**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Численные методы математической физики

Лабораторная работа №1

«Решение краевой задачи для обыкновенного

дифференциального уравнения второго порядка»

Крученкова Евгения Андреевича

студента 3 курса,

специальность «Прикладная математика»

Преподаватель:

доцент кафедры ВычМат,

кандидат физико-математических наук

Будник А.М.

Минск, 2022

*ОГЛАВЛЕНИЕ*

*Постановка задачи. 3*

*1. Аппроксимация разностными операторами и повышение порядка аппроксимации 4*

*Краткие теоретичесие сведения 4*

*Результаты. 7*

*2. Интегро-интерполяционный метод построения консервативных разностных схем 8*

*Краткие теоретичесие сведения. 8*

*Результаты. 10*

*3. Вариационно-разностный метод 11*

*Краткие теоретичесие сведения. 11*

*Результаты 13*

*Листинг программы 14*

Постановка задачи.

Дана третья краевая задача для ОДУ второго порядка следующего вида:

где

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

Необходимо:

1. Используя разностные операторы вместо дифференциальных, аппроксимировать поставленную задачу разностной схемой 2-го порядка на минимальном шаблоне. Для повышения порядка аппроксимации граничных условий выделить главный член погрешности и заменить его, используя следующий вид исходного дифференциального уравнения: .
2. Интегро-интерполяционным методом построить консервативную разностную схему. Для вычисления коэффициентов этой схемы использовать формулу трапеций.
3. Аппроксимировать исходную задачу вариационно-разностным методом. Для вычисления коэффициентов полученной схемы использовать формулу средних прямоугольников.
4. Методом разностной прогонки реализовать полученные в п.п. 1-3 разностные схемы при . Провести сравнительный анализ сеточных решений при разных шагах.

# Аппроксимация разностными операторами и повышение порядка аппроксимации

## Краткие теоретичесие сведения

Рассмотррим исходную задачу:

Используя вид исходного дифференциального уравнения:

, запишем разностную схему, заменияя исходную функцию на сеточную, а дифференциальные операторы на разностные.

Введем сетку с N разбиениями и шагом h:

В нашем случае количество разбиений N = 10, шаг h = 0.1.

Введем сеточные функции:

В данном случае стоит обратить внимание на второе граничное условие. Заметим, что следовательно, будем иметь граничное условие первого рода:

Итак, запишем разностную схему:

где центральная разностная производная,

вторая разностная производная.

Покажем вывод формул и .

Отсюда и получаем искомые коэффициенты:

Второе же граничное условие записано точно.

Запишем индексную форму полученной разностной схемы:

Данная схема имеет второй порядок аппроксимации, т.е. .

Для реализации схемы используется метод разностной прогонки. Приведём схему к следующему виду:

Тогда алгоритм метода:

Приводя нашу схему к нужному виду, получаем выражение для коэффициентов:

В качестве точного решения в дальнейшем будем рассиматривать решение, полученное с помощью данного метода при N = 1000.

## Результаты.

## 

## Выводы.

Построенная схема имеет второй порядок аппроксимации, так как, хоть разностный оператор (правая разностная производная), использованный в аппроксимации граничного условия, обладает первым порядком, засчёт выбора коэффициентов мы повысили порядок до второго. Следовательно, погрешность аппроксимации должна составлять Как видно из вектора невязки точного решения и приближенного, реальная погрешность куда меньше. Это может быть связано с величиной коэффициента при главном члене погрешности. Также это может быть связано с тем, что отдно из граничных условий задаёт точное значение функции в точке.

# Интегро-интерполяционный метод построения консервативных разностных схем

## Краткие теоретичесие сведения.

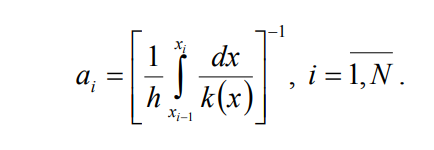
Построим разностную схему методом баланса для данной задачи:

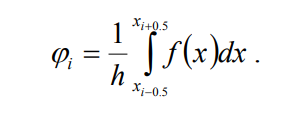
Разностная схема, построенная интегро-интерполяционным методом, будет иметь вид:

где

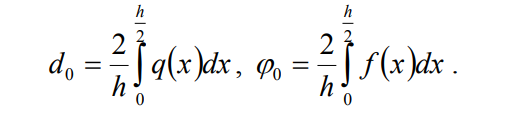
Как и в предыдущем случае, второе граничное условие записали точно.

Коэффициенты данной схемы будут иметь вид:





.



Для вычисления интегралов будет применяться формула средних прямоугольников:

Её погрешность: .

