БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

**Лабораторная работа №3**

**Разностная схема для уравнения Пуассона**

**Вариант 1**

Выполнил: Белоушко Степан

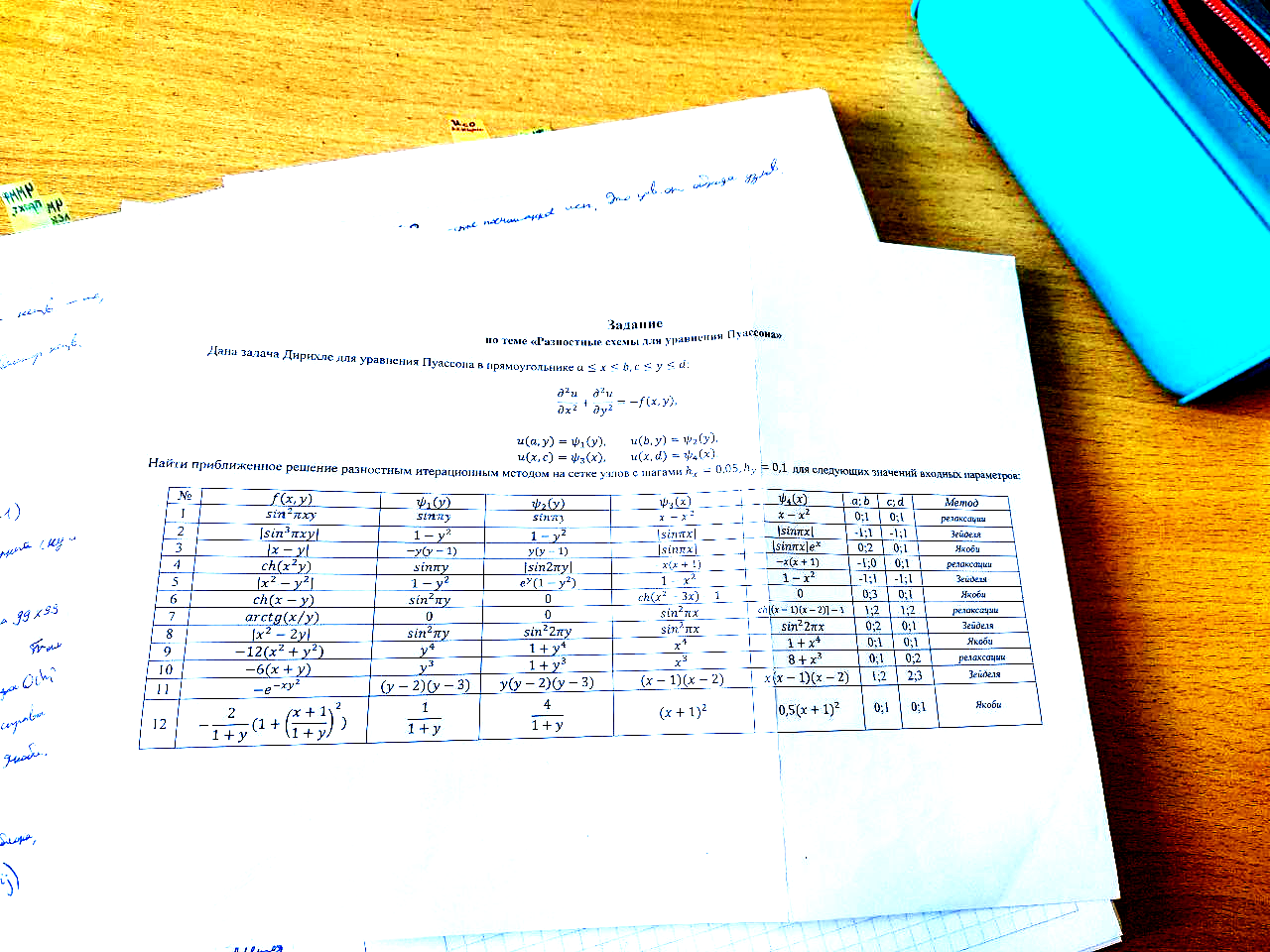
3 курс 7 группа

Преподаватель:

Будник Анатолий Михайлович

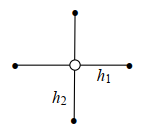
Минск, 2023

# Постановка задачи



# Теоретические сведения

Шаг . Количество разбиений

Возьмём шаблон как на рисунке. Разностная схема для уравнения Пуассона на сетке и на заданном шаблоне примет вид:

Индексная форма разностной схемы:

Она имеет второй порядок аппроксимации по каждой переменной,

Разрешим уравнение относительно центрального элемента шаблона:

Тогда метод Зейделя имеет следующий вид:

Метод релаксации отличается от метода Зейделя наличием итерационного параметра . С ним формула модифицируется следующим образом:

вычисляется так:

где

Вычисления останавливаем, когда . возьмём . В качестве начального приближения возьмём .

В качестве точного решения возьмём приближенное решение, полученное этим же методом с шагами .

# Листинг программы

import numpy as np

import pickle

def Relaxation\_method(ax, bx, ay, by, hx, hy, mu0y, mu1y, mux0, mux1, phi, epsilon):

delta = 2\*hy\*hy/(hx\*hx + hy\*hy)\*np.sin(np.pi\*hx/(2\*(bx-ax)))\*\*2 + 2\*hx\*hx/(hx\*hx + hy\*hy)\*np.sin(np.pi\*hy/(2\*(by-ay)))\*\*2

omega = 2/(1 + np.sqrt(delta\*(2-delta)))

X = np.arange(ax, bx+hx/2, hx)

Y = np.arange(ay, by+hy/2, hy)

matrix = np.zeros(shape=(X.size, Y.size))

for j in range(0, Y.size):

matrix[0][j] = mu0y(Y[j])

matrix[-1][j] = mu1y(Y[j])

for i in range(0, X.size):

matrix[i][0] = mux0(X[i])

matrix[i][-1] = mux1(X[i])

for i in range(1, X.size-1):

for j in range(1, Y.size-1):

matrix[i][j] = phi(X[i], Y[j])

norm = 1

iterations = 0

while(norm > epsilon):

norm = 0

iterations += 1

for i in range(1, X.size-1):

for j in range(1, Y.size-1):

y\_ = (1-omega)\*matrix[i][j] + omega/(2/(hx\*hx) + 2/(hy\*hy)) \* ((matrix[i+1][j] + matrix[i-1][j])/(hx\*hx) + (matrix[i][j+1] + matrix[i][j-1])/(hy\*hy) + phi(X[i], Y[j]))

norm = max(norm, y\_-matrix[i][j])

matrix[i][j] = y\_

return X, Y, matrix, iterations

ax = 0

bx = 1

ay = 0

by = 1

hx = 0.05

hy = 0.1

def mu0y(y):

return np.sin(np.pi\*y)

def mu1y(y):

return np.sin(np.pi\*y)

def mux0(x):

return x-x\*x

def mux1(x):

return x-x\*x

def phi(x, y):

return np.sin(np.pi \* x \* y) \*\* 2

X, Y, matrix, iterations = Relaxation\_method(ax, bx, ay, by, hx, hy, mu0y, mu1y, mux0, mux1, phi, hx\*\*3)

for i in range(0, X.size):

for j in range(0, Y.size):

print('%.4f' % matrix[i][j], end=' ')

print()

print('Количество итераций - ', iterations)

print()

matrix\_ = np.zeros(shape=(1,1))

matrix\_ = pickle.load(open("matrix.bin", "rb"))

iterations = pickle.load(open("iterations.bin", "rb"))

print('Количество итераций - ', iterations)

precision = 0

for i in range(0, X.size):

precision\_ = 0

for j in range(0, Y.size):

precision\_ += abs(matrix[i][j] - matrix\_[i\*10][j\*10])

precision = max(precision, precision\_/Y.size)

print("Норма погрешности - %.5f" % precision)

