**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](#_Toc100719707)

[Анализ порядка аппроксимации уравнения и граничных условий, выражение для главного члена погрешности аппроксимации 9](#_Toc100719708)

[Невязка и порядок погрешность аппроксимации уравнения 9](#_Toc100719709)

[Невязка и порядок погрешности аппроксимации граничного условия 17](#_Toc100719710)

[Решение системы методом сопряженных градиентов 22](#_Toc100719711)

[Тесты 26](#_Toc100719712)

[Константный тест 26](#_Toc100719713)

[Линейный тест 26](#_Toc100719714)

[Нелинейный тест 26](#_Toc100719715)

[Результаты 27](#_Toc100719716)

[Вывод 28](#_Toc100719717)

[Приложение 29](#_Toc100719718)

# 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

Форма (4) отличается от формы (3) тем, что индексы главных диагональных элементов не хранятся и элементы главной диагонали располагаются в отдельном массиве Diag. В массиве A хранятся ненулевые элементы строго верхней треугольной части матрицы. Так как матрица хранится построчно, то в массиве IС 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 A плюс один.

文本, 表格

中度可信度描述已自动生成

**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**.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

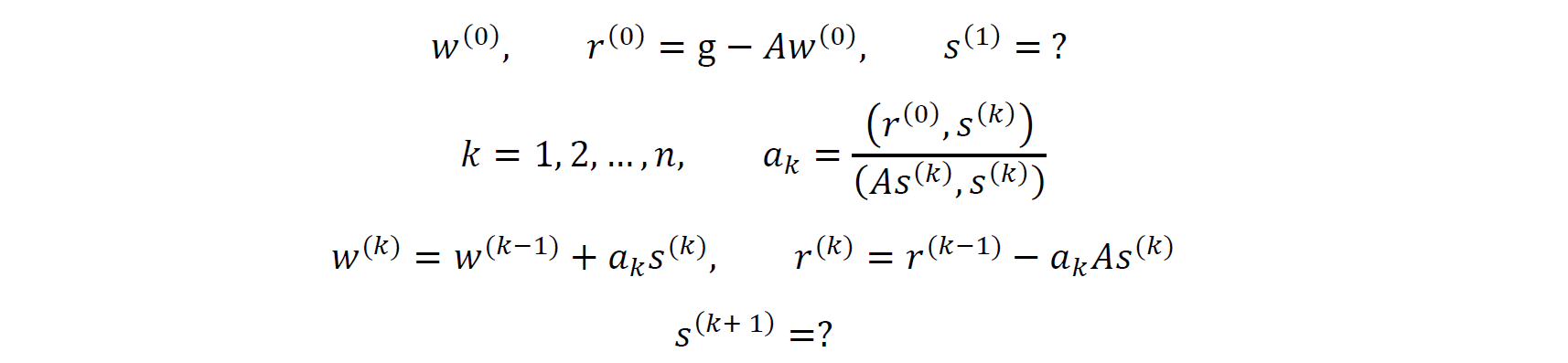
Порядок аппроксимации уравнения по r и z:

Главные члены погрешности

# Решение системы методом сопряженных градиентов

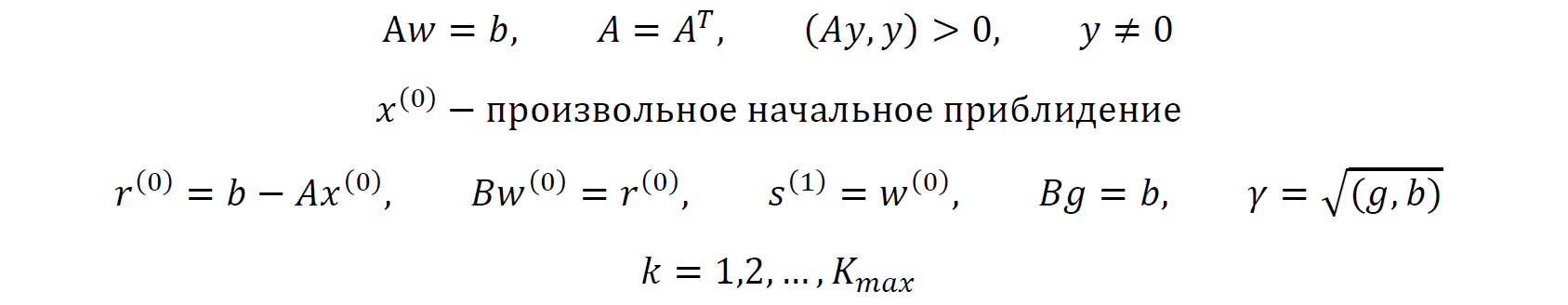
Пусть - произвольное начальное приближение, тогда , что даст нам невязку , предполагается, что у нас есть система из , где 𝑖=1,2,…,𝑛, линейно-независимых векторов, тогда можем разложит по базису этих векторов с соответствующими коэффициентами , найти коэффициенты можем с помощью СЛАУ , решение системы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

