БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №3

Численные методы решения задачи Коши

Вариант 3

Выполнил:
Черепенников Р.М.
3 курс 8 группа

Преподаватель:
Будник А.М.

2021

Найти приближенное решение задачи Коши

$$u'(x) = \frac{3u}{2x} + \frac{3}{2}xu^{\frac{1}{3}}, \qquad x \in [1,2]$$

$$u(1) = \frac{1}{10}$$

на сетке узлов при 10-ти разбиениях отрезка интегрирования.

1. Явный метод Эйлера

1.1. Теория

Метод Эйлера выглядит следующим образом:

$$y_{j+1} = y_j + hf(x_j, y_j)$$

Рассмотрим главный член локальной погрешности:

$$r = u_{j+1} - u_j - hf(x_j, u_j)$$

$$r = u_j + u_j'h + \frac{u_j''h^2}{2} + \cdots - u_j - hu_j'$$

$$r = \frac{u_j''h^2}{2} + \cdots$$

Таким образом порядок локальной погрешности – 2.

А погрешность метода $O(h)$.

1.2. Листинг программы

```
def f(x, u):
    return 3*u/(2*x) + 3/2 * (x*u**(1/3))
u0 = 1/10
n=10
a=1
b=2
def euler(a, b, n, f, y0):
    h = (b - a) / n
    x = [a + h*i for i in range(n+1)]
    y = [y0]
    for i in range(n):
        y.append(y[i] + h*f(x[i], y[i]))
    return x, y
sol = euler(a, b, n, f, u0)
print('Метод Эйлера')
for x, y in zip(sol[0], sol[1]):
```

```
    print(f'u({x:.2f}) = {y:.8f}')
```

1.3. Вывод программы

```
Метод Эйлера
u(1.00) = 0.10000000
u(1.10) = 0.18462383
u(1.20) = 0.30375336
u(1.30) = 0.46272090
u(1.40) = 0.66693712
u(1.50) = 0.92187137
u(1.60) = 1.23303927
u(1.70) = 1.60599423
u(1.80) = 2.04632140
u(1.90) = 2.55963311
u(2.00) = 3.15156530
```

2. Метод последовательного повышения порядка точности

2.1. Теория

По условию необходимо построить метод 2-го порядка при $q = 1, \alpha_0 = 0, \alpha_1 = \frac{1}{2}$.

Для реализации метода будут использоваться формулы:

$$y_{j+\frac{1}{2}} = y_j + \frac{h}{2} f_j$$

$$y_{j+1} = y_j + hf(x_{j+\frac{1}{2}}, y_{j+\frac{1}{2}})$$

Так как метод второго порядка, то локальная погрешность – $O(h^3)$, а погрешность метода $O(h^2)$.

2.2. Листинг программы

```
def f(x, u):
    return 3*u/(2*x) + 3/2 * (x*u**(1/3))
u0 = 1/10
n=10
a=1
b=2
def increase_accuracy(a, b, n, f, y0):
    h = (b - a) / n
    x = [a + h*i for i in range(n+1)]
    y = [y0]
    for j in range(n):
        y_half = y[j] + h/2 * f(x[j], y[j])
        y.append(y[j] + h * f(x[j] + 1/2*h, y_half))
```

```
    return x, y


sol = increase_accuracy(a, b, n, f, u0)

print('Метод последовательного повышения порядка точности')

for x, y in zip(sol[0], sol[1]):

    print(f'u({x:.2f}) = {y:.8f}')
```

2.3. Вывод программы

```
Метод последовательного повышения порядка точности
u(1.00) = 0.10000000
u(1.10) = 0.20255977
u(1.20) = 0.34787642
u(1.30) = 0.54166753
u(1.40) = 0.78978465
u(1.50) = 1.09814089
u(1.60) = 1.47268153
u(1.70) = 1.91937019
u(1.80) = 2.44418163
u(1.90) = 3.05309778
u(2.00) = 3.75210530
```

3. Метод Рунге-Кутта

3.1. Теория

По условию необходимо построить метод Рунге-Кутта при $A_0 = \frac{1}{3},\ A_1 = \frac{2}{3},\ \alpha_1 = \frac{3}{4}, \beta_{10} = = \frac{1}{2}$.

Для реализации метода будут использоваться формула:

$$y_{j+1} = y_j + h[\frac{1}{3}f_j + \frac{2}{3}f\left(x_j + \frac{3}{4}h, y_j + \frac{h}{2}f(x_j, y_j)\right)]$$

Рассмотрим порядок точности данного метода. Так как коэффициенты не удовлетворяют системе $A_0 + A_1 = 1, A_1\alpha_1 = \frac{1}{2}, A_1\beta_{10} = \frac{1}{2}$. То данный метод не может быть методом второго порядка. Но соотношение $A_0 + A_1 = 1$ выполняется, значит рассматриваемый метод имеет порядок $O(h)$.

3.2. Листинг программы

```
def f(x, u):

    return 3*u/(2*x) + 3/2 * (x*u**(1/3))

u0 = 1/10

n=10

a=1

b=2

def runge_cutta(a, b, n, f, y0):
```

```
    h = (b - a) / n

    x = [a + h*i for i in range(n+1)]

    y = [y0]

    for j in range(n):

        y.append(y[j] + h*( 1/3 * f(x[j], y[j]) + 2/3 * f(x[j] + 3/4*h, y[j]
+ h/2  * f(x[j], y[j]))))


    return x, y


sol = runge_cutta(a, b, n, f, u0)

print('Метод Рунге-Кутта')

for x, y in zip(sol[0], sol[1]):

    print(f'u({x:.2f}) = {y:.8f}')
```

3.3. Вывод программы

```
Метод Рунге-Кутта
u(1.00) = 0.10000000
u(1.10) = 0.19757116
u(1.20) = 0.33554255
u(1.30) = 0.51949524
u(1.40) = 0.75512874
u(1.50) = 1.04820558
u(1.60) = 1.40452663
u(1.70) = 1.82991849
u(1.80) = 2.33022633
u(1.90) = 2.91130945
u(2.00) = 3.57903843
```

4. Интерполяционный метод Адамса

4.1. Теория

По условию необходимо построить интерполяционный метод Адамса 3-го порядка.

