Московский авиационный институт
(национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Численные методы»

Студент:	А. А. Каримов
Преподаватель:	Д. В. Беляков
Группа:	М8О-406Б-22
Дата:	
Оценка:	
Подпись:	

Москва, 2025

1 Формулировка задачи №7

Задача: Решить краевую задачу для дифференциального уравнения эллиптического типа. Аппроксимацию уравнения произвести с использованием центрально-разностной схемы. Для решения дискретного аналога применить следующие методы: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $U(x, y)$. Исследовать зависимость погрешности от сеточных параметров h_x, h_y.

0.1 Вариант 8

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2 \cdot \frac{\partial u}{\partial x} - 2 \cdot \frac{\partial u}{\partial y} - 4 \cdot u$$

$$u(0, y) = e^{-y} \cos y$$

$$u(\frac{\pi}{2}, y) = 0$$

$$u(x, 0) = e^{-x} \cdot \cos x$$

$$u(x, \frac{\pi}{2}) = 0$$

Аналитическое решение:

$$U(x, y) = e^{-x-y} \cdot \cos x \cdot \cos y$$

2 Теория

МЕТОД КОНЕЧНЫХ РАЗНОСТЕЙ ДЛЯ РЕШЕНИЯ УРАВНЕНИЙ ЭЛЛИПТИЧЕСКОГО ТИПА

Постановка задач для уравнений эллиптического типа

Классическим примером уравнения эллиптического типа является уравнение Пуассона

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \tag{1}$$

или уравнение Лапласа при $f(x, y) \equiv 0$.

Здесь функция $u(x, y)$ имеет различный физический смысл, а именно: стационарное, независящее от времени, распределение температуры, скорость потенциального (безвихревого) течения идеальной (без трения и теплопроводности) жидкости, распределение напряженностей электрического и магнитного полей, потенциала в силовом поле тяготения и т.п.

Если на границе Γ расчетной области $\bar{\Omega} = \Omega + \Gamma$ задана искомая функция, то соответствующая первая краевая задача для уравнения Лапласа или Пуассона называется **задачей Дирихле**:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (x, y) \in \Omega; \tag{2}$$

$$u(x, y) = \phi(x, y), \quad (x, y) \in \Gamma. \tag{3}$$

Если на границе Γ задается нормальная производная искомой функции, то соответствующая вторая краевая задача называется **задачей Неймана** для уравнения Лапласа или Пуассона:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (x, y) \in \Omega; \tag{4}$$

$$\frac{\partial u(x, y)}{\partial n} = \phi(x, y), \quad (x, y) \in \Gamma. \tag{5}$$

При этом n – направление внешней к границе Γ нормали.

Более приемлемой является координатная форма краевого условия (5)

$$\frac{\partial u}{\partial x} \cos(\hat{n}, i) + \frac{\partial u}{\partial y} \cos(\hat{n}, j) = \phi(x, y),$$

где $\cos(\hat{n}, i)$, $\cos(\hat{n}, j)$ – направляющие косинусы внешнего вектора единичной нормали к границе Γ, i и j – орты базисных векторов.

Наконец **третья краевая задача** для уравнения Пуассона (Лапласа) имеет вид:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (x, y) \in \Omega; \tag{6}$$

$$\frac{\partial u}{\partial n} + \alpha u = \phi(x, y), \quad (x, y) \in \Gamma. \tag{7}$$

Конечно-разностная аппроксимация задач для уравнений эллиптического типа

Рассмотрим краевую задачу для уравнений Лапласа или Пуассона (2), (3) в прямоугольнике $x \in [0, l_1]$, $y \in [0, l_2]$, на который наложим сетку:

$$\omega_{h_1 h_2} = \left\{(x_i, y_j) : x_i = ih_1, i = \overline{0, N_1}; y_j = jh_2, j = \overline{0, N_2}\right\}. \tag{8}$$

На этой сетке аппроксимируем дифференциальную задачу во внутренних узлах с помощью отношения конечных разностей по следующей схеме (вводится сеточная функция $u_{i,j}$, $i = \overline{0, N_1}$, $j = \overline{0, N_2}$):

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_1^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_2^2} = f(x_i, y_j) + O(h_1^2 + h_2^2), \tag{9}$$

$i = \overline{1, N_1 - 1}$, $j = \overline{1, N_2 - 1}$,

которая на шаблоне имеет второй порядок по переменным x и y, поскольку шаблон центрально симметричен.

