## Национальный исследовательский университет «МЭИ»

## (Московский Энергетический Институт)

**Кафедра математического и компьютерного моделирования**

## Численные методы

## Лабораторная работа №8:

## «ЧИСЛЕННОЕ РЕШЕНИЕ СТАЦИОНАРНЫХ ЗАДАЧ ТЕПЛОПРОВОДНОСТИ»

*Выполнил:* Солонин Е. В. А-14-19

*Преподаватель:* Амосова О. А.

Вариант 41

2021 Москва

**Постановка задачи:**

**Задача 8.1.** Найти с точностью  приближенное решение краевой задачи

 где 

### ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ

1. Построить тестовый пример, используя коэффициенты k(x) и q(x), указанные в индивидуальном варианте.
2. Cоставить разностную схему методом баланса. Выписать коэффициенты матрицы системы уравнений и коэффициенты правой части (интегралы, требуемые при построении разностной схемы, вычислить аналитически.)
3. Решение системы разностных уравнений найти прямым методом с помощью встроенной процедуры.
4. Построить на одном чертеже графики приближенного и точного решений для тестового примера и график погрешности. После проверки правильности работы программы перейти к решению основной задачи.
5. Для вычисления решения задачи с заданной точностью произвести расчет с начальным шагом  затем уменьшить шаг вдвое. Проверку достижения заданной точности производить в программе по правилу Рунге.
6. Построить графики найденного решения и графики погрешностей.
7. Используя программу, составленную в задаче, выполнить вычислительные эксперименты: найти решение задачи при следующих вариантах измененной правой части уравнения *F(x*) (*f(x)* - исходная функция правой части ):

A) ; B) 

C) F(x) – точечный источник тепла. Задать точечный источник можно следующим образом: F(x)=  где c – некоторая константа (мощность источника), - дельта функция , -точка из отрезка [a,b] в которую ставится источник. Рассмотреть различные расположения точки c на отрезке [a,b].

1. Построить на одном чертеже графики полученных приближенных решений.
2. Определить при каком варианте из A)-C) приближенное решение задачи достигает максимального и минимального значений.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| 8.1.41 | 1 | 4 |  |  |  | 8 | 6 |

**Непонятно, почему [a,b]=[1,4], когда k(x) определена на [0,3]. Беру [a,b]=[0,3]**

**Теоретический материал:**

Метод баланса: *i=1…N-1*

, .

Здесь , ,  

**Тестовый пример:**

Соблюдение граничных условий:

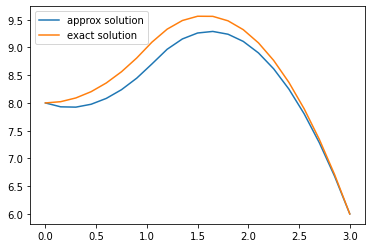
Согласование условий:

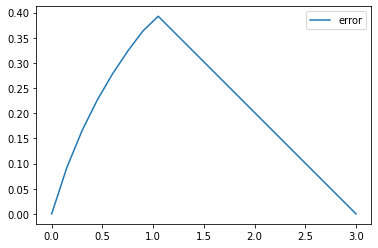
Возьмем

Возьмем уравнение

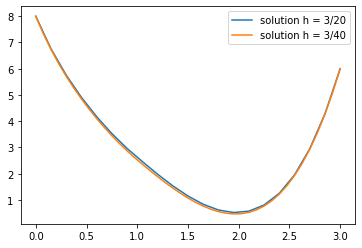
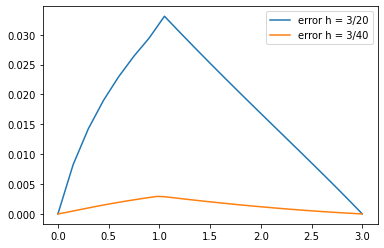
*Задача:* найти приближенное решение к функции u(x), являющейся решением задачи

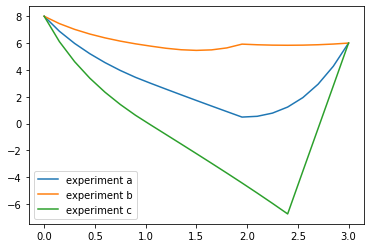
*Будем работать с шагом h=(b-a)/20*

*****Графики точного и приближенного решения:*

*График погрешности:*

**Результаты:**

Исходная задача

**Вычислительные эксперименты:**

\* В эксперименте «с» была выбрана точка и мощность источника с=-80

**Анализ результатов:**

В ходе решения задачи возникла проблема при поиске приближенного решения для тестового примера, а именно из-за того, что функция f(x) в правой части была кусочная - на отрезке, содержащем точку разрыва она была не интегрируема. Поэтому пришлось укоротить отрезок до , где -точка разрыва. Если этого не сделать, вблизи точки разрыва была колоссальная потеря точности.

Для исходной задачи точность на шаге h=3/20 не достигнута, однако на шаге h=3/40 решение найдено с заданной точностью.

Для вычислительных экспериментов максимальное значение решения достигается в граничной точке x=0 для экспериментов a и b. Минимальное – для эксперимента b в точке x=2.

Что касается эксперимета с – его учитывать при поиске экстремумов нерезонно, так как выбрав нужную мощность источника можно добиться максимального или минимального значения именно для этого эксперимента. Например, выбрав мощность с=-80 – получили минимум. Взяв с=100 – максимум.

**Код программы** *(пояснения к коду красным цветом)***:**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** scipy.linalg **as** LA  
**from** sympy **import** DiracDelta, integrate  
**from** sympy.core.symbol **import** Symbol  
  
f = **lambda** x: -8 \* x \*\* 2 + 5 \* x + 1  
q = **lambda** x: x / (x + 3)  
k = **lambda** x: 1 + 2 \* x **if** x <= x\_jump **else** 3  
k\_definite\_integral = **lambda** t: np.log(2 \* t + 1) / 2 **if** t <= x\_jump **else** t / 3  
  
*# точное решение для тестового примера*u\_test = **lambda** x: x \*\* 2 + 8 **if** x <= x\_jump **else** -1.75 \* x \*\* 2 + 5.5 \* x + 5.25  
  
*# расчет числа точек N для шага h*get\_n = **lambda** h: int(np.ceil((b - a) / h))

*# вычисление коэффициента q\_i***def** q\_i(x, h):  
 definite\_integral = **lambda** t: t - 3 \* np.log(t + 3)  
 q\_right = definite\_integral(x + h / 2)  
 q\_left = definite\_integral(x - h / 2)  
  
 **return** (q\_right - q\_left) / h  
  
  
*# вычисление коэффициента f\_i***def** f\_i(x, h, mode):  
 *# различные значения определенного интеграла,  
 # в зависимости от задачи, которая решается  
 # mode = test - тестовые пример  
 # mode = main - исходная задача  
 # mode = experiment\_{a, b, c} - вычислительный эксперимент a, b и с* **def** definite\_integral(t):  
 F = -8 / 3 \* t \*\* 3 + 5 / 2 \* t \*\* 2 + t  
 **if** mode == **"test"**:  
 f\_1 = -(4 \* t \*\* 2) - (2 \* t)  
 f\_2 = 10.5 \* t  
 **return** f\_1 **if** t <= x\_jump **else** f\_2  
 **elif** mode == **"main"**:  
 **return** F  
 **elif** mode == **"experiment\_a"**:  
 **return** F **if** t <= x\_jump **or** t >= x\_jump\_2 **else** 0  
 **elif** mode == **"experiment\_b"**:  
 **return** 0 **if** t <= x\_jump **or** t >= x\_jump\_2 **else** F  
 **else**:  
 **return** 0  
  
 **if** mode == **"experiment\_c"**:  
 c = -80 *# мощность источника* x0 = 2.4  
 z = Symbol(**"z"**)  
 f = c \* integrate(DiracDelta(z - x0), (z, x - h / 2, x + h / 2))  
 **return** f / h  
  
 f\_right = definite\_integral(x + h / 2)  
 f\_left = definite\_integral(x - h / 2)  
  
 *# если на отрезке [x[i-1/2];x[i+1/2]] есть точка разрыва функции,  
 # необходимо обработать отдельно эти случай,  
 # так как функция не интегрируема на этом отрезке* **if** x + h / 2 >= x\_jump > x - h / 2:  
 f\_middle = definite\_integral(x\_jump)  
 **return** (f\_middle - f\_left) / h  
  
 **if** mode == **"experiment\_a" or** mode == **"experiment\_b"**:  
 **if** x + h / 2 >= x\_jump\_2 > x - h / 2:  
 f\_middle = definite\_integral(x\_jump\_2)  
 **return** (f\_middle - f\_left) / h  
  
 **return** (f\_right - f\_left) / h  
  
  
*# построение трехдиагональной матрицы для системы***def** make\_matrix(x\_data, h, mode):  
 n = get\_n(h)  
 A = np.eye(n + 1)  
 **for** i **in** range(1, n):  
 k\_i\_left\_value = k(x\_data[i] - h / 2)  
 k\_i\_right\_value = k(x\_data[i] + h / 2)  
  
 A[i, i - 1] = -k\_i\_left\_value  
 A[i, i] = k\_i\_left\_value + k\_i\_right\_value  
 A[i, i] += q\_i(x\_data[i], h) \* h \*\* 2 **if** mode != **"test" else** 0  
 A[i, i + 1] = -k\_i\_right\_value  
 **return** A  
  
  
*# построение вектора правой части для системы***def** make\_vector(x\_data, h, mode):  
 n = get\_n(h)  
 B = np.ones(n + 1) \* h \*\* 2  
  
 B[0] = u\_a  
 **for** i **in** range(1, n):  
 B[i] \*= f\_i(x\_data[i], h, mode)  
 B[-1] = u\_b  
  
 **return** B  
  
  
*# найти решение задачи***def** solve(h, mode):  
 n = get\_n(h)  
 x\_data = np.linspace(a, b, n + 1)  
 A = make\_matrix(x\_data, h, mode)  
 B = make\_vector(x\_data, h, mode)  
 **return** LA.solve(A, B)  
  
  
*# построение графиков для тестового примера***def** show\_graph\_test(h, u\_data):  
 n = get\_n(h)  
 x\_data = np.linspace(a, b, n + 1)  
 u\_test\_data = [u\_test(x\_data[i]) **for** i **in** range(n + 1)]  
 r\_data = np.abs(u\_test\_data - u\_data)  
  
 plt.subplots()  
 plt.plot(x\_data, u\_data, label=**"approx solution"**)  
 plt.plot(x\_data, u\_test\_data, label=**"exact solution"**)  
 plt.legend()  
 plt.show()  
  
 plt.subplots()  
 plt.plot(x\_data, r\_data, label=**"error"**)  
 plt.legend()  
 plt.show()  
  
  
*# построение графиков для исходной задачи***def** show\_graph\_main(h, u\_data\_h, u\_data\_h2):  
 u\_data\_h4 = solve(h / 4, **"main"**)  
  
 n\_h = get\_n(h)  
 n\_h2 = get\_n(h / 2)  
  
 x\_data\_h = np.linspace(a, b, n\_h + 1)  
 x\_data\_h2 = np.linspace(a, b, n\_h2 + 1)  
  
 r\_data\_h = Runge(u\_data\_h2, u\_data\_h)  
 r\_data\_h2 = Runge(u\_data\_h4, u\_data\_h2)  
  
 plt.subplots()  
 plt.plot(x\_data\_h, u\_data\_h, label=**"solution h = 3/20"**)  
 plt.plot(x\_data\_h2, u\_data\_h2, label=**"solution h = 3/40"**)  
 plt.legend()  
 plt.show()  
  
 plt.subplots()  
 plt.plot(x\_data\_h, r\_data\_h, label=**"error h = 3/20"**)  
 plt.plot(x\_data\_h2, r\_data\_h2, label=**"error h = 3/40"**)  
 plt.legend()  
 plt.show()  
  
  
*# построение графиков для вычислительных экспериментов***def** show\_graph\_experiment(h, u\_data\_a, u\_data\_b, u\_data\_c):  
 n = get\_n(h)  
 x\_data = np.linspace(a, b, n + 1)  
  
 plt.subplots()  
 plt.plot(x\_data, u\_data\_a, label=**"experiment a"**)  
 plt.plot(x\_data, u\_data\_b, label=**"experiment b"**)  
 plt.plot(x\_data, u\_data\_c, label=**"experiment c"**)  
 plt.legend()  
 plt.show()  
  
  
*# погрешность решения по правилу Рунге***def** Runge(y\_data\_h, y\_data\_2h):  
 r\_data = [0]  
 **for** i **in** range(1, len(y\_data\_2h)):  
 r\_data.append(np.abs((y\_data\_h[2 \* i] - y\_data\_2h[i])) / (2 \*\* 2 - 1))  
 **return** r\_data  
  
  
a = 0  
x\_jump = 1  
x\_jump\_2 = 2  
b = 3  
u\_a = 8  
u\_b = 6  
epsilon = 0.01  
h = (b - a) / 20  
  
u\_data = solve(h, **"test"**)  
show\_graph\_test(h, u\_data)  
  
u\_data = solve(h, **"main"**)  
u\_data\_2 = solve(h / 2, **"main"**)  
show\_graph\_main(h, u\_data, u\_data\_2)  
  
u\_data\_a = solve(h, **"experiment\_a"**)  
u\_data\_b = solve(h, **"experiment\_b"**)  
u\_data\_c = solve(h, **"experiment\_c"**)  
show\_graph\_experiment(h, u\_data\_a, u\_data\_b, u\_data\_c)

**Постановка задачи:**

**Задача 8.2.** Решить численно краевую задачу Дирихледля уравненияПуассона в области сложной формы.

**ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ**

1. Подготовить тестовый пример для решения задачи на прямоугольнике.
2. Составить подпрограмму решения системы итерационным методом индивидуального варианта ЛР 4 весеннего семестра.
3. Решить задачу Дирихле в прямоугольнике индивидуальным методом (ЛР4) с точностью 0.001. Для оценки погрешности использовать правило Рунге.
4. Решить задачу Дирихле в области сложной формы.
5. Представить полученные результаты в виде таблиц и графиков поверхностей.

|  |  |
| --- | --- |
| 8.2.41 |  |

**Теоретический материал:**

Задача Дирихле для уравнения Пуассона в прямоугольнике:

**Тестовые примеры:**

**Пример 1**

Найти приближенное решение к функции u(x, y), являющейся решением задачи:

**Таблица** решений(шаг h = 1/9)

+---------+------+-------+-------+--------+--------+-------+-------+--------+--------+------+

| u(x, y) | x[0] | x[1] | x[2] | x[3] | x[4] | x[5] | x[6] | x[7] | x[8] | x[9] |

+---------+------+-------+-------+--------+--------+-------+-------+--------+--------+------+

| y[0] | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

| y[1] | 0.0 | 0.116 | 0.04 | -0.102 | -0.076 | 0.076 | 0.102 | -0.04 | -0.116 | 0.0 |

| y[2] | 0.0 | 0.217 | 0.076 | -0.191 | -0.142 | 0.142 | 0.191 | -0.076 | -0.217 | 0.0 |

| y[3] | 0.0 | 0.293 | 0.102 | -0.258 | -0.191 | 0.191 | 0.258 | -0.102 | -0.293 | 0.0 |

| y[4] | 0.0 | 0.333 | 0.116 | -0.293 | -0.217 | 0.217 | 0.293 | -0.116 | -0.333 | 0.0 |

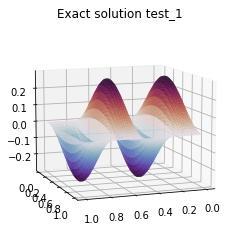
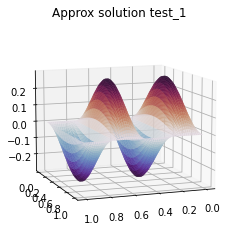
| y[5] | 0.0 | 0.333 | 0.116 | -0.293 | -0.217 | 0.217 | 0.293 | -0.116 | -0.333 | 0.0 |

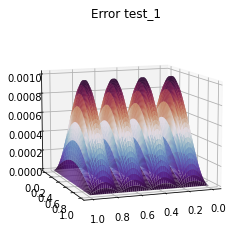
| y[6] | 0.0 | 0.293 | 0.102 | -0.258 | -0.191 | 0.191 | 0.258 | -0.102 | -0.293 | 0.0 |

| y[7] | 0.0 | 0.217 | 0.076 | -0.191 | -0.142 | 0.142 | 0.191 | -0.076 | -0.217 | 0.0 |

| y[8] | 0.0 | 0.116 | 0.04 | -0.102 | -0.076 | 0.076 | 0.102 | -0.04 | -0.116 | 0.0 |

| y[9] | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

****Графики** точного, приближенного решения, а также график погрешности

**

*Для достижения точности использовался шаг 1/60*

**Пример 2**

Найти приближенное решение к функции u(x, y), являющейся решением задачи:

**Таблица** решений(шаг h = 1/9)

+---------+------+-------+-------+-------+-------+-------+-------+-------+-------+------+

| u(x, y) | x[0] | x[1] | x[2] | x[3] | x[4] | x[5] | x[6] | x[7] | x[8] | x[9] |

+---------+------+-------+-------+-------+-------+-------+-------+-------+-------+------+

| y[0] | 0.0 | 0.342 | 0.643 | 0.866 | 0.985 | 0.985 | 0.866 | 0.643 | 0.342 | 0.0 |

| y[1] | 0.0 | 0.307 | 0.576 | 0.777 | 0.883 | 0.883 | 0.777 | 0.576 | 0.307 | 0.0 |

| y[2] | 0.0 | 0.275 | 0.517 | 0.696 | 0.792 | 0.792 | 0.696 | 0.517 | 0.275 | 0.0 |

| y[3] | 0.0 | 0.246 | 0.463 | 0.624 | 0.709 | 0.709 | 0.624 | 0.463 | 0.246 | 0.0 |

| y[4] | 0.0 | 0.221 | 0.415 | 0.559 | 0.635 | 0.635 | 0.559 | 0.415 | 0.221 | 0.0 |

| y[5] | 0.0 | 0.197 | 0.371 | 0.5 | 0.569 | 0.569 | 0.5 | 0.371 | 0.197 | 0.0 |

| y[6] | 0.0 | 0.177 | 0.332 | 0.447 | 0.509 | 0.509 | 0.447 | 0.332 | 0.177 | 0.0 |

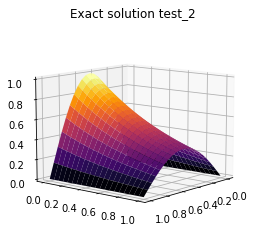
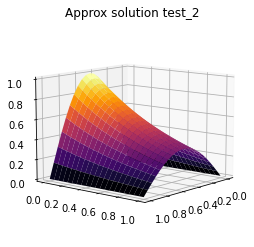
| y[7] | 0.0 | 0.158 | 0.297 | 0.4 | 0.455 | 0.455 | 0.4 | 0.297 | 0.158 | 0.0 |

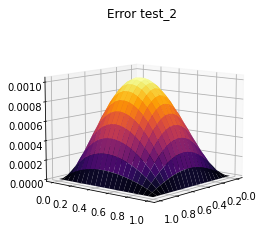
| y[8] | 0.0 | 0.141 | 0.265 | 0.357 | 0.406 | 0.406 | 0.357 | 0.265 | 0.141 | 0.0 |

| y[9] | 0.0 | 0.126 | 0.236 | 0.319 | 0.362 | 0.362 | 0.319 | 0.236 | 0.126 | 0.0 |

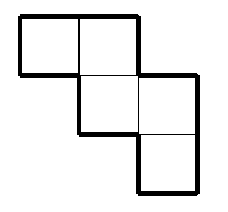
+---------+------+-------+-------+-------+-------+-------+-------+-------+-------+------+

**Графики** точного, приближенного решения, а также график погрешности





Для достижения точности использовался шаг 1/20

**Основная задача:**

+---------+------+-------+-------+-------+-------+-------+-------+-------+------+

| u(x, y) | x[0] | x[1] | x[2] | x[3] | x[4] | x[5] | x[6] | x[7] | x[8] |

+---------+------+-------+-------+-------+-------+-------+-------+-------+------+

| y[0] | 0.0 | 0.383 | 0.707 | 0.924 | 1.0 | 0.924 | 0.0 | 0.0 | 0.0 |

| y[1] | 0.0 | 0.339 | 0.626 | 0.818 | 0.885 | 0.818 | 0.0 | 0.0 | 0.0 |

| y[2] | 0.0 | 0.3 | 0.554 | 0.723 | 0.783 | 0.723 | 0.0 | 0.0 | 0.0 |

| y[3] | 0.0 | 0.0 | 0.0 | 0.639 | 0.692 | 0.639 | 0.489 | 0.265 | 0.0 |

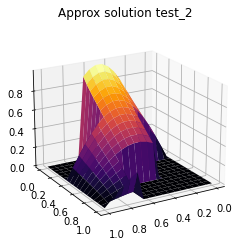
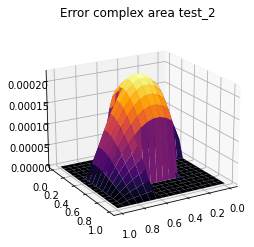
| y[4] | 0.0 | 0.0 | 0.0 | 0.565 | 0.611 | 0.565 | 0.432 | 0.234 | 0.0 |

| y[5] | 0.0 | 0.0 | 0.0 | 0.498 | 0.54 | 0.498 | 0.382 | 0.206 | 0.0 |

| y[6] | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.336 | 0.182 | 0.0 |

| y[7] | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.296 | 0.16 | 0.0 |

| y[8] | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.26 | 0.141 | 0.0 |

+---------+------+-------+-------+-------+-------+-------+-------+-------+------+

**Код программы** *(пояснения к коду красным цветом)***:**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** scipy.linalg **as** LA  
**from** prettytable **import** PrettyTable  
  
*# расчет числа точек N для шага h*get\_n = **lambda** h: int(np.ceil((b - a) / h))  
  
*# расчет размерности системы для шага h*get\_dimensions = **lambda** h: (get\_n(h) - 1) \*\* 2  
  
*# функция правой части***def** f(x, y, mode):  
 **if** mode == **"test\_1"**:  
 **return** -5 \* np.pi \*\* 2 \* np.sin(np.pi \* x \* 4) \* np.sin(np.pi \* y)  
 **if** mode == **"test\_2"**:  
 **return** np.e \*\* -y \* np.sin(np.pi \* x) \* (1 - np.pi \*\* 2)  
 **return** 0  
  
  
*# решение тестового примера***def** u(x, y, mode):  
 **if** mode == **"test\_1"**:  
 **return** 5 / 17 \* np.sin(np.pi \* x \* 4) \* np.sin(np.pi \* y)  
 **if** mode == **"test\_2"**:  
 **return** np.e \*\* -y \* np.sin(np.pi \* x)  
 **return** 0  
  
  
*# значения на границе***def** u\_boundary(x, y, mode, h=0):  
 **if** mode == **"test\_1"**:  
 **return** 0  
 **if** mode == **"test\_2"**:  
 **if** y - h == a:  
 **return** np.sin(np.pi \* x)  
 **if** y + h == b:  
 **return** np.e \*\* -1 \* np.sin(np.pi \* x)  
 **return** 0  
  
  
*# построение пятидиагональной матрицы для прямоугольной области***def** make\_matrix\_test(h):  
 n = get\_n(h)  
 dimension = get\_dimensions(h)  
 A = -2 \* np.eye(dimension)  
 **for** i **in** range(dimension):  
 **if** i + n - 1 < dimension:  
 A[i + n - 1, i] = 1  
  
 **if** i % (n - 1) != 0 **and** i != 0:  
 A[i, i - 1] = 1  
 **return** A + A.T

*# построение вектора правой части для системы***def** make\_vector(h, mode):  
 n = get\_n(h)  
 B = []  
  
 **for** i **in** range(1, n):  
 **for** j **in** range(1, n):  
 x = j \* h  
 y = i \* h  
 res = f(x, y, mode) \* h \*\* 2 - u\_boundary(x, y, mode, h)  
 B.append(res)  
  
 **return** B  
  
  
*# метод Зейделя***def** Gauss\_Seidel(A, B):  
 n = len(A)  
 x = np.ones(n)  
 err = 1  
 **while** err >= epsilon:  
 x\_new = np.copy(x)  
 **for** i **in** range(len(A)):  
 s1 = sum(A[i][j] \* x\_new[j] **for** j **in** range(i))  
 s2 = sum(A[i][j] \* x[j] **for** j **in** range(i + 1, n))  
 x\_new[i] = (B[i] - s1 - s2) / A[i][i]  
 err = sum(abs(x\_new[i] - x[i]) **for** i **in** range(n))  
 x = x\_new  
 **return** x  
  
  
*# найти решение задачи***def** solve(h, mode):  
 dimension = int(np.sqrt(get\_dimensions(h)))  
  
 A = make\_matrix\_test(h)  
 B = make\_vector(h, mode)  
  
 *# solution = LA.solve(A, B)* solution = Gauss\_Seidel(A, B)  
  
 reshaped = np.reshape(solution, (dimension, dimension))  
 u\_data = add\_boundary(reshaped, mode)  
 **return** u\_data  
  
  
*# вырезать лишнюю область из решения***def** cut\_area(u\_data):  
 n = len(u\_data)  
 sq\_1 = n / 3 - 1  
 sq\_2 = n - sq\_1 - 1  
  
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 **if** 0 <= i <= sq\_1 **and** sq\_2 <= j <= n:  
 u\_data[i, j] = 0  
 **if** sq\_1 < i <= n **and** 0 <= j <= sq\_1:  
 u\_data[i, j] = 0  
 **if** sq\_2 <= i <= n **and** 0 <= j < sq\_2:  
 u\_data[i, j] = 0  
  
 **return** u\_data  
  
  
*# добавление к матрице решений на границах***def** add\_boundary(solution, mode):  
 dim = len(solution) + 2  
 x = np.linspace(a, b, dim)  
  
 *# четыре вектора - решения на границе* left = np.array([u\_boundary(a, x[i], mode) **for** i **in** range(dim)]).reshape(-1, 1)  
 right = np.array([u\_boundary(b, x[i], mode) **for** i **in** range(dim)]).reshape(-1, 1)  
 up = [[u\_boundary(x[i], a, mode) **for** i **in** range(dim)]]  
 down = [[u\_boundary(x[i], b, mode) **for** i **in** range(dim)]]  
  
 left = left[1:-1, :]  
 right = right[1:-1, :]  
  
 solution = np.concatenate((left, solution, right), axis=1)  
 solution = np.concatenate((up, solution, down), axis=0)  
  
