|  |  |  |
| --- | --- | --- |
| **ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ** | | |
| по дисциплине  «Математические модели систем с распределёнными параметрами»  Вариант Q12 | | |
| Выполнила | | |
| студентка гр. 3530904/90102 |  | Ли Ицзя |
| Руководитель | | |
| доцент |  | Воскобойников С.П. |
|  | | |

Оглавление

[Постановка задания 3](#_Toc100745561)

[Дискретная модель 4](#_Toc100745562)

[Анализ порядка аппроксимации уравнения и граничных условий, выражение для главного члена погрешности аппроксимации 6](#_Toc100745563)

[Невязка и порядок погрешность аппроксимации уравнения 6](#_Toc100745564)

[Невязка и порядок погрешности аппроксимации граничного условия 10](#_Toc100745565)

[Преобразования разностной схемы для применения метода сопряженных градиентов 13](#_Toc100745566)

[Решение системы методом сопряженных градиентов 17](#_Toc100745567)

[Форма Хранения Матриц 20](#_Toc100745568)

[Тесты 21](#_Toc100745569)

[Константный тест 21](#_Toc100745570)

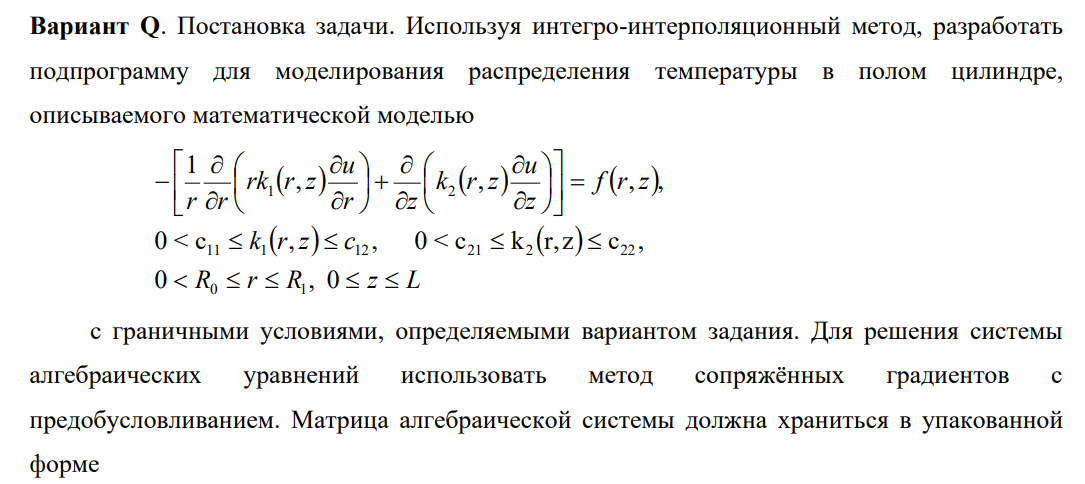
[Линейный тест 21](#_Toc100745571)

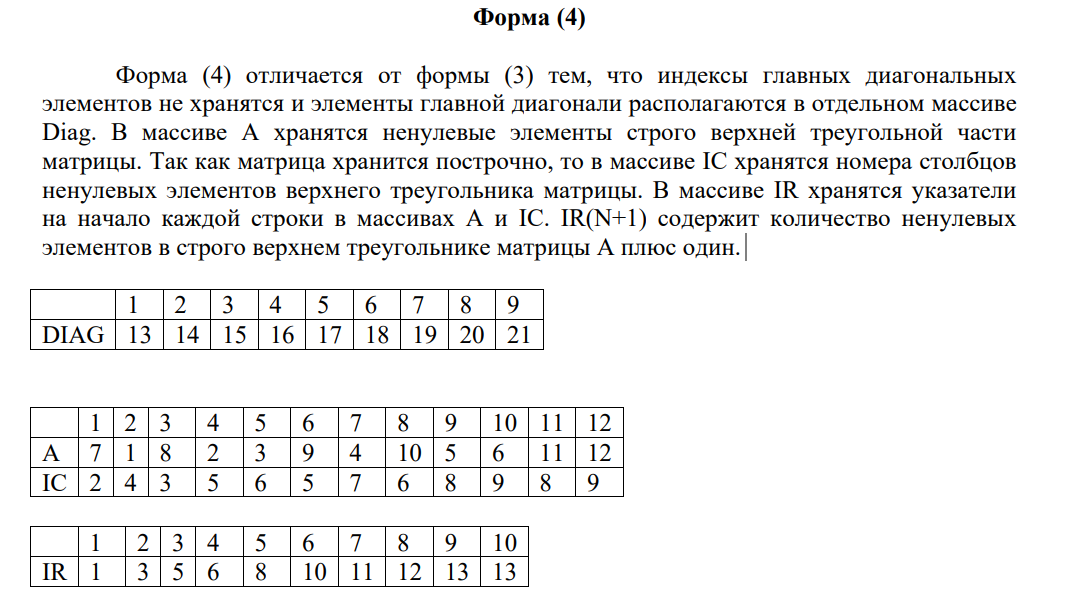
[Нелинейный тест 22](#_Toc100745572)

