Лабораторная работа №8

Тужилкина Н.Г., БПМ-18-2, вариант 23.

Решение краевой задачи для ОДУ 2-го порядка методом конечных разностей

Будем рассматривать линейную краевую задачу:

где к коэффициентам краевых условий предъявля­ется требование

а функции в уравнении должны быть такими, чтобы данная задача имела единственное решение в заданном функциональном пространстве.

Краевые условия определяют третью или смешанную краевую задачу, частными случаями которой являются первая (при ) или вторая (при ) краевые задачи.

**Теорема 1**. *Для того чтобы существовало единственное решение краевой задачи, необходимо и достаточно, чтобы однородная краевая задача*

*имела только тривиальное решение .*

## Цель работы:

Решить краевые задачи методом конечных разностей:

Проанализировать устойчивость: посчитать задачу для и узлов, так, чтобы различия в общих узлах составляли не более 1%.

## Метод конечных разностей

### Изложение метода

Идея МКР решения краевых задач — вместо производных в дифференциальном уравнении использу­ются их конечноразностные аппроксимации.

Сначала на отрезке вводится сетка с шагом :

Считая точным решением данной краевой задачи, через

будем обозначать -ую компоненту искомого каркаса приближен­ного решения

Фиксируя , приходим к равенствам

где целая переменная может принимать значения от 0 до по числу узлов сетки, а ‒ значе­ния точного решения и его производных в -ом узле.

В каждом внутреннем узле сетки, т. е. при , зна­чения производных аппроксимируются конечноразностными отношениями по симметричным формулам второго порядка точ­ности:

В результате подстановки последних при получаем

Отбрасывая неопределенное слагаемое , приходим к разностному уравнению относи­тельно приближенных значений решения:

После приведения подобных членов получаем стан­дартное трехточечное разностное уравнение второго порядка

Число переменных в системе равно – числу значений искомой функции в узловых точках, в то время как число уравнений, составленных для всех внутренних точек отрезка , равно .

Недостающие два уравнения можно получить, используя краевые условия на концах отрезка .

#### Подход двухточечных аппроксимаций

Преимуществом такого подхода двухточечных аппроксимаций является простота и наличие в первом и последнем уравнении только двух переменных, что позволяет упростить решение системы уравнений.

Однако точность у этого подхода имеет порядок , из-за чего нельзя гарантировать лучшую точность при решении системы уравнений с двухточечными аппроксимациями на концах отрезка.

#### Подход трехточечных аппроксимаций

Точность этого подхода заметно выше, поскольку оценка позволяет надеяться на такую же точность при решении системы уравнений с трёхточечными аппроксимациями на концах отрезка.

#### Сведение краевой задачи к СЛАУ с трехдиагональной матрицей

Запишем краевые условия с трёхточечной аппроксимацией на концах. Для левого конца и для правого конца :

Далее, упрощая эти выражения, получим

Вернёмся теперь к СЛАУ и выпишем её первое уравнение, соответствующее :

Положим

Тогда можно написать .

Подставим последнее выражение вместо и получим

где

Выпишем теперь последнее уравнение из СЛАУ , которому соответствует :

Положим

Тогда можно написать .

