МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Павлович Владислав Викторович

МЕТОДЫ ЧИСЛЕННОГО АНАЛИЗА

Отчёт по лабораторной работе №1
студента 3 курса 3 группы

Преподаватель:
Полещук Максим Александрович

Минск, 2016

Содержание

1 Задание 1

Классическим методом Рунге-Кутты четвёртого порядка точности найти приближенное решение задачи Коши дифференциального уравнения:

$$u' = -(g + 0.05s)x^{g-1+0.05s}u\sin(x^{g+0.05s}), u(0) = e, g = 3, s = 2,$$
$$u' = -3.1x^{2.1}u\sin(x^{3.1})$$

2 Теория

Была использована следующая совокупность формул для метода Рунге-Кутты четвёртого порядка точности:

$$k_1 = hf(x, y),$$
$$k_2 = hf\left(x + \frac{h}{2}, y + \frac{k_1}{2}\right),$$
$$k_3 = hf\left(x + \frac{h}{2}, y + \frac{k_2}{2}\right),$$
$$k_4 = hf(x + h, y + k_3),$$
$$y(x + h) = y(x) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

Новое h вычисляется по следующей формуле:

$$h_\varepsilon = \frac{h}{2}\sqrt{\frac{(2^k - 1)\varepsilon}{|y_{h/2} - y_h|}}$$

3 Результаты

Число итераций: $N = 2601$, максимальная ошибка: $E_{gl} = 3.83854e - 05$.

i	h	|u(x)-y(xn)|	$\left\|\frac{u(x) - y(xn)}{u(x)}\right\|$
1	0.0005	0	0
2	0.0005	0	0
3	0.0005	0	0
4	0.0005	0	0
5	0.0005	0	0
6	0.0005	0	0
7	0.0005	0	0
8	0.0005	0	0
9	0.0005	0	0
10	0.0005	0	0
11	0.0005	0	0
12	0.0005	0	0
13	0.0005	0	0
14	0.0005	0	0
15	0.0005	0	0
16	0.0005	0	0
17	0.0005	0	0
18	0.0005	0	0
19	0.0005	0	0
20	0.0005	0	0
N-19	0.000499964	3.31998e-05	6.01572e-05
N-18	0.000499964	3.32594e-05	6.03937e-05
N-17	0.000499964	3.31998e-05	6.04139e-05
N-16	0.000499964	3.32594e-05	6.06511e-05
N-15	0.000499964	3.31998e-05	6.0671e-05
N-14	0.000499964	3.32594e-05	6.09089e-05
N-13	0.000499964	3.32594e-05	6.1038e-05
N-12	0.000499964	3.31998e-05	6.10576e-05
N-11	0.000499964	3.31998e-05	6.11867e-05
N-10	0.000499964	3.3319e-05	6.1536e-05
N-9	0.000499964	3.3319e-05	6.16658e-05
N-8	0.000499964	3.3319e-05	6.17957e-05
N-7	0.000499964	3.3319e-05	6.19257e-05
N-6	0.000499964	3.33786e-05	6.21668e-05
N-5	0.000499964	3.32594e-05	6.20747e-05
N-4	0.000499964	3.3319e-05	6.23163e-05
N-3	0.000499964	3.33786e-05	6.25583e-05
N-2	0.000499964	3.3319e-05	6.25771e-05
N-1	0.000499964	3.33786e-05	6.28199e-05
N-0	2.98023e-06	3.3319e-05	6.27085e-05