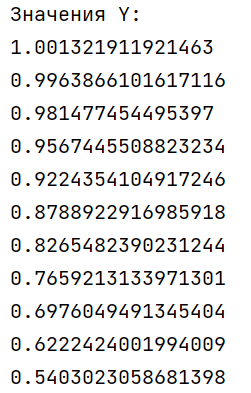
Снова для решения будем применять метод разностной прогонки, поэтому запишем схему в индексной форме:

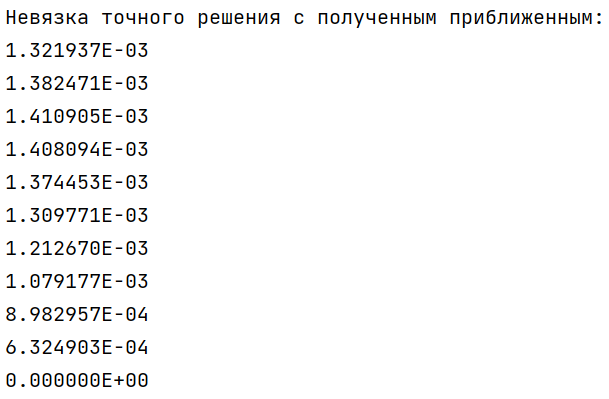
и придедем её к виду:

Получим выражения для коэффициентов:

Алгоритм метода:

## Результаты.





## Выводы.

Разностная схема, построенная в данном пункте, также имеет второй порядок аппроксимации. Следовательно, погрешность должна составлять И снова реальная погрешность, как видно из невязки, получилась меньше, чем теоретическая (опять же, это может быть связано с малым коэффициентом при главном члене погрешности или с тем, что одно из граничных условий задаёт точное значение), несмотря на то, что интегралы в формулах для коэффициентов были вычислены не точно, а приближённо. Для этого использовалась квадратурная формула средних прямоуголиников, имеющая алгебраическую степень точности, равную 1, и, следовательно, погрешность Так как в формулах всех коэффициентов присутствовало деление на h, то эта погрешность составит Рассмотрим вид погрешности квадратурной формулы средних прямоугольников: . Если значения производныx функций, интегралы которых мы находили с использованием квадратурной формулы, достаточно малы на промежутке [0,1], это может стать причиной того, что коэффициент при главном члене погрешности квадратурной формулы будет мал, и приближённое вычисление интегралов не повлечет за собой увеличение погрешности аппроксимации разностной схемы.

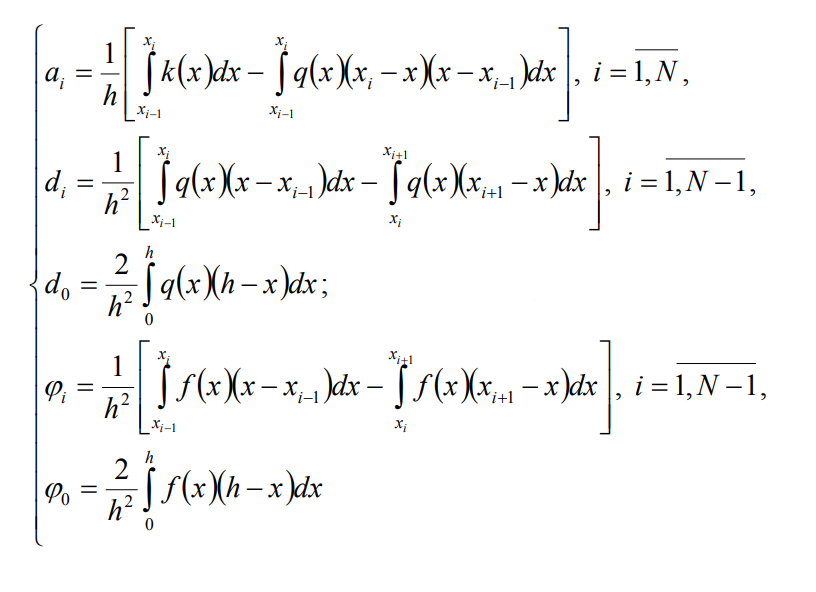
# Вариационно-разностный метод

## Краткие теоретичесие сведения.

Разностная схема, построенная по данному методу, будет иметь такой же вид, как и схема в предыдущем пункте:

где

Отличаться будут только коэффициенты, которые в двнном методе вычисляются по формулам:



При приближённом вычислении интегралов в этом пункте была использована квадратурная формула трапеций:

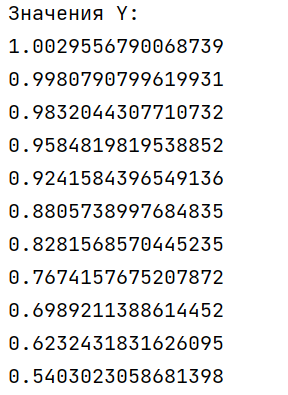
Её погрешность: .

