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 №5
Тема работы: " Численное решение дифференциальных уравнений"
Дисциплина: "Численные методы"

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

Санкт-Петербург
2021

Отчет по лаб. работе №5
Одношаговые методы решения
д/у: метод Рунге-Кутты IV-порядка

1) Постановка задачи: решить заданное д/у численным методом Р-К. IV-порядка с заданной точностью ε. Исследовать зависимость точности от количества итераций по правилу Рунге

№2) Алгоритм метода:
$\exists$ дана задача Коши: $\begin{cases} y' = f(x, y) \\ y(a) = y_0 \end{cases}$

Составим ряд Тейлора:

$$y(x+h) = y(x) + y'(x)h + y''(x)\frac{h^2}{2} + \ldots + y^{(s)}(x)\frac{h^s}{s!} + O(h^{s+1})$$

$$\exists\ \Delta_s y(x) = y(x+h) - y(h) = h f(x, y) + \frac{h^2}{2}\frac{d}{dx} f(x, y) + \ldots +$$
$$+ \frac{h^{s}}{s!}\frac{d^{s-1}}{dx^{s-1}} f(x, y) = \delta_s(x, h)$$

$\delta_s(x, h)$ ищут в виде лин комбинации значений f:

$$\delta_s^{l}(x, h) = h\sum_{i=1}^{l} \rho_i \underbrace{f(x+\delta x_i, y+\delta y_i)}_{K_i}$$

$\Downarrow$

$y_{i+1} = y_i + \delta_s(x, h)$ — общая ф-ла методов Рунге-Кутты

где: $\begin{cases} K_1 = f(x, y) \\ K_2 = f(x+\alpha_2 h, y + h\beta_{21} K_1) \\ \text{—} \quad \text{—} \quad \text{—} \\ K_l = f\left(x + \alpha_l h, y + h\sum_{j=1}^{l-1} \beta_{lj} K_j\right) \end{cases}$

Для метода Р-К $\underline{IV}$ порядка:

$$\begin{cases} k_1 = f(x_k, y_k) \\ k_2 = f(x_k + \frac{h}{2}, y_k + \frac{h}{2} k_1) \\ k_3 = f(x_k + \frac{h}{2}; y_k + \frac{h}{2} k_2) \\ k_4 = f(x_k + \frac{h}{2}; y_k + k_3) \end{cases} \quad y_{k+1} = y_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Для д/у 2-ого порядка: $y'' = f(x, y, y')$; $y(a) = y_0$; $y'(a) = y_0'$

$\exists\ y' = z \Rightarrow z' = f(x, y, z)$

$$\begin{cases} y' = z \\ z' = f(x, y, z) \end{cases} \Rightarrow \begin{cases} y_{k+1} = y_k + \frac{h}{6}(q_1 + 2q_2 + 2q_3 + q_4) \\ z_{k+1} = z_k + \frac{h}{6}(m_1 + 2m_2 + 2m_3 + m_4) \end{cases}$$

$$\begin{cases} q_1 = z_i \\ q_2 = z_i + \frac{h}{2} m_1 \\ q_3 = z_i + \frac{h}{2} m_2 \\ q_4 = z_i + h m_3 \end{cases} \qquad \begin{cases} m_1 = f(x_i, y_i, z_i) \\ m_2 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2} q_1, z_i + \frac{h}{2} m_1) \\ m_3 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2} q_2; z_i + \frac{h}{2} m_2) \\ m_4 = f(x_i + h, y_i + h q_3, z_i + h m_3) \end{cases}$$

Правило Рунге: $\varepsilon_i = \dfrac{y_i|_{h/2} - y_i|_h}{2^r - 1}$ (Р-К $\underline{IV} \Rightarrow r = 4$)

№3) $\exists$ и ! решения

Задача Коши $\begin{cases} y' = f(x, y) \\ y(a) = 0 \end{cases}$ имеет решение и при том единственное, если $f(x, y) \in C([a, b])$ и уд. условию Липшица: $|f(x, \tilde{y}) - f(x, \tilde{\tilde{y}})| \le L|\tilde{y} - \tilde{\tilde{y}}|$, где

L - конст. Липшица, $\tilde{y}$ и $\tilde{\tilde{y}}$ - $\forall$ из $\mathbb{R}^2$

