Московский авиационный институт

(Национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №7

«ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ»

Вариант 3

Выполнил: Дондоков В.И.

Группа: М8О-409Б-20

Проверил: Пивоваров Д.Е.

Дата:

Оценка:

Задание: 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 $u\ (x,\ t)$. Исследовать зависимость погрешности от сеточных параметров τ и h.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0,$$

$$u(0, y) = \cos y,$$

$$u(1, y) = e\cos y,$$

$$u_y(x, 0) = 0,$$

$$u_y(x, \frac{\pi}{2}) = -\exp(x).$$

Аналитическое решение: $U(x, y) = \exp(x)\cos y$.

Теоретическая часть:

Конечно-разностная схема

Будем решать задачу на заданном промежутке от 0 до l_x по координате x и на промежутке от 0 до l_y по координате y.

Рассмотрим конечно-разностную схему решения краевой задачи на сетке с граничными парамерами l_x, l_y и параметрами насыщенности сетки N_x, N_y. Тогда размер шага по каждой из координат определяется:

$$h_x = \frac{l_x}{N_x - 1},\ h_y = \frac{N_y}{N_y - 1}$$

Попробуем определить связь между дискретными значениями функции путем разностной апроксимации производной:

$$\frac{\partial^2 u}{\partial x^2}(x_j, y_i) + \frac{\partial^2 u}{\partial y^2}(x_j, y_i) = \frac{u_{j-1,i} - 2u_{j,i} + u_{j+1,i}}{h_x^2} + \frac{u_{j,i-1} - 2u_{j,i} + u_{j,i+1}}{h_y^2}$$

Тогда выражая из искомого уравнения значение $u_{i,j} = \dfrac{h_y^2(u_{j-1,i} + u_{j+1,i}) + h_x^2(u_{j,i-1} + u_{j,i+1})}{2(h_x^2 + h_y^2)}$, мы получаем основу для применения иттерационных методов решения *СЛАУ*.

Для расчета $u_{j,0}$ и $u_{0,i}$ следует использовать граничные условия.

Начальная инициализация

Поскольку в нашем варианте известны граничные значения $u(x, l_{y0})$ и $u(x, l_{y1})$, то для начальной инициаизации значений в сетке можно использовать линейную интерполяцию при фиксированном $x = x_j$ для улучшения сходимости:

$$u_{j,i} = \frac{u(x_j, l_{y1}) - u(x_j, l_{y0})}{l_{y1} - l_{y0}} \cdot (y_i - l_{y0}) + u(x_j, l_{y0})$$

Граничные значения

Для границ по y координате значения заданы явно граничным условием, и мы можем определить их на начальном этапе при инициализации.

Для границ по x координате аппроксимируем значение производной из граничного условия с помощью трёхточечной аппроксимации в точках $x = 0$ и $x = l$ и получаем 2 новых уравнения в *СЛАУ* соответственно:

$$\frac{-3u_{0,i} + 4u_{1,i} - u_{2,i}}{2h_x} = \phi_0(y_i)$$

$$\frac{3u_{N,i} - 4u_{N-1,i} + u_{N-2,i}}{2h_x} = \phi_1(y_i)$$

Тогда основа для иттерационного метода:

$$u_{0,i} = \frac{-2h_x\phi_0(y_i) + 4u_{1,i} - u_{2,i}}{3}$$

$$u_{N,i} = \frac{2h_x\phi_1(y_i) + 4u_{N-1,i} - u_{N-2,i}}{3}$$

Методы решения СЛАУ

Для решения СЛАУ можно воспользоваться иттерационными методами, такими как *метод простых иттераций*, *метод Зейделя* и *метод верхних релаксаций*. Первые два метода были изучены нами ранее, когда как последний является небольшой модификацией *метода Зейделя* с добавлением параметра *w*, который позволяет регулировать скорость сходимости метода.

