Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Волгоградский государственный технический университет»
Факультет электроники и вычислительной техники
Кафедра «Электронно-вычислительные машины и системы»

Семестровая работа по дисциплине «Вычислительные системы и сетевые технологии» на тему «Решение системы ОДУ методом Адамса»

Вариант №10

Выполнил
студент группы САПР-1.1п
Чечеткин И. А.

Проверил
к. т. н. Андреев А. Е.

Волгоград, 2016

Содержание

1 Метод Адамса

Будем рассматривать систему линейных неоднородных обыкновенных дифференциальных уравнений с постоянными коэффициентами

$$\frac{d\mathbf{y}}{dt} = A\mathbf{y} + \mathbf{f}(t)$$

со следующими начальными условиями:

$$\mathbf{y}(t_0) = \mathbf{y}_0 = \begin{pmatrix} y_{10} \\ y_{20} \\ \dots \\ y_{n0}. \end{pmatrix}$$

Здесь $\mathbf{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \dots \\ y_n(t) \end{pmatrix}$ и $\mathbf{f}(t) = \begin{pmatrix} f_1(t) \\ f_2(t) \\ \dots \\ f_n(t) \end{pmatrix}$, а A – матрица с постоянными коэффициентами размером $n \times n$. Решение такой системы ищется обычно с использованием одношаговых или многошаговых методов четвертого порядка.

При решении задачи Коши методами Рунге–Кутты необходимо вычислять правые части обыкновенных дифференциальных уравнений в нескольких промежуточных точках отрезка интегрирования. Количество таких вычислений зависит от порядка используемого метода. Однако, после того как решение системы ОДУ определено в нескольких точках $t_0, \dots, t_q$, можно применить алгоритмы интерполяции и сократить количество вычислений правых частей ОДУ для получения вектора решения $\mathbf{y}_{q+1}$. Методы такого рода называют многошаговыми или многоточечными. Алгоритмы многоточечных методов основываются на аппроксимации интерполяционными полиномами либо правых частей ОДУ, либо интегральных кривых.

Экстраполяционная формула Адамса–Башфорта 4 порядка:

$$\widetilde{\mathbf{y}}_{m+1} = \mathbf{y}_m + \frac{h}{24}\big[55(A\mathbf{y}_m + \mathbf{f}_m) - 59(A\mathbf{y}_{m-1} + \mathbf{f}_{m-1}) + \\ +37(A\mathbf{y}_{m-2} + \mathbf{f}_{m-2}) - 9(A\mathbf{y}_{m-3} + \mathbf{f}_{m-3})\big].$$

Эта формула часто рассматривается как формула прогноза, значение которой корректируется интерполяционной формулой Адамса–Моултона:

$$\mathbf{y}_{m+1} = \mathbf{y}_m + \frac{h}{24}\big[9(A\widetilde{\mathbf{y}}_{m+1} + \widetilde{\mathbf{f}}_{m+1}) + 19(A\mathbf{y}_m + \mathbf{f}_m) - \\ -5(A\mathbf{y}_{m-1} + \mathbf{f}_{m-1}) + A\mathbf{y}_{m-2} + \mathbf{f}_{m-2}\big].$$

Последовательное применение этих двух формул носит название схемы «предиктор–корректор».

2 Последовательная программа

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctime>

typedef double* Vector;
typedef double** Matrix;

const int m = 3;
int n;
double h;
double x;
int xk;
double eps;

void set(Vector& a, Vector b) {
  int rows = n;

  for (int i = 0; i < rows; i++)
    a[i] = b[i];
}

Vector mul(Matrix a, Vector b) {
  int rows = n;
  int cols = n;
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++)
      c[i] += a[i][j] * b[j];
  }
  return c;
}

Matrix mul(Matrix a, double b) {
  int rows = n;
  int cols = n;
  Matrix c = new Vector[rows];
  for (int i = 0; i < n; i++)
    c[i] = new double[cols];

  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++)
      c[i][j] = a[i][j] * b;
  }
  return c;
}

Vector mul(Vector a, double b) {
```

```
  int rows = n;
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++) {
    c[i] = a[i] * b;
  }
  return c;
}

Vector sum(Vector a, Vector b) {
  int rows = n;
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++)
    c[i] = a[i] + b[i];
  return c;
}