СЛАУ (9) имеет пятидиагональный вид (каждое уравнение содержит пять неизвестных и при соответствующей нумерации переменных матрица имеет ленточную структуру). Решать ее можно различными методами линейной алгебры, например, итерационными методами, методом матричной прогонки и т.п.

Рассмотрим разностно-итерационный метод Либмана численного решения задачи Дирихле (2), (3). Для простоты изложения этого метода примем $h = h_1 = h_2$, тогда из схемы (9) получим (k-номер итерации):

$$u_{i,j}^{(k+1)} = \frac{1}{4}\left[u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)} - h^2 f_{i,j}\right], \tag{10}$$

$i = \overline{1, N_1 - 1}$, $j = \overline{1, N_2 - 1}$,

где $f_{i,j} = f(x_i, y_j)$.

На каждой координатной линии (например, $y_j = \text{const}$, $j = \overline{1, N_2 - 1}$) с помощью линейной интерполяции граничных значений $\phi(x, y)$ определим $u_{i,j}^{(0)}$, подставив которые в (10), получим распределение $u_{i,j}^{(1)}$ на первой итерации:

$$u_{i,j}^{(1)} = \frac{1}{4}\left[u_{i+1,j}^{(0)} + u_{i-1,j}^{(0)} + u_{i,j+1}^{(0)} + u_{i,j-1}^{(0)} - h^2 f_{i,j}\right].$$

Это распределение снова подставляется в (10), получаем распределение $u_{i,j}^{(2)}$ и т.д. Процесс Либмана прекращается, когда

$$\|u^{(k+1)} - u^{(k)}\| \leq \varepsilon, \quad \|u^{(k)}\| = \max_{i,j} |u_{i,j}^{(k)}|,$$

где ε - наперед заданная точность.

3 Исходный код

Здесь располагается реализация задачи.

```
# %%
import matplotlib.cm
import matplotlib.pyplot as plt
import numpy as np
import random

# %%
EPS = 1e-9

# %% [markdown]
#

# %%
omega = 1.5

# %% [markdown]
#

# %%
Lx = np.pi / 2
Ly = np.pi / 2
n = 35
m = 35

# %% [markdown]
#

# %%
a = 1
b = 0
c = 1
d = 2
e = 2
f = 4
g = 0

# %% [markdown]
```

```
#

# %%
alpha_0y = 0
beta_0y = 1

alpha_Ly = 0
beta_Ly = 1

alpha_x0 = 0
beta_x0 = 1

alpha_xL = 0
beta_xL = 1

# %% [markdown]
#

# %%
def gamma_0y(y):
    return np.exp(-y) * np.cos(y)


def gamma_Ly(y):
    return 0


def gamma_x0(x):
    return np.exp(-x) * np.cos(x)


def gamma_xL(x):
    return 0

# %% [markdown]
#

# %%
def analytical(x, y):
    return np.exp(-x - y) * np.cos(x) * np.cos(y)

# %% [markdown]
#  $h_x$  $h_y$

# %%
hx = Lx / (n - 1)
hy = Ly / (m - 1)

hy2 = hy**2
```

```
hx2 = hx**2

# %%
x = [i * hx for i in range(n - 1)]
x.append(Lx)
x = np.array(x)

y = [j * hy for j in range(m - 1)]
y.append(Ly)
y = np.array(y)

# %%
bound_x0 = []
bound_xL = []

for i in range(n):
    bound_x0.append(gamma_x0(x[i]))
    bound_xL.append(gamma_xL(x[i]))

bound_0y = []
bound_Ly = []

for j in range(m):
    bound_0y.append(gamma_0y(y[j]))
    bound_Ly.append(gamma_Ly(y[j]))

# %%
x_plt, y_plt = np.meshgrid(x, y, indexing='ij')
true_data = analytical(x_plt, y_plt)

# %%
def get_delta(u, u_next):
    delta = -1
    for i in range(n):
        for j in range(m):
            delta = max(delta, abs(u[i][j] - u_next[i][j]))
    return delta

# %%
def get_boundary(u_next):
    for i in range(n):
        u_next[i][0] = 0
        u_next[i][m - 1] = 0

    for j in range(m):
        u_next[0][j] = 0
        u_next[n - 1][j] = 0

```

```
    for i in range(n):
        u_next[i][0] += (bound_x0[i] - alpha_x0 / hy * u_next[i][1]) / \
                        (beta_x0 - alpha_x0 / hy)

    for i in range(n):
        u_next[i][m - 1] += (bound_xL[i] - alpha_xL / hy * u_next[i][m - 2]) / \
                            (beta_xL - alpha_xL / hy)


    for j in range(m):
        u_next[0][j] += (bound_0y[j] - alpha_0y / hx * u_next[1][j]) / \
                        (beta_0y - alpha_0y / hx)

    for j in range(m):
        u_next[n - 1][j] += (bound_Ly[j] + alpha_Ly / hx * u_next[n - 2][j]) / \
                            (beta_Ly + alpha_Ly / hx)


    u_next[n - 1][m - 1] /= 2
    u_next[n - 1][0] /= 2
    u_next[0][m - 1] /= 2
    u_next[0][0] /= 2

    return u_next


# %% [markdown]
#

# %%
def liebmann_method():
    delta = 1
    iter_count = 0

    u = np.ones((n, m)) * 0.1

    while delta > EPS:
        u_next = np.zeros((n, m))

        for i in range(1, n - 1):
            for j in range(1, m - 1):
                u_next[i][j] = (c / hy2 - e / (2 * hy)) * u[i][j - 1] + \
                               (c / hy2 + e / (2 * hy)) * u[i][j + 1] + \
                               (a / hx2 - d / (2 * hx)) * u[i - 1][j] + \
                               (a / hx2 + d / (2 * hx)) * u[i + 1][j] + g
                u_next[i][j] /= (2 * (a / hx2 + c / hy2) - f)

        u_next = get_boundary(u_next)
        delta = get_delta(u, u_next)
```

```
            u = u_next
            iter_count += 1

    # print(f"Iteration counter: {iter_count}")
    return u

# %%
liebmann_data = liebmann_method().transpose()

# %%
x_plt, y_plt = np.meshgrid(x, y)
true_data = analytical(x_plt, y_plt)

y_indices = random.sample(range(len(y) - 1), k=min(3, len(y)))
y_indices.sort()

fig = plt.figure(figsize=(20, 12))

for i, y_idx in enumerate(y_indices):
    ax = fig.add_subplot(2, 3, i+1)
    ax.plot(x, liebmann_data[y_idx], label="")
    ax.plot(x, true_data[y_idx], label="")
    ax.set_title(f"y = {y[y_idx]:.2f}")
    ax.set_xlabel('x')
    ax.set_ylabel('u(x,y)')
    ax.legend()
    ax.grid(True)

ax1 = fig.add_subplot(2, 2, 3, projection='3d')
ax1.plot_surface(x_plt, y_plt, liebmann_data, cmap='plasma')
ax1.set_title(' ')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('u(x,y)')

ax2 = fig.add_subplot(2, 2, 4, projection='3d')
ax2.plot_surface(x_plt, y_plt, true_data, cmap='plasma')
ax2.set_title(' ')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('u(x,y)')

plt.show()

# %% [markdown]
#

# %%
def seidel_method():
```

```python
    delta = 1
    iter_count = 0

    u = np.ones((n, m)) * 0.1

    while delta > EPS:
        u_next = u.copy()

        for i in range(1, n - 1):
            for j in range(1, m - 1):
                u_next[i][j] = (c / hy2 - e / (2 * hy)) * u_next[i][j - 1] + \
                               (c / hy2 + e / (2 * hy)) * u[i][j + 1] + \
                               (a / hx2 - d / (2 * hx)) * u_next[i - 1][j] + \
                               (a / hx2 + d / (2 * hx)) * u[i + 1][j] + g
                u_next[i][j] /= (2 * (a / hx2 + c / hy2) - f)

        u_next = get_boundary(u_next)
        delta = get_delta(u, u_next)
        u = u_next
        iter_count += 1

    # print(f"Iteration counter: {iter_count}")
    return u

# %%
seidel_data = seidel_method().transpose()

# %%
x_plt, y_plt = np.meshgrid(x, y)
true_data = analytical(x_plt, y_plt)

y_indices = random.sample(range(len(y) - 1), k=min(3, len(y)))
y_indices.sort()

fig = plt.figure(figsize=(20, 12))

for i, y_idx in enumerate(y_indices):
    ax = fig.add_subplot(2, 3, i+1)
    ax.plot(x, seidel_data[y_idx], label="")
    ax.plot(x, true_data[y_idx], label="")
    ax.set_title(f"y = {y[y_idx]:.2f}")
    ax.set_xlabel('x')
    ax.set_ylabel('u(x,y)')
    ax.legend()
    ax.grid(True)

ax1 = fig.add_subplot(2, 2, 3, projection='3d')
ax1.plot_surface(x_plt, y_plt, seidel_data, cmap='plasma')
ax1.set_title(' ')
```

```
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('u(x,y)')

ax2 = fig.add_subplot(2, 2, 4, projection='3d')
ax2.plot_surface(x_plt, y_plt, true_data, cmap='plasma')
ax2.set_title(' ')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('u(x,y)')

plt.show()

# %% [markdown]
#

# %%
def relaxation_method():
    omega = 1.0038003800380038
    delta = 1
    iter_count = 0

    u = np.ones((n, m)) * 0.1

    while delta > EPS and iter_count < 1e3:
        u_next = np.zeros((n, m))

        for i in range(1, n - 1):
            for j in range(1, m - 1):
                u_next[i][j] = (c / hy2 - e / (2 * hy)) * u[i][j - 1] + \
                               (c / hy2 + e / (2 * hy)) * u[i][j + 1] + \
                               (a / hx2 - d / (2 * hx)) * u[i - 1][j] + \
                               (a / hx2 + d / (2 * hx)) * u[i + 1][j] + g
                u_next[i][j] /= (2 * (a / hx2 + c / hy2) - f)

        u_next = get_boundary(u_next)

        u_next = u_next * omega + (1 - omega) * u
        delta = get_delta(u, u_next)
        u = u_next
        iter_count += 1

    # print(f"Iteration counter: {iter_count}")
    return u

# %%
relaxation_data = relaxation_method().transpose()

# %%
```

```python
x_plt, y_plt = np.meshgrid(x, y)
true_data = analytical(x_plt, y_plt)

y_indices = random.sample(range(len(y) - 1), k=min(3, len(y)))
y_indices.sort()

fig = plt.figure(figsize=(20, 12))

for i, y_idx in enumerate(y_indices):
    ax = fig.add_subplot(2, 3, i+1)
    ax.plot(x, relaxation_data[y_idx], label="")
    ax.plot(x, true_data[y_idx], label="")
    ax.set_title(f"y = {y[y_idx]:.2f}")
    ax.set_xlabel('x')
    ax.set_ylabel('u(x,y)')
    ax.legend()
    ax.grid(True)

ax1 = fig.add_subplot(2, 2, 3, projection='3d')
ax1.plot_surface(x_plt, y_plt, relaxation_data, cmap='plasma')
ax1.set_title(' ')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('u(x,y)')

ax2 = fig.add_subplot(2, 2, 4, projection='3d')
ax2.plot_surface(x_plt, y_plt, true_data, cmap='plasma')
ax2.set_title(' ')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('u(x,y)')

plt.show()

# %%
def seidel_relaxation_method():
    delta = 1
    iter_count = 0

    u = np.ones((n, m)) * 0.1

    while delta > EPS:
        u_next = u.copy()

        for i in range(1, n - 1):
            for j in range(1, m - 1):
                u_next[i][j] = (c / hy2 - e / (2 * hy)) * u_next[i][j - 1] + \
                               (c / hy2 + e / (2 * hy)) * u[i][j + 1] + \
                               (a / hx2 - d / (2 * hx)) * u_next[i - 1][j] + \
```

```
                                (a / hx2 + d / (2 * hx)) * u[i + 1][j] + g
                u_next[i][j] /= (2 * (a / hx2 + c / hy2) - f)
                u_next[i][j] = u_next[i][j] * omega + (1 - omega) * u[i][j]

        u_next = get_boundary(u_next)
        delta = get_delta(u, u_next)
        u = u_next
        iter_count += 1

    # print(f"Iteration counter: {iter_count}")
    return u

# %%
seidel_relaxation_data = seidel_relaxation_method().transpose()

# %%
x_plt, y_plt = np.meshgrid(x, y)
true_data = analytical(x_plt, y_plt)

y_indices = random.sample(range(len(y) - 1), k=min(3, len(y)))
y_indices.sort()

fig = plt.figure(figsize=(20, 12))

for i, y_idx in enumerate(y_indices):
    ax = fig.add_subplot(2, 3, i+1)
    ax.plot(x, seidel_relaxation_data[y_idx], label="")
    ax.plot(x, true_data[y_idx], label="")
    ax.set_title(f"y = {y[y_idx]:.2f}")
    ax.set_xlabel('x')
    ax.set_ylabel('u(x,y)')
    ax.legend()
    ax.grid(True)

ax1 = fig.add_subplot(2, 2, 3, projection='3d')
ax1.plot_surface(x_plt, y_plt, seidel_relaxation_data, cmap='plasma')
ax1.set_title(' ')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('u(x,y)')

ax2 = fig.add_subplot(2, 2, 4, projection='3d')
ax2.plot_surface(x_plt, y_plt, true_data, cmap='plasma')
ax2.set_title(' ')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('u(x,y)')

plt.show()
```

```
# %%
def compute_error(numerical_solution, analytical_solution):
    return np.sqrt(np.mean((numerical_solution - analytical_solution)**2))

fixed_n = 15
m_values = [10, 15, 20, 25, 30, 40]

errors_liebmann_hy = []
errors_seidel_hy = []
errors_relaxation_hy = []
hy_values = []


hx = Lx / (n - 1)
hy = Ly / (m - 1)

hy2 = hy**2
hx2 = hx**2

for m in m_values:
    n = fixed_n
    hy = (Ly - 0) / (m - 1)
    hx = (Lx - 0) / (n - 1)
    hx2 = hx * hx
    hy2 = hy * hy

    x = [i * hx for i in range(n - 1)]
    x.append(Lx)
    x = np.array(x)

    y = [j * hy for j in range(m - 1)]
    y.append(Ly)
    y = np.array(y)

    bound_x0 = []
    bound_xL = []

    for i in range(n):
        bound_x0.append(gamma_x0(x[i]))
        bound_xL.append(gamma_xL(x[i]))

    bound_0y = []
    bound_Ly = []

    for j in range(m):
        bound_0y.append(gamma_0y(y[j]))
        bound_Ly.append(gamma_Ly(y[j]))

```

```
    x_plt, y_plt = np.meshgrid(x, y, indexing='ij')
    true_data = analytical(x_plt, y_plt)

    liebmann_sol = liebmann_method()
    seidel_sol = seidel_method()
    relaxation_sol = seidel_relaxation_method()

    errors_liebmann_hy.append(compute_error(liebmann_sol, true_data))
    errors_seidel_hy.append(compute_error(seidel_sol, true_data))
    errors_relaxation_hy.append(compute_error(relaxation_sol, true_data))
    hy_values.append(hy)

fixed_m = 20
n_values = [10, 15, 20, 25, 30, 40]

errors_liebmann_hx = []
errors_seidel_hx = []
errors_relaxation_hx = []
hx_values = []

for n in n_values:
    m = fixed_m
    hy = (Ly - 0) / (m - 1)
    hx = (Lx - 0) / (n - 1)
    hx2 = hx * hx
    hy2 = hy * hy

    x = [i * hx for i in range(n - 1)]
    x.append(Lx)
    x = np.array(x)

    y = [j * hy for j in range(m - 1)]
    y.append(Ly)
    y = np.array(y)

    bound_x0 = []
    bound_xL = []

    for i in range(n):
        bound_x0.append(gamma_x0(x[i]))
        bound_xL.append(gamma_xL(x[i]))

    bound_0y = []
    bound_Ly = []

    for j in range(m):
        bound_0y.append(gamma_0y(y[j]))
        bound_Ly.append(gamma_Ly(y[j]))

