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 дифференциальных уравнений. Метод Адамса"

Дисциплина: "Численные методы"

Выполнил студент гр. 3630103/90001 Михеев Евгений Викторович

Преподаватель: Павлова Людмила Владимировна

Санкт-Петербург

2021

Лабораторная работа №6

„Численное решение задачи Коши"

Метод Адамса IV порядка

1) Постановка задачи: реализовать метод Адамса IV порядка в форме „предиктор-корректор" для решения задачи Коши ур-я II-го порядка. Решить ее с заданной точностью и исследовать абсолютную погр-ть решения

2) Алгоритм метода

Дана задача Коши $\begin{cases} y' = f(x,y) \\ y(a) = y_0 \end{cases}$

Будем искать y_k как $y_k := y_{k-1} + \int_{x_{k-1}}^{x_k} f(x, y(x))\,dx$

Подинтегральная ф-ция аппроксимируется полиномом Лагранжа $L(x)$ степени m

Если $m = r$ (r - число шагов), то получим неявный метод Адамса - Мультона

$m = r-1$, — явный метод Адамса Башфорта

$$\int_{x_{k-1}}^{x_k} f(x, y(x))\,dx \approx \int_{x_{r-1}}^{x_r} L_m(x)\,dx = \left[\begin{matrix} x = x_0 + th \\ dx = h\,dt \end{matrix}\right] = h\int_{r-1}^{r} L_m(x_0 + ht)\,dt \ (=)$$

$$= h \int_{t-1}^{t} \sum_{j=0}^{m} f_j \frac{(-1)^{m-j}}{(m-j)!\,j!} \frac{\omega(t)}{t-j}\,dt$$

т.о $y_k \approx y_{k-1} + h \int_{t-1}^{t} \sum f_j \frac{(-1)^{m-j}}{(m-j)!\,j!} \frac{\omega(t)}{t-j}\,dt = h \sum_{j=0}^{m} \beta_j f_{k-t+j}$

$$y_k = y_{k-1} + h \sum_{j=0}^{m} \beta_j f_{k-t+j}$$ – общая ф-ла методов Адамса

Явный метод Адамса IV порядка:

$$y_i^{Б} = y_{i-1} + \frac{h}{24}\left[55 f(x_{i-1}, y_{i-1}) - 59 f(x_{i-2}, y_{i-2}) + 37 f(x_{i-3}, y_{i-3}) - 9 f(x_{i-4}, y_{i-4})\right]$$

Неявный (y_i выражается через себя)

$$y_i^{М} = y_{i-1} + \frac{h}{24}\left[9 f(x_i, y_i) + 19 f(x_{i-1}, y_{i-1}) - 5 f(x_{i-2}, y_{i-2}) + f(x_{i-3}, y_{i-3})\right]$$

Предиктор-корректорный метод состоит в том, чтобы находить y_i сначала через явную схему, а потом уточнять его неявной схемой:

$$y_i^{Б} = \Phi(y_{i-1}, y_{i-2}, y_{i-3}, y_{i-4}, x_{i-1}, x_{i-2}, x_{i-3}, x_{i-4})$$

$$y_i^{М} = \tilde{\Phi}(y_i^{Б}, y_{i-1}, y_{i-2}, y_{i-3}, x_i, x_{i-1}, x_{i-2}, x_{i-3})$$

ур-е $y'' = f(x, y, y')$ приводится к виду

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

а затем на каждом шаге метод Адамса применяется к каждому ур-ю системы

№3) ∃ и ! решения

Задача Коши имеет решение и при том единственное, если $f(x, y) \in C([a, b])$ и уд. условию Липшица по y

№4) Тестовый пример:

$$y'' = 2xy' + 2y - 4x$$

$$\begin{matrix} y(0) = 1 \\ y'(0) = 1 \end{matrix} \quad [a, b] = [0, 1] \quad | \quad y(x) = x + e^{x^2}$$

$n = 4 \Rightarrow h = 0{,}25 \Rightarrow x_0 = 0; x_1 = 0{,}25; x_2 = 0{,}5; x_3 = 0{,}75; x_4 = 1$

∃ y_1, y_2, y_3 известны (уже мы найдены методом Эйлера): $y_1 = 1{,}31$ $y_0 = 1$