Vector sub(Vector a, Vector b) {
  int rows = n;
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++)
    c[i] = a[i] - b[i];
  return c;
}

void set_data(Matrix& a, Vector& y0, Matrix& f) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      a[i][j] = (double)rand() / (RAND_MAX) * 2 - 1;
  }

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      f[i][j] = (double)rand() / (RAND_MAX) * 2 - 1;
  }

  for (int i = 0; i < n; i++)
    y0[i] = (double)rand() / (RAND_MAX) * 2 - 1;
}

Vector f_t(Matrix f, double t) {
  int rows = n;
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++)
    c[i] = exp(-t) * cos(log(f[i][0] * f[i][0])) +
      sin(f[i][1]) * tan(t) + exp(-abs(f[i][2]));

  return c;
}
```

```
Vector runge_kutta(Matrix a, Vector y, Matrix f) {
  Vector k1 = new double[n];
  Vector k2 = new double[n];
  Vector k3 = new double[n];
  Vector k4 = new double[n];
  Vector temp = new double[n];

  k1 = sum(mul(mul(a, y), h), mul(f_t(f, x), h));
  k2 = sum(mul(mul(a, sum(y, mul(k1, 0.5))), h),
    mul(f_t(f, x + h / 2), h));
  k3 = sum(mul(mul(a, sum(y, mul(k2, 0.5))), h),
    mul(f_t(f, x + h / 2), h));
  k4 = sum(mul(mul(a, sum(y, k3)), h),
    mul(f_t(f, x + h), h));

  temp = sum(y, mul(sum(sum(k1, k4), mul(sum(k2, k3), 2)), 1.0 / 6));
  return temp;
}

Vector v(Matrix a, Vector y, Matrix f, double t) {
  Vector temp = new double[n];
  temp = sum(mul(a, y), f_t(f, t));
  return temp;
}

double norm(Vector a) {
  int rows = n;
  double temp = 0;

  for (int i = 0; i < rows; i++)
    temp += a[i] * a[i];
  return sqrt(temp);
}

Vector adams(Vector yn, Vector yn1, Vector yn2, Vector yn3,
  Matrix a, Matrix f) {
  Vector v3 = new double[n]; v3 = v(a, yn,  f, x);
  Vector v2 = new double[n]; v2 = v(a, yn1, f, x - h);
  Vector v1 = new double[n]; v1 = v(a, yn2, f, x - 2 * h);
  Vector v0 = new double[n]; v0 = v(a, yn3, f, x - 3 * h);

  Vector yp = new double[n];
  Vector yc = new double[n];
  Vector vn = new double[n];
  double norma;

  while (x < xk * h) {
    norma = 100.0 * eps;

    yp = sum(yn, mul(sum(sub(mul(v3, 55.0), mul(v2, 59.0)),
      sub(mul(v1, 37.0), mul(v0, 9.0))), h / 24));
```

```
    x += h;

    while (norma > eps) {
      vn = v(a, yp, f, x);
      yc = sum(yn, mul(sum(sum(mul(vn, 9.0), mul(v3, 19.0)),
        sub(mul(v2, 5.0), v1)), h / 24));
      norma = norm(sub(yc, yp));
      set(yp, yc);
    }

    set(yn, yc);
    set(v0, v1);
    set(v1, v2);
    set(v2, v3);
    set(v3, v(a, yn, f, x));
  }
  return yn;
}

int main() {
  printf("Input number of equations: ");
  scanf("%d", &n);
  printf("Input step: ");
  scanf("%lf", &h);
  eps = h / 100.0;
  printf("Input number of points: ");
  scanf("%d", &xk);
  clock_t start, finish;

  x = 0;

  Matrix a = new Vector[n];
  for (int i = 0; i < n; i++)
    a[i] = new double[n];
  Matrix f = new Vector[n];
  for (int i = 0; i < n; i++)
    f[i] = new double[m];

  Vector y0 = new double[n];
  set_data(a, y0, f);

  Vector y = new double[n];
  set(y, y0);

  Vector y1 = new double[n]; y1 = runge_kutta(a, y, f);
  x += h;
  Vector y2 = new double[n]; y2 = runge_kutta(a, y1, f);
  x += h;
  y = runge_kutta(a, y2, f);
  x += h;
```

