Федеральное государственное бюджетное образовательное учреждение «Сибирский государственный университет телекоммуникаций и информатики»

Кафедра ПМ и К

КУРСОВАЯ РАБОТА

По дисциплине «Вычислительная математика»
Вариант 6

Выполнил:
студент гр. ИВ-621
Дьяченко Д.В.
Проверил:
Чирихин К. С.

Новосибирск, 2018

Постановка задачи

Решить краевую задачу методом Рунге-Кутта II порядка с усреднением по производной.

$$\begin{cases} (y'')^5 - \cos(\mathrm{x}) * y'' = \sin(x) + 5 * \ln(\mathrm{x}) * y' + y * (x + 3) \\ y(0) = 3 \\ y'(1) = 2 \end{cases}$$

Построить графики функции y(x) и кубического сплайна S(x) (интерполяция по точкам x=0; 0.2;0.4; 0.6; 0.8; 1.0). Найти интеграл

$$\int_0^1 y(x)dx$$

Описание алгоритма

Этапы решения краевой задачи:

1. С помощью метода стрельбы находим значение первой производной.

2. Решаем задачу Коши методом Рунге-Кутта II порядка с усреднением по времени.

Метод стрельбы:

Выбираем параметры: a – y(0) из краевой задачи, а b – произвольно; для того, чтобы найти отрезок, в котором будет искомое значение; корректируем исходные параметры в зависимости от перелёта или недолёта (y(a) – y1 > 0 или y(a) – y1 < 0 соответственно). Как только по a перелет, а по b недолет, останавливаем корректировку, на данном этапе отрезок найден. Для нахождения первой производной остается решить нелинейное уравнение любым известным способом, в частности методом бисекции: y(b) = y2, где y(b) – решение задачи Коши.

Задача Коши:

Применяем метод Рунге-Кутта:

$$\begin{cases} y_{i+1} = y_i + \dfrac{h}{2}\left(f(x_i, y_i) + f(x_i + h, y_{i+1}^*)\right) \\ \text{где} \quad y_{i+1}^* = y_i + \dfrac{h}{2} f(x_i, y_i) \end{cases}$$

Так как по условию дано уравнение, которое не может быть разрешено относительно старшей производной, то каждый раз решаем нелинейное уравнение относительно старшей производной.

Интерполяция:

Для вычисления кубического сплайна на заданной сетке будем использовать формулу:

$$S_i(x) = M_{i-1}\frac{(x_i - x)^3}{6h_i} + M_i\frac{(x - x_{i-1})^3}{6h_i} + \left(y_{i-1} - \frac{M_{i-1}}{6}h_i^2\right)\frac{x_i - x}{h_i} + \left(y_i - \frac{M_i}{6}h_i^2\right)\frac{x - x_{i-1}}{h_i}$$

В данной задаче подразумевается интерполяция естественным кубическим сплайном, т. е. $M0 = Mn$. Шаг для сетки: h = const. Чтобы найти другие Mi составим СЛАУ; получим трехдиагональную матрицу, решаем систему методом прогонки и вычисляем значения сплайна в текущей точке.

Вычисление интеграла:

Численное интегрирование по формуле Симпсона; отрезок [a, b] разбивается на N = 2n частей:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{n-1} f(x_{2j}) + \sum_{j=1}^{n} f(x_{2j-1}) + f(x_n)\right],$$

где $h = \frac{b-a}{N}$ – шаг, а $x_j = a + ih$ – узлы интегрирования.

Результат работы программы

Решение краевой задачи:

i	0	1	2	3	4	5
x_i	0.0	0.2	0.4	0.6	0.8	1.0
y_i	3.000	2.660	2.395	2.199	2.069	2.003
y'_i	-1.907	-1.507	-1.149	-0.813	-0.490	-0.176

Интерполяция:

Рисунок 1 Интерполяция Кубическим Сплайном

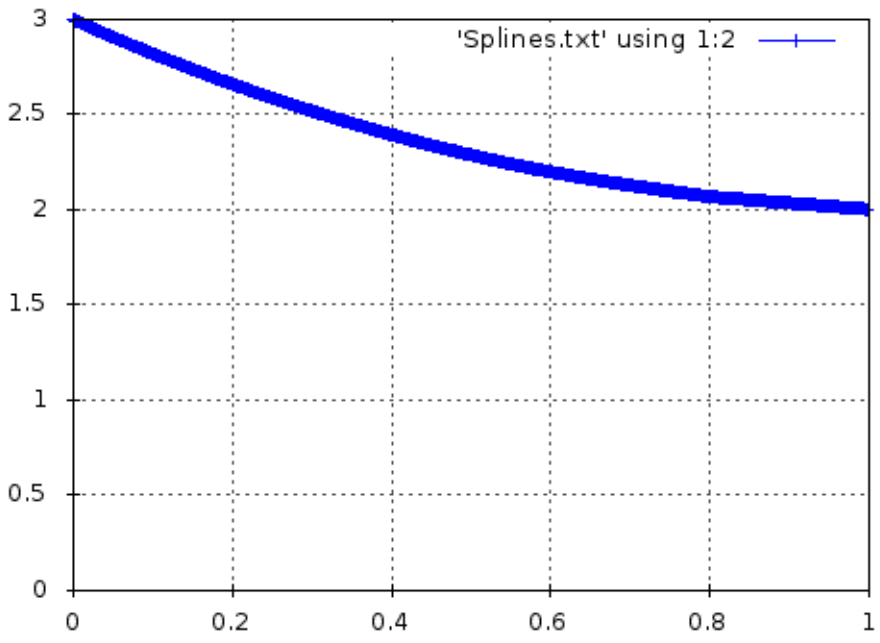


Рисунок 2 Результат функции Рунге-Кутты для y(x)

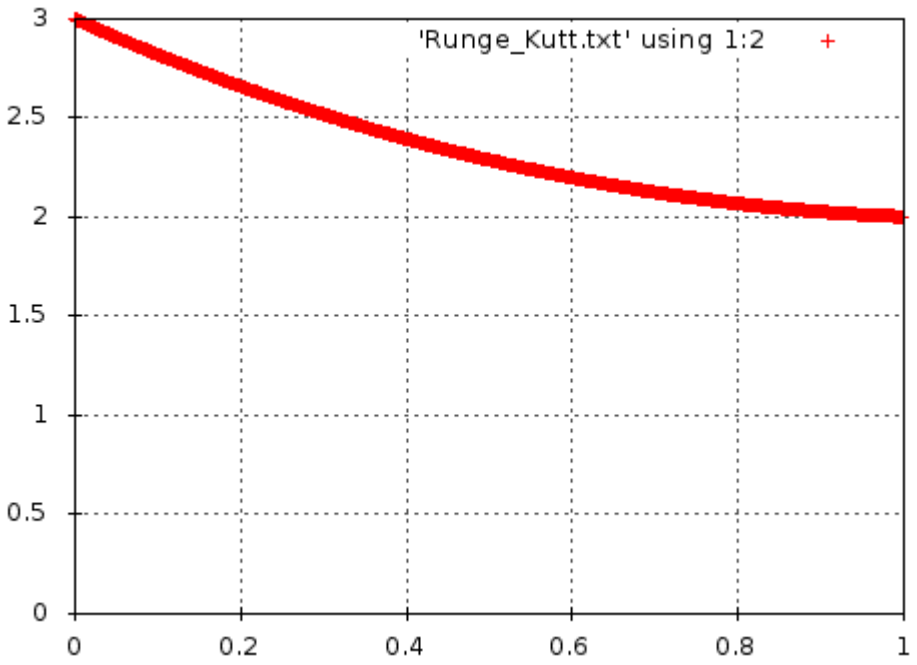
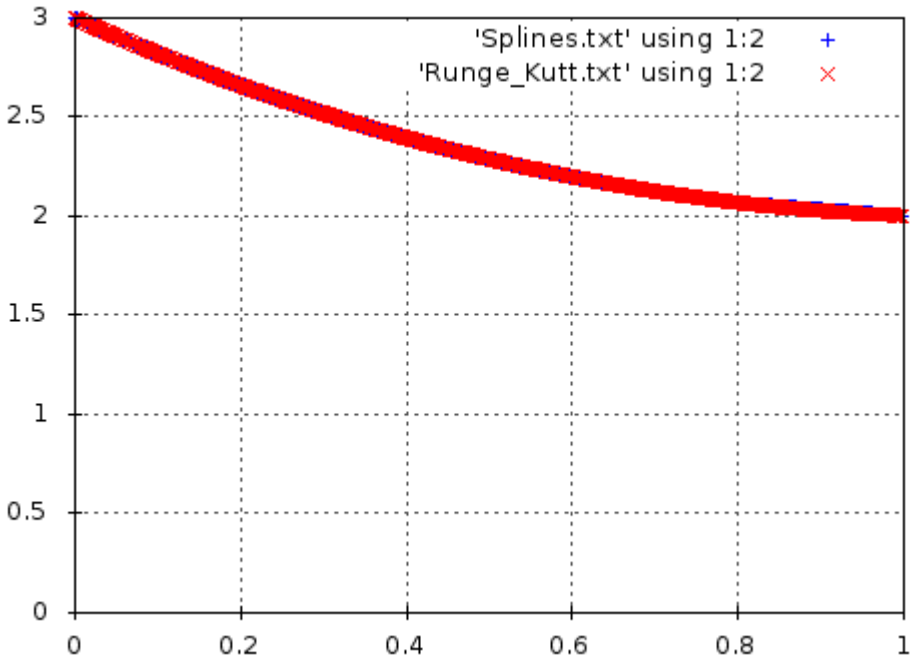