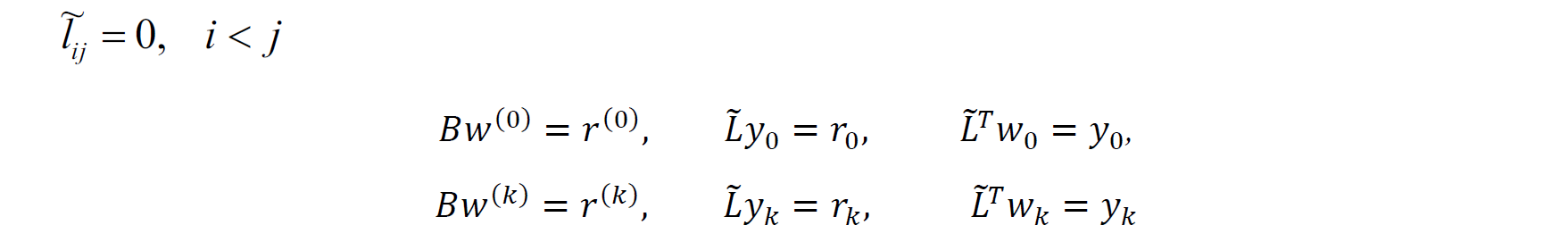
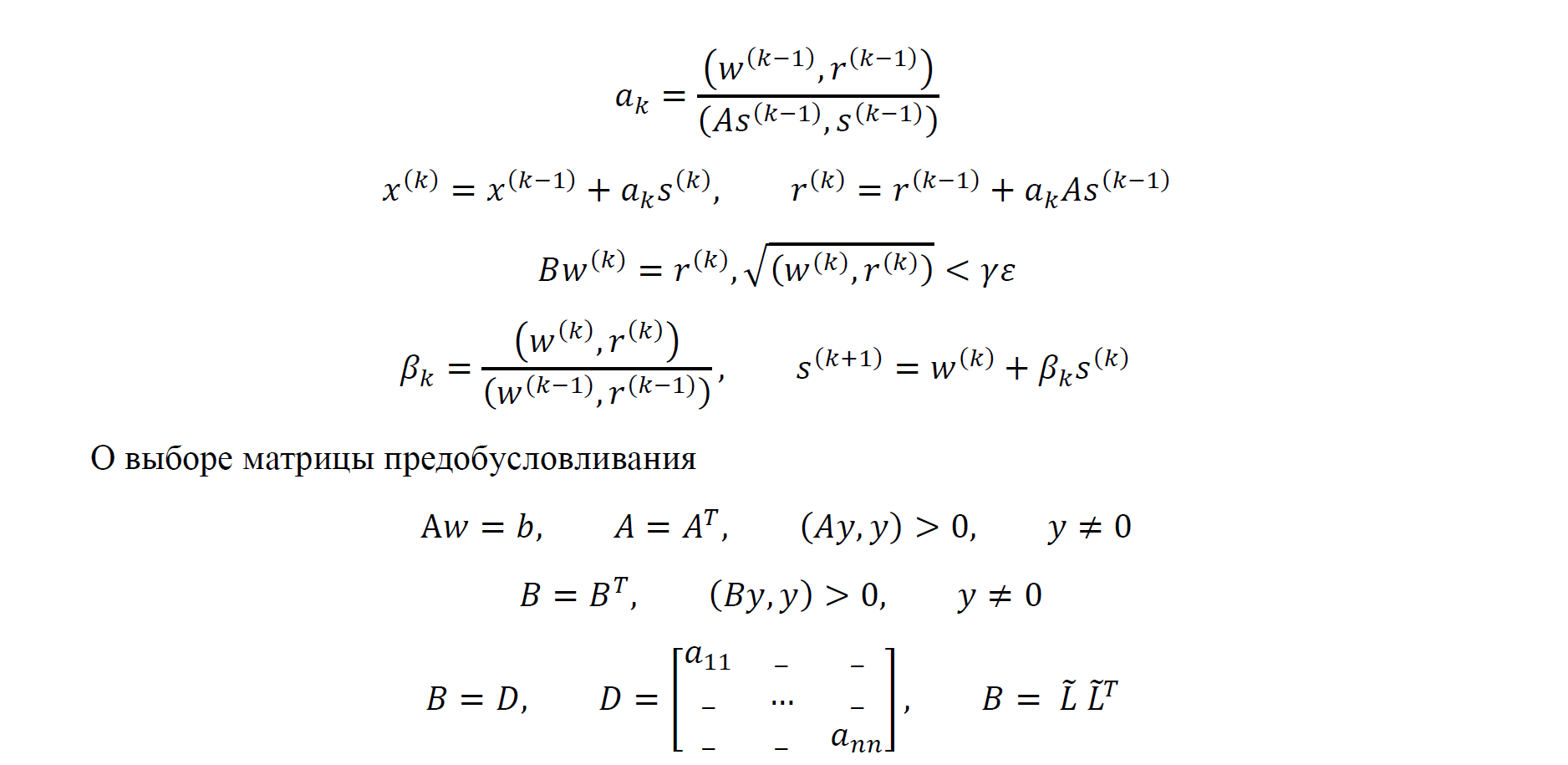
Рассмотрим частичную сумму , , для невязки получим рекуррентное соотношение