```
    start = clock();
    y = adams(y, y2, y1, y0, a, f);
    finish = clock();
    double time = (double)(finish - start) / (double) CLOCKS_PER_SEC;

    printf("Elapsed time: %.3lf\n", time);

    return 0;
}
```

3 Программа, использующая OpenMP

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

typedef double* Vector;
typedef double** Matrix;

const int m = 3;
int n;
int k;
double h;
double x;
int xk;
double eps;
int size;

Vector set(Vector b, int rows) {
  Vector c = new double[rows];
  for (int i = 0; i < rows; i++)
    c[i] = b[i];
  return c;
}

Vector mul(Matrix a, Vector b, int rows, int cols) {
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++)
      c[i] += a[i][j] * b[j];
  }
  return c;
}

Matrix mul(Matrix a, double b, int rows, int cols) {
  Matrix c = new Vector[rows];
  for (int i = 0; i < n; i++)
    c[i] = new double[cols];

  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++)
      c[i][j] = a[i][j] * b;
  }
  return c;
}

Vector mul(Vector a, double b, int rows) {
  Vector c = new double[rows];
```

```
  for (int i = 0; i < rows; i++)
    c[i] = a[i] * b;
  return c;
}

Vector sum(Vector a, Vector b, int rows) {
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++)
    c[i] = a[i] + b[i];
  return c;
}

Vector sub(Vector a, Vector b, int rows) {
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++)
    c[i] = a[i] - b[i];
  return c;
}

void set_data(Matrix& a, Vector& y0, Matrix& f) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      a[i][j] = (double)rand() / (RAND_MAX) * 2 - 1;
  }

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      f[i][j] = (double)rand() / (RAND_MAX) * 2 - 1;
  }

  for (int i = 0; i < n; i++)
    y0[i] = (double)rand() / (RAND_MAX) * 2 - 1;
}

Vector f_t(Matrix f, double t, int rows) {
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++)
    c[i] = exp(-t) * cos(log(f[i][0] * f[i][0])) +
      sin(f[i][1]) * tan(t) + exp(-abs(f[i][2]));
  return c;
}

Vector runge_kutta(Matrix a, Vector y, Matrix f) {
  Vector k1 = new double[n];
  Vector k2 = new double[n];
  Vector k3 = new double[n];
  Vector k4 = new double[n];
  Vector temp = new double[n];
```

```
  k1 = sum(mul(mul(a, y, n, n), h, n), mul(f_t(f, x, n), h, n), n);
  k2 = sum(mul(mul(a, sum(y, mul(k1, 0.5, n), n), n, n), h, n),
    mul(f_t(f, x + h / 2, n), h, n), n);
  k3 = sum(mul(mul(a, sum(y, mul(k2, 0.5, n), n), n, n), h, n),
    mul(f_t(f, x + h / 2, n), h, n), n);
  k4 = sum(mul(mul(a, sum(y, k3, n), n, n), h, n),
    mul(f_t(f, x + h, n), h, n), n);

  temp = sum(y, mul(sum(sum(k1, k4, n),
    mul(sum(k2, k3, n), 2.0, n), n), 1.0 / 6, n), n);
  return temp;
}

Vector v(Matrix a, Vector y, Matrix f, double t, int rows, int cols) {
  Vector temp = new double[n];
  temp = sum(mul(a, y, rows, cols), f_t(f, t, rows), rows);
  return temp;
}

double norm(Vector a, int rows) {
  double temp = 0;

  for (int i = 0; i < rows; i++)
    temp += a[i] * a[i];
  return sqrt(temp);
}

int main() {
  printf("Input number of threads: ");
  scanf("%d", &size);
  omp_set_num_threads(size);
  printf("Input number of equations: ");
  scanf("%d", &n);
  printf("Input step: ");
  scanf("%lf", &h);
  eps = h / 100.0;
  printf("Input number of points: ");
  scanf("%d", &xk);
  k = n / size;
  double start, finish;

  x = 0;

  Matrix a = new Vector[n];
  for (int i = 0; i < n; i++)
    a[i] = new double[n];
  Matrix f0 = new Vector[n];
  for (int i = 0; i < n; i++)
    f0[i] = new double[m];

  Vector y0 = new double[n];
  set_data(a, y0, f0);
```

```
  Vector y1 = new double[n];
  y1 = runge_kutta(a, y0, f0);
  x += h;
  Vector y2 = new double[n];
  y2 = runge_kutta(a, y1, f0);
  x += h;
  Vector y = new double[n];
  y = runge_kutta(a, y2, f0);
  x += h;

#pragma omp parallel shared (y)
{
  Matrix b = new Vector[k];
  for (int i = 0; i < k; i++)
    b[i] = new double[n];
  Matrix f = new Vector[k];
  for (int i = 0; i < k; i++)
    f[i] = new double[m];

  int rank = omp_get_thread_num();
  for (int i = rank * k; i < (rank + 1) * k; i++) {
    b[i - rank * k] = set(a[i], n);
    f[i - rank * k] = set(f0[i], n);
  }

  Vector v3 = new double[k];
  Vector v2 = new double[k];
  Vector v1 = new double[k];
  Vector v0 = new double[k];
  Vector yp = new double[k];
  Vector yc = new double[k];
  Vector vn = new double[k];
  double norma;

#pragma omp master
{
  start = omp_get_wtime();
}

  v3 = v(b, y,  f, x, k, n);
  v2 = v(b, y2, f, x - h, k, n);
  v1 = v(b, y1, f, x - 2 * h, k, n);
  v0 = v(b, y0, f, x - 3 * h, k, n);

  while (x < xk * h) {
    norma = 100.0 * eps;

    yp = sum(y, mul(sum(sub(mul(v3, 55.0, k), mul(v2, 59.0, k), k),
      sub(mul(v1, 37.0, k), mul(v0, 9.0, k), k), k), h / 24, k), k);

    x += h;
```

```
    while (norma > eps) {
      vn = v(b, yp, f, x, k, n);
      yc = sum(y0, mul(sum(sum(mul(vn, 9.0, k), mul(v3, 19.0, k), k),
        sub(mul(v2, 5.0, k), v1, k), k), h / 24, k), k);
      norma = norm(sub(yc, yp, k), k);
      yp = set(yc, k);
    }

    v0 = set(v1, k);
    v1 = set(v2, k);
    v2 = set(v3, k);
    int rank = omp_get_thread_num();
#pragma omp barrier
    for (int i = rank * k; i < (rank + 1) * k; i++)
      y[i] = yp[i - rank * k];
    v3 = set(v(b, y, f, x, k, n), k);
  }

#pragma omp master
{
  finish = omp_get_wtime();
  double time = finish - start;

  printf("Elapsed time: %.3lf\n", time);
}
}
  return 0;
}
```