Рисунок 3 Сравнение результата функции Рунге-Кутты для y(x) и интерполяции Кубическим Сплайном



<u>Численное интегрирование:</u>

$$\int_0^1 y(x)dx \approx 2.36764$$

Заключение

В рамках курсовой работы была решена краевая задача, результаты которой удовлетворяют заданным граничным условиям в концах интервала. Проведена интерполяция кубическими сплайнами, построен график сеточной функции, который иллюстрирует решение дифференциального уравнения. По формуле Симпсона вычислено приближенное значение интеграла для заданной подынтегральной функции.

Листинг

main.cpp

```
#include <cstdlib>
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;

double eps = 1e-4;

double diff(double x, double y, double D1, double D2)
{
	if (x == 0) {
		x = 0.0001;
	}
	return pow(D2, 5) - cos(x) * D2 - sin(x) - 5 * log(x) * D1 - y * (x + 3);
}

double *addition_of_vectors(double *v1, double *v2, int n)
{
	double *v3 = new double[n];

	for (int i = 0; i < n; i++) {
		v3[i] = v1[i] + v2[i];
	}

	return v3;
}

double *multiple_dig_by_vector(double a, double *v, int n)
{
	double *v2 = new double[n];

	for (int i = 0; i < n; i++) {
		v2[i] = a * v[i];
	}

	return v2;
}

double *f(double x, double *y)
{
	double a = 0, b = 2;
	double fa = 0, fb = 0;

	do {
		fa = diff(x, y[0], y[1], a--);
		fb = diff(x, y[0], y[1], b++);
	} while (fa * fb > 0);

	double c = 0;

	while(fabs(b - a) >= eps) {
		c = (a + b) / 2;
```

```
                if(diff(x, y[0], y[1], a) * diff(x, y[0], y[1], c) < 0)
                        b = c;
                else if (diff(x, y[0], y[1], c) * diff(x, y[0], y[1], b) < 0)
                        a = c;
        }

        double *tmp_y = new double[2];
        tmp_y[0] = y[1];
        tmp_y[1] = (a + b) / 2;

        return  tmp_y;
}

double *Runge_Kutt(double a, double b, double h, double *y0)
{
        double *tmp_y;
        double *y = y0;
        for (double i = a + h; i <= b; i += h) {
                tmp_y = addition_of_vectors(y, multiple_dig_by_vector(h, f(i, y), 2), 2);
                y = addition_of_vectors(y, multiple_dig_by_vector(h / 2, addition_of_vectors(f(i, y), f(i + h, tmp_y), 2), 2), 2);
        }

        return y;
}

double **dbl_counting_Runge(double a, double b, double h, double Eps, double *y, int *count, double *h_)
{
        double **y_prev;
        double **y_cur;
        double max;
        int count_elem;
        do {
                count_elem = (b - a) / h;
                y_prev = new double*[count_elem];
                double t = a;
                for (int i = 0; fabs(t - b) >= 1e-8; i++, t += h) {
                        y_prev[i] = new double[2];
                        y_prev[i] = Runge_Kutt(a, t, h, y);
                }

                h /= 2;
                count_elem = (b - a) / h;
                t = a;
                y_cur = new double*[count_elem];
                for (int i = 0; fabs(t - b) >= 1e-8; i++, t += h) {
                        y_cur[i] = new double[2];
                        y_cur[i] = Runge_Kutt(a, t, h, y);
                }

                double mod = 0.0;
                max = 0;
                int j = 0;
                int del = count_elem;
                int i;