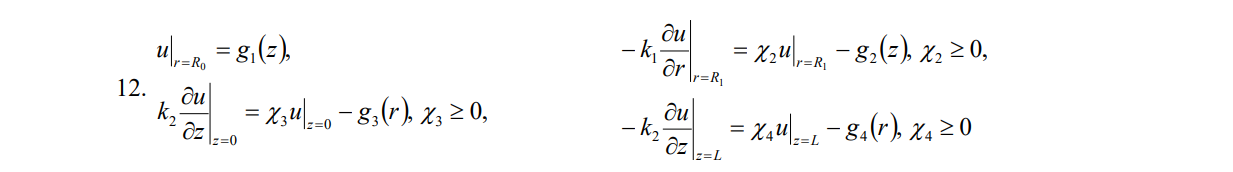
[Вывод 23](#_Toc100745573)

[Приложение 24](#_Toc100745574)

# Постановка задания







# Дискретная модель

Введем обозначения:

Основная сетка:

Введем вспомогательную сетку:

Аналогично произведем разбиения для переменной z

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:

1. При
2. При
3. При
4. При
5. При
6. При
7. При
8. При

# 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 сокрааются

Так как, получаем, что

Подсталяем 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

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

Аналогично выполним обратную замену, получим:

Порядок аппроксимации уравнения по r и z:

Главные члены погрешности:

Аналогично выполним обратную замену, получим:

Порядок аппроксимации уравнения по r и z:

Главные члены погрешности

# Преобразования разностной схемы для применения метода сопряженных градиентов

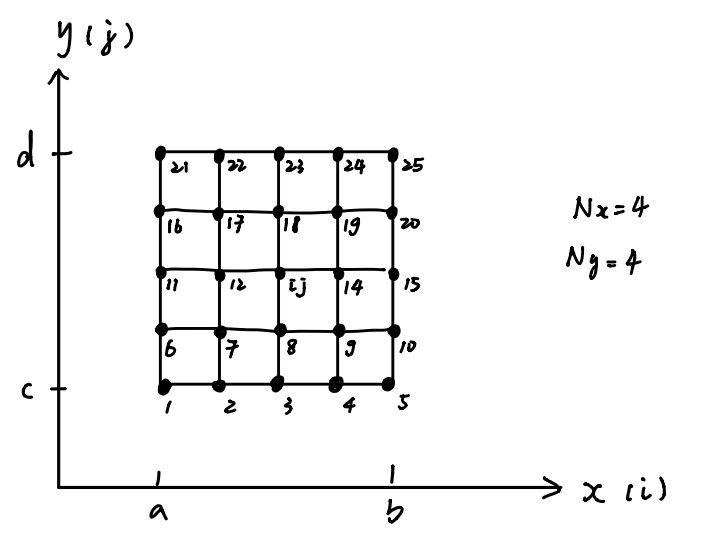
Разностная схема с приведенными подобными членами:

Основная сетка для

Понижение размерности матрицы методом исключения неизвестных

Пронумеруем узлы матрицы следующим образом

Будим принимать, что



Перейдем к одному индексу

Основная сетка

Остальные:

1. При
2. При
3. При
4. При
5. При
6. При
7. При
8. При

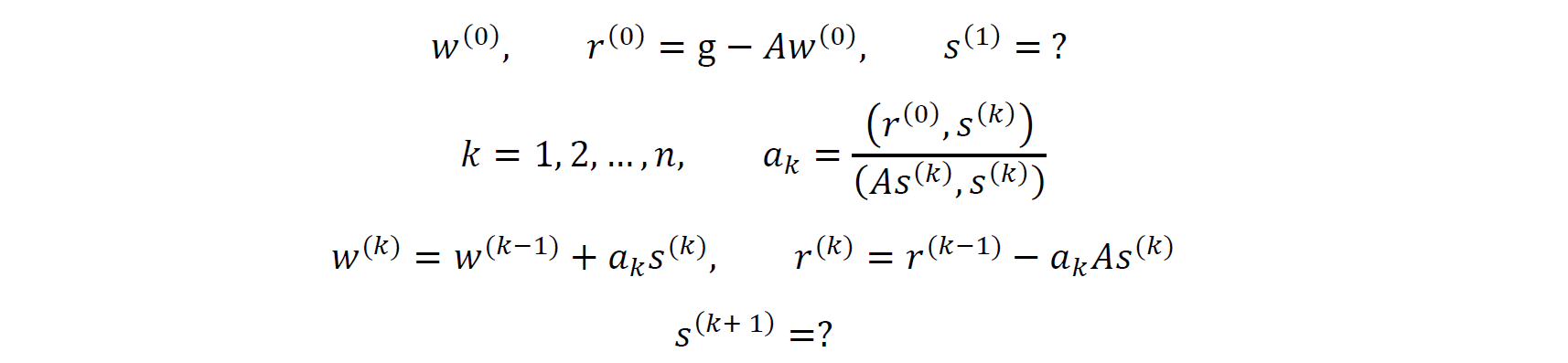
Также мы можем сделать матрицу симметричной. В итоге мы получили СЛАУ:

где A-матрица, w-вектор неизвестных, g-вектор правой части. Решение алгебраической системы проводится метод сопряженных градиентов, для которого необходимо, чтобы матрица A была симметрична и положительно определена.