i	u_i	$f(x_i)$	Δ_i
1	2.71828	2.71828	0
2	2.71828	2.71828	0
3	2.71828	2.71828	0
4	2.71828	2.71828	0
5	2.71828	2.71828	0
6	2.71828	2.71828	0
7	2.71828	2.71828	0
8	2.71828	2.71828	0
9	2.71828	2.71828	0
10	2.71828	2.71828	0
11	2.71828	2.71828	0
12	2.71828	2.71828	0
13	2.71828	2.71828	0
14	2.71828	2.71828	0
15	2.71828	2.71828	0
16	2.71828	2.71828	0
17	2.71828	2.71828	0
18	2.71828	2.71828	0
19	2.71828	2.71828	0
20	2.71828	2.71828	0
N-19	0.55185	0.55185	0
N-18	0.550676	0.550676	0
N-17	0.549506	0.549506	0
N-16	0.54834	0.54834	0
N-15	0.547177	0.547177	0
N-14	0.546018	0.546018	0
N-13	0.544863	0.544863	0
N-12	0.543712	0.543712	3.97364e-09
N-11	0.542565	0.542565	3.97364e-09
N-10	0.541422	0.541422	0
N-9	0.540282	0.540282	0
N-8	0.539147	0.539147	0
N-7	0.538015	0.538015	0
N-6	0.536887	0.536887	0
N-5	0.535763	0.535763	3.97364e-09
N-4	0.534643	0.534643	0
N-3	0.533526	0.533526	0
N-2	0.532414	0.532414	0
N-1	0.531305	0.531305	0
N-0	0.531298	0.531298	3.97364e-09

4 Задание 2

Экстраполяционным методом Адамса четвёртого порядка точности найти приближённое решение задачи Коши дифференциального уравнения из задания 1 на сетке узлов с фиксированным шагом $h = 5 \cdot 10^{-3}$ построив начало таблицы с использованием метода Рунге-Кутты.

5 Теория

Была использована следующая формула для экстраполяционного метода Адамса с $q = 3$:

$$y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}).$$

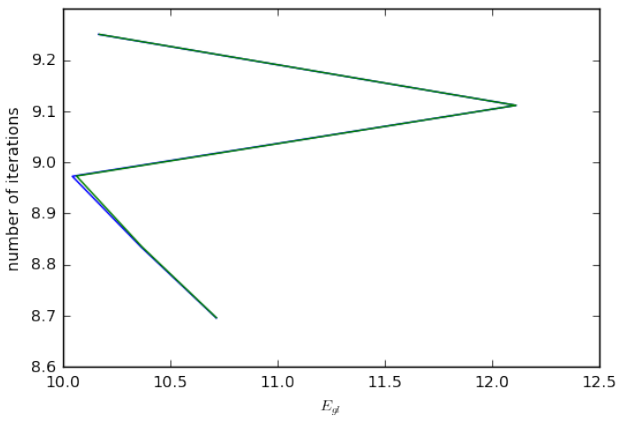
6 Результаты

1. Максимальная глобальная ошибка: $3.82662e - 05$.

2. $251/(720 * 5^5) * max|u^{(5)}| = 0.0081791$.

3. Приближённые значения y_n:

i	y_i
1	2.71828
2	2.71828
3	2.71828
4	2.71828
5	2.71828
6	2.71828
7	2.71828
8	2.71828
9	2.71828
10	2.71828
11	2.71828
12	2.71828
13	2.71828
14	2.71828
15	2.71828
16	2.71828
17	2.71828
18	2.71828
19	2.71828
20	2.71828
N-19	0.55185
N-18	0.550676
N-17	0.549506
N-16	0.548339
N-15	0.547177
N-14	0.546018
N-13	0.544863
N-12	0.543712
N-11	0.542565
N-10	0.541422
N-9	0.540282
N-8	0.539146
N-7	0.538015
N-6	0.536887
N-5	0.535763
N-4	0.534642
N-3	0.533526
N-2	0.532413
N-1	0.531305
N-0	0.5302