                for (i = 0; j < del + (del % 2 ? 1 : 0); i++, j += 2) {
                        mod = fabs(y_prev[i][0] - y_cur[j][0]);
                        if (mod > max) {
                                max = mod;
                        }
                }
        } while (fabs(max) >= Eps);

        if (count) {
                *count = count_elem;
        }

        if (h_) {
                *h_ = h;
        }

        for (int i = 0, j = 0; i < count_elem; i += 2, j++) {
                y_cur[i][0] -= (y_cur[i][0] - y_prev[j][0]) / 3;
                y_cur[i][1] -= (y_cur[i][1] - y_prev[j][1]) / 3;
```

```
	}

	return y_cur;
}

double MethodShooting(double x0, double x1, double y0, double y1, double h)
{
	double al = 1.0;
	double bt = 0.0;
	double fa = 0.0;
	double fb = 0.0;

	double tmp[2];
	tmp[0] = y0;
	double *vt;
	do {
		tmp[1] = al;
		vt = Runge_Kutt(x0, x1, h, tmp);
		fa = vt[0] - y1;
		tmp[1] = bt;
		vt = Runge_Kutt(x0, x1, h, tmp);

		fb = vt[0] - y1;
		al -= h;
		bt += h;
	} while (fa * fb > 0);


	double c = 0.0;
	double *tmp1;
	double *tmp2;
	double *tmp4;
	do {

		tmp[1] = al;
		tmp1 = Runge_Kutt(x0, x1, h, tmp);
		tmp[1] = c;
		tmp2 = Runge_Kutt(x0, x1, h, tmp);
		tmp[1] = bt;
		tmp4 = Runge_Kutt(x0, x1, h, tmp);

		if ((((tmp1[0] - y1) * (tmp2[0] - y1)) < 0)) {
			bt = c;
		} else if ((((tmp2[0] - y1) * (tmp4[0] - y1)) < 0)) {
			al = c;
		}

		c = (al + bt) / 2;
	} while (fabs(y1 - (tmp1[0] + tmp2[0] + tmp4[0]) / 3) > 1e-4);

	return ((al + bt) / 2);
}

double Form_of_Simpson(double a, double b, double h, double *y0)
{
	double res = 0.0;

	double j = a;

	double *tmp;

	for (int i = 1; j <= b - h; i++, j += h) {
		tmp = Runge_Kutt(a, j, h, y0);
		res += (i % 2 ? 4 : 2) * tmp[0];
	}

	tmp = Runge_Kutt(a, 0, h, y0);
	res += tmp[0];
	tmp = Runge_Kutt(a, b, h, y0);
	res += tmp[0];
	res = (res * h) / 3;

	return res;
```

```
}

double double_counting(double (*method)(double, double, double, double *), double a, double b, double h, double Eps, double *y)
{
	h = b - a;
	double prev = method(a, b, h, y);
	h /= 2;
	double cur = method(a, b, h, y);

	int count = 0;
	while (fabs(prev - cur) >= Eps) {
		prev = cur;
		h /= 2;
		cur = method(a, b, h, y);

		count++;
	}
	printf("Count iteration = %d\n", count);

	return cur;
}


double dbl_count_D1(double x0, double x1, double y0, double y1, double h, double Eps)
{
	double prev;
	double cur;
	do {
		prev = MethodShooting(x0, x1, y0, y1, h);
		h /= 2;
		cur = MethodShooting(x0, x1, y0, y1, h);
	} while (fabs(prev - cur) >= Eps);

	return cur;
}

double h_i(int i, const pair<double, double> * xy)
{
	return xy[i].first - xy[i-1].first;
}

double b_i(int i, const pair<double, double> * xy)
{
	return (h_i(i, xy) + h_i(i+1, xy)) / 3;
}

double g_i(int i, const pair<double, double> * xy)
{
	return h_i(i, xy) / 6;
}

double d_i(int i, const pair<double, double> * xy)
{
	return ((xy[i+1].second - xy[i].second) / h_i(i+1, xy) -
					(xy[i].second - xy[i-1].second) / h_i(i, xy));
}

vector <double> thomas_come_on(vector <double>  a, vector <double> b,
                                                                                vector <double>  g, vector <double>  d,
int n)
{
	vector <double> c(n);
	vector <double> p(n);