4 Программа, использующая MPI

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>

/*
  Attention!
  Sometimes program causes Segmentation Fault,
  reason isn't found
*/

typedef double* Vector;
typedef double** Matrix;

const int m = 3;
int n;
int k;
double h;
double x;
int xk;
double eps;
int size, rank;

Vector set(Vector b, int rows) {
  Vector c = new double[rows];
  for (int i = 0; i < rows; i++)
    c[i] = b[i];
  return c;
}

Vector mul(Matrix a, Vector b, int rows, int cols) {
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++)
      c[i] += a[i][j] * b[j];
  }
  return c;
}

Matrix mul(Matrix a, double b, int rows, int cols) {
  Matrix c = new Vector[rows];
  for (int i = 0; i < n; i++)
    c[i] = new double[cols];

  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++)
      c[i][j] = a[i][j] * b;
  }
```

```
  return c;
}

Vector mul(Vector a, double b, int rows) {
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++)
    c[i] = a[i] * b;
  return c;
}

Vector sum(Vector a, Vector b, int rows) {
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++)
    c[i] = a[i] + b[i];
  return c;
}

Vector sub(Vector a, Vector b, int rows) {
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++)
    c[i] = a[i] - b[i];
  return c;
}

void set_data(Matrix a, Vector y0, Matrix f) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      a[i][j] = (double)rand() / (RAND_MAX) * 2 - 1;
  }