# Решение системы методом сопряженных градиентов

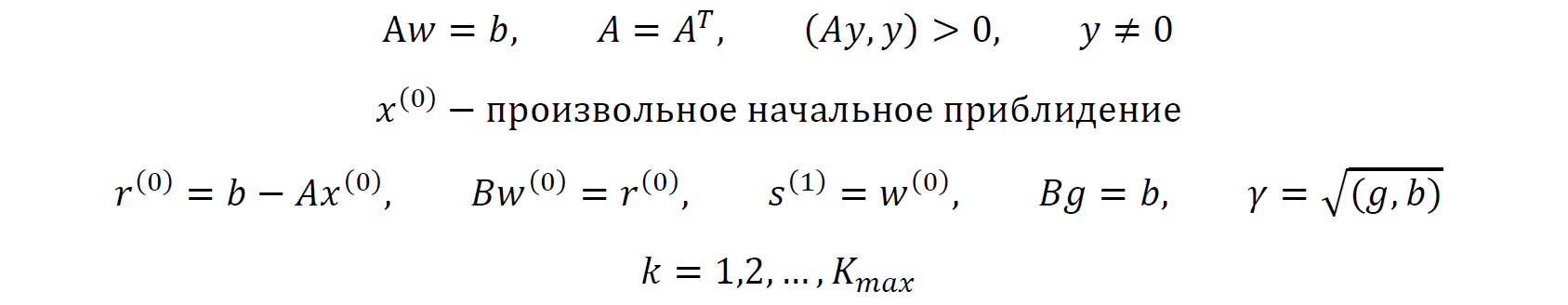
Пусть - произвольное начальное приближение, тогда , что даст нам невязку , предполагается, что у нас есть система из , где 𝑖=1,2,…,𝑛, линейно-независимых векторов, тогда можем разложит по базису этих векторов с соответствующими коэффициентами , найти коэффициенты можем с помощью СЛАУ , решение системы 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 .

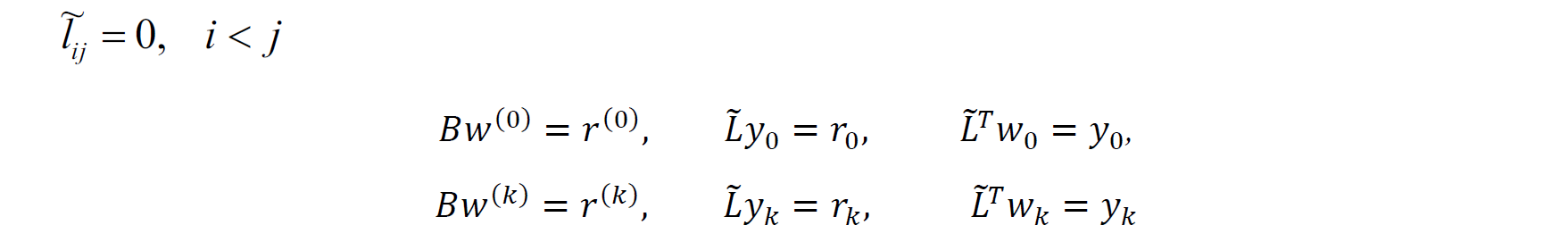
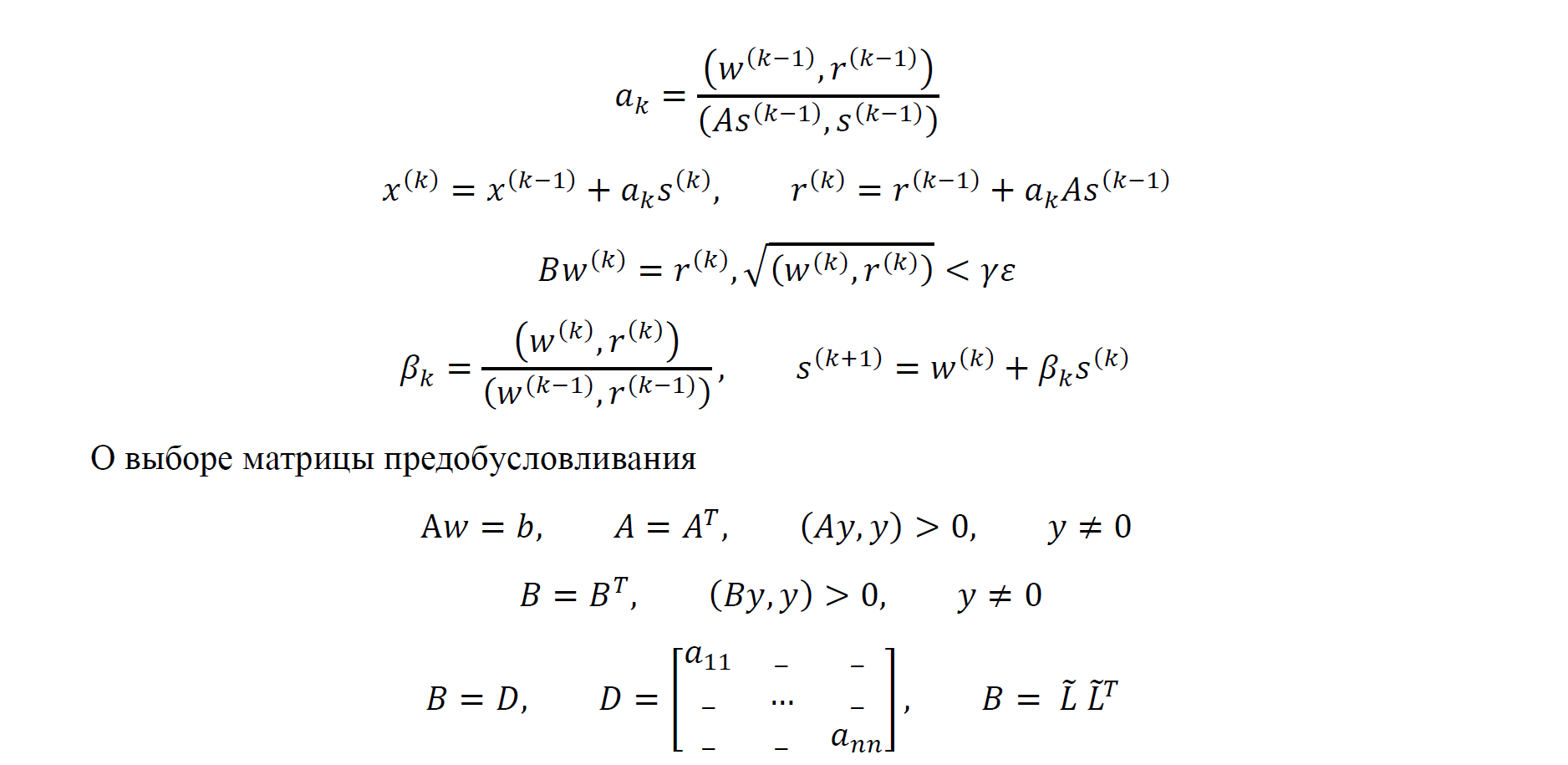
Рассмотрим частичную сумму , , для невязки получим рекуррентное соотношение