№4) –

№5) Тестовый пример:

$$y'' = 2xy' + 2y - 4x$$
$$y(0) = 1, \quad y'(0) = 1, \quad [a, b] = [0, 1]$$

$$y(x) = x + e^{x^2}$$

$$\begin{cases} y' = z \\ z' = 2xz + 2y - 4x \end{cases} \qquad \begin{cases} y(0) = 1 \\ z(0) = 1 \end{cases}$$

$n = \cancel{1} \Rightarrow \cancel{h = 0{,}5}\ h = 1$

$q_1 = 1 \qquad m_1 = 2$

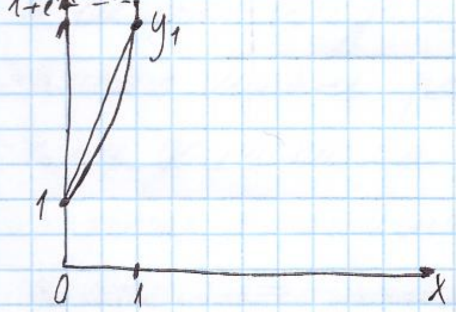
$q_2 = 2 \qquad m_2 = 2\left(1 + \frac{1}{2}\right) = 3$

$q_3 = \frac{5}{2} \qquad m_3 = 2 \cdot \left(1 + \frac{5}{4}\right) = \frac{9}{2}$

$q_4 = 1 + \frac{9}{2} = \frac{11}{2} \qquad m_4 = 2 \cdot \left(1 + \frac{11}{2}\right) = 13$

$$y_1 = 1 + \frac{1}{6}\left(1 + 4 + \frac{5}{\cancel{2}} + \frac{11}{2}\right) = $$
$$\approx 3{,}58$$
$$y(1) = 1 + e \approx 3{,}73$$

График:



6) 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 итераций по правилу Рунге от заданной точности
3) Найти абсолютную погрешность решения для трех разных ε:
4) Заполнить следующую таблицу

Заданная ε	10^{-4}	10^{-8}	10^{-12}
Абс. погрешность приближенного решения			
Кол-во итераций для достижения ε			

7). Структура программы

DE runge.py ⟺ class DE
1) class DE (def RungeIV)
Input: $f(x,y,y')$, $[a,b]$, n, y_0, y_0' Output: массив x_i — y_i — численное решение $(i=0\ldots n)$

2) DE runge.py скрипт
Input: $f(x,y,y')$, $[a,b]$, y_0, y_0', $y=f(x)$ - точное решение
скрипт вызывает def RungeIV, используя вложен-

ную ф-цию def Runge Eps получает решения для ε от 10^{-1} до 10^{-12}

Output: графики: 1) $K(\varepsilon)$ - кол-во итераций по правилу Рунге от точности ε

2) $\varepsilon(k)$ — график показывает достигнутую ~~ю~~ точность на каждой итерации по правилу Рунге и реальную погрешность (Абсолютн.)

3) $\varepsilon_{real}(\varepsilon)$ - абс. погрешность от заданного ε

№8) Численный анализ

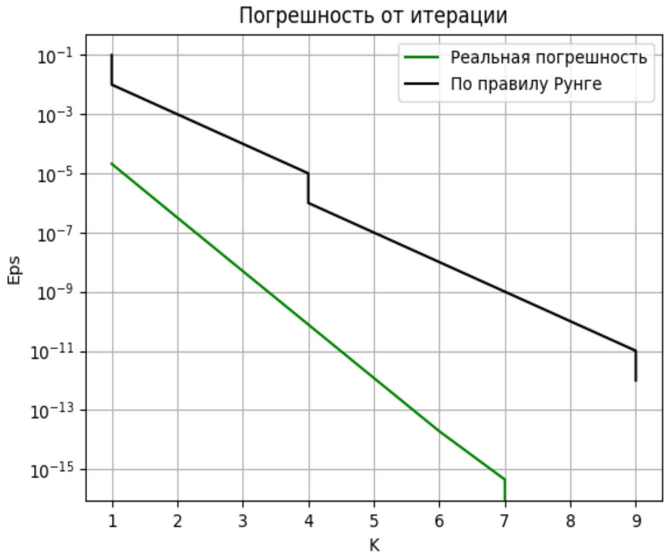
ε	10^{-4}	10^{-8}	10^{-12}
Абсолютная погрешность приближенного решения	$\approx 10^{-9}$	$\approx 10^{-13}$	$\longrightarrow 0$
Значение K, при кот. получ. результат заданной точности ε	3 ($\approx$16 разбиений)	6 (128 разбиений)	9 (1024 разб.)