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**





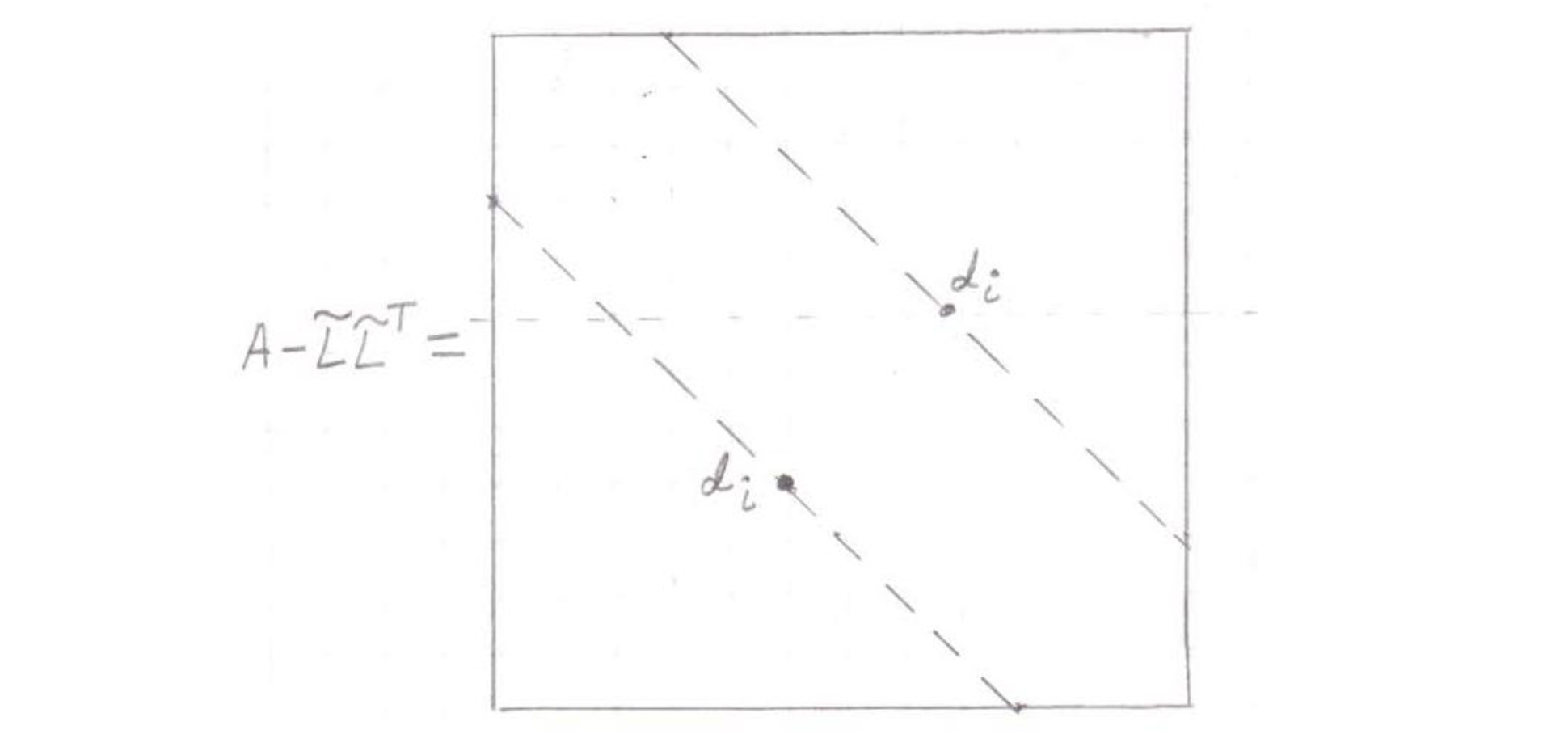
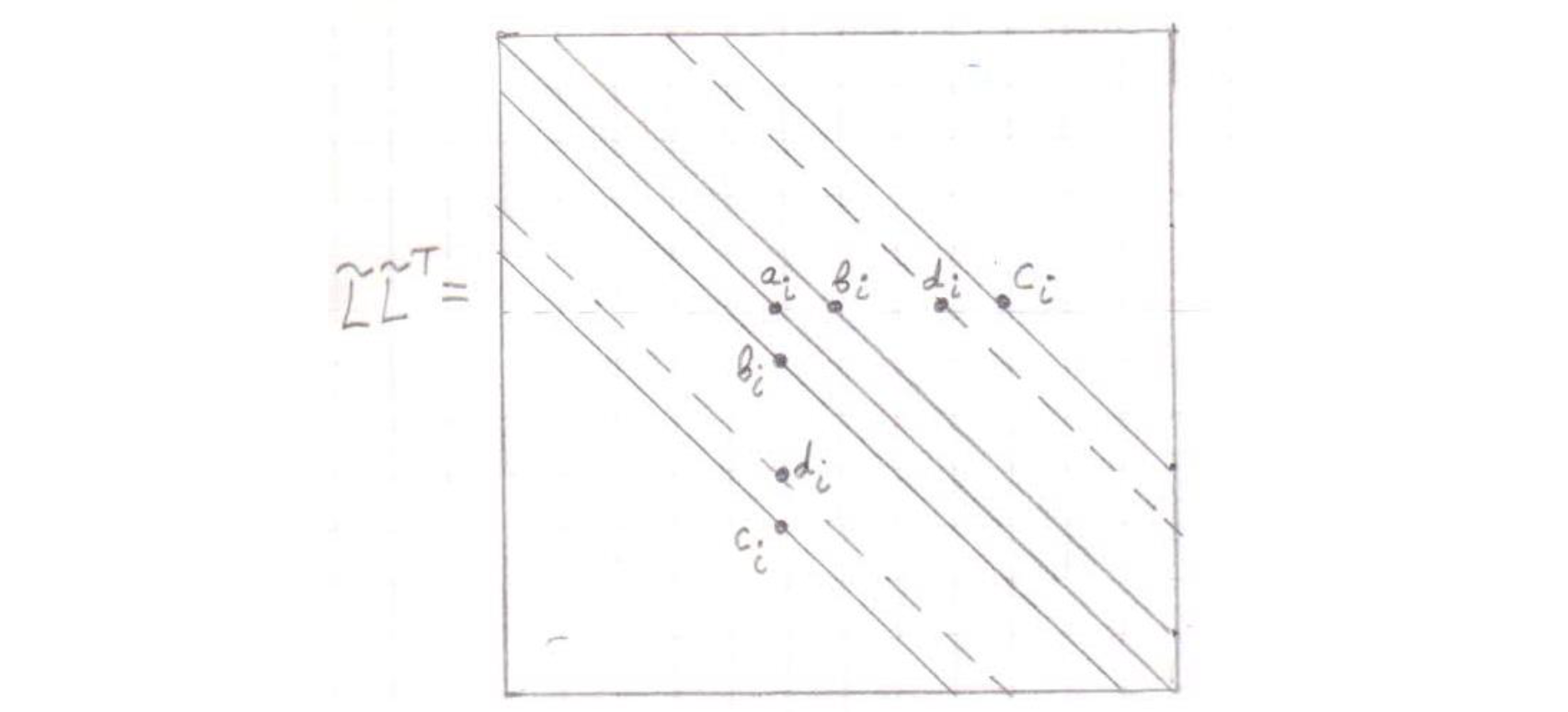
