МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра компьютерных технологий и систем

Лабораторная работа №2

**“Разностные схемы для обыкновенного дифференциального уравнения второго порядка”**

Вариант 2

Выполнил:

Ёда Никита Дмитриевич  
 студент 4 курса 6 группы

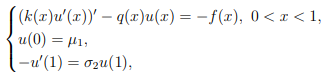
Преподаватель:

Репников Василий Иванович

**Задание 1**

Постановка задачи:

Дана дифференциальная задача, описывающая процесс стационарного распределения тепла в стержне единичной длины:



1. Построить разностную схему, заменяя дифференциальные производные разностными.

2. Методом баланса построить консервативную схему, составив уравнение баланса и проведя интегрирование по [𝑥𝑖−1, 𝑥𝑖].

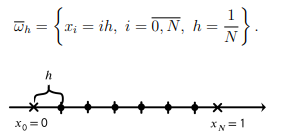
3. Построить вариационно-разностную схему методом Ритца.

4. Используя метод разностной прогонки, составить программу решения исходной задачи с помощью разностных схем п.п. 1-2, выполнить контрольные расчеты на ЭВМ и провести сравнительный анализ результатов.

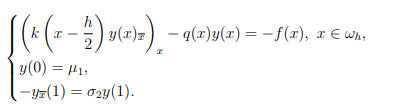
Решение:

1. Построение разностной схемы

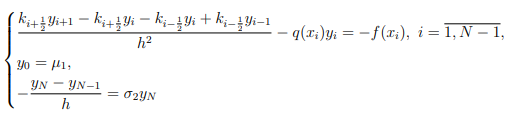
Зададим равномерную сетку узлов на отрезке [0,1]. На сетке определим функцию *y=y(x),* функция является приближенным решением задачи в условии.



Заменим дифференциальные производные разностными и построим схему в безиндексной форме:



Распишем все разностные производные, получим разностную схему в индексной форме:



Найдём погрешность аппроксимации дифференциального уравнения:



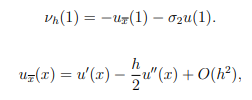




Дифференциальное уравнение аппроксимируется на шаблоне со вторым порядком. Теперь, рассмотрим погрешность аппроксимации левого граничного условия:



Погрешность правого условия:





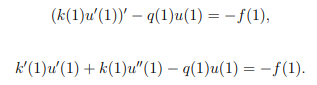
Правое граничное условие аппроксимируется с первым порядком. Общий порядок аппроксимации дифференциальной задачи разностной схемы:



Получили аппроксимацию первого порядка. Повысим порядок аппроксимации разностной схемы, не изменяя минимального шаблона. Правое граничное условие:



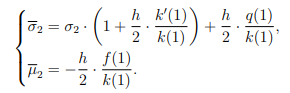
Предположим, что дифференциальное уравнение поставленной задачи выполняется на правой границе:



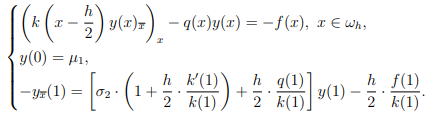




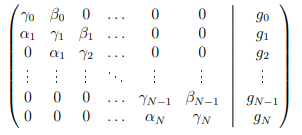


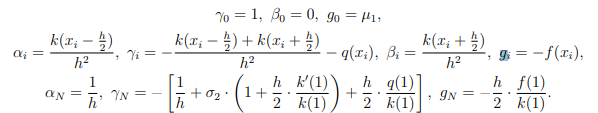


При таком выборе мы получим аппроксимацию правого граничного условия со вторым порядком. Следовательно, вся дифференциальная задача будет аппроксимироваться со вторым порядком разностной схемой вида:



Выпишем коэффициенты, которые будут образовывать трёх диагональную матрицу:





Для сходимости метода прогонки необходимо выполнение следующих условий:



Метод сходится т.к.