# X, Y, matrix, iterations = Relaxation\_method(ax, bx, ay, by, hx/10, hy/10, mu0y, mu1y, mux0, mux1, phi, hx\*\*3)

# for i in range(0, X.size):

# for j in range(0, Y.size):

# print('%.4f' % matrix[i][j], end=' ')

# print()

# print('Количество итераций - ', iterations)

# pickle.dump(matrix, open("matrix.bin", "wb"))

# pickle.dump(iterations, open("iterations.bin", "wb"))

# Результаты

0.0000 0.3090 0.5878 0.8090 0.9511 1.0000 0.9511 0.8090 0.5878 0.3090 0.0000

0.0475 0.3021 0.5369 0.7249 0.8463 0.8885 0.8470 0.7262 0.5383 0.3031 0.0475

0.0900 0.3001 0.4976 0.6574 0.7613 0.7978 0.7628 0.6601 0.5006 0.3023 0.0900

0.1275 0.3014 0.4677 0.6038 0.6931 0.7249 0.6955 0.6080 0.4724 0.3049 0.1275

0.1600 0.3045 0.4453 0.5619 0.6391 0.6671 0.6424 0.5677 0.4518 0.3094 0.1600

0.1875 0.3087 0.4288 0.5297 0.5972 0.6223 0.6015 0.5372 0.4373 0.3150 0.1875

0.2100 0.3129 0.4171 0.5057 0.5656 0.5885 0.5709 0.5149 0.4276 0.3208 0.2100

0.2275 0.3169 0.4091 0.4886 0.5430 0.5644 0.5493 0.4995 0.4215 0.3262 0.2275

0.2400 0.3202 0.4042 0.4775 0.5282 0.5486 0.5353 0.4899 0.4183 0.3307 0.2400

0.2475 0.3224 0.4018 0.4717 0.5205 0.5404 0.5283 0.4853 0.4173 0.3339 0.2475

0.2500 0.3234 0.4017 0.4707 0.5192 0.5392 0.5275 0.4852 0.4181 0.3356 0.2500

0.2475 0.3232 0.4035 0.4744 0.5241 0.5447 0.5328 0.4894 0.4205 0.3357 0.2475

0.2400 0.3217 0.4075 0.4827 0.5353 0.5569 0.5440 0.4978 0.4244 0.3340 0.2400

0.2275 0.3191 0.4138 0.4960 0.5530 0.5761 0.5616 0.5107 0.4301 0.3309 0.2275

0.2100 0.3156 0.4228 0.5148 0.5779 0.6029 0.5859 0.5285 0.4379 0.3264 0.2100

0.1875 0.3115 0.4351 0.5398 0.6108 0.6383 0.6181 0.5522 0.4486 0.3211 0.1875

0.1600 0.3075 0.4517 0.5722 0.6530 0.6834 0.6592 0.5827 0.4631 0.3154 0.1600

0.1275 0.3041 0.4736 0.6134 0.7060 0.7399 0.7109 0.6216 0.4825 0.3101 0.1275

0.0900 0.3023 0.5023 0.6651 0.7717 0.8099 0.7751 0.6708 0.5084 0.3064 0.0900

0.0475 0.3033 0.5396 0.7295 0.8525 0.8956 0.8542 0.7324 0.5427 0.3054 0.0475

0.0000 0.3090 0.5878 0.8090 0.9511 1.0000 0.9511 0.8090 0.5878 0.3090 0.0000

Количество итераций - 29

Количество итераций точного решения - 287

Норма погрешности - 0.00249

# Вывод

Разностная схема, которую мы использовали, имеет второй порядок аппроксимации по каждой переменной. Значит, погрешность аппроксимации должна быть равна Значения на границе вычисляются точно, потому там погрешность около нулевая (имеет место погрешность представления чисел в компьютере).

Из нормы погрешности видно, что реальная погрешность примерно соответствует ожидаемой.

Также, стоит отметить, что для вычисления точного решения требуется намного больше времени. То есть, нет смысла сильно уменьшать шаг из-за вычислительных трудностей метода.