Код программы:

```
def ux0(y):
    return np.cos(y)


def uxl(y):
    return np.e * np.cos(y)


def uy0(x):
    return 0


def uyl(x):
    return -np.exp(x)


def U(x, y):
    return np.exp(x) * np.cos(y)


X_MAX = 1
Y_MAX = np.pi / 2
MAX_ITER = 10000


def simple_iter(hx, hy, eps, verbose=False):
    x = np.arange(0, X_MAX + hx, hx)
    y = np.arange(0, Y_MAX + hy, hy)
    cur = np.zeros((x.size, y.size))
    cur[0] = ux0(y)
    cur[-1] = uxl(y)
    for j in range(y.size):
        for i in range(1, x.size - 1):
            cur[i][j] = cur[i][0] + (cur[i][-1] - cur[i][0]) / (x[-1] - x[0])
* (x[i] - x[0])
```

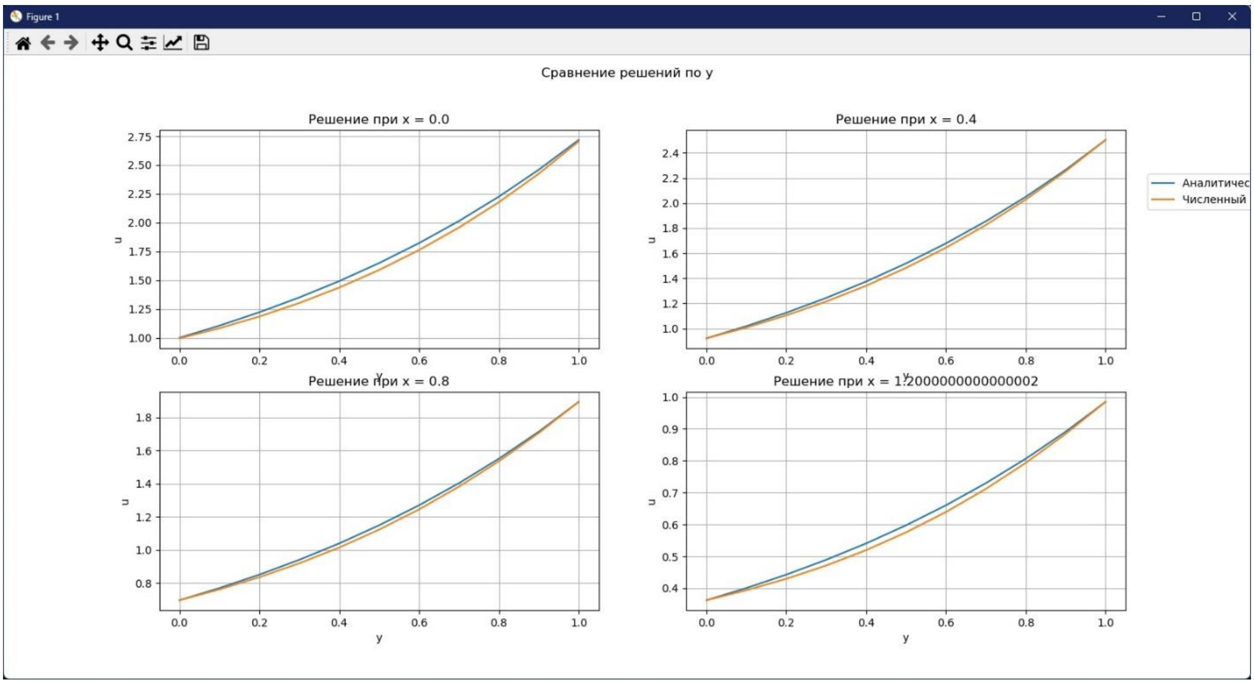
```
    norms = []
    for it in range(MAX_ITER):
        prev = cur.copy()
        for i in range(1, x.size - 1):
            for j in range(1, y.size - 1):
                cur[i][j] = (hx ** 2 * (prev[i - 1][j] + prev[i + 1][j]) +
                             hy ** 2 * (prev[i][j - 1] + prev[i][j + 1])) /
(2 * (hx ** 2 + hy ** 2))
        cur[:, 0] = cur[:, 1] - hy * uy0(x)
        cur[:, -1] = cur[:, -2] + hy * uyl(x)
        norm = np.linalg.norm(cur - prev, np.inf)
        norms.append(norm)
        if verbose:
            print('iter', it, 'norma', norm)
        if (norm <= eps):
            break
    return cur, np.array(norms)

def relax_method(hx, hy, eps, w=1.8, verbose=False):
    x = np.arange(0, X_MAX + hx, hx)
    y = np.arange(0, Y_MAX + hy, hy)
    cur = np.zeros((x.size, y.size))
    cur[0] = ux0(y)
    cur[-1] = uxl(y)
    for j in range(y.size):
        for i in range(1, x.size-1):
            cur[i][j] = cur[i][0] + (cur[i][-1] - cur[i][0]) / (x[-1] - x[0])
* (x[i] - x[0])
    norms = []
    for it in range(MAX_ITER):
        prev = cur.copy()
        for i in range(1, x.size - 1):
            for j in range(1, y.size - 1):
                cur[i][j] = (hx**2 * (cur[i-1][j] + prev[i+1][j]) + hy**2 *
(cur[i][j-1] + prev[i][j+1])) / (2 * (hx**2 + hy**2))
                cur[i][j] *= w
                cur[i][j] += (1-w) * prev[i][j]
        cur[:,0] = cur[:,1] - hy * uy0(x)
        cur[:,-1] = cur[:,-2] + hy * uyl(x)
        norm = np.linalg.norm(cur - prev, np.inf)
        norms.append(norm)
        if verbose:
            print('iter', it, 'norma', norm)
        if (norm <= eps):
            break
    return cur, np.array(norms)

def zeidel_method(hx, hy, eps, verbose=False):
    return relax_method(hx, hy, eps, 1, verbose)

def analytic(hx, hy):
    x = np.arange(0, X_MAX + hx, hx)
    y = np.arange(0, Y_MAX + hy, hy)
    u = np.zeros((x.size, y.size))
    for i in range(x.size):
        for j in range(y.size):
            u[i][j] = U(x[i], y[j])
    return u

solvers = {
    'simple_iter': simple_iter,
    'relax': relax_method,
    'zeidel': zeidel_method
```

