Санкт-Петербургский политехнический университет
Институт прикладной математики и механики
Высшая школа теоретической механики

Направление подготовки
"01.03.03 Механика и математическое моделирование"

Отчет по лабораторной работе №6
Тема работы: " Численное решение краевых задач. Метод пристрелки"
Дисциплина: "Численные методы"

Выполнил студент гр. 3630103/90001 Михеев Евгений Викторович
Преподаватель: Павлова Людмила Владимировна

Санкт-Петербург
2021

Отчет по лабораторной работе №7

"Решение краевой задачи"

метод пристрелки

№1) Постановка задачи:

Реализовать и исследовать численный метод пристрелки решения краевой задачи I рода. Изучить погрешность метода а также исследовать его на устойчивость к возмущениям входных данных.

№2) Алгоритм метода

Поставлена краевая задача: $\begin{cases} y'' + p(x)y' + q(x)y = f(x) \\ y(a) = y_a \\ y(b) = y_b \end{cases}$

Будем устремлять к 0 ф-ию $\varphi(C) = |y(b,C) - y_b|$

где $C = y'(a)$, т.е задача сводится к решению уравнения $|y(b,C) - y_b| = 0$, относительно C

Будем решать его методом секущих:

$$C_{k+1} = C_k - \frac{\varphi(C_k)(C_k - C_{k-1})}{\varphi(C_k) - \varphi(C_{k-1})}$$

Для каждого C_k решения задача Коши:

$$\begin{cases} y'' = f(x, y, z') \\ y(a) = y_a \\ y'(a) = C_k \end{cases}$$

Если C_k является корнем с заданной точностью ε, то прекращаем поиск и решением к.з будет $y = y(C_k)$

№3 $\exists$ и ! решения

Вопрос $\exists$ и ! решения краевой задачи довольно сложен и вообще говоря зависит от конкретной задачи, которая может как иметь одно или ∞ число решений, так и не иметь их вовсе

№4) Пусть расчет: $y(0) = 1$, $y(b) = 3,7$

$y'' = 2xy' + 2y - 4x$ Будем решать задачу с точностью $\varepsilon = 0,1$

$\exists\, C_0 = 0$

$C_1 = 0,1$

I) Имеем задачу Коши: $\begin{cases} y(0) = 1 \\ y'(0) = C_0 \end{cases}$

$\exists$ ее решение найдено: ~~$\exists$~~ (используем метод 3-4 IV порядка)

$x_0 = 0$, $x_1 = 1$

$y_0 = 1$, $y_1 = 1,75 = y_в(C_0)$

II) $\begin{cases} y(0) = 1 \\ y'(0) = C_1 \end{cases} \Rightarrow y(b, C_1) = 1,93$

Применим метод секущих: $\varphi(C_1) = |3,7 - 1,93| = 1,77$

$\varphi(C_0) = |3,7 - 1,75| = 1,95$

$C_2 = 0,1$

$$C_2 = 0{,}1 - 1{,}97\frac{0{,}1-0}{1{,}47-1{,}95} = 1{,}01$$

III) $y(0) = 1$
$y'(0) = C_2 = 1{,}01$ | $y(b, C_2) = 3{,}73$

точное $y(b) = 3.7$ | решение найдено с точностью ε
$\varepsilon = 0{,}1$

$\begin{pmatrix} x_0 = 0, & x_1 = 1 \\ y_0 = 1, & y_1 = 3{,}73 \end{pmatrix}$

№5) Контрольные тесты
Поставлена задача решить краевую задачу.

$$\begin{cases} y'' = 2xy' + 2xy - 4x \\ y(0) = 1 \\ y(1) = 1 + e \end{cases}$$

Для этого на я.п Python при помощи библиотеки numpy написана программа, позволяющая:
- решить краевую задачу с точностью на правом конце $\varepsilon = 10^{-15}$
- исследовать погрешность в каждой точке применяя к ней правило Рунге и сравнить ее с абсолютной погрешностью
- исследовать устойчивость метода к возмущению в $y(0)$

№6) Структура программы:

script shooting.py ⟷ class DE def shooter()
⟷ def tableMaker()
⟷ def disturbance(.)

