**Отчёт по курсовой работе**

«Разработка программы для моделирования стационарного двумерного распределения температуры»

по дисциплине

«Математические модели систем c распределёнными параметрами»

|  |  |
| --- | --- |
| Выполнил студент гр. 3530904/90102 | Мэн Цзянин |
| Руководитель | Воскобойников С.П. |

Оглавление

[Постановка задачи 3](#_Toc100700457)

[Анализ порядка аппроксимации уравнения и граничных условий, выражение для главного члена погрешности аппроксимации 9](#_Toc100700458)

[Невязка и порядок погрешность аппроксимации уравнения 9](#_Toc100700459)

[Невязка и порядок погрешности аппроксимации граничного условия 14](#_Toc100700460)

[Решение системы методом сопряженных градиентов 19](#_Toc100700461)

[Тесты 23](#_Toc100700462)

[Константный тест 23](#_Toc100700463)

[Линейный тест 23](#_Toc100700464)

[Нелинейный тест 23](#_Toc100700465)

[Результаты 24](#_Toc100700466)

[Вывод 25](#_Toc100700467)

[Приложение 26](#_Toc100700468)

# Постановка задачи

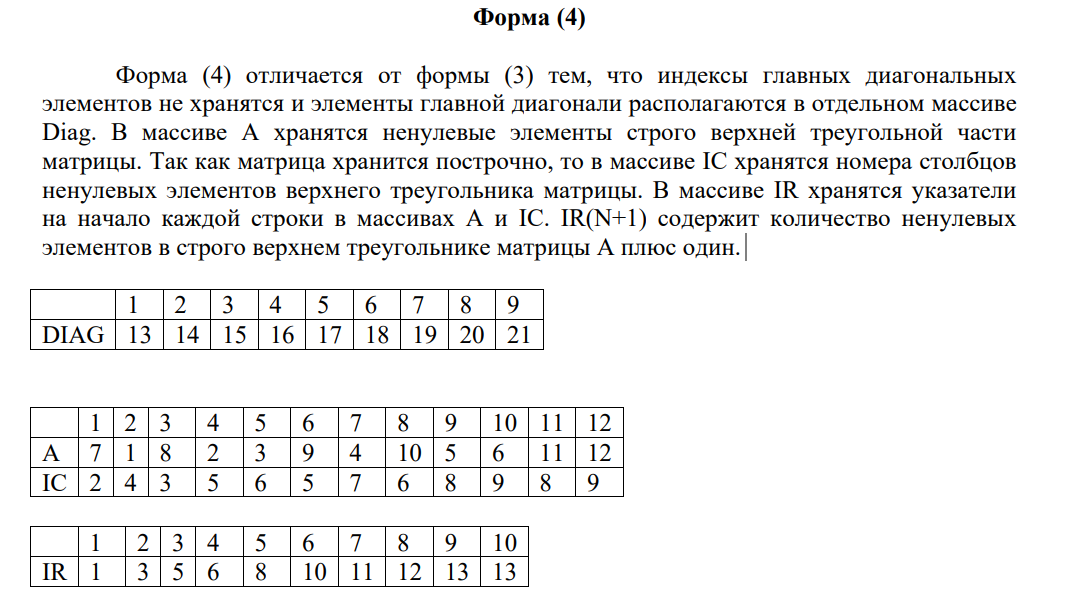
**Вариант N4**.

Постановка задачи. Используя интегро-интерполяционный метод, разработать подпрограмму для моделирования распределения температуры в цилиндре, описываемого математической моделью



, 

с граничными условиями, определяемыми вариантом задания. Для решения системы алгебраических уравнений использовать метод сопряжённых градиентов с предобусловливанием. Матрица алгебраической системы должна храниться в упакованной форме



**Дискретная модель**

Введем в прямоугольнике [0, R]×[0, L] равномерную основную сетку

и вспомогательную сетку





Так как используются равномерные сетки, то шаги вспомогательной сетки определяются как

Умножим исходное уравнение на r, проинтегрируем по вспомогательной сетке:

Воспользуемся формулой средних прямоугольников для вычисления значений интегралов:

Также аппроксимируем производные по формуле центральных разностей:

Получим:

i=1,2,.., ; j = 1,2,..,

Аппроксимация граничных условий:

Аналогично воспользуемся интегро-интерполяционным методом, получим:

В результате получается система линейных алгебраических уравнений вида ***A*u=b** размерности N=(Nz-1)(Nr+1)Рассмотрим более подробно структуру этой системы. Для дальнейшей работы необходимо перенумеровать компоненты векторов u и b. Для этого используем приведенный индекс. Сперва для фиксированного *r* движемся по оси *z*, потом переходим к следующему значению *r.*

При таком обозначении новый индекс k можно рассчитать так: k=i\*(Nz-1)+j

Матрица *А* квадратная, симметричная, пятидиагональная.

Хранить будем только 3 диагонали.

# Анализ порядка аппроксимации уравнения и граничных условий, выражение для главного члена погрешности аппроксимации

## Невязка и порядок погрешность аппроксимации уравнения

Преобразование:

При анализе порядка аппроксимации, для простого, будем писать просто

Невязка определяется как разность между правой и левой частью уравнения при условии, что вместо приближенного решения мы подставляем туда точное:

Раскладываем по степениям h точное решение в узлах и коэффициент k

Сокращаются четные степени

т.к. , получаем, что

Четные степени сокращаются

Так как, получаем, что

Подставляем в невязку получившиеся разложения

Группируем по степени hr и hz

Чтобы вычислить порядок аппроксимации, нормируем невязку

Выполним обратную замену.