$y_2 = 1{,}71$

$y_3 = 2{,}5$

z_1, z_2, z_3 тоже известны: $z_1 = 1{,}53$ $z_0 = 1$

$z_2 = 2{,}28$

$z_3 = 3{,}36$

Тогда

$$y_4^5 = 2{,}5 + \frac{0{,}25}{24}[55 \cdot 3{,}36 - 59 \cdot 2{,}28 + 37 \cdot 1{,}53 - 9 \cdot 1] = 3{,}51$$

$$z_4^5 = 3{,}36 + \frac{0{,}25}{24}[55 \cdot 4{,}94 - 59 \cdot 3{,}84 + 37 \cdot 2{,}38 - 9 \cdot 2] = 5{,}76$$

$$y_4 = 2{,}5 + \frac{0{,}25}{24}\left[9 \cdot 5{,}76 + 19 \cdot 3{,}36 - 5 \cdot 2{,}28 + 1{,}53\right] \approx 3{,}6$$

$$y'_4 = 3{,}36 + \frac{0{,}25}{24}\left[9 \cdot 14{,}54 + 19 \cdot 7{,}04 - 5 \cdot 3{,}17 + 2{,}31\right] \approx 5{,}94$$

Найдена следующая точка $\begin{pmatrix} x_4 \\ y_4 \\ y'_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 3{,}6 \\ 5{,}94 \end{pmatrix}$

№5) Контрольные тесты

Дано д/у: $\begin{cases} y'' = 2xy' + 2y - 4x \\ y(0) = y'(0) = 1 \end{cases}$

Ход работы:

1) Написать алгоритм, реализующий метод
2) Получить зависимость кол-ва ступеней по правилу Рунге от задания точности
3) Найти абс. погрешность решения для пред разных ε:
4) Занести результаты в таблицу

№6) Структура программы

DE adams.py script $\longleftrightarrow$ class DE (def Adams IV)

1) class DE (def Adams IV)

Input: $f(x, y, y')$, $[a, b]$, y_0, y'_0, n

Output: массив $\begin{bmatrix} x_0 & \dots & x_n \\ y_0 & \dots & y_n \end{bmatrix}$ - численное решение

2) def rungeEps

Input: epsMax ь

Output: ф-ция возвращает массив зависимости количества ит. от абс. погрешности от заданных точностей в диапазоне от 0 до epsMax

3) скрипт "DEadams.py

Input: $f(x, y, y')$, y_0, y_0', $y = f(x)$ – точное решение

Output: графики:

1) $K(\varepsilon)$ – кол-во итераций по правилу Рунге от заданной точности ε

2) $\varepsilon_2(\varepsilon)$ – абс. погрешность от заданного ε

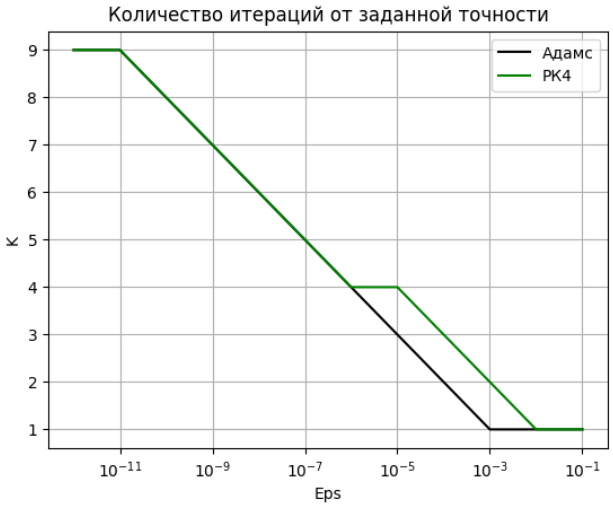
N7) Численный анализ: $n_0 = 4$

ε	10^{-4}	10^{-8}	10^{-12}
абс. погрешность приближенного решения	$\sim 10^{-7}$	10^{-11}	$\to 0$
Кол-во итераций, при кот. получен результат зад. точности	2 (1 разб)	6 (2^4 разб)	9 (2^{10} разб)

Метод показывает сходимость схожую с методом Рунге-Кутты IV. В частности для достижения одинаковой точности обоим методам требуется одинаковое кол-во итераций по правилу Рунге. При этом реальная абс погрешность для метода Адамса в среднем на 10^{-1} меньше, чем у метода Р-К IV.