7 Исходный код

```
#include <iostream>
#include <cmath>
#include <cassert>
#include <vector>

using namespace std;

typedef float datatype;

constexpr datatype kAlpha = 0.8;

datatype F(datatype x, datatype y) {
  return -3.1 * pow(x, 2.1) * y * sin(pow(x, 3.1));
}

datatype RealF(datatype x) {
  return exp(cos(pow(x, 3.1)));
}

vector<datatype> RungeKutta(datatype h, datatype a, datatype b, datatype eta) {
  datatype x = a;
  vector<datatype> ay;
  ay.push_back(eta);
  while (x <= b) {
    datatype y = ay.back();
    datatype k1 = h * F(x, y);
    datatype k2 = h * F(x + h / 2.0, y + k1 / 2.0);
    datatype k3 = h * F(x + h / 2.0, y + k2 / 2.0);
    datatype k4 = h * F(x + h, y + k3);
    datatype newy = y + (k1 + 2.0 * k2 + 2.0 * k3 + k4) / 6.0;
    ay.push_back(newy);
    x += h;
  }
  return ay;
}

datatype CalculateRungeKutta(datatype x, datatype prevy, datatype h) {
  datatype k1 = h * F(x, prevy);
  datatype k2 = h * F(x + h / 2.0, prevy + k1 / 2.0);
  datatype k3 = h * F(x + h / 2.0, prevy + k2 / 2.0);
  datatype k4 = h * F(x + h, prevy + k3);
  datatype newy = prevy + (k1 + 2.0 * k2 + 2.0 * k3 + k4) / 6.0;
  return newy;
}

datatype GetAbsError(datatype x, datatype y, datatype h) {
  datatype res1 = CalculateRungeKutta(x, y, h);
  datatype tmp = CalculateRungeKutta(x, y, h * 0.5);
  datatype res2 = CalculateRungeKutta(x + h * 0.5, tmp, h * 0.5);
  return abs(res1 - res2);
}

pair<vector<datatype>, vector<datatype>> AdaptiveRungeKutta(
  datatype initialh, datatype a, datatype b, datatype eta,
  datatype eps, int& niterations) {
  niterations = 0;
  datatype x = a;
  vector<datatype> ay;
  vector<datatype> ax;
  ay.push_back(eta);
  ax.push_back(0.0);
  datatype h = initialh;
  while (x < b) {
    datatype y = ay.back();
    int i = 0;
    datatype error;
    for (i = 0; i < 5; ++i) {
      error = GetAbsError(x, y, h);
      ++niterations;
      if (error < eps) break;
      // 2^k - 1 = 15
      datatype delta = pow(15.0 * eps / error, 0.2);
      h = h * kAlpha * delta * 0.5;
    }
```

```
    assert(error <= eps);

    if (x + h > b) {
      h = b - x;
    }
    ay.push_back(CalculateRungeKutta(x, y, h));
    x += h;
    ax.push_back(x);
  }
  return make_pair(ax, ay);
}

datatype GetRungeKuttaAbsoluteError(datatype eps, int& niterations) {
  auto v = AdaptiveRungeKutta(0.2, 0.0, 1.3, exp(1.0), eps, niterations);
  datatype error = 0.0;
  for (int i = 0; i < v.first.size(); ++i) {
    error = max(error, abs(v.second[i] - RealF(v.first[i])));
  }
  return error;
}

datatype GetRungeKuttaRelativeError(datatype h) {
  auto v = RungeKutta(h, 0.0, 1.3, exp(1.0));
  datatype x = 0.0;
  datatype error = 0.0;
  for (auto y : v) {
    error = max(error, abs((y - RealF(x)) / RealF(x)));
    x += h;
  }
  return error;
}

class FDiffer {
 public:
  FDiffer(const vector<datatype>& ty, datatype ta, datatype th)
      : y(ty), a(ta), h(th) {}
  datatype Get(int i) {
    return F(a + h * i, y[i]);
  }
 private:
  vector<datatype> y;
  datatype a;
  datatype h;
};

vector<datatype> Adams(datatype h, datatype a, datatype b,
                       datatype eta, int& niterations) {
  auto ty = RungeKutta(h, a, b, eta);
  vector<datatype> y(ty);
  int n = y.size();
  FDiffer d(ty, a, h);
  for (int i = 3; i + 1 < n; ++i) {
    y[i + 1] = y[i] + (55.0 * d.Get(i) - 59.0 * d.Get(i - 1) +
     37.0 * d.Get(i - 2) - 9.0 * d.Get(i - 3)) * h / 24.0;
    ++niterations;
  }
  return y;
}

datatype GetAdamsError(datatype h) {
  int niterations;
  auto v = Adams(h, 0.0, 1.3, exp(1.0), niterations);