оба выражения под модулями слева неотрицательны, так как являются сеточными функциями, а справа при раскрытии модуля меняем знак

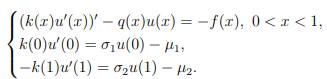






где при раскрытии модулей неравенство будет выполняться. Метод прогони для реализации разностной схемы сходится.

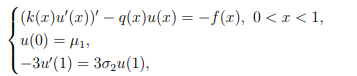
1. Построение консервативной разностной схемы методом баланса



Левое граничное условие:



Дифференциальная задача для метода баланса:

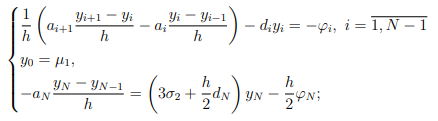


где

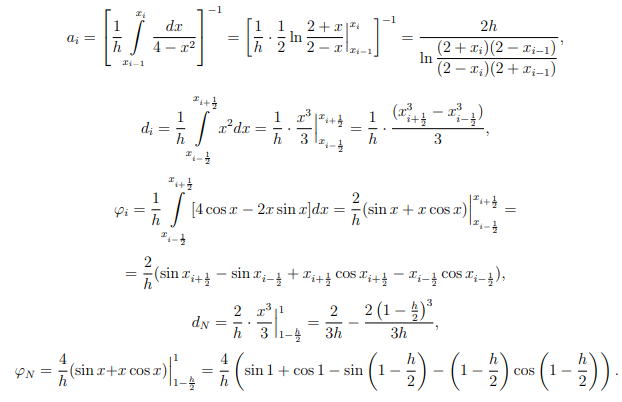




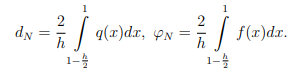
По методу баланса построим разностную схему:



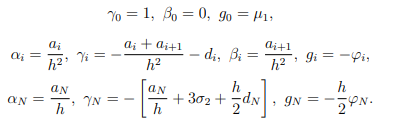
Определим коэффициенты:



Мы имеем общую формулу для итераций:

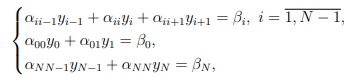


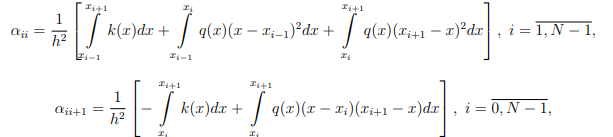
Разностная схема обладает вторым порядком аппроксимации. Метод прогонки сходится. Коэффициенты для метода прогонки:



1. Построение вариационно-разностной схемы

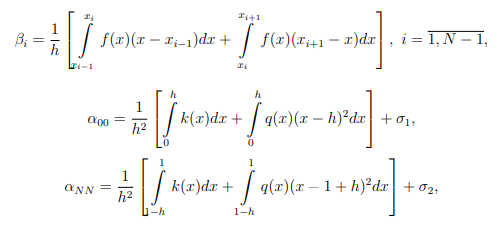
Возьмём заданную сетку узлов и сеточную функцию. По методу Ритца построим трёх диагональную систему вида:

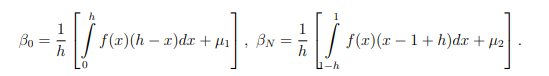




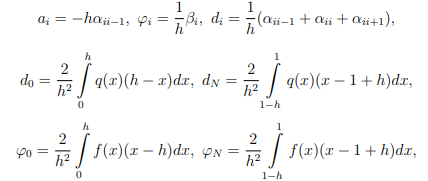


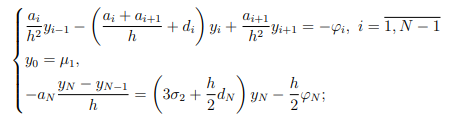
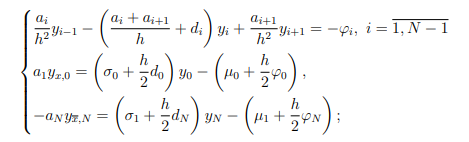
Вычислим