Порядок аппроксимации уравнения по r и z:

Главный член погрешности по r

Главный член погрешности по z

## Невязка и порядок погрешности аппроксимации граничного условия

1. (Лекция10. p27)

Подставляем полученные ранее произведения:

Группируем по степениям hr и hz

Для вычисления порядка аппроксимации нормируем невязку

Аналогично выполним обратную замену, получим:

Порядок аппроксимации уравнения по r и z:

Главный член погрешности по x

Главный член погрешности по y

Подставляем полученные ранее произведения:

Группируем по степениям hr и hz

Для вычисления порядка аппроксимации нормируем невязку

Аналогично выполним обратную замену, получим:

Порядок аппроксимации уравнения по r и z:

Главные члены погрешности

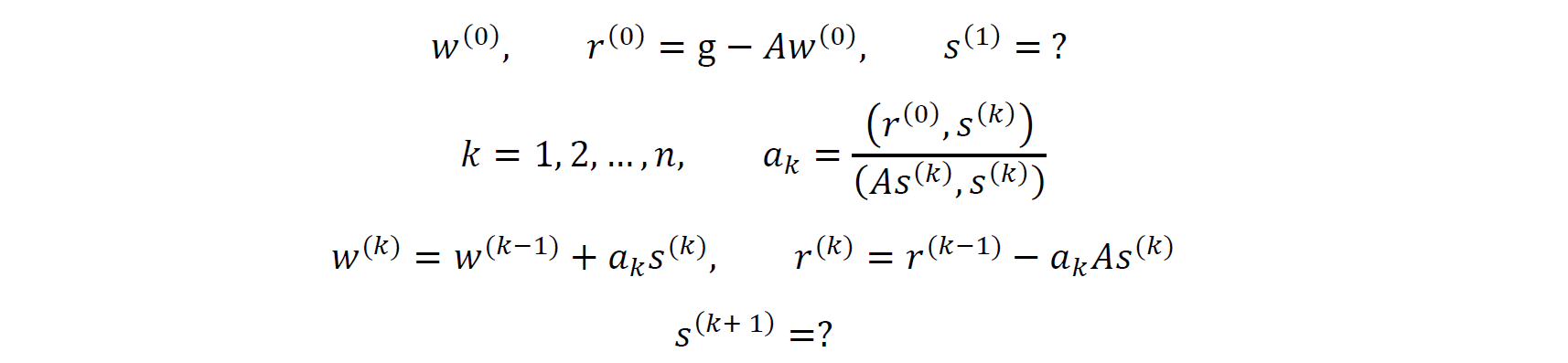
Порядок аппроксимации уравнения по r и z:

Главные члены погрешности

# Решение системы методом сопряженных градиентов

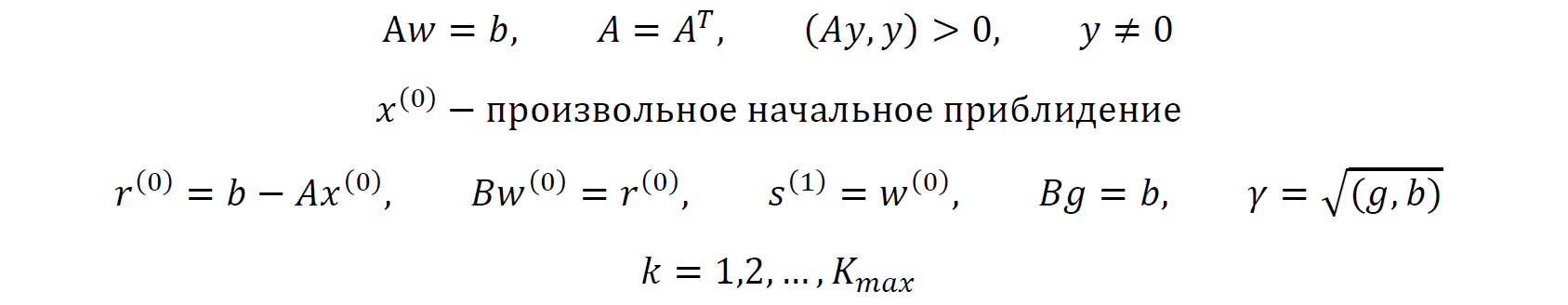
Пусть - произвольное начальное приближение, тогда , что даст нам невязку , предполагается, что у нас есть система из , где 𝑖=1,2,…,𝑛, линейно-независимых векторов, тогда можем разложит по базису этих векторов с соответствующими коэффициентами , найти коэффициенты можем с помощью СЛАУ , решение системы сильно упростится, если при а при скалярное произведение равнялось не 0 значению, в таком случае мы говорим об артогональности. Из этого мы можем выразить коэффициенты , и выразить решение .