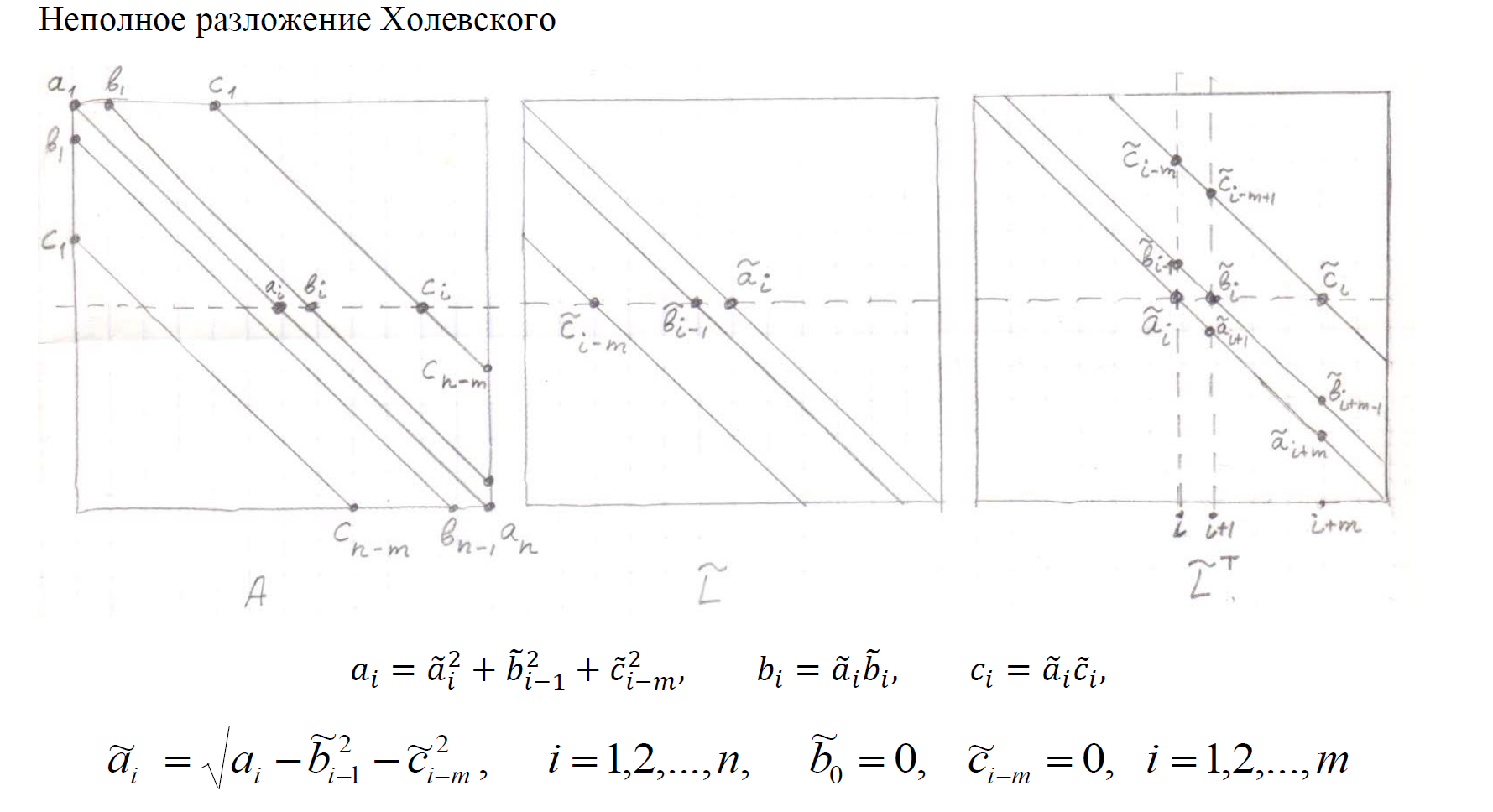
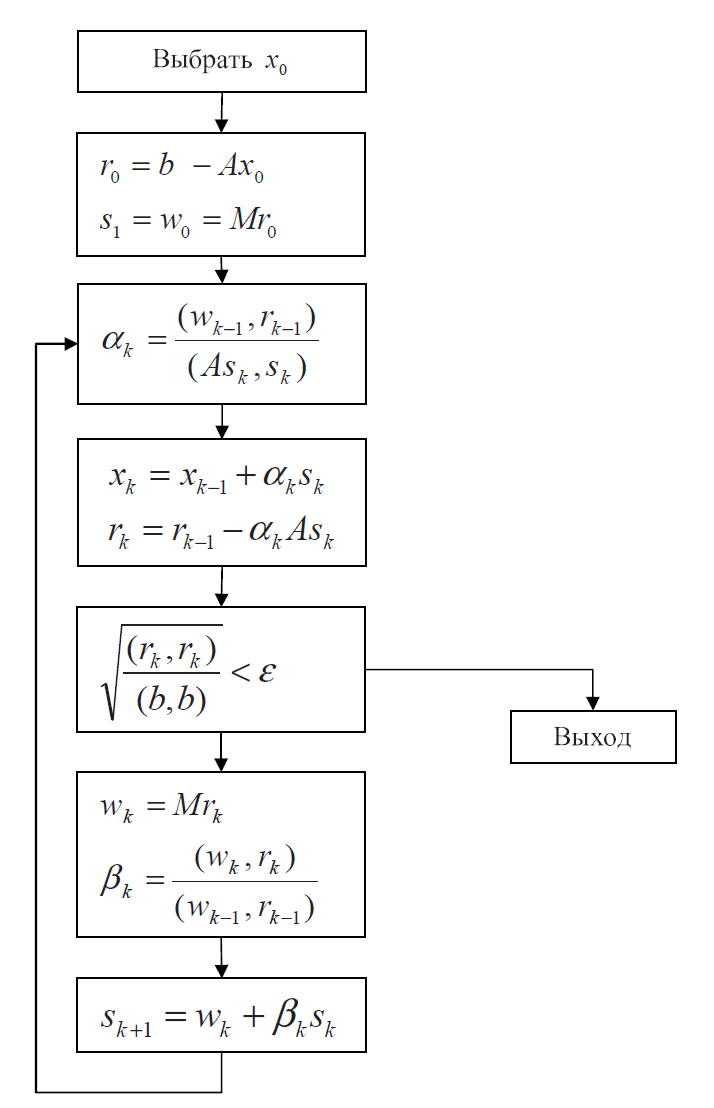