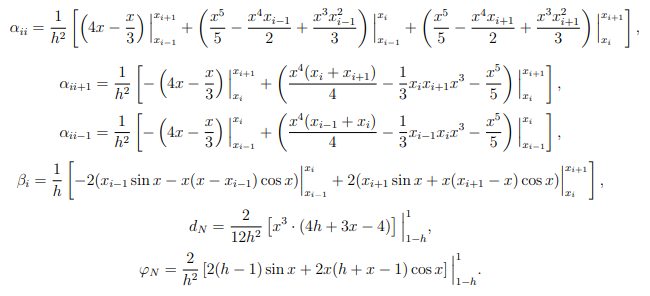


Нас интересуют граничные условия другого вида, поэтому мы будем вычислять:





Подставим неизвестные значения:

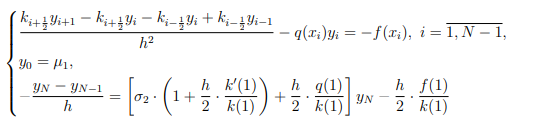


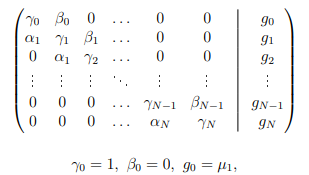
Разностная схема обладает вторым порядком аппроксимации. Метод прогонки сходится.

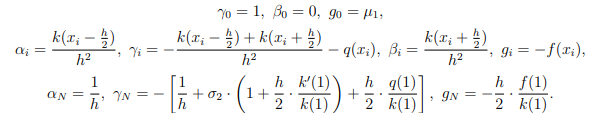
1. Реализация на ЭВМ

|  |
| --- |
| import numpy as np  import math  import matplotlib.pyplot as plt  *def* tridiagonal\_algorithm(*a*,*b*,*c*,*f*):  *a*, *b*, *c*, *f* = tuple(map(*lambda* *k\_list*: list(map(float, *k\_list*)), (*a*,*b*, *c*, *f*)))      alpha = [-*b*[0] / *c*[0]]      beta = [*f*[0] / *c*[0]]      n = len(*f*)      x = [0]\*n      for i in range(1, n):          alpha.append(-*b*[i]/(*a*[i]\*alpha[i-1] + *c*[i]))          beta.append((*f*[i] - *a*[i]\*beta[i-1])/(*a*[i]\*alpha[i-1] + *c*[i]))          x[n-1] = beta[n - 1]      for i in range(n - 1, 0, -1):          x[i - 1] = alpha[i - 1]\*x[i] + beta[i - 1]      return x  *def* k(*x*):      return 4 - *x*\*\*2  *def* dkdx(*x*):      return -2\**x*  *def* q(*x*):      return *x*\*\*2  *def* f(*x*):      return 4\*np.cos(*x*) - 2\**x*\*np.sin(*x*)  mu\_1 = 1  sigma\_2 = np.tan(1)  *def* u(*x*):      return np.cos(*x*)  a, b = 0, 1  N = 5  x = np.linspace(*start*=a, *stop*=b, *num*=N+1)  h = (b-a)/N |