Подставим последнее выражение вместо и получим

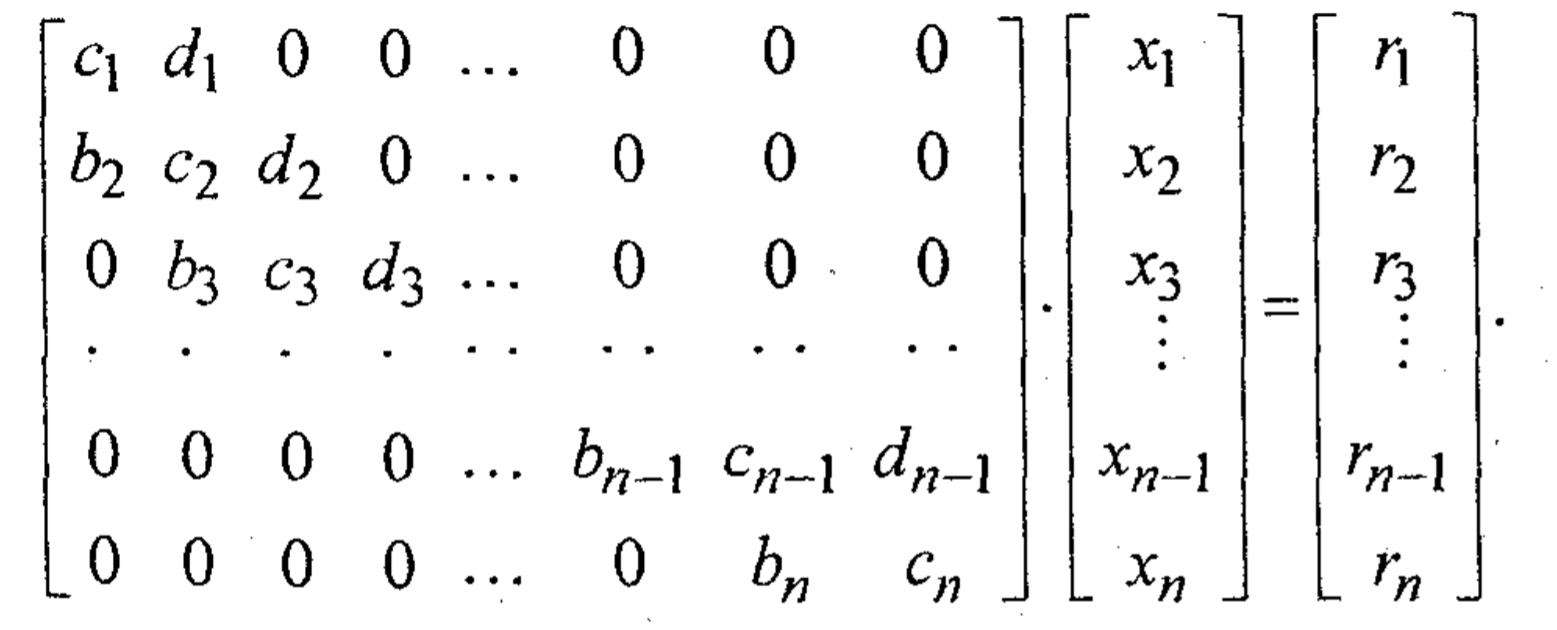
где

Добавим теперь к системе 0-ое уравнение, имеющее вид , и -ое уравнение, имеющее вид .

Остаётся решить построенную СЛАУ:

#### Метод прогонки

Дана система линейных алгебраических уравнений с трехдиагональной матрицей :



Вычислить

Прямая прогонка: найти прогоночные коэффициенты по формулам

Найти

Обратная прогонка: получить неизвестные по формулам

### Описание алгоритма

1. Привести уравнение к виду
2. Вычислить параметры в узловых точках
3. Составить систему уравнений :
4. Составить 0-е и -e уравнения
5. Преобразовать 0-e и -e уравнение
6. Решить СЛАУ методом прогонки

### Текст программы

// Параметры-функции

double P(const double& x) {

return 0;

}

double Q(const double& x) {

return -2 / (x \* x);

}

double F(const double& x) {

return 3 / (x + 1);

}

vector<double> finite\_dif(const double& a, const double& b, const size\_t& n) {

vector<double> x, p, q, f;

double h((b - a) / n);

x.push\_back(a);

// Вычислить параметры p\_i,q\_i,f\_i в узловых точках

for (size\_t i(1); i <= n; ++i)

x.push\_back(x[0] + i \* h);

for (size\_t i(0); i <= n; ++i) {

p.push\_back(P(x[i]));

f.push\_back(F(x[i]));

q.push\_back(Q(x[i]));

}

vector<double> data;

vector<double> fi;

// Краевые условия

double alpha0(1), alpha1(0), A(0), beta0(0), beta1(1), B(1);