Схема применения метода выглядит следующим образом:



Здесь 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

1. import java.util.Arrays;
2. import java.util.HashMap;
3. import java.util.function.Function;
4. public class N4 {
5. private final static double EPS = 1e-6;
6. private static int N = 5;
7. private static final double R0 = 0;
8. private static final double R1 = 1;
9. private static final double L = 1;
10. private static final double Chi2 = 1;
11. private static final double Chi4 = 1;
12. private enum SystemParameters {
13. DIAGONAL\_A, DIAGONAL\_B, DIAGONAL\_C, VECTOR\_G
14. }
15. @FunctionalInterface
16. public interface FunctionTwoArgs<A, B, R> {
17. R apply(A a, B b);
18. }
19. public static void main(String[] args) {
20. System.out.println("N4");
21. System.out.println("---->>> Константый случай");
22. test(   (r, z) -> 1.0,
23. (r, z) -> 1.0,
24. (r, z) -> 0.0,
25. (z) -> 1.0,
26. (r) -> 1.0,
27. (r) -> 1.0,
28. (r, z) -> 1.0);
29. System.out.println("\n\n---->>> Линейный случай");
30. test(   (r, z) -> r + 1.0,
31. (r, z) -> z + 1.0,
32. (r, z) -> -1,
33. (z) -> z,
34. (r) -> 0,
35. (r) -> 3,
36. (r, z) -> z);
37. System.out.println("\n\n---->>> Нелинейный случай");
38. test(   (r, z) -> r \* r + 1,
39. (r, z) -> 1 + z \* z,
40. (r, z) -> -24 \* r\*r\*r\*r - 14 \* r\*r \* z\*z - 18\*r\*r - 4\*z\*z,
41. (z) -> 5\*z\*z + 9,
42. (r) -> r \* r \* r \* r,
43. (r) -> r \* r \* r \* r + 5 \* r \* r,
44. (r, z) -> r \* r \* r \* r + z \* z \* r \* r);
45. }
46. private static void test(FunctionTwoArgs<Double, Double, Double> k1,
47. FunctionTwoArgs<Double, Double, Double> k2,
48. FunctionTwoArgs<Double, Double, Double> f,
49. Function<Double, Double> phi2,
50. Function<Double, Double> phi3,
51. Function<Double, Double> phi4,
52. FunctionTwoArgs<Double, Double, Double> u)
53. {
54. HashMap<SystemParameters, double[]> system;
55. N = 5;
56. double hR = (R1 - R0) / (N - 1);
57. double hZ = L / (N - 1);
58. double r;
59. double z = 0;
60. double[] result = new double[N \* N];
61. system = getSystem(k1, k2, f, phi2, phi3, phi4);
62. for (int i = 0; i < N; ++i) {
63. r = R0;
64. for (int j = 0; j < N; ++j) {
65. result[i \* N + j] = u.apply(r, z);
66. r += hR;
67. }
68. z += hZ;
69. }
70. System.out.println("Отклонения от точного решения\n"
71. + Arrays.toString( sub(multiply(system, result),
72. system.get(SystemParameters.VECTOR\_G))));
73. System.out.println("Ошибка");
74. double prevError = 0;
75. double nowError;
76. N = 5;
77. System.out.println("\t\tN\tError\tRatio\t");
78. for (int i = 2; i <= 8; ++i) {
79. N = (int) Math.round(Math.pow(2, i)) + 1;
80. system = getSystem(k1, k2, f, phi2, phi3, phi4);
81. result = ConjugateGradientMethod(system, system.get(SystemParameters.VECTOR\_G), getEMatrix());
82. nowError = getMaxError(result, u);
83. System.out.println("\t\t " + (N - 1) + "\t " + nowError + " \t " + prevError / nowError);
84. prevError = nowError;
85. }
86. }
87. private static double[] ConjugateGradientMethod(HashMap<SystemParameters, double[]> system,
88. double[] first,
89. HashMap<SystemParameters, double[]> bMatrix) {
90. double[] result = Arrays.copyOf(first, first.length);
91. double[] r = sub(system.get(SystemParameters.VECTOR\_G), multiply(system, first));
92. double[] p = solveB(bMatrix, r);
93. double[] b = solveB(bMatrix, system.get(SystemParameters.VECTOR\_G));
94. double[] s = Arrays.copyOf(p, p.length);
95. double alpha; double beta; double[] newR; double[] newP; int k;
96. for (k = 1; k <= 10000; k++) {
97. alpha = multiply(p, r) / multiply(multiply(system, s), s);
98. result = addition(result, multiply(alpha, s));
99. newR = sub(r, multiply(alpha, multiply(system, s)));
100. newP = solveB(bMatrix, newR);
101. double check = Math.sqrt(multiply(newP, newR) / multiply(b, system.get(SystemParameters.VECTOR\_G)));
102. if (check < EPS) {
103. ++k;
104. break;
105. }
106. beta = multiply(newP, newR) / multiply(p, r);
107. s = addition(newP, multiply(beta, s));
108. r = newR; p = newP;
109. }
110. System.out.println("(Число итераций:\t" + k +")");
111. return result;
112. }
113. private static double[] getADiag(FunctionTwoArgs<Double, Double, Double> k2) {
114. double hR = (R1 - R0) / (N - 1);
115. double hZ = L / (N - 1);
116. double scale = hR / hZ;
117. double[] result = new double[N \* N];
118. double z = hZ;
119. double r;
120. for (int j = 1; j < N - 1; j++) {
121. r = R0;
122. result[j \* N] = -(scale / 4) \* r \* k2.apply(r, z - hZ / 2); *// #(2)*
123. r += hR;
124. for (int i = 1; i < N - 1; i++) {
125. result[j \* N + i] = -(scale) \* r \* k2.apply(r, z - hZ / 2); *// #(1)*
126. r += hR;
127. }
128. result[j \* N + N - 1] = -(scale / 2) \* r \* k2.apply(r, z - hZ / 2); *// #(3)*
129. z += hZ;
130. }
131. r = R0;