Вывод: Метод Адамса IV порядка обладает хорошей сходимостью и позволяет достигать крайне высокой точности, однако метод не показывает какого-то значимого преимущества перед методом Р-К IV (во всяком случае в данной (рассматриваемой) задаче). Также метод потребует расчет 4 разгонных точек, для чего все равно придется применять методы Р-К, а также схема предиктор-корректор так же является вычислительно затратной, т.к требует многократного вычисления значений ф-ции. Таким образом можно сделать вывод, что при равных показателях сходимости метод Адамса уступает методу Р-К в универсальности

Графики



Код

DEadams.py

```
import math
import numpy as np
from NMclasses import DE
from NMclasses import Plots
import matplotlib.pyplot as plt
import DErunge

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 RungeEps(y, epsMax):
    eps = 0.1
    Eps = list()
    EpsReal = list()
    N = list()

    for i in range(1, epsMax):
        eps = math.pow(10, -i)
        n, d, k = 4, 1, 0
        DSp = DE(d2y, a, b, n, y0, dy0).RungeIV()
        while d > eps:
            D = []
            DSn = DE(d2y, a, b, 2*n, y0, dy0).AdamsIV()
            for j in range(1, DSp.shape[1]):
                d = math.fabs(DSn[1][2*j] - DSp[1][j])/15
                D.append(d)
            DSp = DSn
            n *= 2
            d = max(D)
            k+=1
        EpsReal.append(realDifference(y, DSn))
        N.append(k)
        Eps.append(eps)
    return Eps, EpsReal, N

def realDifference(yReal, numericSolution):
    EpsTemp = list()
    for index in range(len(numericSolution)):
        EpsTemp.append(math.fabs(yReal(numericSolution[0][index]) - numericSolution[1][index])
)
    EpsReal = max(EpsTemp)
    return EpsReal

if __name__ == '__main__':

    EpsA, EpsRealA, NA = RungeEps(yR, 13)
    EpsR, EpsRealR, NR = DErunge.RungeEps(yR, 13)
    Plots(1, [[EpsA, NA], [EpsA, NR]], 'Количество итераций от заданной точности', 'Eps', 'K',
['Адамс', 'РК4']).build('logX')
    Plots(2, [[NA, EpsRealA], [NA, EpsA], [NA, EpsRealR], [NA, EpsR]], 'Погрешность от итераци
и', 'K', 'Eps', ['Реальная погрешность Адамс', 'По правилу Рунге Адамс', 'Реальная погрешность
 РК4' ,' По правилу Рунге РК4']).build('logY')
    Plots(3, [[EpsA, EpsRealA], [EpsA, EpsRealR]], 'Абсолютная погршность от заданной точности
', 'Eps', 'EpsReal', ['Адамс', 'РК4']).build('loglog')
    plt.show()
```

DE.AdamsIV

```
def AdamsIV(self):
        h = (self.b - self.a)/self.n
        x = [self.a + i*h for i in range(self.n+1)]

        startSol = self.__RungeIVShort()
        y = list(startSol[0])
        z = list(startSol[1])
        for i in range(3, self.n):
            yB =  y[i] + h/24*(55*z[i] - 59*z[i-1] + 37*z[i-2] - 9*z[i-3])
            zB =  z[i] + h/24*(55*self.func(x[i], y[i], z[i]) - 59*self.func(x[i-1], y[i-
1], z[i-1]) + 37*self.func(x[i-2], y[i-2], z[i-2]) - 9*self.func(x[i-3], y[i-3], z[i-3]))
            y.append( y[i] + h/24 * (9*zB + 19*z[i] - 5*z[i-1] + z[i-2]) )
            z.append( z[i] + h/24 * (9*self.func(x[i+1], yB, zB) + 19*self.func(x[i], y[i], z[
i]) - 5*self.func(x[i-1], y[i-1], z[i-1]) + self.func(x[i-2], y[i-2], z[i-2])) )
        result = np.array([x, y, z])
        return result
```