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      f[i][j] = (double)rand() / (RAND_MAX) * 2 - 1;
  }

  for (int i = 0; i < n; i++)
    y0[i] = (double)rand() / (RAND_MAX) * 2 - 1;
}

Vector f_t(Matrix f, double t, int rows) {
  Vector c = new double[rows];

  for (int i = 0; i < rows; i++)
    c[i] = exp(-t) * cos(log(f[i][0] * f[i][0])) +
      sin(f[i][1]) * tan(t) + exp(-abs(f[i][2]));
  return c;
}

Vector runge_kutta(Matrix& a, Vector& y, Matrix& f) {
```

```
  Vector k1 = new double[n];
  Vector k2 = new double[n];
  Vector k3 = new double[n];
  Vector k4 = new double[n];
  Vector temp = new double[n];

  k1 = sum(mul(mul(a, y, n, n), h, n), mul(f_t(f, x, n), h, n), n);
  k2 = sum(mul(mul(a, sum(y, mul(k1, 0.5, n), n), n, n), h, n),
    mul(f_t(f, x + h / 2, n), h, n), n);
  k3 = sum(mul(mul(a, sum(y, mul(k2, 0.5, n), n), n, n), h, n),
    mul(f_t(f, x + h / 2, n), h, n), n);
  k4 = sum(mul(mul(a, sum(y, k3, n), n, n), h, n),
    mul(f_t(f, x + h, n), h, n), n);

  temp = sum(y, mul(sum(sum(k1, k4, n),
    mul(sum(k2, k3, n), 2.0, n), n), 1.0 / 6, n), n);
  return temp;
}

Vector v(Matrix a, Vector y, Matrix f, double t, int rows, int cols) {
  Vector temp = new double[n];
  temp = sum(mul(a, y, rows, cols), f_t(f, t, rows), rows);
  return temp;
}

double norm(Vector a, int rows) {
  double temp = 0;

  for (int i = 0; i < rows; i++)
    temp += a[i] * a[i];
  return sqrt(temp);
}

int main(int argc, char** argv) {
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Status status;

  Vector y0 = new double[n];
  Vector y = new double[n];
  Vector y1 = new double[n];
  Vector y2 = new double[n];
  double start, finish;

  if (rank == 0) {
    printf("Input number of equations: ");
    scanf("%d", &n);
    printf("Input step: ");
    scanf("%lf", &h);
    eps = h / 100.0;
    printf("Input number of points: ");
```

```
  scanf("%d", &xk);
  k = n / size;
}

MPI_Barrier(MPI_COMM_WORLD);
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&h, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&eps, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&xk, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&k, 1, MPI_INT, 0, MPI_COMM_WORLD);

x = 0;
Matrix b = new Vector[n];
for (int i = 0; i < k; i++)
  b[i] = new double[n];
Matrix f = new Vector[n];
for (int i = 0; i < k; i++)
  f[i] = new double[m];

if (rank == 0) {
  Matrix a = new Vector[n];
  for (int i = 0; i < n; i++)
    a[i] = new double[n];
  Matrix f0 = new Vector[n];
  for (int i = 0; i < n; i++)
    f0[i] = new double[m];

  set_data(a, y0, f0);
  y1 = runge_kutta(a, y0, f0);
  x += h;
  y2 = runge_kutta(a, y1, f0);
  x += h;
  y = runge_kutta(a, y2, f0);
  x += h;

  for (int i = 1; i < size; i++) {
    for (int j = i * k; j < (i + 1) * k; j++) {
      MPI_Send(a[j], n, MPI_DOUBLE, i, j, MPI_COMM_WORLD);
      MPI_Send(f0[j], m, MPI_DOUBLE, i, j, MPI_COMM_WORLD);
    }
  }
  for (int j = 0; j < k; j++) {
    b[j] = set(a[j], n);
    f[j] = set(f0[j], m);
  }

  for (int i = 0; i < n; i++) {
    delete [] f0[i];
    delete [] a[i];
  }
  delete [] f0;
  delete [] a;
```

