МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский Авиационный Институт» (Национальный Исследовательский Университет)

Институт: №8 «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №4
по курсу «Численные методы»

Группа: М8О-307Б-22

Студент(ка): П. В. Лебедько

Преподаватель: Д. Л. Ревизников

Оценка:

Дата: 02.05.2025

Москва, 2025

Оглавление

1 Задание 1

Задание

Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h. С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Вариант

Вариант 17

$xy''-(x+1)y'+y=0$, $y(1)=2+e,$ $y'(1)=1+e,$ $x\in[1,2], h=0.1$	$y=x+1+e^x$

Ход лабораторной работы

Необходимо преобразовать исходное уравнение второго порядка в систему ОДУ первого порядка:

$$y' = z$$
$$z' = (x+1)/x * z - 1/x * y$$

Хранить переменные системы y и z будем в одной массиве, чтобы с помощью программы можно было решать ОДУ любого порядка. В соответствие с этим введем начальные данные и реализуем вектор-функцию f(x):

```
# Функции
def f(x: float, y: np.ndarray):
    return np.array([
        y[1],
        ((x + 1) * y[1] - y[0]) / x
    ])

def getTrueY(x):
    return x + 1 + np.exp(x)

# Интервал
a = 1
```

```
b = 2
# Начальные условия
y0 = np.array([2 + np.e, 1 + np.e])
```

Реализуем методы Рунге-Кутты, Эйлера и Адамса:

```
# Рунге-Кутт 4го порядка
p = 4
as_ = [0, 0.5, 0.5, 1]
bs = [[0.5], [0, 0.5], [0, 0, 1]]
cs = [1/6, 1/3, 1/3, 1/6]

def getKs(x: float, y: np.ndarray, h):
    dim = y.shape[0]
    Ks = np.empty((p, dim))

    for i in range(p):
        newX = x + as_[i] * h
        newY = np.copy(y)
        for j in range(i):
            newY += bs[i - 1][j] * Ks[j]

        K = h * f(newX, newY)
        if debug: print(f"\tK{i + 1} = {K}")
        Ks[i] = K

    return Ks

def getDeltaY(x: float, y: np.ndarray, h):
    Ks = getKs(x, y, h)
    dim = Ks.shape[1]
    sum_ = np.zeros(dim)
    for i in range(p):
        sum_ += cs[i] * Ks[i]
    if debug: print(f"\tdeltaY = {sum_}")
    return sum_

def RungeKutta(xs: list, y0: np.ndarray, h):
    N = len(xs) - 1
    dim = y0.shape[0]
    ys = np.empty((N + 1, dim))
    ys[0] = y0

    if debug: print(f"N = {N}, dim = {dim}")

    for k in range(1, N + 1):
```

```
        if debug: print(f"Шаг {k}")
        ys[k] = ys[k - 1] + getDeltaY(xs[k - 1], ys[k - 1], h)
        if debug: print(f"\ty = {ys[k]}")

    return ys


def Euler(xs: list, y0: np.ndarray, h):
    N = len(xs) - 1
    dim = y0.shape[0]
    ys = np.empty((N + 1, dim))
    ys[0] = y0

    for k in range(N):
        ys[k + 1] = ys[k] + h * f(xs[k], ys[k])

    return ys


def Adams(xs: list, y0s: np.ndarray, h):
    N = len(xs) - 1
    dim = y0s.shape[1]
    ys = np.empty((N + 1, dim))

    fs = np.empty((N + 1, dim))
    for i in range(4):
        ys[i] = np.copy(y0s[i])
        fs[i] = f(xs[i], ys[i])

    for k in range(4, N + 1):
        ys[k] = ys[k - 1] + h/24 * (55 * fs[k - 1] - 59 * fs[k - 2] + 37 * fs[k - 
3] - 9 * fs[k - 4])
        fs[k] = f(xs[k], ys[k])

    return ys
```

А также метод Рунге для оценки погрешности. Он будет принимать массивы рассчитанных значений ys для шага h и ys2 для шага h/2:

```
def RungeError(ys: np.ndarray, ys2: np.ndarray, p):
    k = 2
    error = 0
    for i in range(ys.shape[0]):
        error = max(error, abs(ys2[i * 2][0] - ys[i][0]) / (k ** p - 1))
    return error
```

Остается только провести вычисления.