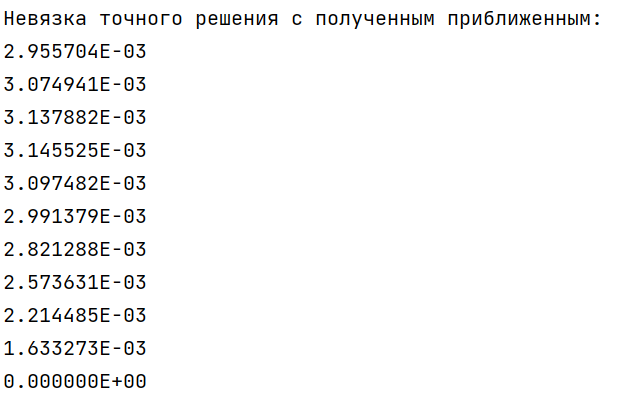
Для решения снова применяется метод разностной прогонки. Так как схема имеет тот же вид, что и прежде, метод будет иметь тот же вид:

Получим выражения для коэффициентов:

Алгоритм метода:

## Результаты.





## Выводы.

Данная разностная схема также имеет второй порядок аппроксимации, погрешность должна составлять Реальная погрешность, как видно из невязки, получилась около , что больше, чем в двух предыдущих пунктах, но всё ещё меньше, чем ожидаемая погрешность

Интегралы в формулах для коэффициентов были вычислены приближённо. Для этого использовалась квадратурная формула трапеций, имеющая алгебраическую степень точности, равную 1, и, следовательно, погрешность Так как в формулах всех коэффициентов присутствовало деление на , а под интегралам величина, приблизительно равная , то эта погрешность составит Рассмотрим вид погрешности квадратурной формулы трапеций: .

Так как под в формулах коэффициентов под интегралами находились другие фунции, то погрешность аппроксимации могла несколько увеличиться по сравнению с предыдущим пунктом засчёт бо́льших значений для подынтегральных функций, так как данные значения присутствуют в коэффициенте при главном члене погрешности квадратурной формулы.

Листинг программы

import math  
*################1*def K(x):  
 return (1-x\*x)  
def K\_(x):  
 return (-2\*x)  
def F(x):  
 return(math.cos(x)-2\*x\*math.sin(x))  
def q(x):  
 return(x\*x)  
def kappa0():  
 return(1/2)  
def g0():  
 return(1/2)  
def kappa1():  
 return(1)  
def g1():  
 return (math.cos(1))  
*#######*def A(x, h):  
 return ((-K\_(x)/(2\*h))+K(x)/(h\*h))  
def B(x, h):  
 return ((K\_(x) / (2 \* h)) + K(x) / (h\*h))  
def C(x, h):  
 return((2\*K(x))/(h\*h)+q(x))  
def K1(h):  
 return(K(0)/(h\*(kappa0()-(h/2)\*kappa0()\*(K\_(0)/K(0))+(h/2)\*q(0))+K(0)))  
def V1(h):  
 return((h\*(g0()-(h/2)\*g0()\*(K\_(0)/K(0))+(h/2)\*F(0)))/(h\*(kappa0()-(h/2)\*kappa0()\*(K\_(0)/K(0))+(h/2)\*q(0))+K(0)))  
def Progonka\_1(N):  
 h = 1/N  
 Y = [.0]\*(N+1)  
 alpha = [.0]\*(N+1)  
 beta = [.0]\*(N+1)  
 alpha[1] = K1(h)  
 beta[1] = V1(h)  
 for i in range(1, N):  
 x\_i = i\*h  
 alpha[i+1] = B(x\_i,h)/(C(x\_i, h)-A(x\_i, h)\*alpha[i])  
 beta[i+1] = (F(x\_i)+beta[i]\*A(x\_i, h))/(C(x\_i, h)-A(x\_i, h)\*alpha[i])  
 Y[N] = g1()/kappa1()  
  
 for i in range((N-1), -1, -1):  
 Y[i] = alpha[i+1]\*Y[i+1]+beta[i+1]  
 return (Y)  
  
def print\_Y(Y):  
 for i in range(len(Y)):  
 print(str(Y[i]))  
def print\_pogr(Y, Y\_t):  
 for i in range(len(Y)):  
 print(f"{abs(Y[i]-Y\_t[i]):E}")  
*####################2*def a(x, h):  
 return (K(x-(h/2)))  
def phi(x,h):  
 return (F(x))  
def d(x,h):  
 return (q(x))  
def d0(h):  
 return (q(h/4))  
def phi0(h):  
 return (F(h/4))  
  