1) DE.shooter()

In: y'', a, b, n, y_a, y_b

ф-ция решает краевую задачу с точностью $\varepsilon = 10^{-15}$ на правой границе

Out: массив типу ([x, y])

2) def tableMaker()

In: y'', a, b, n = 10, y_a, y_b

При фиксированном n = 10 функция вычисляет в каждой точке абс. погрешность и оценивает погрешность по правилу Рунге

Out: таблица: абс и Рунге - погрешность от точки график этой таблицы

3) def disturbance()

In: y'', a, b, n = 5, y_a, y_b

ф-ция вносит произвольную погрешность из промежутка [0, 1] в наш условие $y(0)$ и вычисляет погрешность относительно невозмущенного решения в 5 точках промежутка

Out: таблица: ошибка от внесенного возмущения

№7) Численный анализ:

– Сходимость метода и ее скорость существенно зависят от метода поиска $y'(0)$. При выбранном нами методе секущих и начальных $C_0 = 0$, $C_1 = 0{,}1$ (шаг равный = 1) верный C находится всего за 2 итерации, т.е требуется решить 4 задачи Коши, однако скорее всего в некоторых случаях будет затруднителен выбор C_0, C_1. Также С можно искать методом половинного деления, но это потребует бо́льшего решения задач Коши.

– Устойчивость решения д/у в первую очередь зависит от устойчивости самого д/у и только потом от выбора численного метода. В случае данного уравнения при внесении ошибки в $y(0)$ в районе $[-1, 1]$ возмущение решения ни в одной из точек не превосходит внесенного возмущения (см. таблицу ниже) что говорит об устойчивости метода

Возмущение	0.008286920	0.22612338	0.202594356	0.613731136	0.886461977
Ошибка в точке 0	0,00828692	0,22612338	0,202594356	0,613731137	0,886461977
Ошибка в точке 1	0,0063457	0,173153741	0,155136416	0,469963974	0,67880733
Ошибка в точке 2	0,004780969	0,130457252	0,116882664	0,354079607	0,511426078
Ошибка в точке 3	0,003366213	0,091853109	0,082295433	0,249302451	0,360087879
Ошибка в точке 4	0,001875776	0,05118389	0,045858005	0,138920384	0,20065405
Ошибка в точке 5	4,44089E-16	4,44089E-16	4,44089E-16	4,44089E-16	4,44089E-16

– Изучим график зависимостей абс. погр. и погр. по Рунге от узла

Погрешность ε нахождения $C = 10^{-15}$ $(\text{т.е} |y(b,c) - y_B| < \varepsilon)$