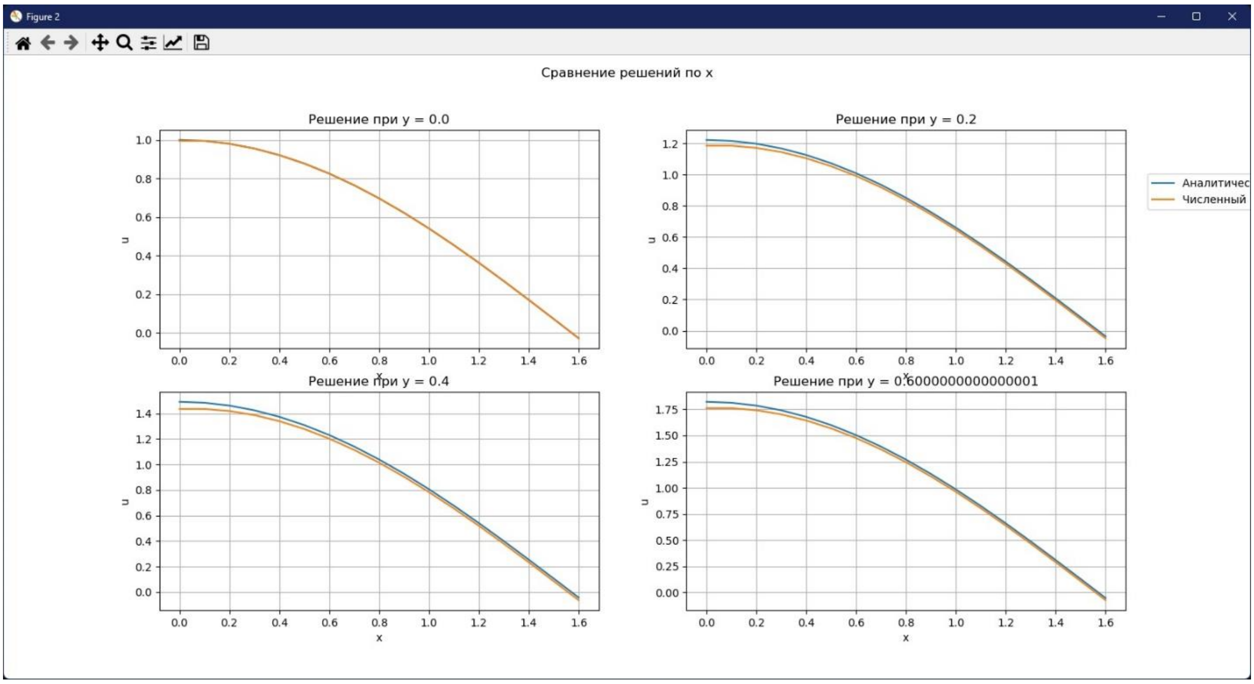
```
}


def plot_solutions(x, y, sol, u):
    n = 2
    m = 2
    x_step = x.size // (n * m)
    y_step = y.size // (n * m)
    p_x = [k for k in range(0, x.size-1, x_step)]
    p_y = [k for k in range(0, y.size-1, y_step)]
    fig, ax = plt.subplots(n, m)
    fig.suptitle('Сравнение решений по y')
    fig.set_figheight(8)
    fig.set_figwidth(16)
    k = 0
    for i in range(n):
        for j in range(m):
            ax[i][j].set_title(f'Решение при x = {y[p_y[k]]}')
            ax[i][j].plot(x, sol[:,p_y[k]], label='Аналитическое решение')
            ax[i][j].plot(x, u[:,p_y[k]], label='Численный метод')
            ax[i][j].grid(True)
            ax[i][j].set_xlabel('y')
            ax[i][j].set_ylabel('u')
            k += 1
    plt.legend(bbox_to_anchor=(1.05, 2), loc='upper left', borderaxespad=0.)
    fig, ax = plt.subplots(n, m)
    fig.suptitle('Сравнение решений по x')
    fig.set_figheight(8)
    fig.set_figwidth(16)
    k = 0
    for i in range(n):
        for j in range(m):
            ax[i][j].set_title(f'Решение при y = {x[p_x[k]]}')
            ax[i][j].plot(y, sol[p_x[k]], label='Аналитическое решение')
            ax[i][j].plot(y, u[p_x[k]], label='Численный метод')
            ax[i][j].grid(True)
            ax[i][j].set_xlabel('x')
            ax[i][j].set_ylabel('u')
            k += 1
    plt.legend(bbox_to_anchor=(1.05, 2), loc='upper left', borderaxespad=0.)


def plot_norm(norms):
    fig, ax = plt.subplots()
    fig.set_figwidth(16)
    fig.suptitle('Изменение нормы от итерации')
    ax.plot(np.arange(norms.size), norms)
    ax.grid(True)
    ax.set_xlabel('Итерация')
    ax.set_ylabel('Норма')


def plot_errors(x, y, sol, u):
    x_error = np.zeros(x.size)
    y_error = np.zeros(y.size)
    for i in range(x.size):
        x_error[i] = np.max(abs(sol[i] - u[i]))
    for i in range(y.size):
        y_error[i] = np.max(abs(sol[:,i] - u[:,i]))
    fig, ax = plt.subplots(1, 2)
    fig.set_figheight(4)
    fig.set_figwidth(16)
    ax[0].plot(x, x_error)
    ax[0].grid(True)
```