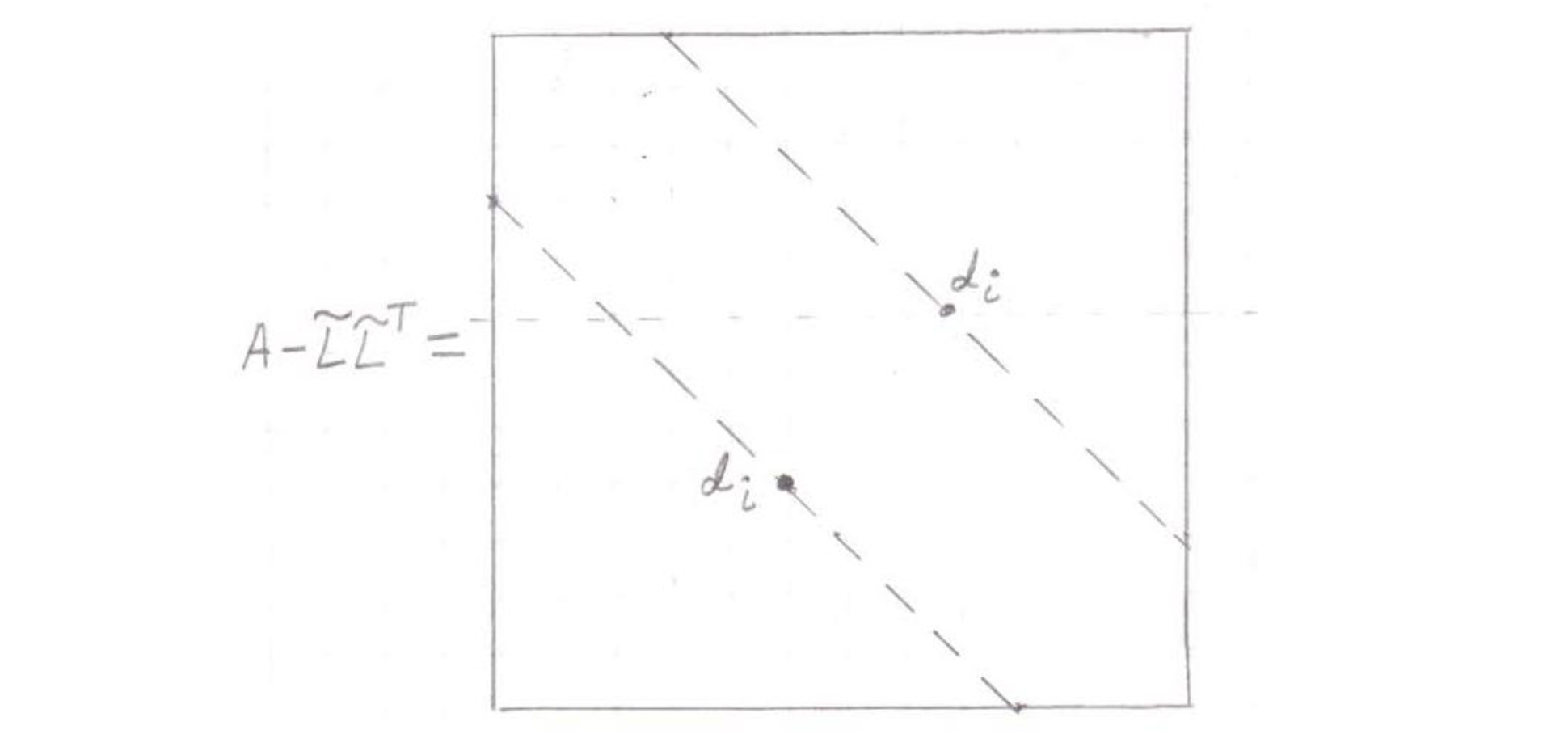
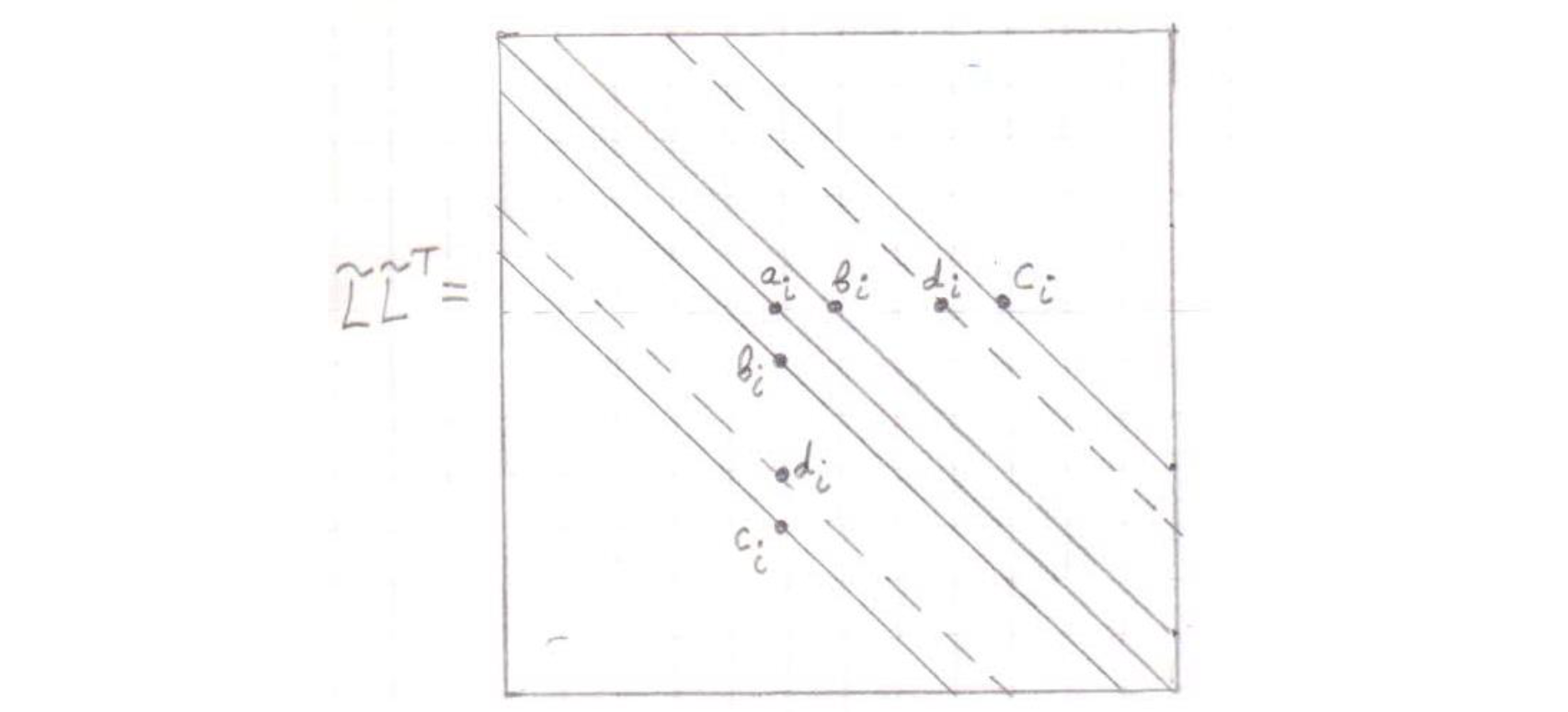
