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

Факультет №8 «Информационные технологии и прикладная математика»

Курсовая работа
по курсу «Численные методы»
На тему: “Численное решение нелинейных интегральных уравнений”

Выполнил:
гр. М8О-402Б-20
Ткачёв Г.А.
Преподаватель:
Пивоваров Д.Е.
Дата сдачи:

Подпись:

_____

Москва, 2024

Необходимо рассмотреть численное решение нелинейных интегральных уравнений.

Метод решения

В данной курсовой работе я реализовал три метода: метод Рунге-Ромберга, метод квадратур и метод Ньютона-Канторовича, а также рассмотрел решение трех примеров уравнений.

а) Метод Рунге-Ромберга

Метод Рунге-Ромберга представляет собой технику улучшения точности численных методов, основанных на аппроксимации. Этот метод часто используется для оценки ошибок и улучшения результатов при численном решении дифференциальных уравнений, но он также может применяться к другим численным методам, включая методы интегрирования и методы решения нелинейных уравнений.

Идея метода Рунге-Ромберга основана на использовании двух или более приближенных значений решения с различными шагами интегрирования (или другими параметрами метода). Обозначим результаты таких приближенных вычислений как *h1* и *h2*, где *h1* – более крупный шаг, чем *h2*.

Рассмотрим формулу численного метода в общем виде:

$$y_1 = M(h_1), \quad y_2 = M(h_2)$$

где M - численный метод, h_1 и h_2 - различные шаги интегрирования, а y_1 и y_2 - соответствующие приближенные значения решения.

Метод Рунге-Ромберга предполагает использование разностей между результатами с различными шагами для уточнения оценки ошибки и улучшения приближенного результата. Для этого используется следующая формула:

$$\text{Improved result} = y_2 + \frac{y_2 - y_1}{k^p - 1}$$

где p - порядок метода (обычно известен для конкретного численного метода), а k - отношение шагов интегрирования:

$$k = \frac{h_1}{h_2}$$

Таким образом, метод Рунге-Ромберга позволяет уточнить результат численного метода, учитывая разницу между результатами при различных шагах интегрирования. Это особенно полезно при автоматическом выборе шага интегрирования, так как он позволяет адаптировать шаг так, чтобы достичь заданной точности решения.

Общий алгоритм применения метода Рунге-Ромберга:

1) Вычислить приближенные значения *y1* и *y2* с различными шагами интегрирования.
2) Рассчитать оценку ошибки и улучшить результат с использованием формулы Рунге-Ромберга.
3) Повторять процесс, уменьшая шаг интегрирования, пока не достигнута необходимая точность.

Этот метод помогает повысить точность численных результатов и является важным инструментом в численных вычислениях.

б**) Метод квадратур**

Метод квадратур - это численный метод решения интегральных уравнений, основанный на приближенном вычислении интегралов с использованием квадратурных формул. Этот метод часто применяется для решения нелинейных интегральных уравнений, где аналитическое решение может быть сложным или даже невозможным.

Рассмотрим интегральное уравнение в общем виде:

$$F(x) = \int_a^b K(x,t) \cdot \phi(t)\, dt + f(x)$$

где $F(x)$ - нелинейная функция, зависящая от неизвестной функции $\phi(t)$, $K(x,t)$ - ядро интегрального уравнения, $f(x)$ - заданная функция, а интервал интегрирования - от a до b.

Одним из подходов к решению таких уравнений численно является использование квадратурных формул для приближенного вычисления интегралов. Квадратурные формулы представляют собой аппроксимацию интегралов с использованием взвешенной суммы значений функции в узлах интегрирования.

Для решения нелинейных интегральных уравнений с использованием метода квадратур, обычно применяются итерационные процессы. Процедура выглядит следующим образом:

1) Задаются начальные значения для неизвестной функции.
2) Вычисляется интеграл в правой части уравнения, используя выбранную квадратурную формулу.
3) Подставляется значение интеграла в уравнение и решается полученное нелинейное уравнение относительно неизвестной функции
4) Шаги 2-3 повторяются до тех пор, пока не будет достигнута заданная точность или выполнено условие сходимости.

Этот итерационный процесс продолжается до достижения сходимости, когда изменения в значениях функции перестают значительно влиять на результат. Метод квадратур предоставляет эффективный численный подход к решению нелинейных интегральных уравнений, особенно в случаях, когда аналитическое решение недоступно.