// Составление системы уравнений

double S((2 - h \* h \* q[1]) / (1 - h \* p[1] / 2));

double T(-(1 + h \* p[1] / 2) / (1 - h \* p[1] / 2));

double U(h \* h \* f[1] / (1 - h \* p[1] / 2));

double temp(2 \* h \* alpha0 - 3 \* alpha1);

double L(temp \* S + 4 \* alpha1);

double M(temp \* T - alpha1);

double N(2 \* A \* h - temp \* U);

double V(-(1 - h \* p[n - 1] / 2) / (1 + h \* p[n - 1] / 2));

double W((2 - h \* h \* q[n - 1]) / (1 + h \* p[n - 1] / 2));

double Z(h \* h \* f[n - 1] / (1 + h \* p[n - 1] / 2));

temp = (2 \* h \* beta0 + 3 \* beta1);

double P(temp \* V + beta1);

double Q(temp \* W - 4 \* beta1);

double R(2 \* B \* h - temp \* Z);

size\_t m(n - 1);

data.push\_back(L); data.push\_back(M); fi.push\_back(N);

for (size\_t i(1); i <= m - 2; ++i)

data.push\_back(0);

for (size\_t i(1); i <= m - 2; ++i) {

for (size\_t j(1); j <= i - 1; ++j)

data.push\_back(0);

data.push\_back(1 - h \* p[i + 1] / 2);

data.push\_back(-(2 - h \* h \* q[i]));

data.push\_back(1 + h \* p[i - 1] / 2);

for (size\_t j(1); j <= m - i - 2; ++j)

data.push\_back(0);

fi.push\_back(h \* h \* f[i]);

}

for (size\_t i(1); i <= m - 2; ++i)

data.push\_back(0);

data.push\_back(P); data.push\_back(Q); fi.push\_back(R);

// Решение получившейся трехдиагональной системы уравнений методом прогонки

matrix SLAE{ m, data, fi };

SLAE.print();

vector<double> y0(SLAE.TD\_method());

vector<double> y;

y.push\_back(S \* y0[0] + T \* y0[1] + U);

y.insert(y.end(), y0.begin(), y0.end());

y.push\_back(V \* y0[n - 3] + W \* y0[n - 2] + Z);

for (size\_t i(0); i < y.size(); ++i) {

printf("(%.5f, %.5f)", x[i], y[i]);

if (i != y.size() - 1) cout << ", ";

}

printf("\n\n\n");

return y;

}

vector<double> matrix::TD\_method() const {

// δ\_1 = - d\_1 / c\_1

vector<double> delta{ -A(1, 2) / A(1, 1) };

// λ\_1 = r\_1 / c\_1

vector<double> lambda{ B(1) / A(1, 1) };

// Прямая прогонка

for (size\_t i(2); i <= n - 1; ++i) {

// δ\_i = - d\_i / (c\_i + b\_i \* δ\_(i-1))

double d\_i(-A(i, i + 1) / (A(i, i) + A(i, i - 1) \* delta.back()));

// λ\_i = (r\_i - b\_i \* λ\_(i-1)) / (c\_i + b\_i \* δ\_(i-1))

double l\_i((B(i) - A(i, i - 1) \* lambda.back()) / (A(i, i) + A(i, i - 1) \* delta.back()));

delta.push\_back(d\_i);

lambda.push\_back(l\_i);

}

vector<double> x{ b };

// x\_n = (r\_n - b\_n \* λ\_(n-1)) / (c\_n + b\_n \* δ\_(n-1)0)

x.back() = (b.back() - A(n, n - 1) \* lambda.back()) / (A(n, n) + A(n, n - 1) \* delta.back());

// Обратная прогонка

for (size\_t i(n - 1); i >= 1; --i)

x[i - 1] = delta[i - 1] \* x[i] + lambda[i - 1];

return x;