132. for (int i = 0; i < N; i++) {
133. result[N \* (N - 1) + i] = -scale \* r \* k2.apply(r, z - hZ / 2); *// #（5）*
134. r += hR;
135. }
136. return result;
137. }
138. private static double[] getCDiag(FunctionTwoArgs<Double, Double, Double> k1,
139. FunctionTwoArgs<Double, Double, Double> k2) {
140. double hR = (R1 - R0) / (N - 1); double hZ = L / (N - 1);
141. double scale = hZ / hR; double z = hZ;
142. double r;
143. double[] result = new double[N \* N];
144. for (int i = 0; i < N; i++) { *// #(4)*
145. result[i] = 1;
146. }
147. for (int j = 1; j < N - 1; j++) {
148. r = R0;
149. result[j \* N] = scale \* (r + hR / 2) \* k1.apply(r + hR / 2, z) *// #(2)*
150. + (1 / scale) \* r \* k2.apply(r, z + hZ / 2)
151. + (1 / scale) \* r \* k2.apply(r, z - hZ / 2);
152. r += hR;
153. for (int i = 1; i < N - 1; i++) {
154. result[j \* N + i] = scale \* (r + hR / 2) \* k1.apply(r + hR /2, z) *// #(1)*
155. + scale \* (r - hR / 2) \* k1.apply(r - hR / 2, z)
156. + (1 / scale) \* r \* k2.apply(r, z + hZ / 2)
157. + (1 / scale) \* r \* k2.apply(r, z - hZ / 2);
158. r += hR;
159. }
160. result[j \* N + N - 1] = hZ \* r \* Chi2 *// #(3)*
161. + scale \* (r - hR / 2) \* k1.apply(r - hR / 2, z)
162. + (1 / scale / 2) \* r \* k2.apply(r, z + hZ / 2)
163. + (1 / scale / 2) \* r \* k2.apply(r, z - hZ / 2);
164. z += hZ;
165. }
166. r = R0;
167. for (int i = 0; i < N; i++) {  *// #(5)*
168. result[N \* (N - 1) + i] =  scale \* (r + hR / 2) \* k1.apply(r + hR /2, z)
169. + scale \* (r - hR / 2) \* k1.apply(r - hR / 2, z)
170. + hR \* r \* Chi4
171. + (1 / scale) \* r \* k2.apply(r, z - hZ / 2);
172. r += hR;
173. }
174. return result;
175. }
176. private static double[] getDDiag(FunctionTwoArgs<Double, Double, Double> k1) {
177. double hR = (R1 - R0) / (N - 1);
178. double hZ = L / (N - 1);
179. double scale = hZ / hR;
180. double z = hZ;
181. double r;
182. double[] result = new double[N \* N];
183. for (int j = 1; j < N - 1; j++) {
184. r = R0;
185. for (int i = 0; i < N - 1; i++) {
186. result[j \* N + i] = -scale \* (r + hR / 2) \* k1.apply(r + hR /2, z); *// #(2) & (1)*
187. r += hR;
188. }
189. z += hZ;
190. }
191. return result;
192. }
193. private static double[] getEDiag(FunctionTwoArgs<Double, Double, Double> k2) {
194. double hR = (R1 - R0) / (N - 1); double hZ = L / (N - 1);
195. double scale = hR / hZ;
196. double[] result = new double[N \* N]; double z = hZ;
197. double r;
198. for (int j = 1; j < N - 1; j++) {
199. r = R0;
200. result[j \* N] = -scale \* r \* k2.apply(r, z + hZ / 2) / 2; r += hR; *// #(2)*
201. for (int i = 1; i < N - 1; i++) {
202. result[j \* N + i] = -scale \* r \* k2.apply(r, z + hZ / 2); r += hR; *// #(1)*
203. }
204. result[j \* N + N - 1] = -scale \* r \* k2.apply(r, z + hZ / 2) / 2; z += hZ; *// #(3)*
205. }
206. return result;
207. }
208. private static double[] getVectorG(FunctionTwoArgs<Double, Double, Double> f,
209. Function<Double, Double> phi2,
210. Function<Double, Double> phi3,
211. Function<Double, Double> phi4) {
212. double hR = (R1 - R0) / (N - 1); double hZ = L / (N - 1);
213. double[] result = new double[N \* N]; double z = hZ;
214. double r = R0;
215. for (int i = 0; i < N; i++) { *// #（4）*
216. result[i] = phi3.apply(r);
217. r += hR;
218. }
219. for (int j = 1; j < N - 1; j++) {
220. r = R0;
221. result[j \* N] = hR \* hZ \* r \* f.apply(r, z) / 4;  *// #（2）*
222. r += hR;
223. for (int i = 1; i < N - 1; i++) {
224. result[j \* N + i] = hR \* hZ \* r \* f.apply(r, z);  *// #（1）*
225. r += hR;
226. }
227. result[j \* N + N - 1] = hR \* hZ \* r \* f.apply(r, z) / 2 + hZ \* r \* phi2.apply(z); *// #（3）*
228. z += hZ;
229. }
230. r = R0;
231. for (int i = 0; i < N; i++) {
232. result[N \* (N - 1) + i] = hR \* r \* phi4.apply(r) + hR \* hZ \* r \* f.apply(r, z) / 2; *// #（5）*
233. r += hR;
234. }
235. return result;
236. }
237. private static HashMap<SystemParameters, double[]> getSystem(FunctionTwoArgs<Double, Double, Double> k1,
238. FunctionTwoArgs<Double, Double, Double> k2,
239. FunctionTwoArgs<Double, Double, Double> f,
240. Function<Double, Double> phi2,
241. Function<Double, Double> phi3,
242. Function<Double, Double> phi4)
243. {
244. double[] a = getADiag(k2);
245. double[] c = getCDiag(k1, k2);
246. double[] d = getDDiag(k1);
247. double[] e = getEDiag(k2);
248. double[] g = getVectorG(f, phi2, phi3, phi4);
249. for (int i = 0; i < N; i++) {
250. g[N + i] -= g[i] \* a[N + i];
251. a[N + i] = 0;
252. g[N \* (N - 2) + i] -= g[N \* (N - 1) + i] \* e[N \* (N - 2) + i];
253. e[N \* (N - 2) + i] = 0;
254. }
255. HashMap<SystemParameters, double[]> system = new HashMap<>();
256. system.put(SystemParameters.DIAGONAL\_A, c);
257. system.put(SystemParameters.DIAGONAL\_B, d);
258. system.put(SystemParameters.DIAGONAL\_C, e);