  datatype x = 0.0;
  datatype error = 0.0;
  for (auto y : v) {
    error = max(error, abs(y - RealF(x)));
    x += h;
  }
  return error;
}

void PrintRungeKutta() {
  cout << "Runge Kutta method: " << endl;
  datatype h = 0.2;
  datatype eps = 0.001;
  int niterations;
```

```
auto res = AdaptiveRungeKutta(h, 0.0, 1.3, exp(1.0), eps, niterations);
datatype max_error = 0.0;
int max_error_id = -1;
cout << "Size: " << res.first.size() << " Iterations: " << niterations << endl;
for (int i = 0; i < res.first.size(); ++i) {
  datatype cur_error = abs(RealF(res.first[i]) - res.second[i]);
  if (cur_error > max_error) {
    max_error = cur_error;
    max_error_id = i;
  }
}
cout << max_error << endl;
//cout << "Accepted h: " << h << " (" << niterations << " iterations, " << GetRelError(h) << "
    error)" << endl;
auto v1 = RungeKutta(h, 0.0, 1.3, exp(1.0));
auto v2 = RungeKutta(h / 2.0, 0.0, 1.3, exp(1.0));
for (int i = 0; i < v1.size(); ++i) {
  cout << h * i << " & ";
}
cout << endl;
for (auto y : v1) {
  cout << y << " & ";
}
cout << endl;
for (int i = 0; i < v2.size(); ++i) {
  cout << h * i / 2 << " & ";
}
cout << endl;
for (auto y : v2) {
  cout << y << " & ";
}
cout << endl;
cout << "Absolute error: " << endl;
datatype curx = 0.0;
for (int i = 0; i < v1.size(); ++i) {
  cout << " & " << curx;
  curx += h;
}
cout << endl;
curx = 0.0;
for (auto y : v1) {
  cout << " & " << abs(y - RealF(curx));
  curx += h;
}
cout << endl;
cout << "Relative error: " << endl;
curx = 0.0;
for (int i = 0; i < v1.size(); ++i) {
  cout << " & " << curx;
  curx += h;
}
cout << endl;
curx = 0.0;
for (auto y : v1) {
  cout << " & " << abs((y - RealF(curx)) / RealF(curx));
  curx += h;
}
cout << endl;
cout << "Major error summand: " << endl;
curx = 0.0;
for (int i = 0; i < v1.size() && i * 2 < v2.size(); ++i) {
  cout << " & " << curx;
  curx += h;
}
cout << endl;
for (int i = 0; i < v1.size() && i * 2 < v2.size(); ++i) {
  cout << " & " << abs(v1[i] - v2[i * 2]) / 15.0;
  curx += h;
}
cout << endl << endl;

cout << "Errors graph: ";
cout << endl;
constexpr int total_q = 5;
vector<int> niters;
for (int i = 1; i <= total_q; ++i) {
```

```
    datatype eps = pow(5, datatype(i) / total_q) * 0.00001;
    cout << eps << "-";
    int niterations;
    cout << GetRungeKuttaAbsoluteError(eps, niterations) << ", ";
    niters.push_back(niterations);
  }
  cout << endl;
  for (auto t : niters) {
    cout << t << ", ";
  }
  cout << endl;
}

void PrintAdams() {
  cout << "Adams method: " << endl;
  int niterations;
  auto data = Adams(0.005, 0.0, 1.3, exp(1.0), niterations);
  cout << "Adams iterations: " << niterations << endl;

  for (int i = 0; i < 20; ++i) {
    cout << 0.005 * i << " & ";
  }
  cout << endl;
  for (int i = 0; i < 20; ++i) {
    cout << data[i] << " & ";
  }
  cout << endl;

  for (int i = data.size() - 20; i < data.size(); ++i) {
    cout << 0.005 * i << " & ";
  }
  cout << endl;
  for (int i = 0; i < 20; ++i) {
    cout << data[i] << " & ";
  }
  cout << endl;
  for (int i = data.size() - 20; i < data.size(); ++i) {
    cout << data[i] << " & ";
  }
  cout << endl;

  constexpr int total_q = 5;
  for (int i = 0; i < total_q; ++i) {
    datatype h = pow(5, datatype(i) / total_q) * 0.0001;
    cout << 1.3 / h * 4 << ", ";
  }
  cout << endl;
  for (int i = 0; i < total_q; ++i) {
    datatype h = pow(5, datatype(i) / total_q) * 0.0001;
    cout << GetAdamsError(h) << ", ";
  }
  cout << endl;
}

int main() {
  PrintRungeKutta();
  PrintAdams();
  return 0;
}
```