 **return** solution  
  
  
*# построение графика***def** show\_graph(h, u\_data, mode, label):  
  
 n\_h = get\_n(h)  
  
 x = np.linspace(a, b, n\_h + 1)  
 y = np.linspace(a, b, n\_h + 1)  
 X, Y = np.meshgrid(x, y)  
  
 Z = []  
 **if** label == **"Exact solution"**:  
 Z = u(X, Y, mode)  
 **elif** label == **"Approx solution"**:  
 Z = u\_data  
 **elif** label == **"Error"**:  
 Z = np.abs(u\_data - u(X, Y, mode))  
 **elif** label == **"Error complex area"**:  
 Z = u\_data  
  
 plt.figure()  
 ax = plt.axes(projection=**"3d"**)  
 ax.set\_title(label + **" "** + mode)  
 ax.view\_init(10, 70)  
 ax.plot\_surface(X, Y, Z, rstride=1, cstride=1, cmap=**"inferno"**)  
 *# inferno # twilight\_shifted* plt.show()  
  
  
*# вывод таблицы результатов***def** print\_table(u\_test):  
 table = PrettyTable()  
  
 field\_names = np.array([**"x["** + str(i) + **"]" for** i **in** range(len(u\_test))])  
 field\_names = np.concatenate(([**"u(x, y)"**], field\_names), axis=0)  
 table.field\_names = field\_names  
  
 **for** row **in** range(len(u\_test)):  
 label = **"y["** + str(row) + **"]"** data = np.round(u\_test[row], 3)  
 table.add\_row([label, \*data])  
 print(table)  
  
  
*# погрешность решения по правилу Рунге***def** Runge(y\_h, y\_2h, h):  
 n = len(y\_2h)  
 r\_data = np.zeros(n)  
  
 **for** i **in** range(1, n - 1):  
 **for** j **in** range(1, n - 1):  
 r\_data = np.append(  
 r\_data, np.abs((y\_h[2 \* i, 2 \* j] - y\_2h[i, j])) / (2 \*\* 2 - 1)  
 )  
 r\_data = np.append(r\_data, [0, 0])  
 r\_data = np.append(r\_data, np.zeros(n))  
  
 **if** max(r\_data) > epsilon:  
 h \*= 3 / 4  
 u\_main = cut\_area(solve(h, **"test\_2"**))  
 u\_main\_runge = cut\_area(solve(h / 2, **"test\_2"**))  
 **return** Runge(u\_main\_runge, u\_main, h)  
  
 show\_graph(h, y\_2h, **"test\_2"**, **"Approx solution"**)  
 **return** np.reshape(r\_data, (n, n)), h  
  
  
a = 0  
b = 1  
h = 1 / 4  
epsilon = 0.001  
  
  
u\_test\_1 = solve(h, **"test\_1"**)  
show\_graph(h, 0, **"test\_1"**, **"Exact solution"**)  
show\_graph(h, u\_test\_1, **"test\_1"**, **"Approx solution"**)  
show\_graph(h, u\_test\_1, **"test\_1"**, **"Error"**)  
print\_table(u\_test\_1)  
  
u\_test\_2 = solve(h, **"test\_2"**)  
show\_graph(h, 0, **"test\_2"**, **"Exact solution"**)  
show\_graph(h, u\_test\_2, **"test\_2"**, **"Approx solution"**)  
show\_graph(h, u\_test\_2, **"test\_2"**, **"Error"**)  
print\_table(u\_test\_2)  
  
  
u\_main = cut\_area(u\_test\_2)  
u\_main\_runge = cut\_area(solve(h / 2, **"test\_2"**))  
r\_main, h = Runge(u\_main\_runge, u\_main, h)  
show\_graph(h, r\_main, **"test\_2"**, **"Error complex area"**)  
print\_table(u\_main)