```
}
MPI_Barrier(MPI_COMM_WORLD);
MPI_Bcast(y0, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(y1, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(y2, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(y, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);

if (rank != 0) {
  for (int i = rank * k; i < (rank + 1) * k; i++) {
    Vector temp_b = new double[n];
    Vector temp_f = new double[m];
    MPI_Recv(temp_b, n, MPI_DOUBLE, 0, i, MPI_COMM_WORLD, &status);
    b[i - rank * k] = set(temp_b, n);
    MPI_Recv(temp_f, m, MPI_DOUBLE, 0, i, MPI_COMM_WORLD, &status);
    f[i - rank * k] = set(temp_f, m);
    delete [] temp_b;
    delete [] temp_f;
  }
}

Vector v3 = new double[k];
Vector v2 = new double[k];
Vector v1 = new double[k];
Vector v0 = new double[k];

Vector yp = new double[k];
Vector yc = new double[k];
Vector vn = new double[k];
Vector yn = new double[k];
double norma;
bool ft = true;

Vector temp_yn = new double[k];
MPI_Barrier(MPI_COMM_WORLD);

if (rank == 0)
  start = MPI_Wtime();
for (int point = 4; point < xk; point++) {
  if (ft) {
    v3 = v(b, y, f, x, k, n);
    v2 = v(b, y2, f, x - h, k, n);
    v1 = v(b, y1, f, x - 2 * h, k, n);
    v0 = v(b, y0, f, x - 3 * h, k, n);
    ft = false;
  }
  for (int i = 0; i < k; i++)
    yn[i] = y[rank * k + i];

  norma = 100.0 * eps;
  yp = sum(yn, mul(sum(sub(mul(v3, 55.0, k), mul(v2, 59.0, k), k),
    sub(mul(v1, 37.0, k), mul(v0, 9.0, k), k), k), h / 24, k), k);
```

```
    x += h;

    while (norma > eps) {
      vn = v(b, yp, f, x, k, n);
      yc = sum(yn, mul(sum(sum(mul(vn, 9.0, k), mul(v3, 19.0, k), k),
        sub(mul(v2, 5.0, k), v1, k), k), h / 24, k), k);
      norma = norm(sub(yc, yp, k), k);
      yp = set(yc, k);
    }

    yn = set(yc, k);
    v0 = set(v1, k);
    v1 = set(v2, k);
    v2 = set(v3, k);
    v3 = set(v(b, yn, f, x, k, n), k);

    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Allgather(yn, k, MPI_DOUBLE, y, k, MPI_DOUBLE, MPI_COMM_WORLD);
    printf("%d ended pt: %d\n", rank, point);
  }

  if (rank == 0) {
    finish = MPI_Wtime();
    double time = finish - start;