259. system.put(SystemParameters.VECTOR\_G, g);
260. return system;
261. }
262. private static double getMaxError(double[] solve, FunctionTwoArgs<Double, Double, Double> u) {
263. double hR = (R1 - R0) / (N - 1); double hZ = L / (N - 1);
264. double z = 0; double r;
265. double maxError = 0; double nowError;
266. for (int j = 0; j < N; j++) { r = R0;
267. for (int i = 0; i < N; i++) {
268. nowError = Math.abs(u.apply(r, z) - solve[j \* N + i]); if (nowError > maxError) {
269. maxError = nowError;
270. }
271. r += hR;
272. }
273. z += hZ;
274. }
275. return maxError;
276. }
277. private static HashMap<SystemParameters, double[]> getBMatrix(HashMap<SystemParameters, double[]> system)
278. {
279. HashMap<SystemParameters, double[]> result = new HashMap<>(); int squareN = N \* N;
280. double[] a = new double[squareN];
281. double[] b = new double[squareN];
282. double[] c = new double[squareN];
283. result.put(SystemParameters.DIAGONAL\_A, a);
284. result.put(SystemParameters.DIAGONAL\_B, b);
285. result.put(SystemParameters.DIAGONAL\_C, c);
286. a[0] = Math.sqrt(system.get(SystemParameters.DIAGONAL\_A)[0]);
287. for (int i = 1; i < N; i++) {
288. b[i - 1] = system.get(SystemParameters.DIAGONAL\_B)[i - 1] / a[i - 1];
289. a[i] = Math.sqrt(system.get(SystemParameters.DIAGONAL\_A)[i] - Math.pow(b[i - 1], 2));
290. }
291. for (int i = N; i < squareN; i++) {
292. c[i - N] = system.get(SystemParameters.DIAGONAL\_C)[i - N];
293. b[i - 1] = system.get(SystemParameters.DIAGONAL\_B)[i - 1] / a[i - 1];
294. a[i] = Math.sqrt(system.get(SystemParameters.DIAGONAL\_A)[i] - Math.pow(b[i - 1], 2) - Math.pow(c[i - N], 2));
295. }
296. return result;
297. }
298. private static double[] solveB(HashMap<SystemParameters, double[]> bMatrix, double[] g) {
299. int squareN = N \* N;
300. double[] y = new double[squareN];
301. double[] a = bMatrix.get(SystemParameters.DIAGONAL\_A);
302. double[] b = bMatrix.get(SystemParameters.DIAGONAL\_B);
303. double[] c = bMatrix.get(SystemParameters.DIAGONAL\_C);
304. y[0] = g[0] / a[0];
305. for (int i = 1; i < N; i++) {
306. y[i] = (g[i] - b[i - 1] \* y[i - 1]) / a[i];
307. }
308. for (int i = N; i < squareN; i++) {
309. y[i] = (g[i] - b[i - 1] \* y[i - 1] - c[i - N] \* y[i - N]) / a[i];
310. }
311. double[] result = new double[squareN];
312. result[squareN - 1] = y[squareN - 1] / a[squareN - 1];
313. for (int i = squareN - 2; i >= N \* (N - 1); i--) {
314. result[i] = (y[i] - b[i] \* result[i + 1]) / a[i];
315. }
316. for (int i = N \* (N - 1) - 1; i >= 0; i--) {
317. result[i] = (y[i] - b[i] \* result[i + 1] - c[i] \* result[i + N])/ a[i];
318. }
319. return result;
320. }
321. private static HashMap<SystemParameters, double[]> getEMatrix() {
322. HashMap<SystemParameters, double[]> e = new HashMap<>();
323. int squareN = N \* N;
324. double[] a = new double[squareN];
325. for (int j = 0; j < squareN; j++) {
326. a[j] = 1;
327. }
328. e.put(SystemParameters.DIAGONAL\_A, a); e.put(SystemParameters.DIAGONAL\_B, new double[squareN]); e.put(SystemParameters.DIAGONAL\_C, new double[squareN]); return e;
329. }
330. private static double multiply(double[] leftVector, double[] rightVector)
331. {
332. double result = 0;
333. for (int i = 0; i < leftVector.length; i++) {
334. result += leftVector[i] \* rightVector[i];
335. }
336. return result;
337. }
338. private static double[] multiply(HashMap<SystemParameters, double[]> system, double[] vector) {
339. double[] result = new double[vector.length];
340. 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++) {
341. result[i] = diagA[i] \* vector[i];
342. }
343. for (int i = 0; i < vector.length - 1; i++) { result[i] += diagB[i] \* vector[i + 1];
344. }
345. for (int i = 0; i < vector.length - N; i++) { result[i] += diagC[i] \* vector[i + N];
346. }
347. for (int i = 1; i < vector.length; i++) { result[i] += diagB[i - 1] \* vector[i - 1];
348. }
349. for (int i = N; i < vector.length; i++) { result[i] += diagC[i - N] \* vector[i - N];
350. }
351. return result;
352. }
353. private static double[] multiply(double number, double[] vector) { double[] result = new double[vector.length];
354. for (int i = 0; i < vector.length; i++)
355. {
356. result[i] = vector[i] \* number;
357. }
358. return result;
359. }
360. private static double[] addition(double[] leftVector, double[] rightVector) {
361. double[] result = new double[leftVector.length];
362. for (int i = 0; i < leftVector.length; i++) {
363. result[i] = leftVector[i] + rightVector[i];
364. }
365. return result;
366. }
367. private static double[] sub(double[] leftVector, double[] rightVector) { double[] result = new double[leftVector.length];
368. for (int i = 0; i < leftVector.length; i++) {
369. result[i] = leftVector[i] - rightVector[i];
370. }
371. return result;
372. }
373. }