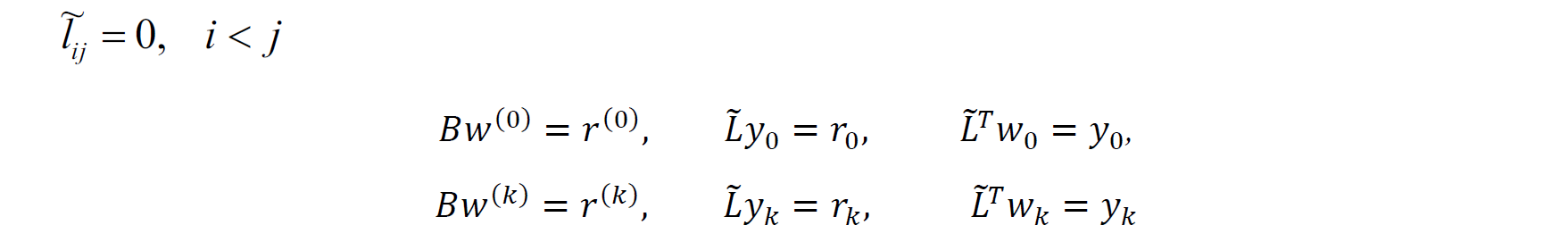
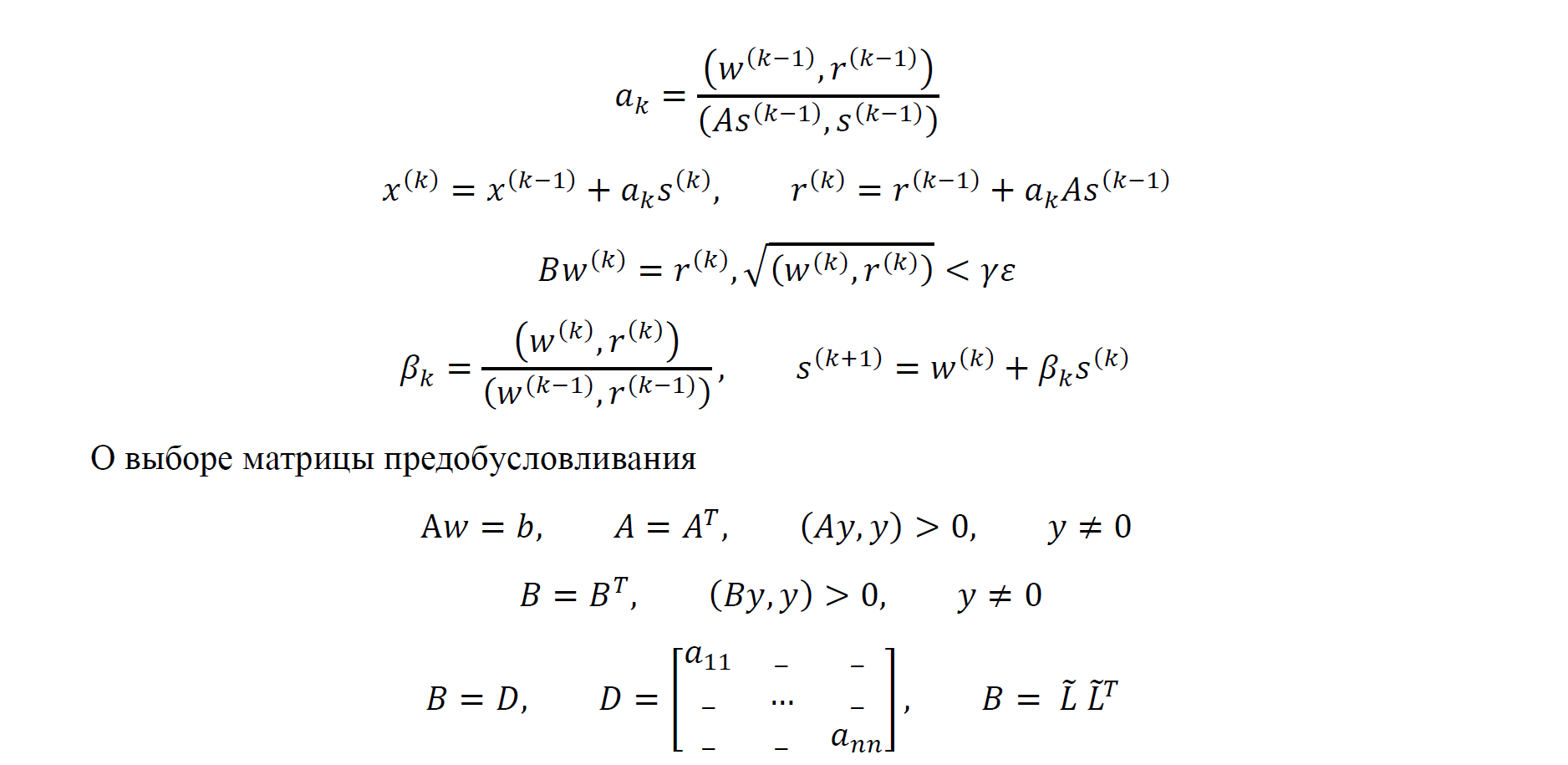
Рассмотрим частичную сумму , , для невязки получим рекуррентное соотношение



При явном методе сопряженных градиентов берут равным , с вводом дополнительного коэффициента при явный метод обладает тем свойством что при отсутствии ошибок округления мы можем получить точное решение не позднее чем на n-ом шаге, но возникает двойственность, из-за ошибок округления происходит разрушение артогональности последовательности s и в результате к неточности, и метод становится итерационным.