* 1. Разностная аппроксимация







|  |
| --- |
| gamma = [1]  beta = [0]  g = [mu\_1]  alpha = [0]  for i in range(1, N):      alpha.append(k(x[i] - h/2) / h\*\*2)      gamma.append(-(k(x[i] - h/2) + k(x[i] + h/2)) / h\*\*2 - q(x[i]))      beta.append(k(x[i] + h/2) / h\*\*2)      g.append(-f(x[i]))  alpha.append(1/h)  gamma.append(- (1/h + sigma\_2 \* (1 + h/2 \* dkdx(1) / k(1)) + h/2 \* q(1) / k(1)))  g.append(-h/2 \* f(1) / k(1))  beta.append(0)  y = tridiagonal\_algorithm(alpha,beta,gamma,g)  plt.figure(*figsize*=(16, 8))  plt.plot(x, u(x), *label*='exact temperature')  plt.plot(x, y, *label*='numerical temperature')  plt.title('Аппроксимация разностными производными')  plt.grid(True)  plt.xlabel('x')  plt.ylabel('u(x)')  plt.legend()  plt.show() |

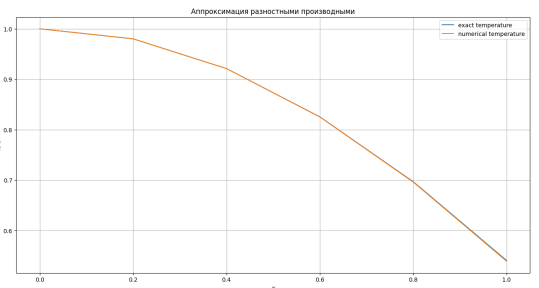
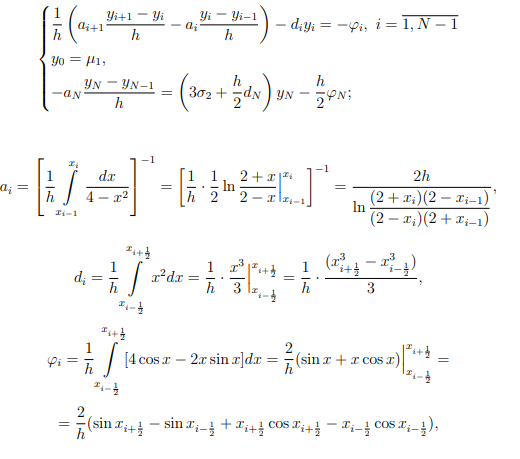
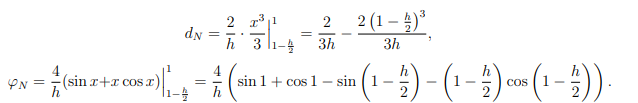


Рисунок 1 – Приближённое и точное решение

* 1. Метод баланса





|  |
| --- |
| def a\_i(x, h):      return 2 \* h / np.log((2+x)\*(2-(x-h)) / ((2-x)\*(2+(x-h))))  def d\_i(x, h):      return ((x+h/2)\*\*3 - (x-h/2)\*\*3)/(3\*h)  def phi\_i(x, h):      return 2/h \* (np.sin(x+h/2) - np.sin(x-h/2) + (x+h/2)\*np.cos(x+h/2)␣- (x-h/2)\*np.cos(x-h/2))  def d\_N(h):      return 2/(3\*h) \* (1 - (1-h/2)\*\*3)  def phi\_N(h):      return 4/h \* (np.sin(1) + np.cos(1) - np.sin(1-h/2) - (1-h/2)\*np.cos(1-h/2))  gamma = [1]  beta = [0]  g = [mu\_1]  alpha = [0]  for i in range(1, N):      alpha.append(a\_i(x[i], h) / h\*\*2)      gamma.append(-(a\_i(x[i],h) + a\_i(x[i+1],h))/h\*\*2 - d\_i(x[i], h))      beta.append(a\_i(x[i+1], h) / h\*\*2)      g.append(-phi\_i(x[i], h))  alpha.append(a\_i(x[N], h)/h)  gamma.append(-(a\_i(x[N], h) / h + 3 \* sigma\_2 + h/2 \* d\_N(h)))  g.append(-h/2 \* phi\_N(h))  beta.append(0)  y = tridiagonal\_algorithm(alpha,beta,gamma,g)  plt.figure(*figsize*=(16, 8))  plt.plot(x, u(x), *label*='exact temperature')  plt.plot(x, y, *label*='numerical temperature')  plt.title('Аппроксимация методом баланса')  plt.grid(True)  plt.xlabel('x')  plt.ylabel('u(x)')  plt.legend()  plt.show() |