```

```
    x_plt, y_plt = np.meshgrid(x, y, indexing='ij')
    true_data = analytical(x_plt, y_plt)

    liebmann_sol = liebmann_method()
    seidel_sol = seidel_method()
    relaxation_sol = seidel_relaxation_method()

    errors_liebmann_hx.append(compute_error(liebmann_sol, true_data))
    errors_seidel_hx.append(compute_error(seidel_sol, true_data))
    errors_relaxation_hx.append(compute_error(relaxation_sol, true_data))
    hx_values.append(hx)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

ax1.plot(hy_values, errors_liebmann_hy, 'o-', label=' ', linewidth=2)
ax1.plot(hy_values, errors_seidel_hy, 's-', label=' ', linewidth=2)
ax1.plot(hy_values, errors_relaxation_hy, '^-', label=' ', linewidth=2)
ax1.set_xlabel(' hy')
ax1.set_ylabel(' L2')
ax1.set_title(f'   hy')
ax1.legend()
ax1.grid(True, alpha=0.3)
ax1.set_yscale('log')
ax1.set_xscale('log')

ax2.plot(hx_values, errors_liebmann_hx, 'o-', label=' ', linewidth=2)
ax2.plot(hx_values, errors_seidel_hx, 's-', label=' ', linewidth=2)
ax2.plot(hx_values, errors_relaxation_hx, '^-', label=' ', linewidth=2)
ax2.set_xlabel(' hx')
ax2.set_ylabel(' L2')
ax2.set_title(f'   hx')
ax2.legend()
ax2.grid(True, alpha=0.3)
ax2.set_yscale('log')
ax2.set_xscale('log')

plt.tight_layout()
plt.show()
```

4 Выводы

В ходе выполнения лабораторной работы реализованы метод простых итераций (метод Либмана), метод Зейделя и метод простых итераций с верхней релаксацией для решения уравнени эллиптического типа. Также была реализована визуализация поверхности численного решения, а также графики срезов численного и аналитического решения вместе с графиками зависимости погрешности от шага h_x и h_y.

Список литературы

[1] Каханер Д., Моулер К., Нэш С. *Численные методы и программное обеспечение.* М.: Мир, 1998. 575 с.

[2] Golub G. H., Van Loan C. F. *Matrix Computations*. 4th ed. Baltimore: Johns Hopkins University Press, 2013. 756 p.