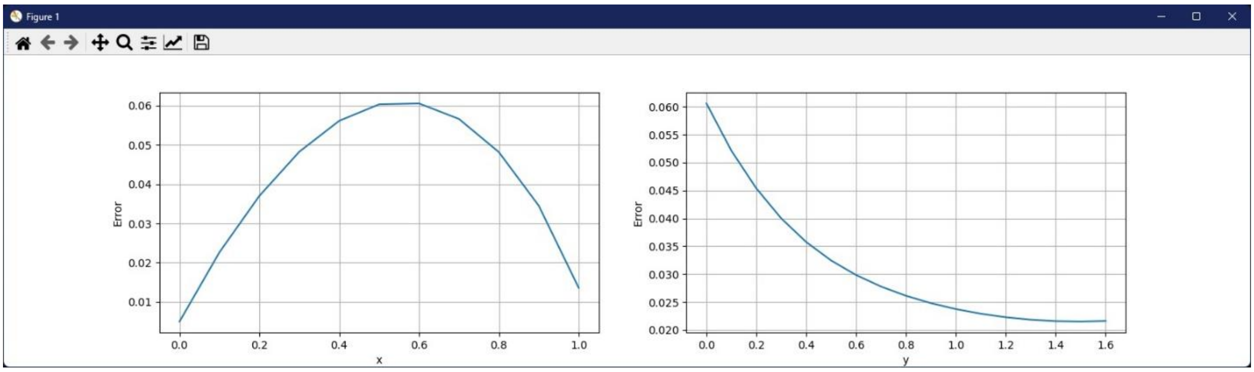
```
    ax[0].set_xlabel('x')
    ax[0].set_ylabel('Error')
    ax[1].plot(y, y_error)
    ax[1].grid(True)
    ax[1].set_xlabel('y')
    ax[1].set_ylabel('Error')


def visualize(method: str, hx: float, hy: float, eps: float):
    x = np.arange(0, X_MAX + hx, hx)
    y = np.arange(0, Y_MAX + hy, hy)
    sol = analytic(hx, hy)
    u, norms = solvers[method](hx, hy, eps)
    print('Iter count', norms.size)
    print('Norma', norms[-1])
    plot_solutions(x, y, sol, u)
    plot_errors(x, y, sol, u)
    plot_norm(norms)

visualize('simple_iter', 0.1, 0.1, 0.01)

visualize('zeidel', 0.1, 0.1, 0.01)

visualize('relax', 0.1, 0.1, 0.01)
```

