Численные метода математической физики

Лабораторная работа №5

Тема: **“Разностные схемы для уравнения Пуассона”**

Дунаев Виктор, 6 группа, 3 курс

Вариант 4

Преподаватель

Будник А.М.

# Постановка задачи

Дана задача Дирихле для уравнения Пуассона в прямоугольнике

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Найти приближенное решение разностными итерационными методами на сетке узлов с шагами

# Условие

Вариант 4.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | Метод |
|  |  |  |  |  |  |  | Якоби |

Таким образом:

# Ход работы

Так как то имеет место неравномерный шаг сетки на предпоследнем узел сетки. Тогда разностная схема для неравномерного шага имеет вид:

где

Представим уравнение в виде разностной схемы. Будем считать, что сетка является неравномерной по двум переменным (для большей явности решения).

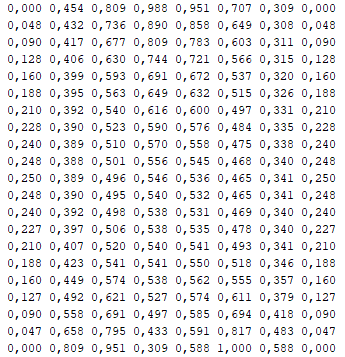
Перепишем в другом виде, чтобы можно было применить метод Якоби.

Недостающие условия доберем из доп. условий.

Для нулевой итерации возьмем:

Таким образом, границы сетки будет постоянны, а меняться будут только значения внутри сетки. В качестве критерия остановки возьмем:

# Результаты:



Для нахождения результатов с заданной точностью потребовалось 134 итерации.

# Листинг программы (на языке Java)

**import** java.util.Arrays;

**public class** DirichletEquation {  
 **private double h1** = 0.05; *//шаги* **private double h2** = 0.15;  
 **private double**[] **h1s**;  
 **private double**[] **h2s**;  
 **private double a** = -1; *//прямоугольник* **private double b** = 0;  
 **private double c** = 0;  
 **private double d** = 1;  
 **private double epsilon** = Math.*min*(**this**.**h1**, **this**.**h2**);  
 **private int n1** = (**int**) Math.*ceil*((**this**.**b** - **this**.**a**) / **this**.**h1**);  
 **private int n2** = (**int**) Math.*ceil*((**this**.**d** - **this**.**c**) / **this**.**h2**);  
  
 DirichletEquation() {  
 **this**.**h1s** = **new double**[**this**.**n1**];  
 **this**.**h2s** = **new double**[**this**.**n2**];  
 Arrays.*fill*(**this**.**h1s**, **this**.**h1**);  
 Arrays.*fill*(**this**.**h2s**, **this**.**h2**);  
 **this**.**h1s**[**this**.**n1** - 1] = (**this**.**b** - **this**.**a**) - (**this**.**n1** - 1) \* **this**.**h1**;  
 **this**.**h2s**[**this**.**n2** - 1] = (**this**.**d** - **this**.**c**) - (**this**.**n2** - 1) \* **this**.**h2**;  
 }  
  
 **public double** f(**double** x, **double** y) {  
 **return** Math.*cosh*(x\*x\*y);  
 }  
  
 **public double** psi1(**double** y) {  
 **return** Math.*sin*(Math.***PI*** \* y);  
 }  
  
 **public double** psi2(**double** y) {  
 **return** Math.*abs*(Math.*sin*(Math.***PI*** \*y\*2));  
 }  
  
 **public double** psi3(**double** x) {  
 **return** -x\*(x+1);  
 }  
  
 **public double** psi4(**double** x) {  
 **return** -x\*(x+1);  
 }  
  
 **public double** getH1Multiplier(**int** i, **int** j) {  
 **return** 1 / (0.5 \* (**this**.**h1s**[i] + **this**.**h1s**[j])) \* (1 / **this**.**h1s**[i] + 1 / **this**.**h1s**[j]);  
 }  
  
 **public double** getH2Multiplier(**int** i, **int** j) {  
 **return** 1 / (0.5 \* (**this**.**h2s**[i] + **this**.**h2s**[j])) \* (1 / **this**.**h2s**[i] + 1 / **this**.**h2s**[j]);  
 }  
  