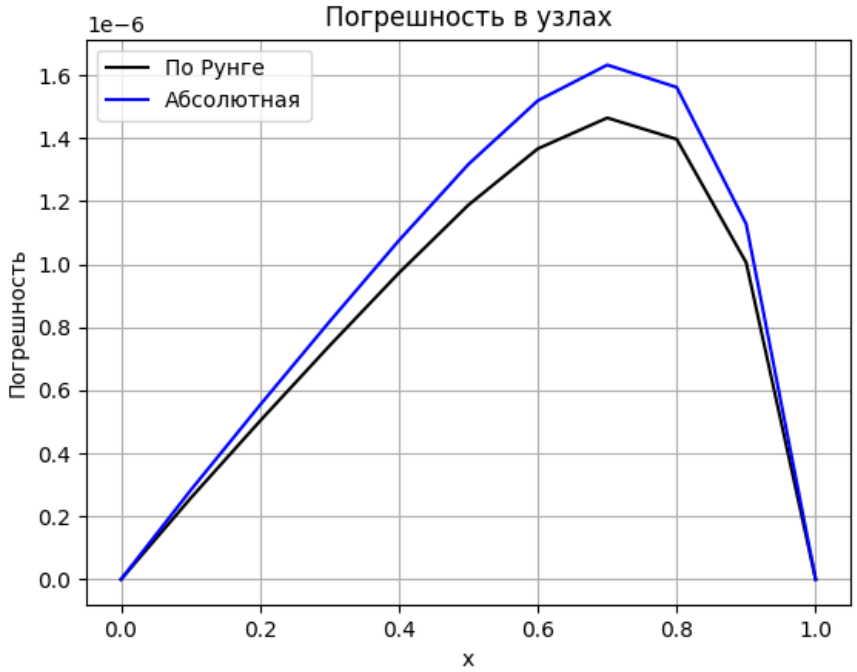
$n = 10$

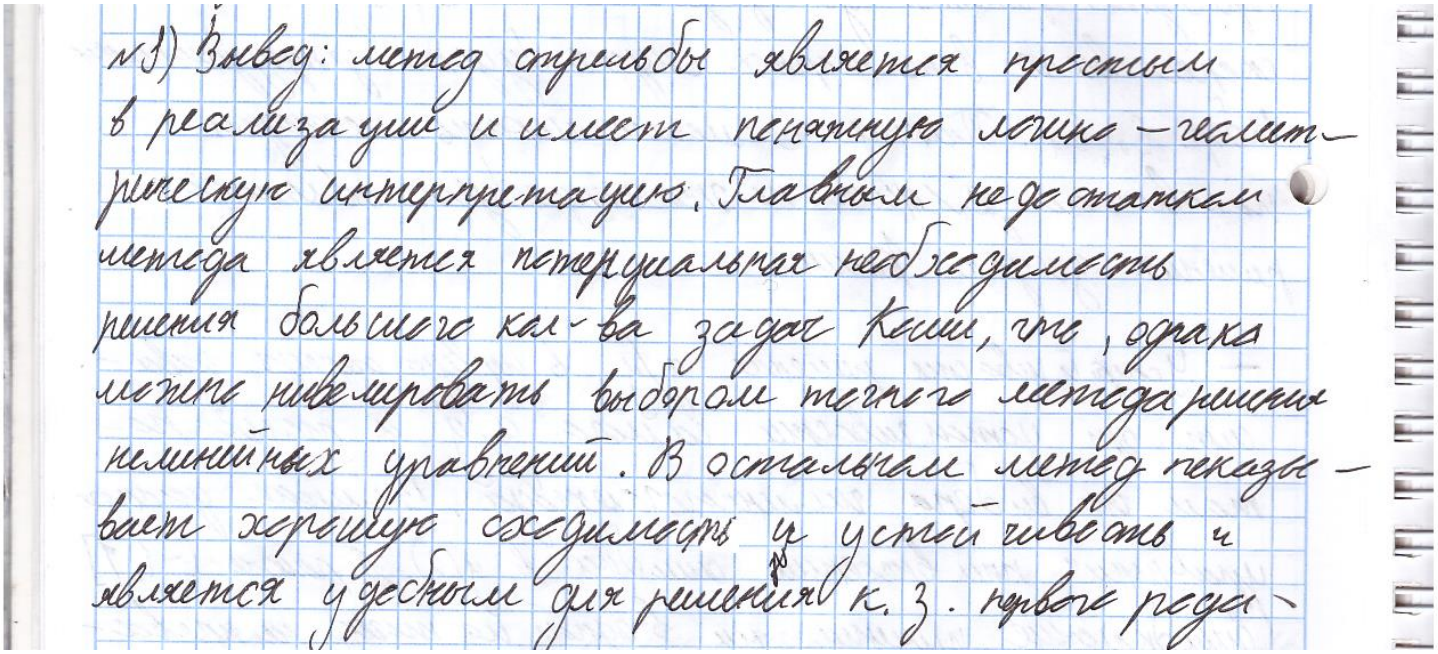
Из графика видно, что основная краевая задача решена – на концах промежутка погрешность достаточно мала, т.е полученное решение удовлетворяет своим краевым условиям: $y(0) = 0$

$y(1) = 1 + e$

Внутри промежутка погрешность ожидаемо больше (кстати из графика также видна корреляция оценки по Рунге и абс. погрешности – правило Рунге с точностью до порядка оценивает абс. погрешность), однако ошибка падает к краям промежутка, что подтверждает сходимость метода.