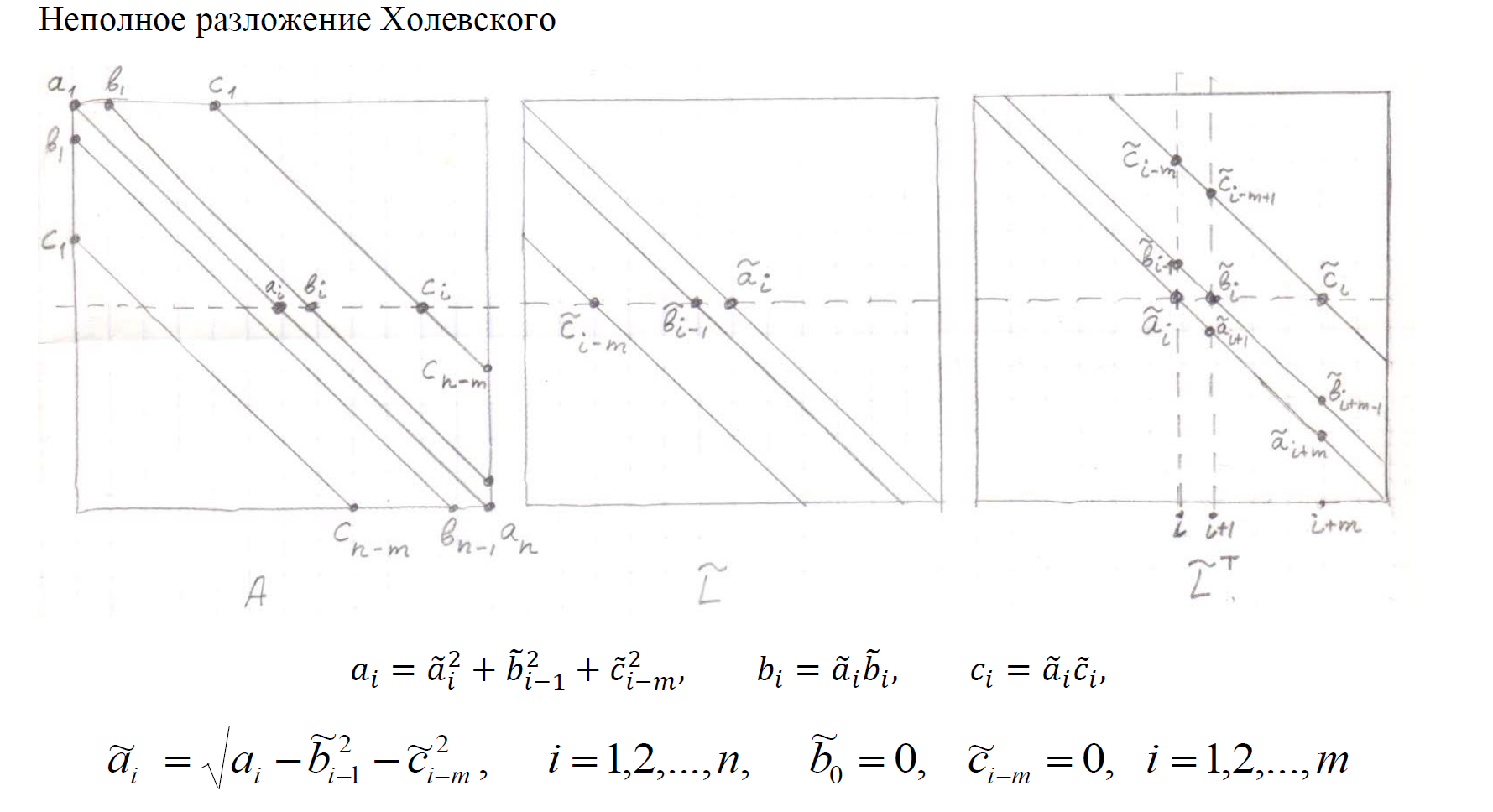