 **public double**[] getX1() {  
 **double** step = **this**.**a**;  
 **double**[] x1 = **new double**[**this**.**n1** + 2];  
 x1[0] = step;  
 **for** (**int** i = 0; i <= **this**.**n1**; i++) {  
 step += **this**.**h1s**[i];  
 x1[i + 1] = step;  
 }  
 **return** x1;  
 }  
  
 **public double**[] getX2() {  
 **double** step = **this**.**c**;  
 **double**[] x2 = **new double**[**this**.**n2** + 2];  
 x2[0] = step;  
 **for** (**int** i = 0; i <= **this**.**n2**; i++) {  
 step += **this**.**h2s**[i];  
 x2[i + 1] = step;  
 }  
 **return** x2;  
 }  
  
 **public double**[][] getZeroIteration() {  
 **double**[][] result = **new double**[**this**.**n1** + 1][**this**.**n2** + 1];  
 **double** h1 = 0;  
 **for** (**int** j = 0; j < **this**.**n2**; j++) {  
 result[0][j] = psi1(**this**.**c** + h1);  
 result[**this**.**n1**][j] = psi2(**this**.**c** + h1);  
 h1 += **this**.**h2s**[j];  
 }  
 h1 = 0;  
 **for** (**int** i = 0; i < **this**.**n1**; i++) {  
 h1 += **this**.**h1s**[i];  
 result[i][0] = psi3(**this**.**a** + **this**.**h1** \* i);  
 result[i][**this**.**n2**] = psi4(**this**.**a** + **this**.**h1** \* i);  
 }  
 **double** h2 = 0;  
 h1 = 0;  
 **for** (**int** i = 1; i < **this**.**n1**; i++) {  
 h2 = **this**.**h2s**[1];  
 h1 += **this**.**h1s**[i];  
 **for** (**int** j = 1; j < **this**.**n2**; j++) {  
 h2 += **this**.**h2s**[j];  
 result[i][j] = f(**this**.**a** + h1, **this**.**c** + h2);  
 }  
 }  
 **return** result;  
 }  
  
 **public double**[][] getSolution() {  
 **double**[][] prevIteration = getZeroIteration();  
 **double**[][] curIteration = getZeroIteration();  
 **int** iterations = 1;  
 **while** (**true**) {  
 **for** (**int** i = 1; i < **this**.**n1**; i++) {  
 **for** (**int** j = 1; j < **this**.**n2**; j++) {  
 **double** first = 1 / (getH1Multiplier(i, i - 1) + getH2Multiplier(j, j - 1));  
 **double** second = (curIteration[i + 1][j] / **this**.**h1s**[i] + curIteration[i - 1][j] / **this**.**h1s**[i - 1]) \* 1 / ((**this**.**h1s**[i - 1] + **h1s**[i]) / 2);  
 **double** third = (curIteration[i][j + 1] / **this**.**h2s**[j] + curIteration[i][j - 1] / **this**.**h2s**[j - 1]) \* 1 / ((**this**.**h2s**[j - 1] + **h2s**[j]) / 2);  
 curIteration[i][j] = first \* (f(**this**.**a** + **this**.**h1** \* i, **this**.**c** + **this**.**h2** \* j) + second + third);  
 }  
 }  
 **if** (Utils.*norm*(Utils.*subtractMatrices*(prevIteration, curIteration)) < 0.0001) {  
 **break**;  
 } **else** {  
 System.***out***.println(Utils.*norm*(Utils.*subtractMatrices*(prevIteration, curIteration)));  
 Utils.*matrixCopy*(curIteration, prevIteration);  
 iterations++;  
 }  
 }  
 System.***out***.println(iterations);  
 **return** curIteration;  
 }  
  
  
 **public static void** main(String[] args) {  
 DirichletEquation dirichletEquation = **new** DirichletEquation();  
 **double**[][] res = dirichletEquation.getSolution();  
 **for** (**int** i = 0; i < res.**length**; i++) {  
 **for** (**int** j = 0; j < res[i].**length**; j++) {  
 System.***out***.printf(**"%.3f "**, res[i][j]);  
 }  
 System.***out***.println();  
 }  
  
 }  
}