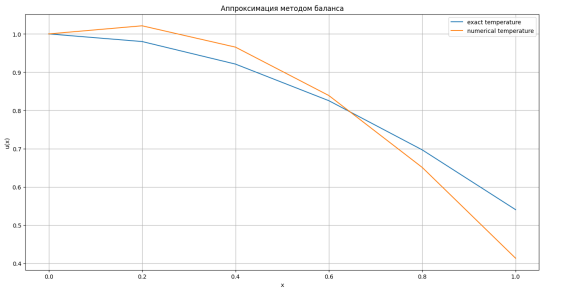
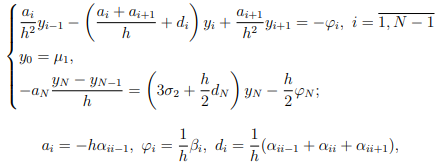
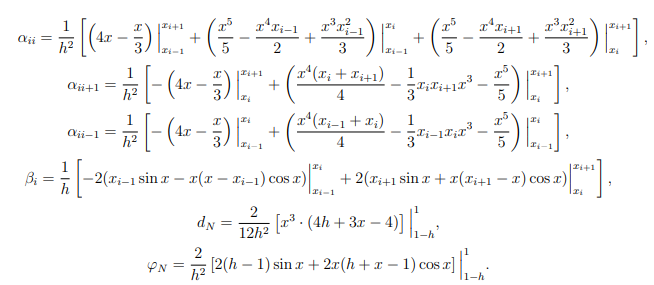