При явном методе сопряженных градиентов берут равным , с вводом дополнительного коэффициента при явный метод обладает тем свойством что при отсутствии ошибок округления мы можем получить точное решение не позднее чем на n-ом шаге, но возникает двойственность, из-за ошибок округления происходит разрушение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

**Неявный метод**







# Форма Хранения Матриц

Индексы главных диагональных элементов не хранятся, элементы главной диагонали располагаются в отдельном массиве Diag. В массиве А хранятся ненулевые элементы строго верхней треугольной части матрицы. Так как матрица хранится построчно, то в массиве IC хранятся номера столбцов ненулевых элементов верхнего треугольника матрицы. В массиве IR хранятся указатели на начало каждой строки в массивах A и IC. IR(N+1) содержит количество ненулевых элементов в строго верхнем треугольнике матрицы А плюс один.

# Тесты

Для всех тектов:

## Константный тест

|  |  |  |  |
| --- | --- | --- | --- |
| **Число разбиений Nr, Nz** | **Максимальная погрешность** | **Отношение погрешностей** | **Число итераций метода** |
| **4** | 8.881784197E-16 | 0 | 11 |
| **8** | 3.648100533E-08 | 2.4E-08 | 29 |
| **16** | 6.796758723E-08 | 0.5367412 | 57 |
| **32** | 1.328761980E-07 | 0.5115106 | 115 |
| **64** | 4.501897028E-07 | 0.295156 | 231 |
| **128** | 6.328432882E-07 | 0.7113763 | 461 |

## Линейный тест

|  |  |  |  |
| --- | --- | --- | --- |
| **Число разбиений Nr, Nz** | **Максимальная погрешность** | **Отношение погрешностей** | **Число итераций метода** |
| **4** | 1.953992523E-14 | 0 | 16 |
| **8** | 1.664610227E-07 | 1.2E-07 | 35 |
| **16** | 3.506694122E-07 | 0.474695 | 70 |
| **32** | 7.220940421E-07 | 0.4856285 | 141 |
| **64** | 1.049764290E-06 | 0.687863 | 279 |
| **128** | 1.941950505E-06 | 0.5405721 | 553 |