	vector <double> q(n);

	for(int i = 0; i < n; i++) {
		if (i == 0) {
			p[i] = -g[i] / b[i];
			q[i] =  d[i] / b[i];
			continue;
		}
		p[i] = g[i] / (-b[i] - a[i] * p[i-1]);
		q[i] = (a[i] * q[i-1] - d[i])/(-b[i] - a[i] * p[i-1]);
```

```
	}

	for(int i = n - 1 ; i >= 0; i--) {
		if( i == n) {
			c[i] = (a[i] * q[i-1] - d[i]) / (-b[i] - a[i] * p[i-1]);
			continue;
		}
		c[i] = p[i] * c[i + 1] + q[i];
	}
	return c;
}

int binary_search (pair<double, double> * xy, double x, int n)
{
	int idx = 0;
	if(x <= xy[0].first) {
		idx = 1;
	}
	else if (x >= xy[n-1].first) {
		idx = n-1;
	}
	else {
		int i = 0, j = n-1;
		while(i + 1 < j) {
			int k = i + (j - i) / 2;
			if (x <= xy[k].first) {
				j = k;
			} else {
				i = k;
			}
		}
		idx = j;
	}
	return idx;
}

double spline_eval(int i, double * M, pair<double, double> * v, double x)
{
	double s1 = M[i-1] * pow(v[i].first - x, 3) / (6 * h_i(i, v));

	double s2 = M[i] * pow(x - v[i-1].first, 3) / (6 * h_i(i, v));

	double s3 = (v[i-1].second - M[i-1] * pow(h_i(i, v), 2) / 6) *
			(v[i].first - x) / h_i(i, v);

	double s4 = (v[i].second - M[i] * pow(h_i(i, v), 2) / 6) *
			(x - v[i-1].first) / h_i(i, v);

	return s1 + s2 + s3 + s4;
}

double cubic(double x, int n, pair<double, double> * v)
{
	int i = 0;
	double s = 0;
	vector <double> M(n - 2);
	vector <double> a(n - 2), b(n - 2), g(n - 2), d(n - 2);

	a[0] = g[n-3] = 0;
	for(int i = 1; i < n - 1; i++)
		b[i - 1] = b_i(i, v);
	for(int i = 2; i < n - 1; i++)
		a[i - 1] = g_i(i, v);
	for(int i = 0; i < n - 3; i++)
		g[i] = g_i(i+1, v);
	for(int i = 1; i < n - 1; i++)
		d[i-1] = d_i(i, v);

	M = thomas_come_on(a, b, g, d, n-2);
	i = binary_search(v, x, n);
	s = spline_eval(i, M.data(), v, x);

	return s;
```

```
}

int main()
{
	double a = 0.0;
	double b = 1.0;
	double h = 0.2;
	int size = (a + b) / h;

	double y[size + 1];

	double x0 = 0.0;
	double y0 = 3.0;
	double x1 = 1.0;
	double y1 = 2.0;

	double D1 = dbl_count_D1(x0, x1, y0, y1, h, 1e-3);
	printf("D1 = %.3lf\n", D1);

	FILE *out = fopen("Runge_Kutt.txt", "w");
	printf("x\ty(x)\ty\'(x)\n");
	double Eps = 1e-3;

	double tmp[2] = { y0, D1 };
	int count_elem;
	double h_;
	double **yt = dbl_counting_Runge(a, b, h, Eps, tmp, &count_elem, &h_);

	int i_count[6];
	i_count[0] = 0;
	double x[6] = { 0.0, 0.2, 0.4, 0.6, 0.8, 1.0 };
	int t = count_elem / 5;
	for (int j = 1; j < 6; j++) {
		i_count[j] = t * j;
	}
	i_count[5]--;

	double ta = a;
	for (int i = 0; i < count_elem; i++, ta += h_) {
		fprintf(out, "%.4lf %lf\n", ta, yt[i][0]);
	}

	vector<pair<double, double>> v;

	double m = 0.0;
	for (int i = 0; i < 6; i++, m += h) {
		printf("%.2lf\t", m);
		printf("%.3lf\t", yt[i_count[i]][0]);
		printf("%.3lf\n", yt[i_count[i]][1]);
		y[i] = yt[i_count[i]][0];

		v.push_back(make_pair(x[i], y[i]));
	}
	printf("\n");

	fclose(out);

	printf("Spleins interpolation:\n");
	FILE *splines_out = fopen("Splines.txt", "w");


	for (double i = a; i <= b; i += h_) {
		double tmp = cubic(i, v.size(), v.data());
		fprintf(splines_out, "%lf %lf\n", i, tmp);
	}
	printf("\n");

	fclose(splines_out);

	Eps = 1e-2;
	printf("Integration = %.10lf\n", double_counting(Form_of_Simpson, a, b, h_, Eps, tmp));

	return 0;
```

```
}
```