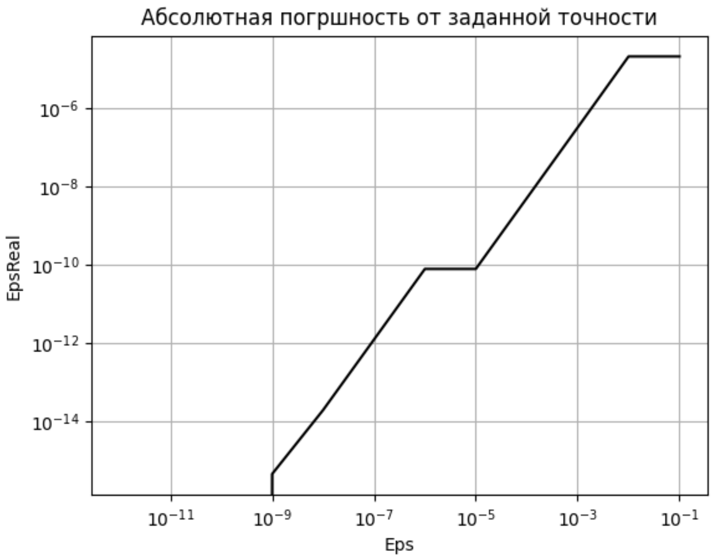
Из полученных результатов можно сделать вывод о высокой сходимости метода: реальная погрешность намного меньше расчетной и с помощью метода вполне достижима точность 10^{-16} и меньше, т.е уходящая в машинный 0.

~~Впрочем из недостатков метода можно выде~~

№9) Вывод: метод Рунге-Кутты $\underline{IV}$ порядка является весьма точным и быстросходящимся методом решения ОДУ и систем ОДУ, главным недостатком которого является необходимость многократного вычисления значений ф-ции: 4 вычисления на один шаг h, т.е 4n вычислений на весь ~~пр~~ промежуток.

Графики





Код

DErunge.py

```
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 = 2, 1, 0
        DSp = DE(d2y, a, b, n, y0, dy0).RungeIV()
        while d > eps:
            D = []
            DSn = DE(d2y, a, b, 2*n, y0, dy0).RungeIV()
            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
        # K[f'10E-{i}'] = n
        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__':

    Eps, EpsReal, N = RungeEps(yR, 13)
    print(EpsReal)
    Plots(1, [[Eps, N]], 'Количество итераций от заданной точности', 'Eps', 'K').build('logX')
    Plots(2, [[N, EpsReal], [N, Eps]], 'Погрешность от итерации', 'K', 'Eps', ['Реальная погре
шность', 'По правилу Рунге']).build('logY')
    Plots(3, [[Eps, EpsReal]], 'Абсолютная погршность от заданной точности', 'Eps', 'EpsReal')
.build('loglog')
    plt.show()
```

class DE (def RungeIV)

```
def RungeIV(self):
        h = (self.b - self.a)/self.n
        x = [self.a + i*h for i in range(self.n+1)]
        y = [self.y0]
        z = [self.dy0]

        q1 = lambda x,y,z: self.func(x,y,z)
        q2 = lambda x,y,z: self.func(x + h/2, y + h*k1(z)/2, z + h*q1(x,y,z)/2)
        q3 = lambda x,y,z: self.func(x + h/2, y + h*k2(x,y,z)/2, z + h*q2(x,y,z)/2)
        q4 = lambda x,y,z: self.func(x + h, y + h*k3(x,y,z), z + h*q3(x,y,z))
        k1 = lambda z: z
        k2 = lambda x,y,z: z + h*q1(x,y,z)/2
        k3 = lambda x,y,z: z + h*q2(x,y,z)/2
        k4 = lambda x,y,z: z + h*q3(x,y,z)
```

```
        for i in range(self.n):
            y.append( y[i] + ( h/6 * ( k1(z[i]) + 2*k2(x[i], y[i], z[i]) + 2*k3(x[i], y[i], z[i]) + k4(x[i], y[i], z[i]) ) ) )
            z.append( z[i] + ( h/6 * ( q1(x[i], y[i], z[i]) + 2*q2(x[i], y[i], z[i]) + 2*q3(x[i], y[i], z[i]) + q4(x[i], y[i], z[i]) ) ) )

        result = np.array([x, y])
        return result
```