## Нелинейный тест

|  |  |  |  |
| --- | --- | --- | --- |
| **Число разбиений Nr, Nz** | **Максимальная погрешность** | **Отношение погрешностей** | **Число итераций метода** |
| **4** | 3.381672021E-02 | 0 | 16 |
| **8** | 8.920468110E-03 | 3.7909132 | 48 |
| **16** | 2.253769401E-03 | 3.9580217 | 97 |
| **32** | 5.648927730E-04 | 3.9897296 | 195 |
| **64** | 1.402223429E-04 | 4.0285504 | 401 |
| **128** | 3.811650674E-05 | 3.6787826 | 796 |

# Вывод

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 Q12 {  
 private final static double *EPS* = 1e-6;  
 private static int *N* = 5;  
 private static final double *R0* = 1;  
 private static final double *R1* = 2;  
 private static final double *L* = 1;  
 private static final double *Chi2* = 1;  
 private static final double *Chi3* = 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(" >>>>> Константый случай");  
 *test*( (r, z) -> 1.0,  
 (r, z) -> 1.0,  
 (r, z) -> 0.0,  
 (z) -> 1.0,  
 (z) -> 2.0,  
 (r) -> 1.0,  
 (r) -> 1.0,  
 (r, z) -> 1.0);  
  
 System.*out*.println(" >>>>>> Линейный случай");  
 *test*( (r, z) -> r + 1.0,  
 (r, z) -> z + 1.0,  
 (r, z) -> -8 - 3/r,  
 (z) -> 3 + 2 \* z,  
 (z) -> 2 \* z + 15,  
 (r) -> 3 \* r - 2,  
 (r) -> 3 \* r + 6,  
 (r, z) -> 3 \* r + 2 \* z);  
  
 System.*out*.println(" >>>>> Нелинейный случай");  
 *test*( (r, z) -> r + z,  
 (r, z) -> r + z,  
 (r, z) -> -8 \* z - 8 \* r,  
 (z) -> z \* z + 1,  
 (z) -> z \* z + 4 \* z + 12,  
 (r) -> r \* r,  
 (r) -> r \* r + 2 \* r + 3,  
 (r, z) -> r \* r + z \* z);  
 }  
  
 private static void test(Q12.FunctionTwoArgs<Double, Double, Double> k1,  
 Q12.FunctionTwoArgs<Double, Double, Double> k2,  
 Q12.FunctionTwoArgs<Double, Double, Double> f,  
 Function<Double, Double> g1,  
 Function<Double, Double> g2,  
 Function<Double, Double> g3,  
 Function<Double, Double> g4,  
 Q12.FunctionTwoArgs<Double, Double, Double> u) {  
 HashMap<Q12.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, g1, g2, g3, g4); 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(Q12.SystemParameters.*VECTOR\_G*)))); System.*out*.println("Ошибка");  
 double prevError = 0; double nowError;  
 *N* = 5;  
 System.*out*.println("\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, g1, g2, g3, g4); result = *leastGradientMethod*(system,  
 system.get(Q12.SystemParameters.*VECTOR\_G*), *getEMatrix*()); nowError = *getMaxError*(result, u);  
 System.*out*.println("\t " + (*N* - 1) + "\t " + nowError + " \t " + prevError / nowError);  
 prevError = nowError;  
 }  
 }  
  
  
 private static double[] leastGradientMethod(HashMap<Q12.SystemParameters, double[]> system, double[] first,  
 HashMap<Q12.SystemParameters,  
 double[]> bMatrix) {  
 double[] result = Arrays.*copyOf*(first, first.length); double[] r = *sub*(system.get(Q12.SystemParameters.*VECTOR\_G*),  
 *multiply*(system, first));  
 double[] p = *solveB*(bMatrix, r);  
 double[] b = *solveB*(bMatrix, system.get(Q12.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(Q12.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("K\t" + k);  
 return result;  
 }  
  
  
 private static double[] getADiag(Q12.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 / 2) \* r \* k2.apply(r, z - hZ / 2);  
 r += hR;  
 for (int i = 1; i < *N* - 1; i++) {  
 result[j \* *N* + i] = -(scale) \* r \* k2.apply(r, z - hZ / 2);  
 r += hR;  
 }  
 result[j \* *N* + *N* - 1] = -(scale / 2) \* r \* k2.apply(r, z - hZ / 2);  
 z += hZ;  
 }  
 return result;  
 }  
  
 private static double[] getCDiag(Q12.FunctionTwoArgs<Double, Double, Double> k1,  
 Q12.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++) {  
 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)  
 + hZ \* r \* *Chi2* + (1 / scale / 2) \* r \* k2.apply(r, z + hZ / 2)  
 + (1 / scale / 2) \* 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)  
 + 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* + 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;  
 }  
 for (int i = 0; i < *N*; i++) { result[*N* \* (*N* - 1) + i] = 1;  
 }  
 return result;  
 }  
  
 private static double[] getDDiag(Q12.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);  
 r += hR;  
 }  
 z += hZ;  
 }  
 return result;  
 }  
  
 private static double[] getEDiag(Q12.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;  
 for (int i = 1; i < *N* - 1; i++) {  
 result[j \* *N* + i] = -scale \* r \* k2.apply(r, z + hZ / 2); r += hR;  
 }  
 result[j \* *N* + *N* - 1] = -scale \* r \* k2.apply(r, z + hZ / 2) / 2; z += hZ;  
 }  
 return result;  
 }  
  
 private static double[] getVectorG(Q12.FunctionTwoArgs<Double, Double, Double> f,  
 Function<Double, Double> g1, Function<Double, Double> g2, Function<Double, Double> g3, Function<Double, Double> g4) {  
  