в**) Метод Ньютона-Канторовича**

Метод Ньютона-Канторовича, также известный как метод Ньютона с оценкой Канторовича, представляет собой численный метод для нахождения корня нелинейного уравнения. Этот метод является итерационным и основан на использовании производных функции. Важной особенностью метода является возможность оценки области сходимости для достижения успешной сходимости приближения к корню.

Предположим, что у нас есть нелинейное уравнение вида *f(x) = 0*, и мы хотим найти его корень. Метод Ньютона-Канторовича начинает с выбора начального приближения *x0* и затем использует итерационную формулу:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

где $f'(x_n)$ - производная функции f по переменной x в точке x_n.

Оценка Канторовича включает в себя оценку величины производной второго порядка в некоторой окрестности корня. Если оценка величины производной второго порядка остается ограниченной в этой окрестности, то метод сходится к корню. Это условие является частью проверки наличия корня в выбранной области.

Основные шаги метода Ньютона-Канторовича:

1) Выбор начального приближения *x0* – выбирается начальная точка, близкая к корню.

2) Итерации:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

3) Оценка Канторовича:

Оценивается величина второй производной функции в некоторой окрестности корня для проверки условия сходимости.

4) Проверка условия сходимости:
Если оценка Канторовича удовлетворяет условиям, метод завершается.

5) Повторение шагов 2-4: Повторяется до достижения заданной точности или максимального числа итераций.

Метод Ньютона-Канторовича обычно сходится к корню быстро в том случае, если начальное приближение достаточно близко к реальному корню и если выполняются условия сходимости оценки Канторовича. Однако, метод может не сходиться или сходиться медленно, если начальное приближение далеко от корня или если условия оценки Канторовича не выполняются в выбранной области.

Код программы

```
import matplotlib.pyplot as plt
import numpy as np

kernel = lambda x, s: np.exp(s - x)
right_side = lambda x: np.exp(-x)
exact_solution = lambda x: 1 + x * 0
start = 0
end = 1
step_size = 0.05

def quad_trapezoid_method(start, end, step_size, kernel, right_side):
    num_intervals = int((end - start) / step_size)
    x_values = np.linspace(start, end, num_intervals + 1)
    y_values = np.empty_like(x_values)
```

```
    y_values[0] = right_side(x_values[0])
    kernel_ii = kernel(x_values[0], x_values[0])

    for i in range(1, num_intervals + 1):
        c_i = right_side(x_values[i]) + sum(
            step_size * kernel(x_values[i], x_values[j]) * y_values[j] *
y_values[j] for j in range(0, i))
        y_values[i] = (1 - np.sqrt(1 - 2 * step_size * kernel_ii * c_i)) /
(step_size * kernel_ii)

    return x_values, y_values

def runge_romberg_error_estimation(method, order, step_size):
    x_h, y_h = method(start, end, step_size, kernel, right_side)
    x_2h, y_2h = method(start, end, 2 * step_size, kernel, right_side)
    num_intervals = len(x_2h)
    error = max(abs((y_h[2 * i] - y_2h[i])) / (2 ** order - 1) for i in
range(num_intervals))
    print("Рунге-Ромберг =", error)

figsize_result = (8, 6)
figsize_error = (8, 4)
x_result, y_result = quad_trapezoid_method(start, end, step_size, kernel,
right_side)
plt.figure(figsize=figsize_result)
plt.plot(x_result, y_result, 'o', label='Результат', color='blue')
plt.plot(x_result, exact_solution(x_result), '-', lw=2, label='Решение',
color='green')
plt.grid(True)
plt.title('Метод квадратур')
plt.show()
plt.figure(figsize=figsize_error)
plt.grid(True)
plt.title('График ошибки')
plt.plot(x_result, abs(y_result - exact_solution(x_result)), '-',
color='red')
plt.show()
max_error = max(abs(y_result - exact_solution(x_result)))
print("Ошибка = ", max_error)
runge_romberg_error_estimation(quad_trapezoid_method, 2, step_size)

kernel = lambda x, s: 1 + x * s * 0
right_side = lambda x: np.sin(x) - x / 2 + np.sin(2 * x) / 4
exact_solution = lambda x: np.sin(x)
start = 0
end = np.pi / 2
step_size = np.pi / 2 / 14

x_result, y_result = quad_trapezoid_method(start, end, step_size, kernel,
right_side)
plt.figure(figsize=figsize_result)
plt.plot(x_result, y_result, 'o', label='Результат', color='blue')
plt.plot(x_result, exact_solution(x_result), '-', lw=2, label='Решение',
color='green')
plt.grid(True)
plt.title('Метод квадратур')
plt.show()
plt.figure(figsize=figsize_error)
plt.grid(True)
plt.title('График ошибки')
```

```
plt.plot(x_result, abs(y_result - exact_solution(x_result)), '-',
color='red')
plt.show()
max_error = max(abs(y_result - exact_solution(x_result)))
print("Ошибка = ", max_error)
runge_romberg_error_estimation(quad_trapezoid_method, 2, step_size)


kernel = lambda x, s, y: np.cos(np.pi * x) * np.sin(np.pi * s) * y ** 3 / 5
kernel_derivative = lambda x, s, y: np.cos(np.pi * x) * np.sin(np.pi * s) * y
* y * 3 / 5
right_hand_side = lambda x: np.sin(np.pi * x)
true_solution = lambda x: np.sin(np.pi * x) + np.cos(np.pi * x) * (20 -
np.sqrt(391)) / 3

a = 0
b = 1
h = 0.05
m = 2

def runge_romberg_2(method, p, h):
    x_h, y_h = method(kernel, kernel_derivative, right_hand_side, a, b, h, m)
    x_2h, y_2h = method(kernel, kernel_derivative, right_hand_side, a, b, 2 *
h, m)
    N = len(x_2h)
    R = max(abs((y_h[2 * i] - y_2h[i])) / (2 ** p - 1) for i in range(N))
    print("Рунге-Ромберг = ", R)

def newton_kantorovich(K, K_y, f, a, b, h, m):
    k = 0
    N = int((b - a) / h)
    x = np.linspace(a, b, N + 1)
    I = np.eye(N + 1)
    Y = np.ones_like(x)
    F = np.empty_like(Y)
    A = np.empty((N + 1, N + 1))
    while k < m:
        for i in range(N + 1):
            F[i] = f(x[i]) + sum(
                h * (K(x[i], x[j], Y[j]) - K_y(x[i], x[j], Y[j]) * Y[j]) for
j in range(1, N)) + h / 2 * (
                                K(x[i], x[0], Y[0]) + K(x[i], x[N], Y[N]) -
K_y(x[i], x[0], Y[0]) * Y[0] - K_y(x[i], x[N],

Y[N]) * Y[N])
        for i in range(N + 1):
            A[i, 0] = K_y(x[i], x[0], Y[0]) * h / 2
            A[i, N] = K_y(x[i], x[N], Y[N]) * h / 2
            for j in range(1, N):
                A[i, j] = K_y(x[i], x[j], Y[j]) * h
        Y = np.linalg.solve(I - A, F)
        k += 1
    return x, Y

x_result, y_result = newton_kantorovich(kernel, kernel_derivative,
right hand side, a, b, h, m)
plt.plot(x_result, y_result, 'o', label='Результат', color='blue')
plt.plot(x_result, true_solution(x_result), '-', lw=2, label='Решение',
color='green')
plt.grid(True)
```

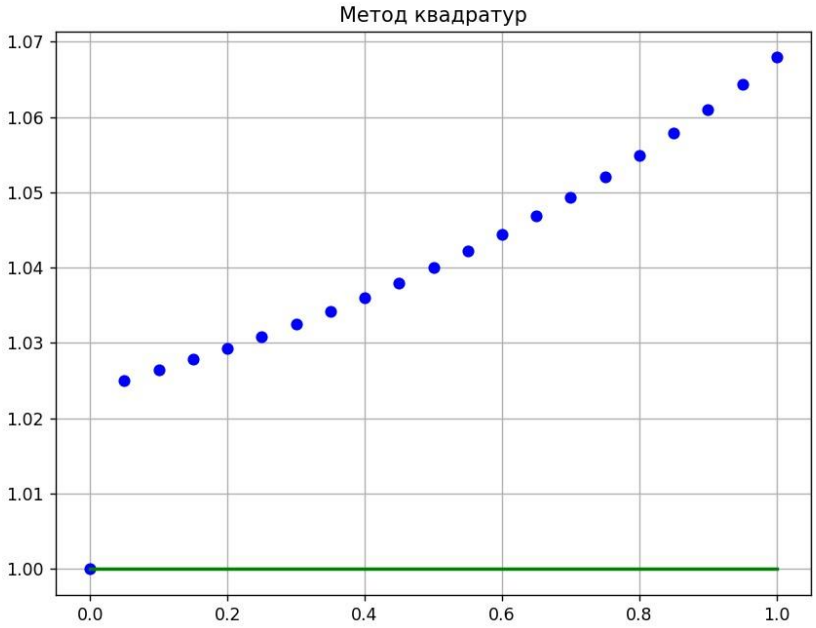
```
plt.title('Ньютон-Канторович')
plt.show()
plt.grid(True)
plt.title('График ошибки')
plt.plot(x_result, abs(y_result - true_solution(x_result)), '-', color='red')
plt.show()
print("Ошибка = ", max(abs(y_result - true_solution(x_result))))
runge_romberg_2(newton_kantorovich, 2, h)
```

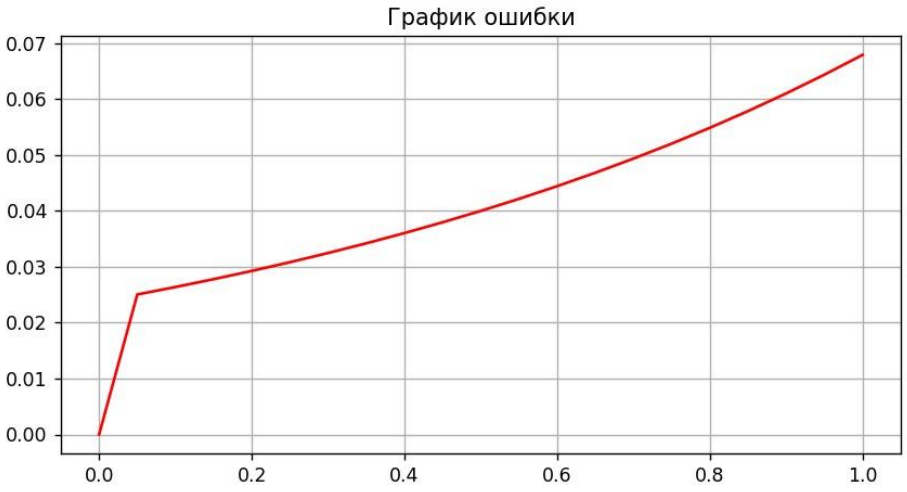
Пример №1 (y – точное решение):

$$y(x) - \int_0^x e^{-(x-s)} y^2(s)\, ds = e^{-x}$$

$$y = 1$$

Результат:



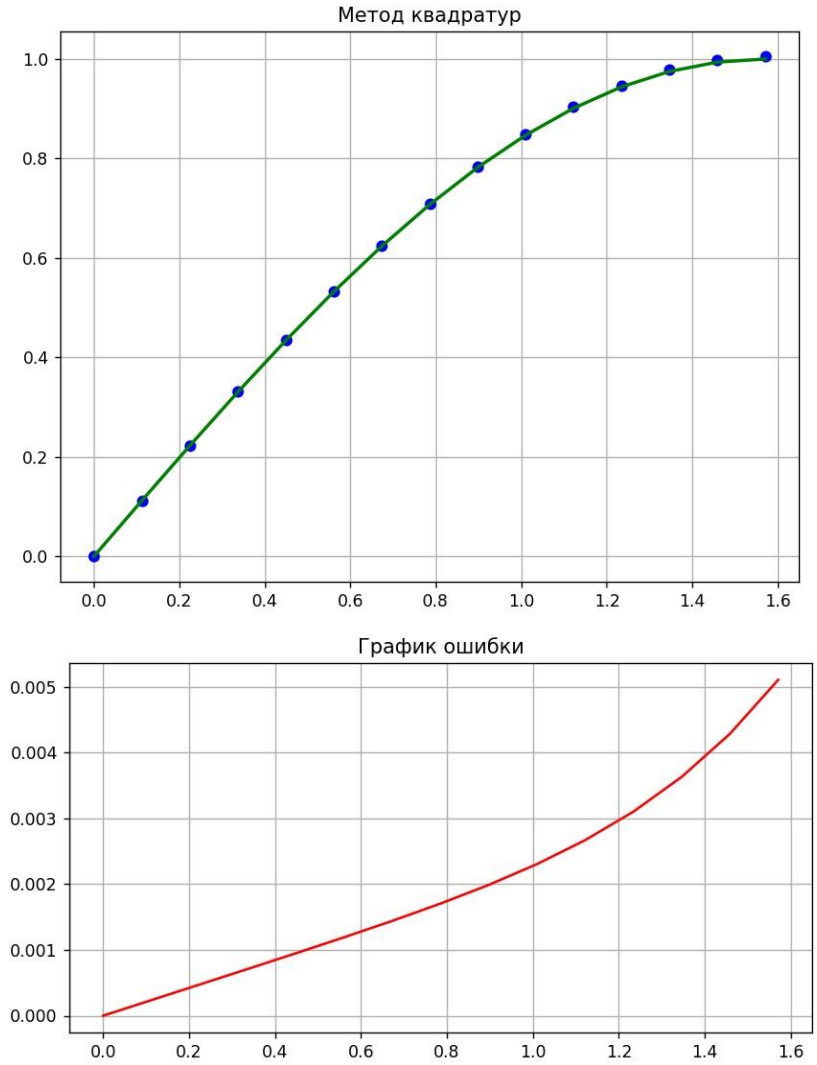


Пример №2 (y - точное решение):

$$y(x) - \int_0^x y^2(s)\,ds = \sin x - \frac{x}{2} + \frac{\sin 2x}{4}$$

$$y = \sin x$$

Результат:

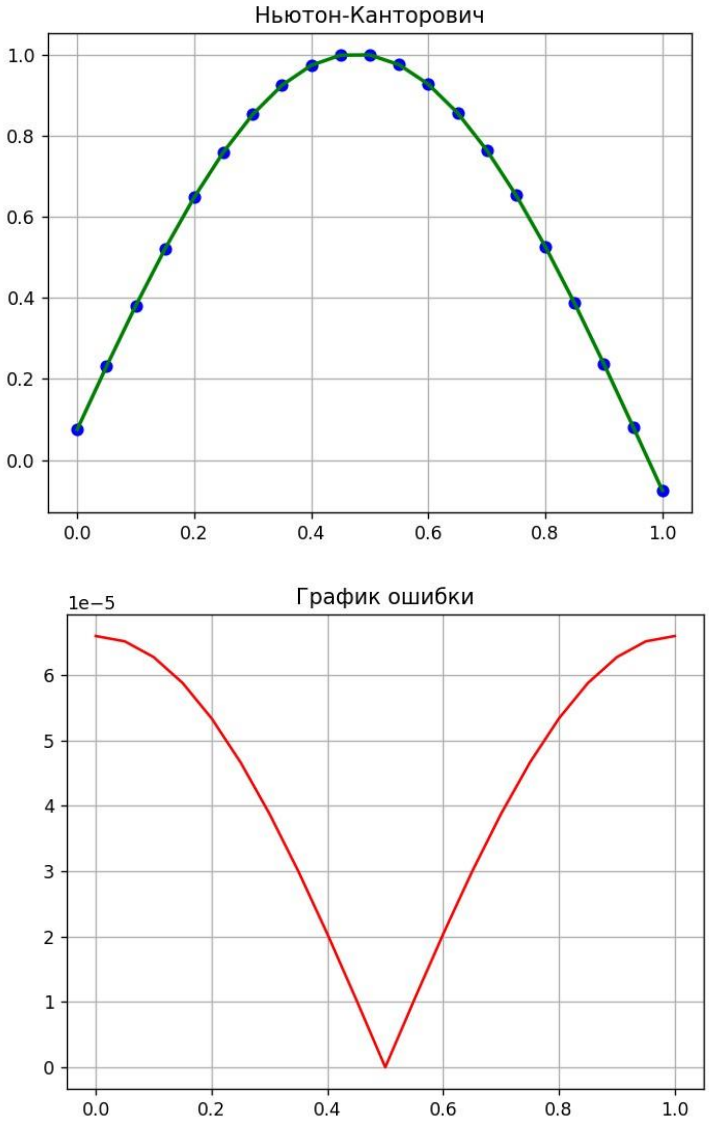


Пример №3 (y - точное решение):

$$y(x) = \sin \pi x + \frac{1}{5} \int_0^1 \cos(\pi x) \sin(\pi s) y^3(s)\, ds$$

$$y(x) = \sin \pi x + \frac{1}{3}(20 - \sqrt{391}) \cos \pi x$$

Результат:



Вывод по курсовой работе

Благодаря данному курсовому проекту, я приобрел знания в области численных методов для решения нелинейных интегральных уравнений: были реализованы методы РунгеРомберга, квадратур, Ньютона-Канторовича, а также эти методы были проиллюстрированы тремя примерами решений уравнений Вольтерры и Фредгольма с графиками решений и графиками ошибок.