**Неявный метод**





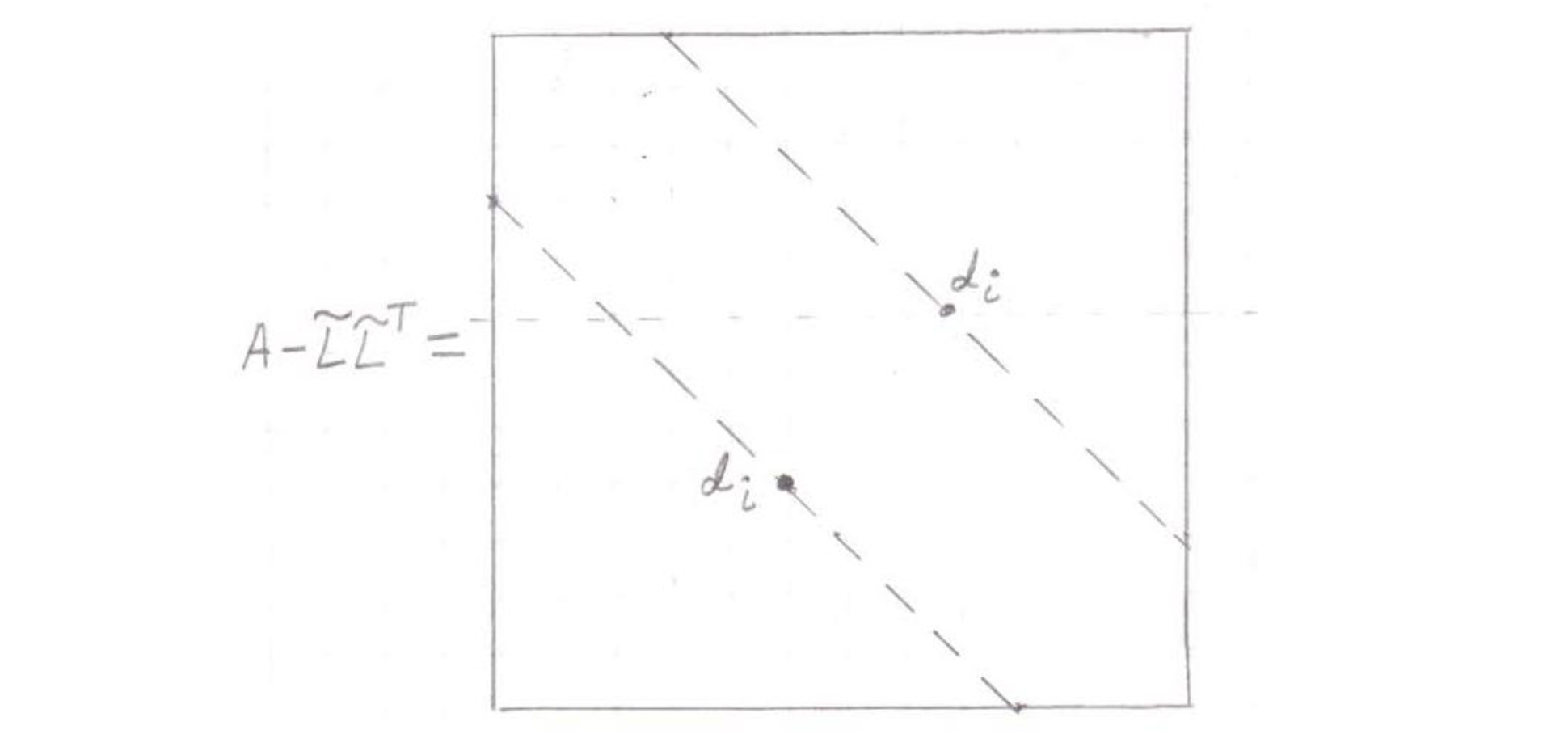
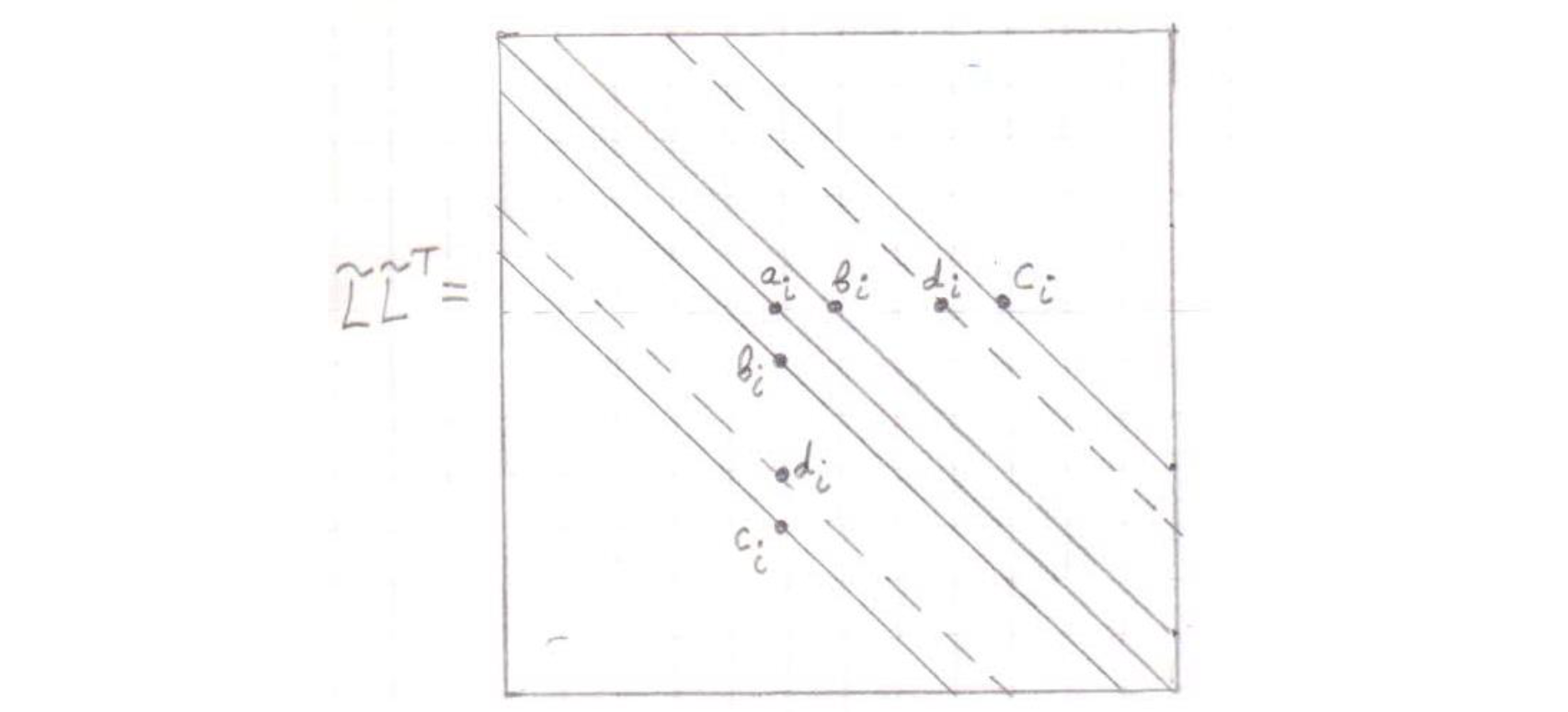
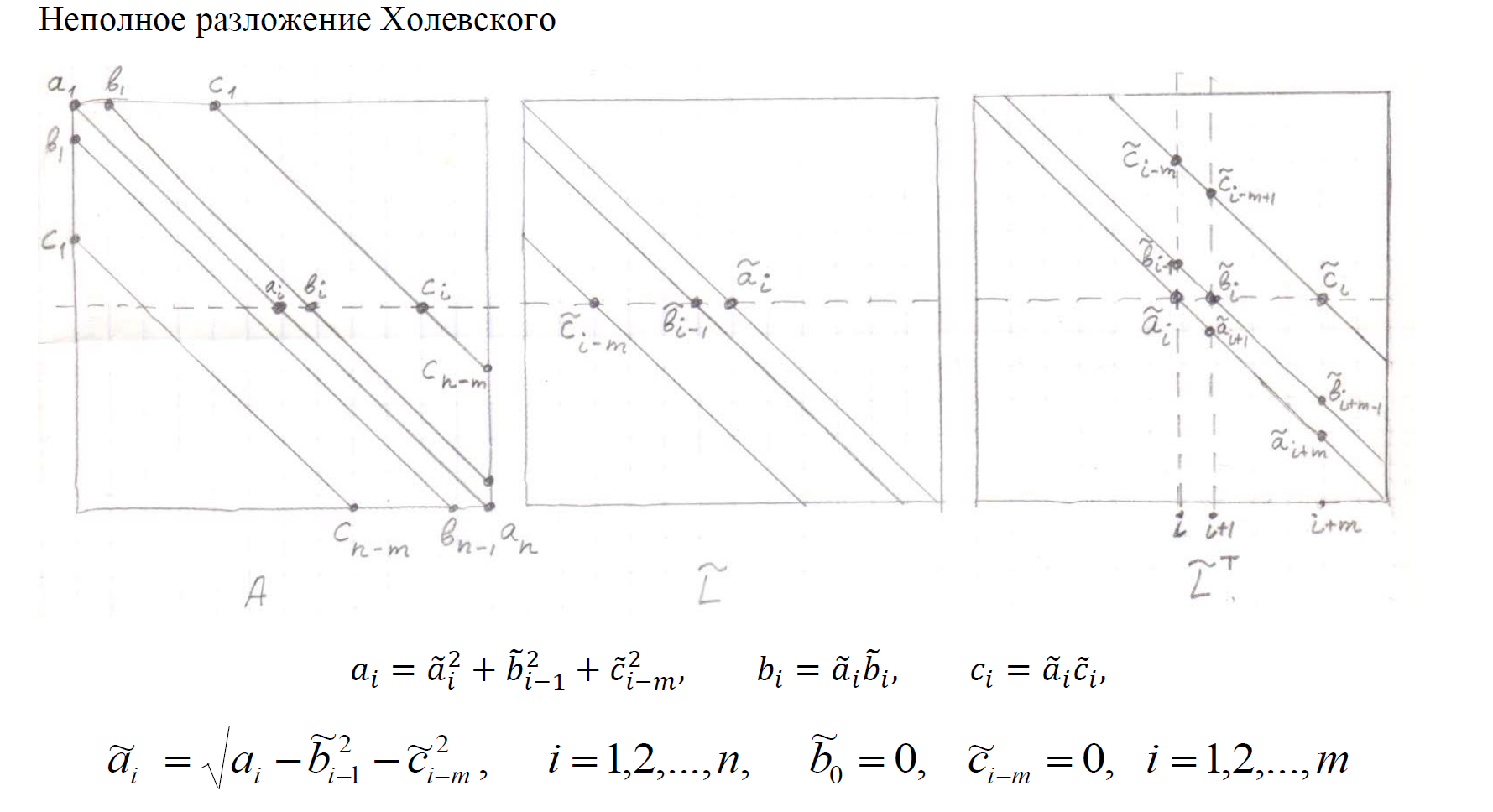
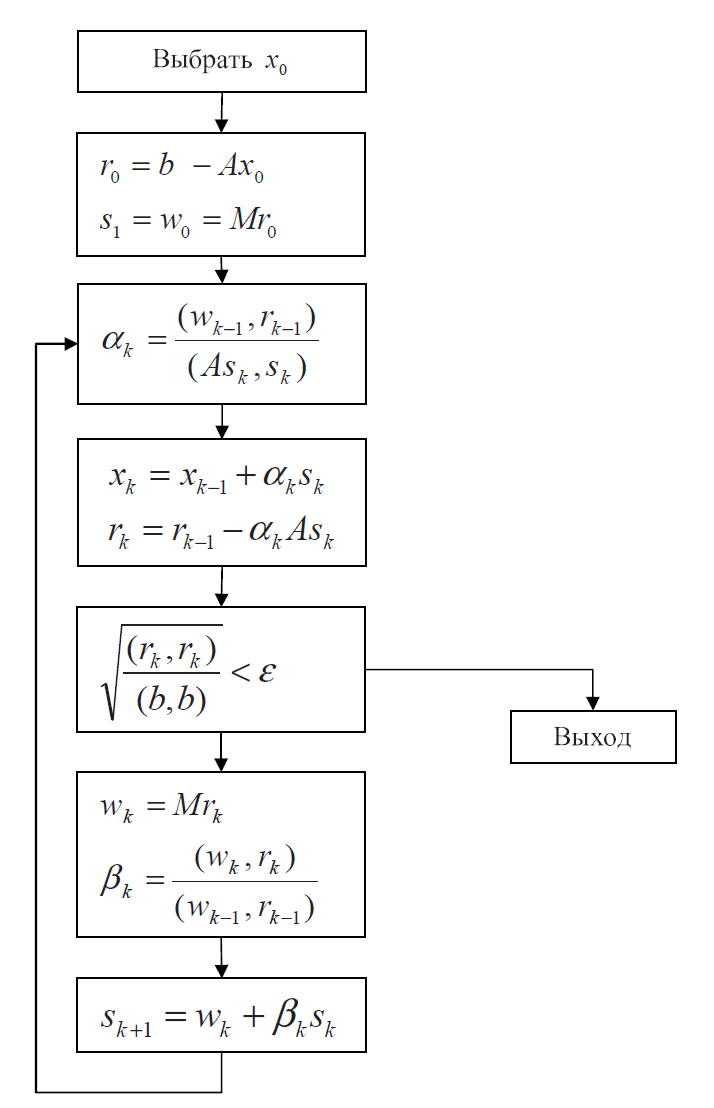