Рисунок 2 – Приближённое и точное решение

* 1. Метод Ритца





|  |
| --- |
| def alpha\_ii(x\_i, x\_im1, x\_ip1, h):      term1 = (4 \* x\_ip1 - x\_ip1 / 3) - (4 \* x\_im1 - x\_im1 / 3)      term2 = ((x\_i\*\*5 / 5 - x\_i\*\*4 \* x\_im1 / 2 + x\_i\*\*3 \* x\_im1\*\*2 / 3) -(x\_im1\*\*5 / 5 - x\_im1\*\*4 \* x\_im1 / 2 + x\_im1\*\*3 \* x\_im1\*\*2/ 3))      term3 = ((x\_ip1\*\*5 / 5 - x\_ip1\*\*4 \* x\_ip1 / 2 + x\_ip1\*\*3 \* x\_ip1\*\*2/ 3) -(x\_i\*\*5 / 5 - x\_i\*\*4 \* x\_ip1 / 2 + x\_i\*\*3 \* x\_ip1\*\*2 / 3))      result = (1 / h\*\*2) \* (term1 + term2 + term3)  return result  def alpha\_ii\_plus\_1(x\_i, x\_ip1, h):      term1 = -(4 \* x\_ip1 - x\_ip1 / 3) + (4 \* x\_i - x\_i / 3)      term2 = ((x\_ip1\*\*4 \* (x\_i + x\_ip1) / 4 - 1/3 \* x\_i \* x\_ip1 \*x\_ip1\*\*3 - x\_ip1\*\*5 / 5) -      (x\_i\*\*4 \* (x\_i + x\_ip1) / 4 - 1/3 \* x\_i \* x\_ip1 \* x\_i\*\*3 -x\_i\*\*5 / 5))      result = (1 / h\*\*2) \* (term1 + term2)  return result  def alpha\_ii\_minus\_1(x\_im1, x\_i, h):      term1 = -(4 \* x\_i - x\_i / 3) + (4 \* x\_im1 - x\_im1 / 3)      term2 = ((x\_i\*\*4 \* (x\_im1 + x\_i) / 4 - 1/3 \* x\_im1 \* x\_i \* x\_i\*\*3 -x\_i\*\*5 / 5) -(x\_im1\*\*4 \* (x\_im1 + x\_i) / 4 - 1/3 \* x\_im1 \* x\_i \*x\_im1\*\*3 - x\_im1\*\*5 / 5))      result = (1 / h\*\*2) \* (term1 + term2)  return result  def beta\_i(x\_i, x\_im1, x\_ip1, h):      term1 = -2 \* ((x\_i \* np.sin(x\_i) - x\_i \* (x\_i - x\_im1) \* npcos(x\_i)) -(x\_im1 \* np.sin(x\_im1) - x\_im1 \* (x\_im1 - x\_im1) \* np.cos(x\_im1)))      term2 = 2 \* ((x\_ip1 \* np.sin(x\_ip1) + x\_ip1 \* (x\_ip1 - x\_ip1) \* npcos(x\_ip1)) -(x\_i \* np.sin(x\_i) + x\_i \* (x\_ip1 - x\_i) \* np.cos(x\_i)))      result = (1 / h) \* (term1 + term2)  return result  def d\_N(h):      term1 = (1 - h)\*\*3 \* (4 \* h + 3 \* (1 - h) - 4)      term2 = 1\*\*3 \* (4 \* h + 3 \* 1 - 4)      result = (2 / (12 \* h\*\*2)) \* (term2 - term1)  return result  def phi\_N(h):      term1 = 2 \* ((h - 1) \* np.sin(1) + 1 \* (h + 1 - 1) \* np.cos(1)) - \2 \* ((h - 1) \* np.sin(1 - h) + (1 - h) \* (h + (1 - h) - 1)\* np.cos(1 - h))      result = (2 / h\*\*2) \* term1  return result  def a\_i(x, h):      return -h \* alpha\_ii\_minus\_1(x-h, x, h)  def d\_i(x, h):      return 1/h \* (alpha\_ii\_minus\_1(x-h, x, h) + alpha\_ii(x, x-h, x+h,h) + alpha\_ii\_plus\_1(x, x+h, h))  def phi\_i(x, h):      return 1/h \* beta\_i(x, x-h, x+h, h)  gamma = [1]  beta = [0]  g = [mu\_1]  alpha = [0]  for i in range(1, N):      alpha.append(a\_i(x[i], h) / h\*\*2)      gamma.append(-(a\_i(x[i],h) + a\_i(x[i+1],h))/h\*\*2 - d\_i(x[i], h))      beta.append(a\_i(x[i+1], h) / h\*\*2)      g.append(-phi\_i(x[i], h))  alpha.append(a\_i(x[N], h)/h)  gamma.append(-(a\_i(x[N], h) / h + 3 \* sigma\_2 + h/2 \* d\_N(h)))  g.append(-h/2 \* phi\_N(h))  beta.append(0)  y = tridiagonal\_algorithm(alpha,beta,gamma,g)  plt.figure(figsize=(16, 8))  plt.plot(x, u(x), label='exact temperature')  plt.plot(x, y, label='numerical temperature')  plt.title('Аппроксимация методом Ритца')  plt.grid(True)  plt.xlabel('x')  plt.ylabel('u(x)')  plt.legend()  plt.show() |

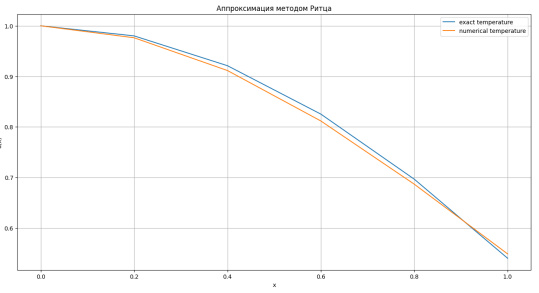


Рисунок 3 – Приближённое и точное решение