Для реализации метода будут использоваться формулы:

$$y_{j+1} = y_j + \frac{h}{12}(5f_{j+1} + 8f_j - f_{j-1}), \qquad j = \overline{1,n}$$

Так как полученный метод неявный, то для нахождения y_{j+1} необходимо воспользоваться методом простой итерации:

$$y_{j+1}^{(l+1)} = y_j + \frac{h}{12}\left(5f_{j+1}^{(l)} + 8f_j - f_{j-1}\right)$$

Итерационный процесс будем продолжать пока

$$\left|y_{j+1}^{(l+1)} - y_{j+1}^{(l)}\right| > h^5$$

А в качестве начального приближения будем брать y_j

Начало таблицы – y_0, y_1. y_0 известно по условию, а y_1 найдем с помощью метода предиктор-корректор того же порядка точности:

$$y_{j+\frac{1}{3}} = y_j + \frac{h}{3} f_j$$

$$y_{j+\frac{2}{3}} = y_j + \frac{2h}{3} f_{j+\frac{1}{3}}$$

$$y_{j+1} = y_j + \frac{h}{4} f_j + \frac{3h}{4} f_{j+\frac{2}{3}}$$

Порядок локальной погрешности метода – $O(h^4)$

4.2. Листинг программы

```
def f(x, u):
    return 3*u/(2*x) + 3/2 * (x*u**(1/3))
u0 = 1/10
n=10
a=1
b=2


def simple_it(f, eps, y0):
    y1 = f(y0)
    while abs(y1-y0) > eps:
        y0 = y1
        y1 = f(y0)
    return y1



def adams(a, b, n, f, y0):
    h = (b-a)/n
    x = [a + h*i for i in range(n+1)]
    y = [y0]
    # Начало таблицы
    y_1_3 = y[0] + h/3 * f(x[0], y[0])
    y_2_3 = y[0] + 2*h/3 * f(x[0] + 1/3 *h, y_1_3)
    y1 = y[0] + h/4*f(x[0], y[0]) + 3*h/4*f(x[0] + 2/3 *h, y_2_3)
    y.append(y1)
    # Метод Адамса
```

```
    for j in range(1, n):
        # Реализация метода простой итерации
        y_new = simple_it(
            lambda y_j: y[j] + h/12 * (5 * f(x[j+1], y_j) + 8*f(x[j], y[j]) -
f(x[j-1], y[j-1]))
            , h**5, y[j]
        )
        #
        y.append(y_new)
    return x, y


sol = adams(a, b, n, f, u0)
print('Метод Адамса')
for x, y in zip(sol[0], sol[1]):
    print(f'u({x:.2f}) = {y:.8f}')
```

4.3. Вывод программы

```
Метод Адамса
u(1.00) = 0.10000000
u(1.10) = 0.20425959
u(1.20) = 0.33536652
u(1.30) = 0.50884034
u(1.40) = 0.73042440
u(1.50) = 1.00567795
u(1.60) = 1.34019782
u(1.70) = 1.73961417
u(1.80) = 2.20958463
u(1.90) = 2.78557573
u(2.00) = 3.44996714
```

5. Анализ полученных результатов

Таблица полученных результатов

	Метод Эйлера	Метод последовательного повышения порядка точности	Метод Рунге-Кутта	Интерполяционный метод Адамса
1.0	0.1	0.1	0.1	0.1
1.1	0.18462383	0.20255977	0.19757116	0.20425959
1.2	0.30375336	0.34787642	0.33554255	0.35182950
1.3	0.46272090	0.54166753	0.51949524	0.54830671
1.4	0.66693712	0.78978465	0.75512874	0.79953731
1.5	0.92187137	1.09814089	1.04820558	1.11143367
1.6	1.23303927	1.47268153	1.40452663	1.48994007
1.7	1.60599423	1.91937019	1.82991849	1.94101942
1.8	2.05632140	2.44418163	2.33022633	2.47064605

1.9	2.55963311	3.05309778	2.91130945	3.08480156
2.0	3.15156530	3.75210530	3.57903843	3.78947239

Точными значениями считаем значения, полученные с помощью интерполяционного метода Адамса.

Погрешности:

```
Адамс - Эйлер:
[0.         0.01963576 0.04807614 0.0855858  0.13260019 0.1895623
 0.2569008  0.33502519 0.42432465 0.52516845 0.6379071 ]
Адамс - Последовательное повышение точности:
[0.         0.00169982 0.00395308 0.00663917 0.00975265 0.01329278
 0.01725854 0.02164924 0.02646443 0.03170378 0.03736709]
Адамс - Рунге-Кутта:
[0.         0.00668844 0.01628694 0.02881147 0.04440857 0.0632281
 0.08541344 0.11110093 0.14041973 0.17349211 0.21043396]
```

Самым точным оказался метод последовательного повышения порядка точности, который из всех рассмотренных методов обладает наивысшим порядком точности. Методы Рунге-Кутта и Эйлера имеют одинаковый порядок точности, но при этом погрешность метода Рунге-Кутта меньше, это связано с коэффициентом при главном члене погрешности.