Схема применения метода выглядит следующим образом:



Здесь я использовал параметр .

# Тесты

Для всех тектов:

## Константный тест

## Линейный тест

## Нелинейный тест

# Результаты

Константный случай

|  |  |  |  |
| --- | --- | --- | --- |
| Число разбиений Nr, Nz | Максимальная погрешность | Отношение погрешностей | Число итераций метода |
| 4 | 8.88178419700E-16 | 0 | 11 |
| 8 | 3.82518982489E-07 | 2.32E-09 | 27 |
| 16 | 6.30674762192E-07 | 0.60652337056 | 16 |
| 32 | 2.62734831580E-06 | 0.24004231125 | 107 |
| 64 | 5.99528580780E-06 | 0.43823570719 | 208 |
| 128 | 6.36365821094E-06 | 0.94211310681 | 413 |

Линейный случай

|  |  |  |  |
| --- | --- | --- | --- |
| **Число разбиений Nr, Nz** | **Максимальная погрешность** | **Отношение погрешностей** | **Число итераций метода** |
| **4** | 1.95399252334E-14 | 0 | 16 |
| **8** | 1.14468746837E-06 | 1.71E-08 | 32 |
| **16** | 2.26973880935E-06 | 0.504325636 | 65 |
| **32** | 5.50640104335E-06 | 0.412200054 | 130 |
| **64** | 1.24175795404E-05 | 0.443435939 | 252 |
| **128** | 2.36257855251E-05 | 0.525594357 | 486 |

Нелинейный случай

|  |  |  |  |
| --- | --- | --- | --- |
| **Число разбиений Nr, Nz** | **Максимальная погрешность** | **Отношение погрешностей** | **Число итераций метода** |
| **4** | 3.38167202120E-02 | 0 | 16 |
| **8** | 8.92482305903E-03 | 3.789063378 | 42 |
| **16** | 2.25182768122E-03 | 3.963368571 | 87 |
| **32** | 5.80594786091E-04 | 3.87848416 | 179 |
| **64** | 1.70294493403E-04 | 3.40935737 | 359 |
| **128** | 4.16449411629E-05 | 4.0892 | 717 |

# Вывод

Погрешность решения дифференциального уравнения складывается из двух: погрешности аппроксимации (появляется при переходе от непрерывного уравнения к системе разностных) и погрешности решения алгебраической системы.

В линейном и константном случаях погрешность аппроксимации отсутствует, ее небольшой рост с увеличением количества разбиений связано с накоплением ошибки округления.