def A\_(x, h):  
 return (a(x,h)/(h\*h))  
def B\_(x, h):  
 return (a(x+h,h)/(h\*h))  
def C\_(x, h):  
 return((a(x+h,h)+a(x,h))/(h\*h)+d(x,h))  
def F\_(x,h):  
 return phi(x,h)  
def K1\_(h):  
 return(a(h,h)/(h\*(kappa0()+(h/2)\*d0(h))+a(h,h)))  
def V1\_(h):  
 return((h\*(g0()+(h/2)\*phi0(h)))/(h\*(kappa0()+(h/2)\*d0(h))+a(h,h)))  
def Progonka\_2(N):  
 h = 1/N  
 Y = [.0]\*(N+1)  
 alpha = [.0]\*(N+1)  
 beta = [.0]\*(N+1)  
 alpha[1] = K1\_(h)  
 beta[1] = V1\_(h)  
 for i in range(1, N):  
 x\_i = i\*h  
 alpha[i+1] = B\_(x\_i,h)/(C\_(x\_i, h)-A\_(x\_i, h)\*alpha[i])  
 beta[i+1] = (F\_(x\_i, h)+beta[i]\*A\_(x\_i, h))/(C\_(x\_i, h)-A\_(x\_i, h)\*alpha[i])  
 Y[N] = g1()/kappa1()  
  
 for i in range((N-1), -1, -1):  
 Y[i] = alpha[i+1]\*Y[i+1]+beta[i+1]  
 return (Y)  
*#################3*def a\_(x, h):  
 return ((K(x)+K(x-h))/2)  
def phi\_(x,h):  
 return (F(x))  
def d\_(x,h):  
 return (q(x))  
def d0\_(h):  
 return (q(0))  
def phi0\_(h):  
 return (F(0))  
  
def A\_\_(x, h):  
 return (a\_(x,h)/(h\*h))  
def B\_\_(x, h):  
 return (a\_(x+h,h)/(h\*h))  
def C\_\_(x, h):  
 return((a\_(x+h,h)+a\_(x,h))/(h\*h)+d\_(x,h))  
def F\_\_(x,h):  
 return phi\_(x,h)  
def K1\_\_(h):  
 return(a\_(h,h)/(h\*(kappa0()+(h/2)\*d0\_(h))+a\_(h,h)))  
def V1\_\_(h):  
 return((h\*(g0()+(h/2)\*phi0\_(h)))/(h\*(kappa0()+(h/2)\*d0\_(h))+a\_(h,h)))  
def Progonka\_3(N):  
 h = 1/N  
 Y = [.0]\*(N+1)  
 alpha = [.0]\*(N+1)  
 beta = [.0]\*(N+1)  
 alpha[1] = K1\_\_(h)  
 beta[1] = V1\_\_(h)  
 for i in range(1, N):  
 x\_i = i\*h  
 alpha[i+1] = B\_\_(x\_i,h)/(C\_\_(x\_i, h)-A\_\_(x\_i, h)\*alpha[i])  
 beta[i+1] = (F\_\_(x\_i, h)+beta[i]\*A\_\_(x\_i, h))/(C\_\_(x\_i, h)-A\_\_(x\_i, h)\*alpha[i])  
 Y[N] = g1()/kappa1()  
  
 for i in range((N-1), -1, -1):  
 Y[i] = alpha[i+1]\*Y[i+1]+beta[i+1]  
 return (Y)  
*##################*def Print\_Res(N1, N2):  
 Y\_t1000 = Progonka\_1(N2)  
 Y\_t = [.0] \* (N1+1)  
 for i in range(N1+1):  
 Y\_t[i] = Y\_t1000[i\*(N2//N1)]  
 print("Аппроксимация разностными операторами и повышение порядка точности:")  
 Y1 = Progonka\_1(N1)  
 print("Значения Y1:")  
 print\_Y(Y1)  
 print("Невязка точного решения с полученным приближенным:")  
 print\_pogr(Y\_t, Y1)  
 print("Интегро-интерполяционный метод:")  
 Y2 = Progonka\_2(N1)  
 print("Значения Y2:")  
 print\_Y(Y2)  
 print("Невязка точного решения с полученным приближенным:")  
 print\_pogr(Y\_t, Y2)  
 print("Вариационно-разностный метод:")  
 Y3 = Progonka\_3(N1)  
 print("Значения Y3:")  
 print\_Y(Y3)  
 print("Невязка точного решения с полученным приближенным:")  
 print\_pogr(Y\_t, Y3)  
*###################main*Print\_Res(10,1000)