    printf("Elapsed time: %.3lf\n", time);
  }

  MPI_Finalize();
  return 0;
}
```

5 Оценка алгоритма

Время последовательного алгоритма для одного уравнения при реализации метода Адамса может быть описано следующим соотношением:

$$T_1 = s \cdot T_f + N \cdot \left[s^2 \cdot t_{\text{умн}} + s^2 \cdot t_{\text{сл}} + s \cdot T_f\right] + s \cdot t_{\text{умн}} + s \cdot t_{\text{сл}}$$

где T_f – время вычисления функции f (правой части исходного дифференциального уравнения); $t_{\text{умн}}$ – время выполнения операции одиночного умножения; $t_{\text{сл}}$ – время выполнения операции одиночного сложения; N – число итераций в методе. Для представленного алгоритма время выполнения на p процессорах без учета обменов и других накладных расходов задается соотношением:

$$T_p = T_f + N \cdot [s \cdot t_{\text{умн}} + s \cdot t_{\text{сл}} + T_f] + t_{\text{умн}} + t_{\text{сл}}$$

При подсчете коэффициента ускорения будем считать, что $t_{\text{сл}} = t_{\text{умн}} = t_*$ – любая арифметическая операция с плавающей точкой выполняется за одно и то же время независимо от вида операции. Соответственно, коэффициент ускорения равен

$$S_p = \frac{T_1}{T_p} = \frac{s \cdot T_f + N \cdot \left[2s^2 \cdot t_* + s \cdot T_f\right] + 2s \cdot t_*}{T_f + N \cdot [2s \cdot t_* + T_f] + 2 \cdot t_*}$$

Коэффициент ускорения для сложных функций правой части коэффициент ускорения практически равен числу процессоров:

$$S_p \approx \frac{s \cdot T_f + N \cdot s \cdot T_f}{T_f + N \cdot T_f} = s = p$$

Тогда показатель эффективности будет равен:

$$E_p = \frac{T_1(n)}{pT_p(n)} = \frac{1}{p} S_p(n) = 1$$

Оценим максимальное достижимое ускорение по закону Густавсона – Барсиса:

$$g = \frac{\tau(n)}{\tau(n) + \pi(n)/p} \tag{1}$$

здесь $\tau(n)$ и $\pi(n)$ есть времена последовательной и параллельной частей выполняемых вычислений, т.е.

$$T_1(n) = \tau(n) + \pi(n), \quad T_p = \tau(n) + \pi(n)/p$$

Преобразуем (1) к следующему виду:

$$g = \left(1 - \frac{p}{p+1}\right) \frac{T_1}{T_p} - \frac{p}{p+1}$$

Используя ранее полученное значение для ускорения, получим значение достижимого ускорения:

$$g = \left(p - \frac{p^2}{p+1}\right) - \frac{p}{p+1}$$

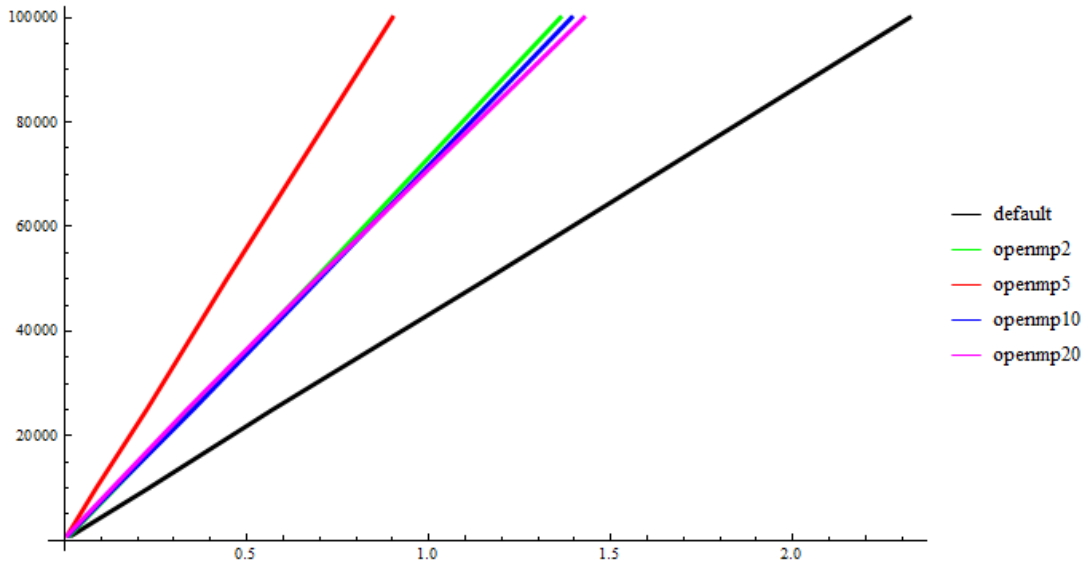
Используем формулу для ускорения масштабирования, подставляя в нее полученный ранее коэффициент g:

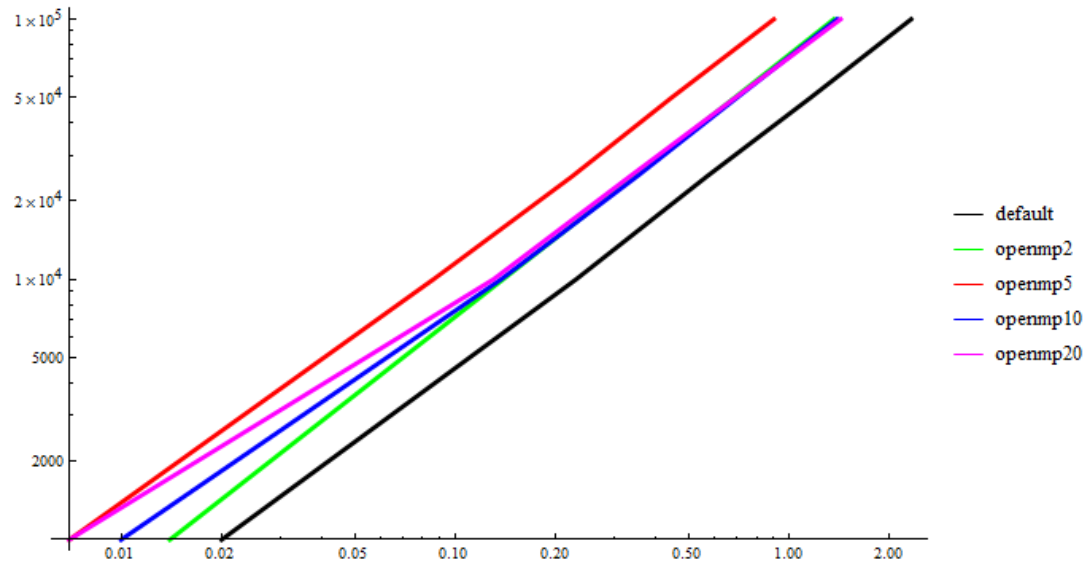
$$S_p = p + (1-p) \cdot g = p + p\,(1-p)\left(1 - \frac{p}{p+1}\right) - p\left(\frac{1-p}{p+1}\right)$$

6 Результаты

Неизменяемые параметры запуска программ: количество уравнений — 20; шаг — 10^{-6}; точность — 10^{-8}.

Кол-во точек	default	OpenMP 2	OpenMP 5	OpenMP 10	OpenMP 20
1000	0.020	0.014	0.007	0.010	0.007
10000	0.230	0.139	0.086	0.137	0.130
25000	0.570	0.346	0.225	0.352	0.335
50000	1.160	0.687	0.445	0.699	0.691
100000	2.320	1.361	0.900	1.392	1.426
1000000	38.370	18.705	14.076	19.328	20.051





7 Сертификаты с Intuit

ИНТУИТ
НАЦИОНАЛЬНЫЙ ОТКРЫТЫЙ УНИВЕРСИТЕТ

Сертификат

Серия О Регистрационный № 100838168

ВЫДАН

ЧЕЧЕТКИНУ ИЛЬЕ АЛЕКСАНДРОВИЧУ

В ТОМ, ЧТО ОН С 5 ЯНВАРЯ 2015 ПО 19 ЯНВАРЯ 2015 ПРОШЁЛ ОБУЧЕНИЕ
В НАЦИОНАЛЬНОМ ОТКРЫТОМ УНИВЕРСИТЕТЕ «ИНТУИТ» ПО КУРСУ
«ВВЕДЕНИЕ В ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ»
В ОБЪЕМЕ 72 ЧАСОВ

А.В. Шкред

Мо[illegible] января 2015 года

ИНТУИТ
НАЦИОНАЛЬНЫЙ ОТКРЫТЫЙ УНИВЕРСИТЕТ

Сертификат

Серия О Регистрационный № 100838148

ВЫДАН

ЧЕЧЕТКИНУ ИЛЬЕ АЛЕКСАНДРОВИЧУ

В ТОМ, ЧТО ОН С 5 ЯНВАРЯ 2015 ПО 19 ЯНВАРЯ 2015 ПРОШЁЛ ОБУЧЕНИЕ
В НАЦИОНАЛЬНОМ ОТКРЫТОМ УНИВЕРСИТЕТЕ «ИНТУИТ» ПО КУРСУ
«ОСНОВЫ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ С ИСПОЛЬЗОВАНИЕМ MPI»
В ОБЪЕМЕ 72 ЧАСОВ

А.В. Шкред

Мо[illegible] января 2015 года

Список литературы

[1] Высокопроизводительные вычисления на кластерах: Учебное пособие / Под ред. А. В. Старченко. — Томск: Изд-во Том. ун-та, 2008. — 198 с.

[2] Антонов, А. С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие. / А. С. Антонов, НИВЦ МГУ. — Москва: Изд-во МГУ, 2009. — 77 с.

[3] Заусаев, А. Ф. Применение разностных методов для решения обыкновенных дифференциальных уравнений: лабораторный практикум / А. Ф. Заусаев, В. Е. Зотеев. — Самара, Самар. гос. техн. ун-т, 2010. — 34 с.