Результаты

```
PS C:\Users\plato\Documents\Prog\numeric-methods\4> python3 4.1.py
Шаг: 0.1
xk = 1, y(xk) = 4.71828
        Эйлер:      yk = 4.71828, e = 0.0
        Рунге-Кутт: yk = 4.71828, e = 0.0
        Адамс:      yk = 4.71828, e = 0.0
xk = 1.1, y(xk) = 5.10417
        Эйлер:      yk = 5.09011, e = 0.01405601
        Рунге-Кутт: yk = 5.10417, e = 2.3e-07
        Адамс:      yk = 5.10417, e = 2.3e-07
xk = 1.2, y(xk) = 5.52012
        Эйлер:      yk = 5.48912, e = 0.03099591
        Рунге-Кутт: yk = 5.52012, e = 5.1e-07
        Адамс:      yk = 5.52012, e = 5.1e-07
xk = 1.3, y(xk) = 5.9693
        Эйлер:      yk = 5.91803, e = 0.05126355
        Рунге-Кутт: yk = 5.9693, e = 8.4e-07
        Адамс:      yk = 5.9693, e = 8.4e-07
xk = 1.4, y(xk) = 6.4552
        Эйлер:      yk = 6.37984, e = 0.07536354
        Рунге-Кутт: yk = 6.4552, e = 1.24e-06
        Адамс:      yk = 6.45519, e = 1.248e-05
xk = 1.5, y(xk) = 6.98169
        Эйлер:      yk = 6.87782, e = 0.103869
        Рунге-Кутт: yk = 6.98169, e = 1.72e-06
        Адамс:      yk = 6.98166, e = 2.794e-05
xk = 1.6, y(xk) = 7.55303
        Эйлер:      yk = 7.4156, e = 0.13743035
        Рунге-Кутт: yk = 7.55303, e = 2.28e-06
        Адамс:      yk = 7.55299, e = 4.547e-05
xk = 1.7, y(xk) = 8.17395
        Эйлер:      yk = 7.99716, e = 0.17678511
        Рунге-Кутт: yk = 8.17394, e = 2.94e-06
        Адамс:      yk = 8.17388, e = 6.648e-05
xk = 1.8, y(xk) = 8.84965
        Эйлер:      yk = 8.62688, e = 0.22276895
        Рунге-Кутт: yk = 8.84964, e = 3.71e-06
        Адамс:      yk = 8.84956, e = 9.157e-05
xk = 1.9, y(xk) = 9.58589
        Эйлер:      yk = 9.30957, e = 0.27632808
        Рунге-Кутт: yk = 9.58589, e = 4.61e-06
        Адамс:      yk = 9.58577, e = 0.00012119
xk = 2, y(xk) = 10.38906
        Эйлер:      yk = 10.05052, e = 0.3385331
        Рунге-Кутт: yk = 10.38905, e = 5.67e-06
        Адамс:      yk = 10.3889, e = 0.000156
===================================================================
Апостериорные оценки погрешности по Рунге:
        Эйлер:      0.16188794039367416
        Рунге-Кутт: 3.531086456121102e-07
        Адамс:      2.0469011518896097e-05
PS C:\Users\plato\Documents\Prog\numeric-methods\4>
```

Заметим, что Рунге-Кутт дает самые точные результаты. Это ожидаемо, поскольку этот метод имеет четвертый порядок точности, что выше методов Эйлера и Адамса.

2 Задание 2

Задание

Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Вариант

Вариант 17

$(x^2-1)\,y''+(x-3)y'-y=0$, $y'(0)=0$, $y'(1)+y(1)=-0.75$	$y(x) = x-3+\dfrac{1}{x+1}$

Ход лабораторной работы

Метод стрельбы

Необходимо преобразовать исходную краевую задачу второго порядка в задачу Коши:

$$y' = z$$
$$z' = (y - (x-3) * z) \;/\; (x \;**\; 2 \;-\; 1)$$

Так же определим вектор-функцию f:

```
    return np.array([
        y[1],
        y[1] / 2
        if x == 1 else
        (y[0] - (x - 3) * y[1]) / (x ** 2 - 1)
    ])
```

Причем начальные условия заданы только для z(0) = 0. Начальное условие y(0) будем подбирать с помощью метода секущих из условия на значение в точке 1:

```
def getTrueY(x):
    return x - 3 + 1 / (x + 1)

def y1(ys: np.ndarray):
```

```
    return -0.75 - ys[-1][1]
```
Так как мы знаем истинное решение ОДУ, можно посчитать, что на самом деле y(0) = -2. Введем данное условие в алгоритм Рунге-Кутта и сравним его решение с решением, полученным методом стрельбы также через алгоритм Рунге-Кутта.

Метод Рунге-Кутта берем из предыдущего задания, реализуем метод стрельбы:

```
def Shooting(xs, y0, h, eps):
    eta0 = randint(-2166, 2166)
    eta1 = randint(-2166, 2166)

    ys0 = RungeKutta(xs, np.array([eta0, y0]), h)
    ys1 = RungeKutta(xs, np.array([eta1, y0]), h)
    F0 = ys0[-1][0] - y1(ys0)
    F1 = ys1[-1][0] - y1(ys1)

    iter = 1
    while True:
        eta = eta1 - (eta1 - eta0) / (F1 - F0) * F1
        ys = RungeKutta(xs, np.array([eta, y0]), h)
        F0 = F1
        F1 = ys[-1][0] - y1(ys)

        if abs(F1) < eps:
            return ys, iter, eta

        iter += 1
        eta0 = eta1
        eta1 = eta
```