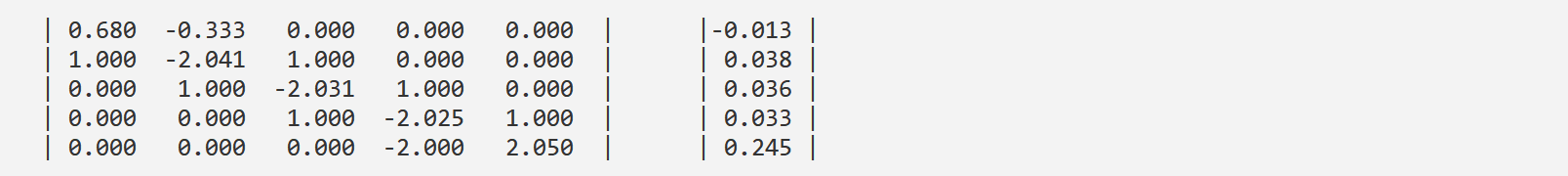
}

### Представление результатов

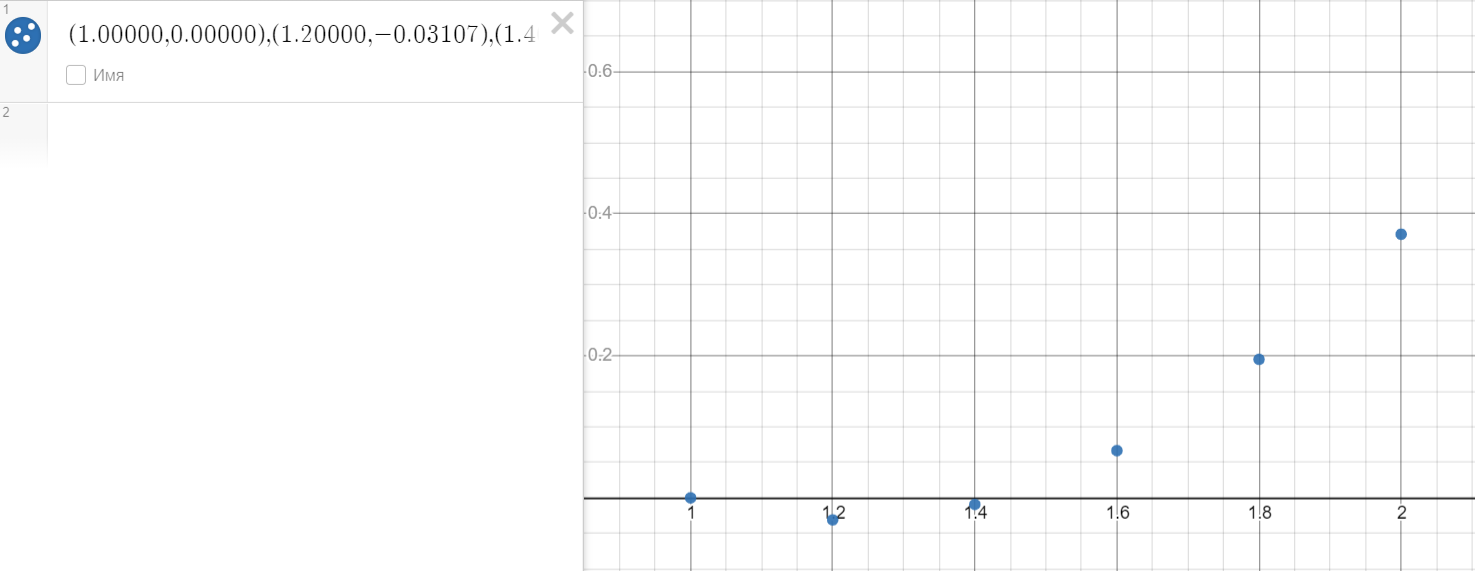
Приведенное к нужному виду уравнение:

Параметры:

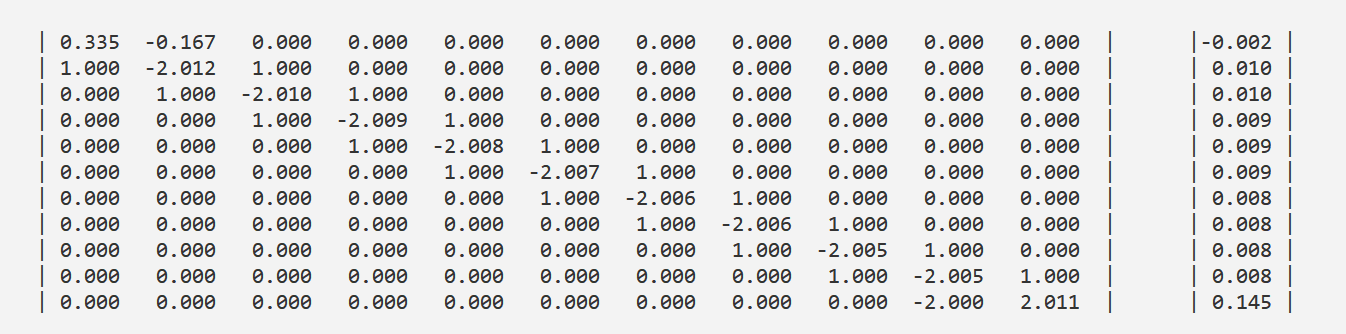
Решим задачу для узлов. Построенная система имеет вид:

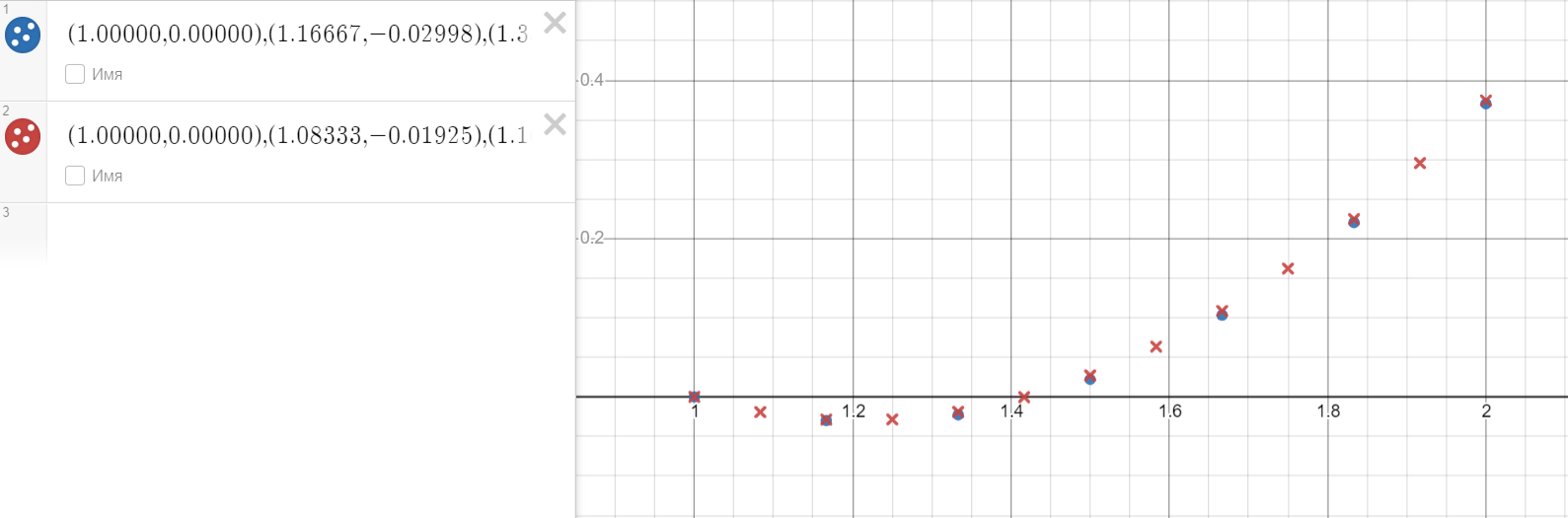


Решение задачи:

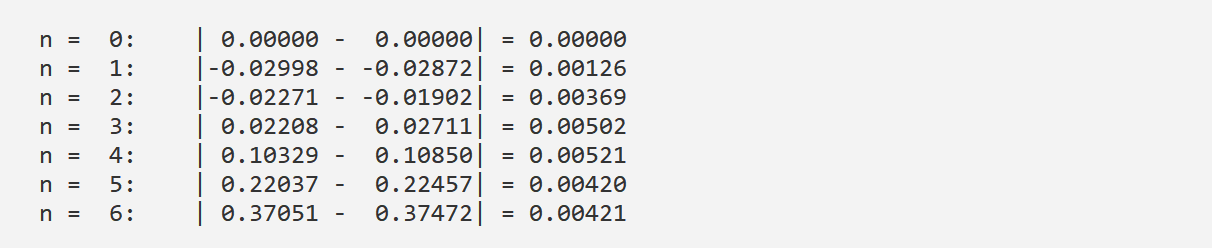


Проанализируем устойчивость, рассмотрев 12 узлов:





Получившаяся разница между значениями:



Таким образом, данная краевая задача имеет единственное решение.