Погрешность	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Рунге	0	2,57E-07	5,03E-07	7,41123E-07	9,72E-07	1,19E-06	1,37E-06	1,46E-06	1,4E-06	1,01E-06	2,96E-17
Абсолютная	0	2,82E-07	5,53E-07	8,17656E-07	1,08E-06	1,32E-06	1,52E-06	1,63E-06	1,56E-06	1,13E-06	4,44E-16




№3) Вывод: метод стрельбы является простым в реализации и имеет понятную логика – геометрическую интерпретацию. Главным недостатком метода является потенциальная необходимость решения большого кол-ва задач Коши, что, однако легко нивелировать выбором такого метода решения нелинейных уравнений. В остальном метод показывает хорошую сходимость и устойчивость и является удобным для решения к.з. первого рода

Код

Class DE, def Shooter

```
def Shooter(self, y_exact):

        Cpp, Cp = 0, 0.1
        self.yb = y_exact(self.b)
        self.dy0 = Cpp
        solpp = self.RungeIV()
        self.dy0 = Cp
        solp = self.RungeIV()
        k = 0
        while True:
            k+=1
            eps = math.pow(10,-15)
            deltap = solp[1][self.n] - self.yb
            deltapp = solpp[1][self.n] - self.yb
            Cn = Cp - deltap*(Cp-Cpp)/(deltap - deltapp)
            self.dy0 = Cn
            soln = self.RungeIV()
            delta = soln[1][self.n]-self.yb
            if math.fabs(delta)<eps:
                break
            else:
                Cpp = Cp
                Cp = Cn
                solpp = solp
                solp = soln

                continue
        return soln
```

Script shooter.py

```
import math
import random
import statistics
import numpy as np
```

```
from NMclasses import DE
from NMclasses import Plots
import matplotlib.pyplot as plt
import pandas as pd

d2y = lambda x, y, dy: 2*x*dy +2*y - 4*x
yR = np.vectorize(lambda x: x + np.exp(x**2))

y0 = 1
dy0 = 1
a, b = 0, 1
xR = np.linspace(a, b, 100)

def tableMaker():
    de1 = DE(d2y, a, b, 20, y0, dy0)
    de2 = DE(d2y, a, b, 10, y0, dy0)
    sol10 = de1.Shooter(yR)
    sol5 = de2.Shooter(yR)
    result = {}
    Eps, Epsr = [], []
    for i in range(sol5.shape[1]):
        E = math.fabs(sol5[1][i]-sol10[1][2*i])/15
        Er = math.fabs(yR(sol10[0][2*i])-sol10[1][2*i])
        result[f"{sol10[0][2*i]}"] = [E, Er]
        Eps.append(E)
        Epsr.append(Er)
    table = pd.DataFrame(result, index = ['Рунге','Абсолютная'])
    table.to_excel(r"C:\Users\mchav\OneDrive\Учеба\Предметы\ЧМ\2 сем\7\Отчет\table.xlsx")

    return Eps, Epsr

def disturbance():
    result = {}
    de = DE(d2y, a, b, 5, y0, dy0)
    undisturbedSolution = de.Shooter(yR)
    for i in range(5):
        r = random.random()
        de.y0 += r
        disturbedSolution = de.Shooter(yR)
        d = []
        for j in range(disturbedSolution.shape[1]):
            d.append( math.fabs(disturbedSolution[1][j] - undisturbedSolution[1][j]) )
        de.y0 -=r
        result[f"{r}"] = d
    table = pd.DataFrame(result, index = ['Ошибка в точке 0', 'Ошибка в точке 1', 'Ошибка в то
чке 2', 'Ошибка в точке 3', 'Ошибка в точке 4', 'Ошибка в точке 5'])
    table.to_excel(r"C:\Users\mchav\OneDrive\Учеба\Предметы\ЧМ\2 сем\7\Отчет\tableDisturbance.
xlsx")

if __name__ == '__main__':
    Eps, Epsr = tableMaker()
    disturbance()
    de = DE(d2y, a, b, 10, y0, dy0)
    sol = de.Shooter(yR)
```

```
    testSol = DE(d2y, a, b, 10, y0, 1.08).RungeIV()
    Plots(1, [[sol[0], Eps], [sol[0], Epsr]], 'Погрешность в узлах', 'x', 'Погрешность', ['По
Рунге', 'Абсолютная']).build()
    plt.show()
```