А в нелинейном случае наблюдается уменьшение ошибки в 4 раза при увеличении в 2 раза разбиений по оси 𝑟 и 𝑧. Погрешность решения алгебраической системы мала по сравнению с погрешностью аппроксимации, она возрастает незаметно. Погрешность аппроксимации, в свою очередь, уменьшается, т.к. мы увеличиваем количество разбиений. Причем, согласно теории, при одновременном удвоении числа разбиений погрешность аппроксимации должна уменьшаться в 4 раза, т.к. порядок аппроксимации метода равен 2. Как видим, наблюдаемые результаты очень близок к теоретическому.

# Приложение

import java.util.Arrays;  
import java.util.HashMap;  
import java.util.function.Function;  
  
  
public class N4 {  
 private final static double EPS = 1e-6;  
 private static int N = 5;  
 private static final double R0 = 0;  
 private static final double R1 = 1;  
 private static final double L = 1;  
 private static final double Chi2 = 1;  
 private static final double Chi4 = 1;  
  
  
 private enum SystemParameters {  
 DIAGONAL\_A, DIAGONAL\_B, DIAGONAL\_C, VECTOR\_G  
 }  
  
 @FunctionalInterface  
 public interface FunctionTwoArgs<A, B, R> {  
 R apply(A a, B b);  
 }  
  
 public static void main(String[] args) {  
  
 System.out.println("N4");  
 System.out.println("---->>> Константый случай");  
 test( (r, z) -> 1.0,  
 (r, z) -> 1.0,  
 (r, z) -> 0.0,  
 (z) -> 1.0,  
 (r) -> 1.0,  
 (r) -> 1.0,  
 (r, z) -> 1.0);  
  
  
 System.out.println("\n\n---->>> Линейный случай");  
 test( (r, z) -> r + 1.0,  
 (r, z) -> z + 1.0,  
 (r, z) -> -6 \* r - 4,  
 (z) -> 5.0,  
 (r) -> r \* r,  
 (r) -> r \* r,  
 (r, z) -> r \* r);  
  
 System.out.println("\n\n---->>> Нелинейный случай");  
 test( (r, z) -> r \* r + 1,  
 (r, z) -> 1 + z \* z,  
 (r, z) -> -24 \* r\*r\*r\*r - 14 \* r\*r \* z\*z - 18\*r\*r - 4\*z\*z,  
 (z) -> 5\*z\*z + 9,  
 (r) -> r \* r \* r \* r,  
 (r) -> r \* r \* r \* r + 5 \* r \* r,  
 (r, z) -> r \* r \* r \* r + z \* z \* r \* r);  
 }  
  
 private static void test(FunctionTwoArgs<Double, Double, Double> k1,  
 FunctionTwoArgs<Double, Double, Double> k2,  
 FunctionTwoArgs<Double, Double, Double> f,  
 Function<Double, Double> phi2,  
 Function<Double, Double> phi3,  
 Function<Double, Double> phi4,  
 FunctionTwoArgs<Double, Double, Double> u)  
 {  
 HashMap<SystemParameters, double[]> system;  
 N = 5;  
 double hR = (R1 - R0) / (N - 1);  
 double hZ = L / (N - 1);  
 double r;  
 double z = 0;  
 double[] result = new double[N \* N];  
 system = getSystem(k1, k2, f, phi2, phi3, phi4);  
 for (int i = 0; i < N; ++i) {  
 r = R0;  
 for (int j = 0; j < N; ++j) {  
 result[i \* N + j] = u.apply(r, z);  
 r += hR;  
 }  
 z += hZ;  
 }  
 System.out.println("Отклонения от точного решения\n"  
 + Arrays.toString( sub(multiply(system, result),  
 system.get(SystemParameters.VECTOR\_G))));  
  
 System.out.println("Ошибка");  
 double prevError = 0;  
 double nowError;  
 N = 5;  
 System.out.println("\t\tN\tError\tRatio\t");  
 for (int i = 2; i <= 8; ++i) {  
 N = (int) Math.round(Math.pow(2, i)) + 1;  
 system = getSystem(k1, k2, f, phi2, phi3, phi4);  
 result = ConjugateGradientMethod(system, system.get(SystemParameters.VECTOR\_G), getEMatrix());  
 nowError = getMaxError(result, u);  
 System.out.println("\t\t " + (N - 1) + "\t " + nowError + " \t " + prevError / nowError);  
 prevError = nowError;  
 }  
 }  
  
  
 private static double[] ConjugateGradientMethod(HashMap<SystemParameters, double[]> system,  
 double[] first,  
 HashMap<SystemParameters, double[]> bMatrix) {  
 double[] result = Arrays.copyOf(first, first.length);  
 double[] r = sub(system.get(SystemParameters.VECTOR\_G), multiply(system, first));  
 double[] p = solveB(bMatrix, r);  
 double[] b = solveB(bMatrix, system.get(SystemParameters.VECTOR\_G));  
 double[] s = Arrays.copyOf(p, p.length);  
 double alpha; double beta; double[] newR; double[] newP; int k;  
 for (k = 1; k <= 10000; k++) {  
 alpha = multiply(p, r) / multiply(multiply(system, s), s);  
 result = addition(result, multiply(alpha, s));  
 newR = sub(r, multiply(alpha, multiply(system, s)));  
 newP = solveB(bMatrix, newR);  
 double check = Math.sqrt(multiply(newP, newR) / multiply(b, system.get(SystemParameters.VECTOR\_G)));  
 if (check < EPS) {  
 ++k;  
 break;  
 }  
 beta = multiply(newP, newR) / multiply(p, r);  
 s = addition(newP, multiply(beta, s));  
 r = newR; p = newP;  
 }  
 System.out.println("(Число итераций:\t" + k +")");  
 return result;  
 }  
  
  
 private static double[] getADiag(FunctionTwoArgs<Double, Double, Double> k2) {  
 double hR = (R1 - R0) / (N - 1);  
 double hZ = L / (N - 1);  
 double scale = hR / hZ;  
 double[] result = new double[N \* N];  
 double z = hZ;  
 double r;  
  
 for (int j = 1; j < N - 1; j++) {  
 r = R0;  
 result[j \* N] = -(scale / 4) \* r \* k2.apply(r, z - hZ / 2); // #(2)  
 r += hR;  
 for (int i = 1; i < N - 1; i++) {  
 result[j \* N + i] = -(scale) \* r \* k2.apply(r, z - hZ / 2); // #(1)  
 r += hR;  
 }  
 result[j \* N + N - 1] = -(scale / 2) \* r \* k2.apply(r, z - hZ / 2); // #(3)  
 z += hZ;  
 }  
  