Функции-параметры:

double P(const double& x) {

return 0;

}

double Q(const double& x) {

return 1;

}

double F(const double& x) {

return 1;

}

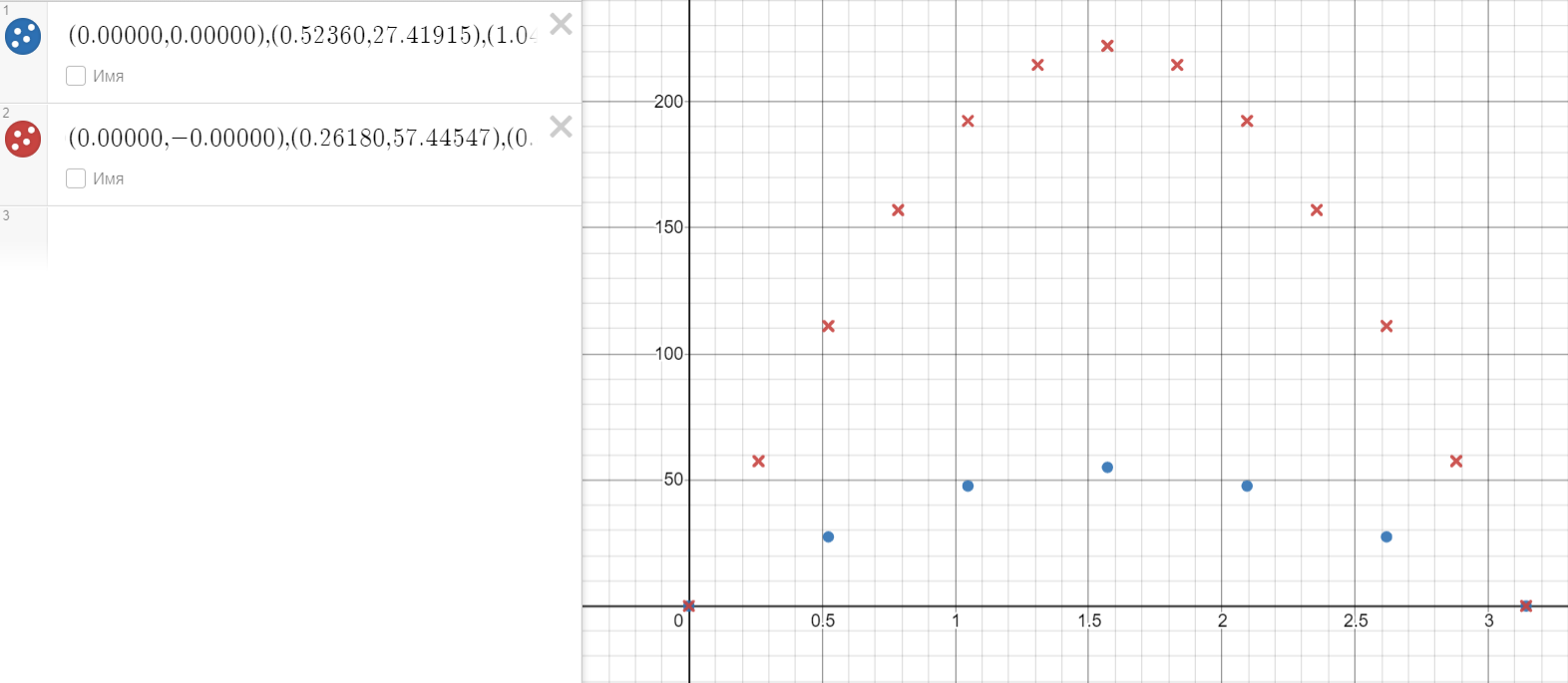
Краевые условия:

double alpha0(1), alpha1(0), A(0), beta0(1), beta1(0), B(0);

Вывод программы при :

При :

По графику, представленном ниже, видно, что решения не сходятся друг к другу, поэтому данная краевая задача имеет несколько решений или вовсе не имеет решений:



Получившаяся разница между значениями:

