Санкт - Петербургский государственный университет
Математико - механический факультет

Отчёт по практическому заданию №4

Численное решение интегрального уравнения Фредгольма 2-го рода.

Выполнил:
студент 451 гр.
Челабов А.Г.

Санкт – Петербург
2024

Постановка задачи

Интегральное уравнение Фредгольма второго рода

$$u(x) - \int_a^b H(x,y)u(y)dy = f(x), \quad f(x) \in C_{[a,b]}, \tag{1}$$

где ядро $H(x, y)$ - достаточное количество раз непрерывно дифференцируемо.

Метод замены ядра на вырожденное

Вырожденным называется ядро, представимое в виде

$$\widetilde{H}(x,y) = \sum_{i=1}^{n} \alpha_i(x)\beta_i(y). \tag{2}$$

Пусть $H(x,y) \approx \widetilde{H}(x,y)$ и будем решать уравнение

$$\widetilde{u}^n(x) - \int_a^b \widetilde{H}(x,y)\widetilde{u}^n(y)dy = f(x). \tag{3}$$

Если уравнение (3) имеет решение, то оно представимо в виде

$$\widetilde{u}^n(x) = f(x) + \sum_{i=1}^{n} c_i\alpha_i(x), \tag{4}$$

где

$$c_i = \int_a^b \beta_i(y)\widetilde{u}^n(y)dy = \int_a^b \beta_i(y)(f(y) + \sum_{j1}^{n} c_j\alpha_j(y))\,dy =$$

$$= \sum_{j=1}^{n} \int_a^b \beta_i(y)\alpha_j(y)dy\, c_j + \int_a^b \beta_i(y)f(y)dy. \tag{5}$$

Обозначим

$$\gamma_{ij} = \int_a^b \beta_i(y)\alpha_j(y)dy, \; b_i = \int_a^b \beta_i(y)f(y)dy, \; a_{ij} = \delta_{ij} - \gamma_{ij}, \tag{6}$$

($\delta_{ij} -$ символ Кронекера), тогда c_i является решением системы линейных алгебраических уравнений $AC = B$. Здесь $A = (a_{ij})^n{}_{i,j=1} -$ матрица коэффициентов, $B = (b_1, b_2, \dots, b_n)^T -$ вектор правых частей, $C = (c_1, c_2, \dots, c_n)^T -$ искомый вектор.

Метод механических квадратур

Используем для рассмотрения, квадратурную формулу

$$\int_a^b v(x)dx \approx \sum_{k=1}^{n} A_k v(x_k), \qquad (14)$$

где узлы $x_k \epsilon [a, b], x_k \neq x_j$ *при* $k \neq j$.

Заменив интеграл в уравнении (1) приближенно на квадратурную сумму, получим новое уравнение относительно новой неизвестной функции $u^n(x)$

$$u^n(x) - \sum_{k=1}^{n} A_k H(x,x_k) u^n(x_k) = f(x). \qquad (15)$$

Если квадратурная сумма достаточно хорошо приближает интеграл, то есть основания надеяться, что решение $u^n(x)$уравнения (15) близко к решению $u(x)$ уравнения (1).

Для решения уравнения (15) будем полагать x поочередно равными $x_1, x_2, \ldots, x_n$.

Обозначим $\varsigma_j = u^n(x_j)$, тогда ς_j обязаны удовлетворять системе уравнений

$$\varsigma_j - \sum_{k=1}^{n} A_k H(x_j, x_k) \varsigma_k = f(x_j), \quad j = 1, 2, \ldots, n \qquad (16)$$

или, в матричной записи, $Dz = g$, где

$$D = (d_{jk})_{j,k=1}^{n}, \quad d_{jk} = \delta_{jk} - A_k H(x_j, x_k), \quad g = (f(x_1), f(x_2), \ldots, f(x_n)), \qquad (17)$$

$z = (\varsigma_1, \varsigma_2, \ldots, \varsigma_n)$ – искомый вектор.

После вычисления решения системы (16) $z = (\varsigma_1, \varsigma_2, \ldots, \varsigma_n)$ решение уравнения (15) может быть получено по формуле

$$u^n(x) = \sum_{k=1}^{n} A_k H(x, x_k) \varsigma_k + f(x). \qquad (18)$$

Решение

$$u(x) + 0.5 \int_0^1 e^{xy}\, u(y)\, dy = x + 0.5.$$

```
H = lambda x, y: np.exp(x * y)
f = lambda x: x+ .5
a, b = 0, 1
# Метод замены ядра на вырожденное
def taylor(k):
    return lambda x: x**k / np.sqrt(math.factorial(k))

alpha = [taylor (k) for k in range(4)]
beta = [taylor (k) for k in range(4)]
H_3 = lambda x, y: np. sum( [alpha[i] (x) * beta[i] (y) for i in range(3)])
H_4 = lambda x, y: np. sum( [alpha[i] (x) * beta[i] (y) for i in range(4)])

# ---------------------------------------------------------------------------


def taylor(k):
    return lambda x: x**k / np.sqrt(math.factorial(k))

alpha = [taylor(k) for k in range(4)]
beta = [taylor(k) for k in range(4)]

def B(k):
    res = np.zeros(k)
    for i in range(k):
        res[i], _ = quad(lambda x: beta[i](x) * f(x), a, b)
    return res

def G(k):
    res = np.zeros((k, k))
    for i in range(k):
        for j in range(k):
            res[i][j], _ = quad(lambda x: beta[i](x) * alpha[j](x), a, b)
    return res
```

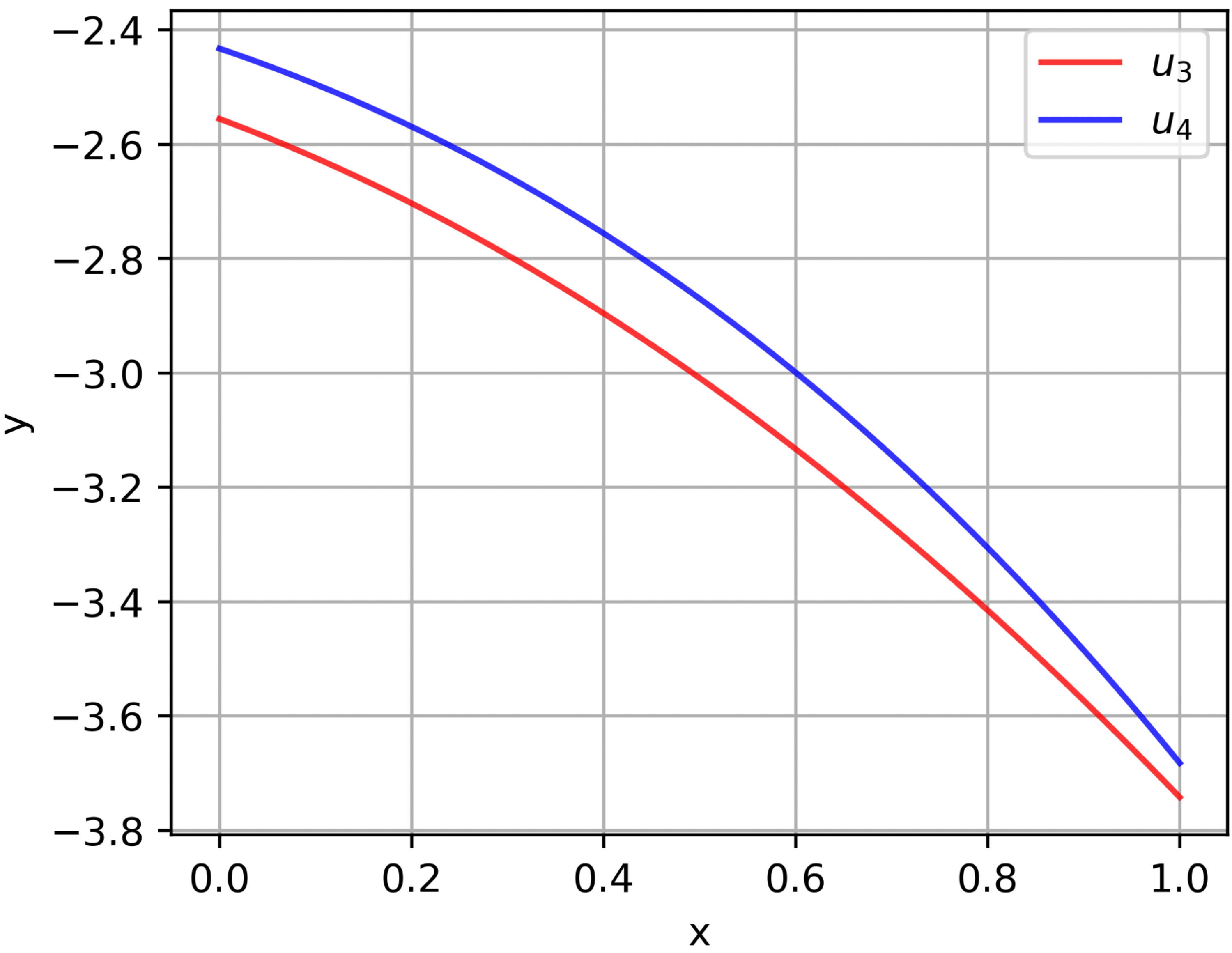
```python
def A(k):
    G_k = G(k)
    res = np.zeros((k, k))
    for i in range(k):
        for j in range(k):
            res[i][j] = (i == j) - G_k[i][j]
    return res

def C(k):
    return np.linalg.solve(A(k), B(k))

# # Вывод результатов
# print("A(3) = ", A(3), "\nB(3) = ", B(3), "\nC(3) = ", C(3))
# print("A(4) = ", A(4), "\nB(4) = ", B(4), "\nC(4) = ", C(4))


# Функция для форматированного вывода матриц
def print_matrix(matrix, name):
    print(f"{name} = ")
    for row in matrix:
        print("  [", "  ".join(f"{val:.6f}" for val in row), "]")
    print()

# Вывод результатов
print_matrix(A(3), "A(3)")
print_matrix(B(3).reshape(1, -1), "B(3)")
print_matrix(C(3).reshape(1, -1), "C(3)")

print_matrix(A(4), "A(4)")
print_matrix(B(4).reshape(1, -1), "B(4)")
print_matrix(C(4).reshape(1, -1), "C(4)")
```

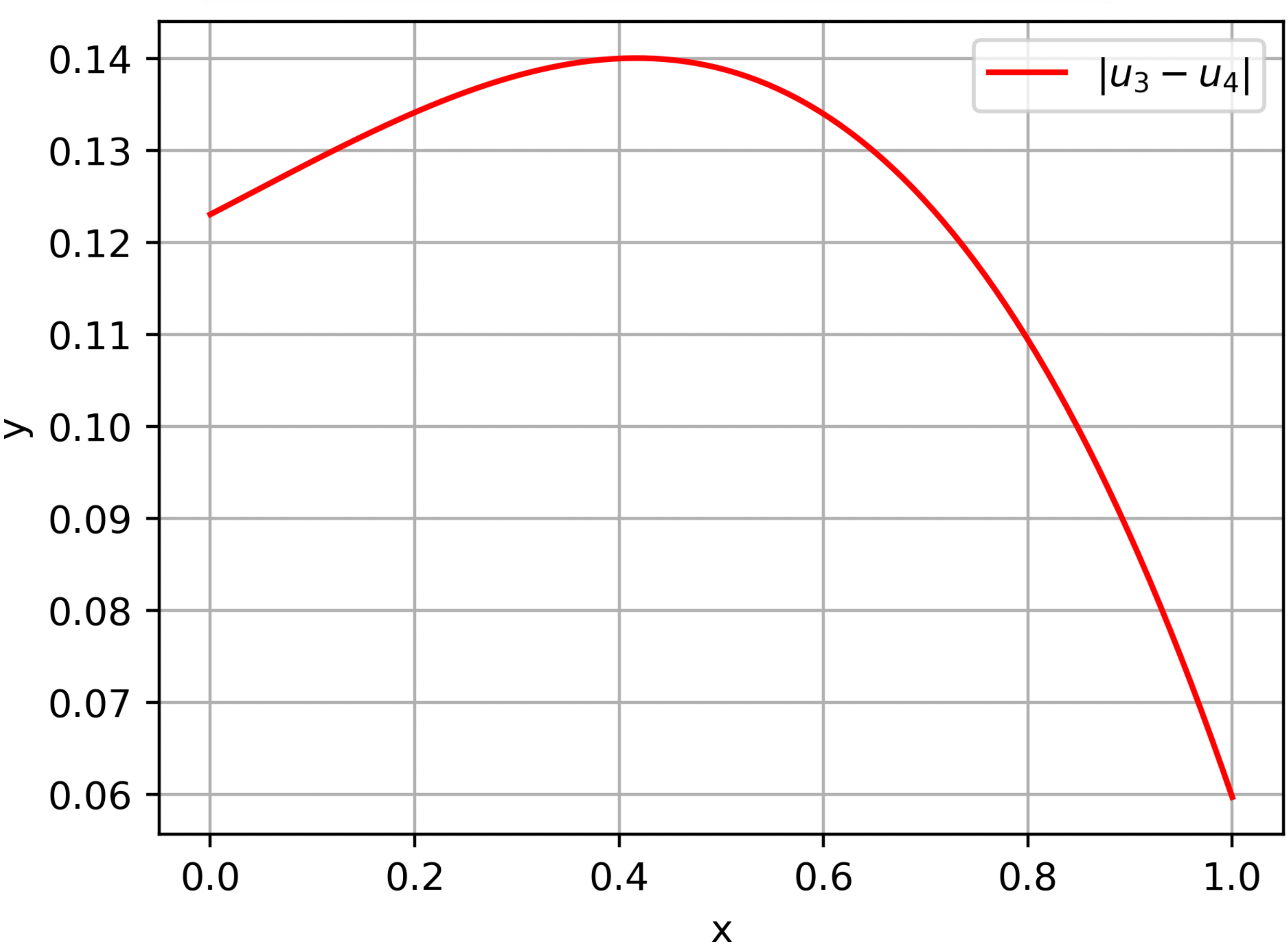
```
A(3) =
  [ 0.000000  -0.500000  -0.235702 ]
  [ -0.500000  0.666667  -0.176777 ]
  [ -0.235702  -0.176777  0.900000 ]

B(3) =
  [ 1.000000  0.583333  0.294628 ]

C(3) =
  [ -3.055259  -1.626524  -0.792261 ]

A(4) =
  [ 0.000000  -0.500000  -0.235702  -0.102062 ]
  [ -0.500000  0.666667  -0.176777  -0.081650 ]
  [ -0.235702  -0.176777  0.900000  -0.048113 ]
  [ -0.102062  -0.081650  -0.048113  0.976190 ]

B(4) =
  [ 1.000000  0.583333  0.294628  0.132681 ]

C(4) =
  [ -2.932218  -1.569129  -0.766924  -0.339693 ]

Delta = 0.1388825690799833
```

```python
def U(k):
    C_k = C(k)
    return lambda x: f(x) + sum(C_k[i] * alpha[i](x) for i in range(k))

u_3 = U(3)
u_4 = U(4)

pts = [0, 0.5, 1]

# Вычисление максимальной разности
delta = np.max([abs(u_3(p) - u_4(p)) for p in pts])
print(f"Delta = {delta}")

# Создание DataFrame
data = {
    "U3": {
        "0": u_3(0),
        "0.5": u_3(0.5),
        "1": u_3(1)
    },
    "U4": {
        "0": u_4(0),
        "0.5": u_4(0.5),
        "1": u_4(1)
    }
}

df = pd.DataFrame(data)
print(df)
```

```
Delta = 0.1388825690799833
           U3         U4
0   -2.555259 -2.432218
0.5 -3.008575 -2.869692
1   -3.741997 -3.682324
```

```python
# Построение графика
fig, ax = plt.subplots(figsize=(10, 6), dpi=200)

x = np.linspace(a, b, 100)

plt.plot(x, u_3(x), color='red', label=r'$u_3$', alpha=0.8)
plt.plot(x, u_4(x), color='blue', label=r'$u_4$', alpha=0.8)

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend()
ax.grid()

plt.show()
```





0.14
0.13
0.12
0.11
0.10
0.09
0.08
0.07
0.06
y
0.0
0.2
0.4
0.6
0.8
1.0
x
$|u_3 - u_4|$

```python
# Метод механических квадратур

n = 200
eps = 1.e-8

def target(u: Callable) -> float:
    return abs(max(u(a), u((a + b) / 2), u(b)))

var = 1.e+8
u_n = u_3

while var > eps:
    n *= 2
    h = (b - a) / (n + 1)
    x_ = np.linspace(a + h, b - h, n)
    D = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            D[i][j] = (i == j) - h * H(x_[i], x_[j])

    q = f(x_)
    z = np.linalg.solve(D, q)
    t_prev = target(u_n)

    def sol(x):
        s = f(x)
        for i in range(n):
            s += h * H(x, x_[i]) * z[i]
        return s

    u_n = sol
    t_next = target(u_n)
    var = abs(t_prev - t_next)
```

```python
n = 800

fig, ax = plt.subplots(figsize=(10, 6), dpi=200)
x = np.linspace(a, b, 100)
plt.plot(x, u_3(x), color="red", label=r"$u_3$", alpha=0.8)
plt.plot(x, u_4(x), color="blue", label=r"$u_4$", alpha=0.8)
plt.plot(x, u_n(x), color="green", label=r"$u_n$", alpha=0.8)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.legend()
ax.grid()
plt.show()
```