 r = R0;  
 for (int i = 0; i < N; i++) {  
 result[N \* (N - 1) + i] = -scale \* r \* k2.apply(r, z - hZ / 2); // #（5）  
 r += hR;  
 }  
  
 return result;  
 }  
  
 private static double[] getCDiag(FunctionTwoArgs<Double, Double, Double> k1,  
 FunctionTwoArgs<Double, Double, Double> k2) {  
  
 double hR = (R1 - R0) / (N - 1); double hZ = L / (N - 1);  
 double scale = hZ / hR; double z = hZ;  
 double r;  
 double[] result = new double[N \* N];  
  
 for (int i = 0; i < N; i++) { // #(4)  
 result[i] = 1;  
 }  
  
 for (int j = 1; j < N - 1; j++) {  
 r = R0;  
 result[j \* N] = scale \* (r + hR / 2) \* k1.apply(r + hR / 2, z) // #(2)  
 + (1 / scale) \* r \* k2.apply(r, z + hZ / 2)  
 + (1 / scale) \* r \* k2.apply(r, z - hZ / 2);  
 r += hR;  
  
  
 for (int i = 1; i < N - 1; i++) {  
 result[j \* N + i] = scale \* (r + hR / 2) \* k1.apply(r + hR /2, z) // #(1)  
 + scale \* (r - hR / 2) \* k1.apply(r - hR / 2, z)  
 + (1 / scale) \* r \* k2.apply(r, z + hZ / 2)  
 + (1 / scale) \* r \* k2.apply(r, z - hZ / 2);  
  
 r += hR;  
 }  
  
 result[j \* N + N - 1] = hZ \* r \* Chi2 // #(3)  
 + scale \* (r - hR / 2) \* k1.apply(r - hR / 2, z)  
 + (1 / scale / 2) \* r \* k2.apply(r, z + hZ / 2)  
 + (1 / scale / 2) \* r \* k2.apply(r, z - hZ / 2);  
 z += hZ;  
 }  
  
 r = R0;  
 for (int i = 0; i < N; i++) { // #(5)  
 result[N \* (N - 1) + i] = scale \* (r + hR / 2) \* k1.apply(r + hR /2, z)  
 + scale \* (r - hR / 2) \* k1.apply(r - hR / 2, z)  
 + hR \* r \* Chi4  
 + (1 / scale) \* r \* k2.apply(r, z - hZ / 2);  
 r += hR;  
 }  
 return result;  
 }  
  
 private static double[] getDDiag(FunctionTwoArgs<Double, Double, Double> k1) {  
  
 double hR = (R1 - R0) / (N - 1);  
 double hZ = L / (N - 1);  
 double scale = hZ / hR;  
 double z = hZ;  
 double r;  
 double[] result = new double[N \* N];  
  
 for (int j = 1; j < N - 1; j++) {  
 r = R0;  
 for (int i = 0; i < N - 1; i++) {  
 result[j \* N + i] = -scale \* (r + hR / 2) \* k1.apply(r + hR /2, z); // #(2) & (1)  
 r += hR;  
 }  
 z += hZ;  
 }  
 return result;  
 }  
  
 private static double[] getEDiag(FunctionTwoArgs<Double, Double, Double> k2) {  
 double hR = (R1 - R0) / (N - 1); double hZ = L / (N - 1);  
 double scale = hR / hZ;  
 double[] result = new double[N \* N]; double z = hZ;  
 double r;  
 for (int j = 1; j < N - 1; j++) {  
 r = R0;  
 result[j \* N] = -scale \* r \* k2.apply(r, z + hZ / 2) / 2; r += hR; // #(2)  
 for (int i = 1; i < N - 1; i++) {  
 result[j \* N + i] = -scale \* r \* k2.apply(r, z + hZ / 2); r += hR; // #(1)  
 }  
 result[j \* N + N - 1] = -scale \* r \* k2.apply(r, z + hZ / 2) / 2; z += hZ; // #(3)  
 }  
 return result;  
 }  
  
 private static double[] getVectorG(FunctionTwoArgs<Double, Double, Double> f,  
 Function<Double, Double> phi2,  
 Function<Double, Double> phi3,  
 Function<Double, Double> phi4) {  
  
 double hR = (R1 - R0) / (N - 1); double hZ = L / (N - 1);  
 double[] result = new double[N \* N]; double z = hZ;  
  
 double r = R0;  
 for (int i = 0; i < N; i++) { // #（4）  
 result[i] = phi3.apply(r);  
 r += hR;  
 }  
  
 for (int j = 1; j < N - 1; j++) {  
 r = R0;  
 result[j \* N] = hR \* hZ \* r \* f.apply(r, z) / 4; // #（2）  
 r += hR;  
 for (int i = 1; i < N - 1; i++) {  
 result[j \* N + i] = hR \* hZ \* r \* f.apply(r, z); // #（1）  
 r += hR;  
 }  
 result[j \* N + N - 1] = hR \* hZ \* r \* f.apply(r, z) / 2 + hZ \* r \* phi2.apply(z); // #（3）  
 z += hZ;  
 }  
  
 r = R0;  
 for (int i = 0; i < N; i++) {  
 result[N \* (N - 1) + i] = hR \* r \* phi4.apply(r) + hR \* hZ \* r \* f.apply(r, z) / 2; // #（5）  
 r += hR;  
 }  
 return result;  
 }  
  
 private static HashMap<SystemParameters, double[]> getSystem(FunctionTwoArgs<Double, Double, Double> k1,  
 FunctionTwoArgs<Double, Double, Double> k2,  
 FunctionTwoArgs<Double, Double, Double> f,  
 Function<Double, Double> phi2,  
 Function<Double, Double> phi3,  
 Function<Double, Double> phi4)  
 {  
 double[] a = getADiag(k2);  
 double[] c = getCDiag(k1, k2);  
 double[] d = getDDiag(k1);  
 double[] e = getEDiag(k2);  
 double[] g = getVectorG(f, phi2, phi3, phi4);  
  
 for (int i = 0; i < N; i++) {  
 g[N + i] -= g[i] \* a[N + i];  
 a[N + i] = 0;  
 g[N \* (N - 2) + i] -= g[N \* (N - 1) + i] \* e[N \* (N - 2) + i];  
 e[N \* (N - 2) + i] = 0;  
 }  
  