Конечно-разностный метод

Так как граничные условия в задаче не первого рода, будем аппроксимировать производные с помощью односторонних разностей первого порядка (для сохранения трех диагональной структуры). Определим реализацию конечно-разностного метода, вручную задавая коэффициенты для первой и последней строки системы:

```
def FiniteDifference(n, xs, h, A_b1, A_c1, A_an, A_bn, b1, bn):
    A = np.zeros((n, 3))
    b = np.empty(n)
    A[0][1] = A_b1
    A[0][2] = A_c1
```

```
    A[n - 1][0] = A_an
    A[n - 1][1] = A_bn
    b[0] = b1
    b[n - 1] = bn
    for k in range(1, n - 1):
        A[k][0] = 1 - p(xs[k]) * h / 2
        A[k][1] = -2 + h ** 2 * q(xs[k])
        A[k][2] = 1 + p(xs[k]) * h / 2
        b[k] = h ** 2 * f(xs[k])

    ys = tridiagonalMatrixAlgorithm(A, b)
    return ys
```

Определим функции p(x), q(x), f(x) по исходной задаче:

```
def p(x):
    return (x - 3) / (x ** 2 - 1)
def q(x):
    return -1 / (x ** 2 - 1)
def f(x):
    return 0
```

Результаты

```
PS C:\Users\plato\Documents\Prog\numeric-methods\4> python3 4.2.1.py
Шаг: 0.125
Точность: 1e-09
Итераций в стрельбе: 1, Вычисленная y(0) = -2.0082606276009756
xk = 0, y(xk) = -2.0
        Рунге-Кутт: yk = -2.0, e = 0.0
        Стрельба:   yk = -2.00826, e = 0.0082606276009756
xk = 0.125, y(xk) = -1.98611
        Рунге-Кутт: yk = -1.9861, e = 9.7718826093e-06
        Стрельба:   yk = -1.9943, e = 0.0081934498879734
xk = 0.25, y(xk) = -1.95
        Рунге-Кутт: yk = -1.94999, e = 1.32127141483e-05
        Стрельба:   yk = -1.95804, e = 0.0080408446241471
xk = 0.375, y(xk) = -1.89773
        Рунге-Кутт: yk = -1.89771, e = 1.38470464592e-05
        Стрельба:   yk = -1.90555, e = 0.0078243049050011
xk = 0.5, y(xk) = -1.83333
        Рунге-Кутт: yk = -1.83332, e = 1.3293191395e-05
        Стрельба:   yk = -1.84089, e = 0.007558893871114
xk = 0.625, y(xk) = -1.75962
        Рунге-Кутт: yk = -1.7596, e = 1.23940608017e-05
        Стрельба:   yk = -1.76687, e = 0.0072553184544655
xk = 0.75, y(xk) = -1.67857
        Рунге-Кутт: yk = -1.67856, e = 1.17641558435e-05
        Стрельба:   yk = -1.68549, e = 0.0069212139910342
xk = 0.875, y(xk) = -1.59167
        Рунге-Кутт: yk = -1.59165, e = 1.2675427183e-05
        Стрельба:   yk = -1.59823, e = 0.0065613550184347
xk = 1, y(xk) = -1.5
        Рунге-Кутт: yk = -1.49997, e = 3.43864223331e-05
        Стрельба:   yk = -1.50616, e = 0.0061609422516835
===================================================================
Апостериорная оценка погрешности по Рунге: 0.0003013855324752512
PS C:\Users\plato\Documents\Prog\numeric-methods\4>
```

Метод стрельбы за одну итерацию смог вычислить y(0) = -2.008, то есть решение с точностью до двух порядков. И приблизительно на два порядка отличается полученное от решение от решения Рунге-Кутты, которое знало y(0) = -2.

```
PS C:\Users\plato\Documents\Prog\numeric-methods\4> python3 4.2.2.py
Шаг: 0.125

xk = 0, y(xk) = -2.0
yk = -1.91264, e = 0.0873631915756687

xk = 0.125, y(xk) = -1.98611
yk = -1.91264, e = 0.0734743026867799

xk = 0.25, y(xk) = -1.95
yk = -1.88696, e = 0.063036168869955

xk = 0.375, y(xk) = -1.89773
yk = -1.84267, e = 0.0550583672106089

xk = 0.5, y(xk) = -1.83333
yk = -1.78444, e = 0.0488902766000605

xk = 0.625, y(xk) = -1.75962
yk = -1.71553, e = 0.0440865573735272

xk = 0.75, y(xk) = -1.67857
yk = -1.63824, e = 0.0403312322004705

xk = 0.875, y(xk) = -1.59167
yk = -1.55427, e = 0.0373919887529532

xk = 1, y(xk) = -1.5
yk = -1.46491, e = 0.0350891751878102

===================================================================
Апостериорная оценка погрешности по Рунге: 0.044584925606153236
PS C:\Users\plato\Documents\Prog\numeric-methods\4>
```

Заметно, что конечно-разностный метод дает менее точные результаты, поскольку, из-за приближения производных разностями первого порядка, имеет точность первого порядка.

3 Выводы

В ходе лабораторной работы я реализовал численные методы для решения задачи Коши для ОДУ второго порядка, а также для решения краевой задачи.