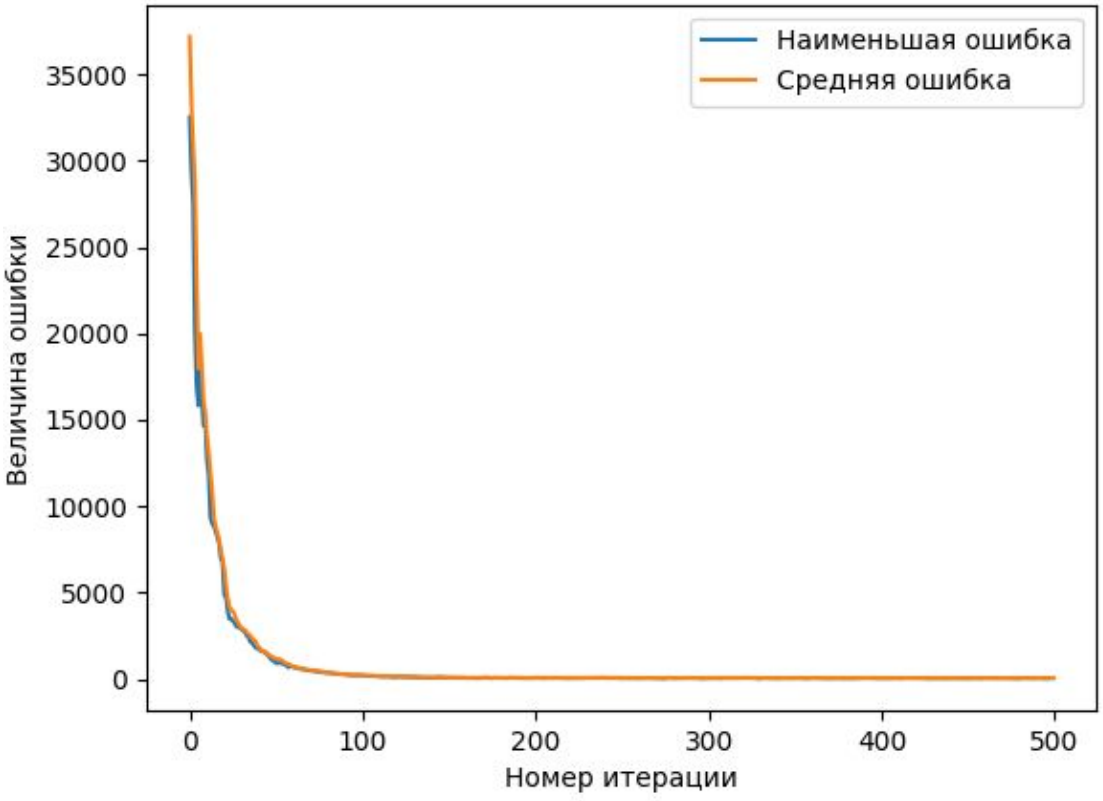

Рис. 3: Функция Растригина

Листинг 1: Код программы

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

using namespace std;


double genDouble ( const double a , const double b ) {
    static random_device rd ;
    static mt19937 gen (rd()) ;
    uniform_real_distribution <> dis (a,b) ;
    return dis(gen) ;
}

double f0(double *P, int n, int num) {
    double sum = 0;
    for(int i = 0 ; i < n; ++i) {
        sum += P[n * num + i] * P[n * num + i];
    }
    return sum;
}

double pow2(double x) {
    return x * x;
}

double f1(double *P, int n, int num) {
    double sum = 0;
    for(int i = 0 ; i < n - 1; ++i) {
        sum += 100 * pow2(pow2(P[n * num + i]) - P[num * n + i + 1]) + pow2(P[num * n
+ i] - 1);
    }
    return sum;
}

double f2(double *P, int n, int num) {
    double sum = 0;
    static const double PI = acos(-1.);
    for(int i = 0 ; i < n; ++i) {
        sum += pow2(P[n * num + i]) - 10 * cos(2 * PI * P[n * num + 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] = genDouble(-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 = f1(P, n, a);
        double fb = f1(P, n, b);
        double p = genDouble(0., 1.);
        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( genDouble(0., 1.) < pmut )
                P[k*n+i] += genDouble(-1.,1.);
}

double printthebest(double* P, int m, int n)
{
    int idx_best = -1;
    double best_val = 1e18;
    for(int i = 0 ; i < m; ++i) {
        double cur_val = f1(P, n, i);
        if (cur_val < best_val) {
            best_val = cur_val;
            idx_best = i;
        }
    }

    return best_val;
}

void migration(double *P, int m, int n, int rank, int size, int migration_size) {
    int left = (rank + size - 1) % size;
    int right = (rank + 1) % size;
    double *sendl = new double[n * migration_size];
    double *sendr = new double[n * migration_size];
    for(int i = 0 ; i < migration_size * n; ++i) {
        sendl[i] = P[i];
        sendr[i] = P[n * m - migration_size * n + i - 1];
    }
```

```
   MPI_Sendrecv(sendl, migration_size * n, MPI_DOUBLE, left, 0, P + n *  m -
migration_size * n - 1,
      migration_size * n, MPI_DOUBLE, right, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
   MPI_Sendrecv(sendr, migration_size * n, MPI_DOUBLE, right, 0, P,
      migration_size * n, MPI_DOUBLE, left, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
}

void runGA(int n, int m, int it, int migration_size, int migration_frequency, int rank, int
size)
{
   double* P = new double[n*m];
   init(P, m, n);
   double best, sum;
   ofstream bFile("best2.txt");
   ofstream aFile("av2.txt");
   for( int t = 1; t <= it; t++ )
   {
      select(P, m, n);
      crossover(P, m, n);
      mutate(P, m, n);
      if (t % migration_frequency == 0) {
         migration(P, m, n, rank, size, migration_size);
      }
      sum = best = printthebest(P, m, n);
      if (rank == 0) {
         MPI_Reduce(MPI_IN_PLACE, &sum, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
         MPI_Reduce(MPI_IN_PLACE, &best, 1, MPI_DOUBLE, MPI_MIN, 0,
MPI_COMM_WORLD);
         bFile << best << "\n";
         aFile << sum / size << "\n";
         cout << "It num - " << t << " best - " << best << " average - " << sum / size <<
"\n";
      } else {
         MPI_Reduce(&sum, 0, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
         MPI_Reduce(&best, 0, 1, MPI_DOUBLE, MPI_MIN, 0,
MPI_COMM_WORLD);
      }
   }

   delete[] P;
```

```
}

int main(int argc, char** argv)
{
    MPI_Init(&argc, &argv);
    int rank;
    int size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank) ;
    MPI_Comm_size(MPI_COMM_WORLD, &size) ;
    int n = atoi(argv[1]);
    int m = atoi(argv[2]);
    int it = atoi(argv[3]);
    int migration_size = atoi(argv[4]);
    int migration_frequency = atoi(argv[5]);


    runGA(n, m, it, migration_size, migration_frequency, rank, size);
    MPI_Finalize();
    return 0;
}
```