 HashMap<SystemParameters, double[]> system = new HashMap<>();  
  
 system.put(SystemParameters.DIAGONAL\_A, c);  
 system.put(SystemParameters.DIAGONAL\_B, d);  
 system.put(SystemParameters.DIAGONAL\_C, e);  
 system.put(SystemParameters.VECTOR\_G, g);  
 return system;  
 }  
  
 private static double getMaxError(double[] solve, FunctionTwoArgs<Double, Double, Double> u) {  
 double hR = (R1 - R0) / (N - 1); double hZ = L / (N - 1);  
 double z = 0; double r;  
 double maxError = 0; double nowError;  
 for (int j = 0; j < N; j++) { r = R0;  
 for (int i = 0; i < N; i++) {  
 nowError = Math.abs(u.apply(r, z) - solve[j \* N + i]); if (nowError > maxError) {  
 maxError = nowError;  
 }  
 r += hR;  
 }  
 z += hZ;  
 }  
 return maxError;  
 }  
  
 private static HashMap<SystemParameters, double[]> getBMatrix(HashMap<SystemParameters, double[]> system)  
 {  
 HashMap<SystemParameters, double[]> result = new HashMap<>(); int squareN = N \* N;  
 double[] a = new double[squareN];  
 double[] b = new double[squareN];  
 double[] c = new double[squareN];  
 result.put(SystemParameters.DIAGONAL\_A, a);  
 result.put(SystemParameters.DIAGONAL\_B, b);  
 result.put(SystemParameters.DIAGONAL\_C, c);  
 a[0] = Math.sqrt(system.get(SystemParameters.DIAGONAL\_A)[0]);  
 for (int i = 1; i < N; i++) {  
 b[i - 1] = system.get(SystemParameters.DIAGONAL\_B)[i - 1] / a[i - 1];  
 a[i] = Math.sqrt(system.get(SystemParameters.DIAGONAL\_A)[i] - Math.pow(b[i - 1], 2));  
 }  
 for (int i = N; i < squareN; i++) {  
 c[i - N] = system.get(SystemParameters.DIAGONAL\_C)[i - N];  
 b[i - 1] = system.get(SystemParameters.DIAGONAL\_B)[i - 1] / a[i - 1];  
 a[i] = Math.sqrt(system.get(SystemParameters.DIAGONAL\_A)[i] - Math.pow(b[i - 1], 2) - Math.pow(c[i - N], 2));  
 }  
 return result;  
 }  
  
 private static double[] solveB(HashMap<SystemParameters, double[]> bMatrix, double[] g) {  
 int squareN = N \* N;  
  
 double[] y = new double[squareN];  
 double[] a = bMatrix.get(SystemParameters.DIAGONAL\_A);  
 double[] b = bMatrix.get(SystemParameters.DIAGONAL\_B);  
 double[] c = bMatrix.get(SystemParameters.DIAGONAL\_C);  
  
 y[0] = g[0] / a[0];  
 for (int i = 1; i < N; i++) {  
 y[i] = (g[i] - b[i - 1] \* y[i - 1]) / a[i];  
 }  
  
 for (int i = N; i < squareN; i++) {  
 y[i] = (g[i] - b[i - 1] \* y[i - 1] - c[i - N] \* y[i - N]) / a[i];  
 }  
  
 double[] result = new double[squareN];  
 result[squareN - 1] = y[squareN - 1] / a[squareN - 1];  
  
 for (int i = squareN - 2; i >= N \* (N - 1); i--) {  
 result[i] = (y[i] - b[i] \* result[i + 1]) / a[i];  
 }  
 for (int i = N \* (N - 1) - 1; i >= 0; i--) {  
 result[i] = (y[i] - b[i] \* result[i + 1] - c[i] \* result[i + N])/ a[i];  
 }  
 return result;  
 }  
  
 private static HashMap<SystemParameters, double[]> getEMatrix() {  
 HashMap<SystemParameters, double[]> e = new HashMap<>();  
 int squareN = N \* N;  
 double[] a = new double[squareN];  
 for (int j = 0; j < squareN; j++) {  
 a[j] = 1;  
 }  
 e.put(SystemParameters.DIAGONAL\_A, a); e.put(SystemParameters.DIAGONAL\_B, new double[squareN]); e.put(SystemParameters.DIAGONAL\_C, new double[squareN]); return e;  
 }  
  
 private static double multiply(double[] leftVector, double[] rightVector)  
 {  
 double result = 0;  
 for (int i = 0; i < leftVector.length; i++) {   
 result += leftVector[i] \* rightVector[i];  
 }  
 return result;  
 }  
  
 private static double[] multiply(HashMap<SystemParameters, double[]> system, double[] vector) {  
 double[] result = new double[vector.length];  
 double[] diagA = system.get(SystemParameters.DIAGONAL\_A); double[] diagB = system.get(SystemParameters.DIAGONAL\_B); double[] diagC = system.get(SystemParameters.DIAGONAL\_C); for (int i = 0; i < vector.length; i++) {  
 result[i] = diagA[i] \* vector[i];  
 }  
 for (int i = 0; i < vector.length - 1; i++) { result[i] += diagB[i] \* vector[i + 1];  
 }  
 for (int i = 0; i < vector.length - N; i++) { result[i] += diagC[i] \* vector[i + N];  
 }  
 for (int i = 1; i < vector.length; i++) { result[i] += diagB[i - 1] \* vector[i - 1];  
 }  
 for (int i = N; i < vector.length; i++) { result[i] += diagC[i - N] \* vector[i - N];  
  
 }  
 return result;  
 }  
  
 private static double[] multiply(double number, double[] vector) { double[] result = new double[vector.length];  
 for (int i = 0; i < vector.length; i++)  
 {  
 result[i] = vector[i] \* number;  
 }  
 return result;  
 }  
  
 private static double[] addition(double[] leftVector, double[] rightVector) {  
 double[] result = new double[leftVector.length];  
 for (int i = 0; i < leftVector.length; i++) {  
 result[i] = leftVector[i] + rightVector[i];  
 }  
 return result;  
 }  
  
 private static double[] sub(double[] leftVector, double[] rightVector) { double[] result = new double[leftVector.length];  
 for (int i = 0; i < leftVector.length; i++) {  
 result[i] = leftVector[i] - rightVector[i];  
 }  
 return result;  
 }  
}