 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++) { result[i] = g3.apply(r); r += hR;  
 }  
 for (int j = 1; j < *N* - 1; j++) { r = *R0*;  
 result[j \* *N*] = hR \* hZ \* r \* f.apply(r, z) / 2  
 + hZ \* r \* g1.apply(z);  
 r += hR;  
 for (int i = 1; i < *N* - 1; i++) {  
 result[j \* *N* + i] = hR \* hZ \* r \* f.apply(r, z); r += hR;  
 }  
 result[j \* *N* + *N* - 1] = hR \* hZ \* r \* f.apply(r, z) / 2  
 + hZ \* r \* g2.apply(z);  
 z += hZ;  
 }  
 r = *R0*;  
 for (int i = 0; i < *N*; i++) {  
 result[*N* \* (*N* - 1) + i] = g4.apply(r); r += hR;  
 }  
 return result;  
 }  
  
 private static HashMap<Q12.SystemParameters, double[]> getSystem(Q12.FunctionTwoArgs<Double, Double, Double> k1,  
 Q12.FunctionTwoArgs<Double, Double, Double> k2,  
 Q12.FunctionTwoArgs<Double, Double, Double> f,  
 Function<Double, Double> g1,  
 Function<Double, Double> g2,  
 Function<Double, Double> g3,  
 Function<Double, Double> g4)  
 {  
 double[] a = *getADiag*(k2); double[] c = *getCDiag*(k1, k2); double[] d = *getDDiag*(k1); double[] e = *getEDiag*(k2);  
 double[] g = *getVectorG*(f, g1, g2, g3, g4);  
  
 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<Q12.SystemParameters, double[]> system = new HashMap<>();  
  
 system.put(Q12.SystemParameters.*DIAGONAL\_A*, c); system.put(Q12.SystemParameters.*DIAGONAL\_B*, d); system.put(Q12.SystemParameters.*DIAGONAL\_C*, e); system.put(Q12.SystemParameters.*VECTOR\_G*, g); return system;  
 }  
  
 private static double getMaxError(double[] solve, Q12.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<Q12.SystemParameters, double[]> getBMatrix(HashMap<Q12.SystemParameters, double[]> system) {  
 HashMap<Q12.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(Q12.SystemParameters.*DIAGONAL\_A*, a); result.put(Q12.SystemParameters.*DIAGONAL\_B*, b); result.put(Q12.SystemParameters.*DIAGONAL\_C*, c);  
 a[0] = Math.*sqrt*(system.get(Q12.SystemParameters.*DIAGONAL\_A*)[0]); for (int i = 1; i < *N*; i++) {  
 b[i - 1] = system.get(Q12.SystemParameters.*DIAGONAL\_B*)[i - 1] / a[i -  
 1];  
 a[i] = Math.*sqrt*(system.get(Q12.SystemParameters.*DIAGONAL\_A*)[i] -  
 Math.*pow*(b[i - 1], 2));  
 }  
 for (int i = *N*; i < squareN; i++) {  
 c[i - *N*] = system.get(Q12.SystemParameters.*DIAGONAL\_C*)[i - *N*];  
 b[i - 1] = system.get(Q12.SystemParameters.*DIAGONAL\_B*)[i - 1] / a[i -  
 1];  
 a[i] = Math.*sqrt*(system.get(Q12.SystemParameters.*DIAGONAL\_A*)[i] -  
 Math.*pow*(b[i - 1], 2) - Math.*pow*(c[i - *N*], 2));  
 }  
 return result;  
 }  
  
 private static double[] solveB(HashMap<Q12.SystemParameters, double[]> bMatrix, double[] g) {  
 int squareN = *N* \* *N*;  
  
 double[] y = new double[squareN];  
 double[] a = bMatrix.get(Q12.SystemParameters.*DIAGONAL\_A*); double[] b = bMatrix.get(Q12.SystemParameters.*DIAGONAL\_B*);  
  
 double[] c = bMatrix.get(Q12.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<Q12.SystemParameters, double[]> getEMatrix() { HashMap<Q12.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(Q12.SystemParameters.*DIAGONAL\_A*, a); e.put(Q12.SystemParameters.*DIAGONAL\_B*, new double[squareN]); e.put(Q12.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<Q12.SystemParameters, double[]> system, double[] vector) {  
 double[] result = new double[vector.length];  
 double[] diagA = system.get(Q12.SystemParameters.*DIAGONAL\_A*); double[] diagB = system.get(Q12.SystemParameters.*DIAGONAL\_B*); double[] diagC = system.get(Q12.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;  
 }  
}