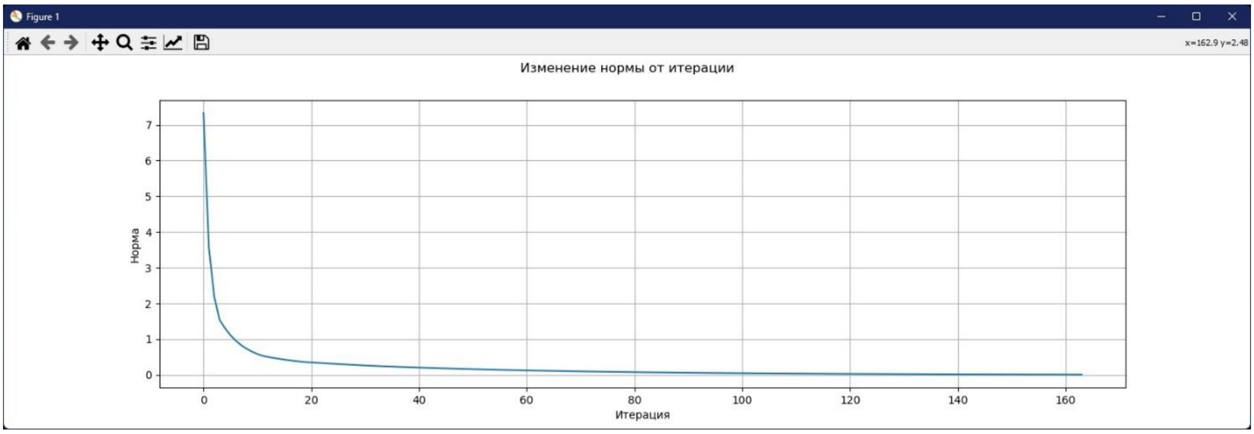
Результат:

Метод простых итераций









Метод Зейделя

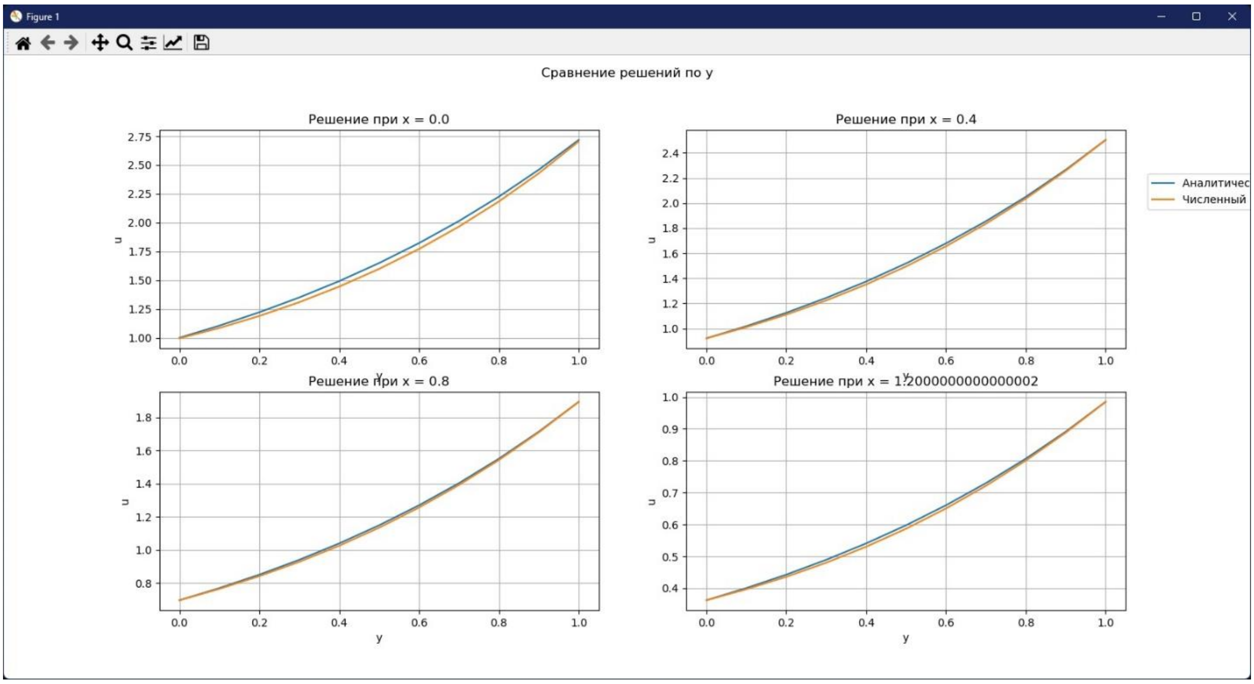


Figure 1
Сравнение решений по y
Решение при x = 0.0
Решение при x = 0.4
Решение при x = 0.8
Решение при x = 1.2000000000000002
Аналитичес
Численный
u
y

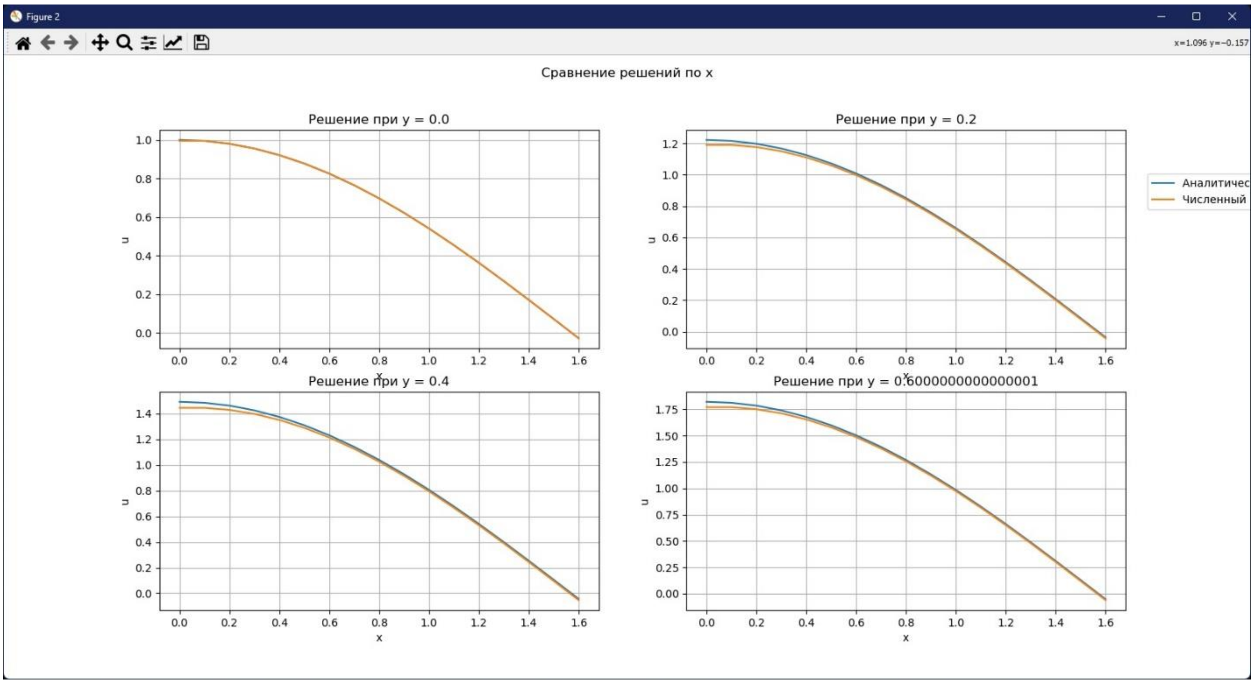


Figure 2
x=1.096 y=-0.157
Сравнение решений по x
Решение при y = 0.0
Решение при y = 0.2
Решение при y = 0.4
Решение при y = 0.6000000000000001
Аналитичес
Численный
u
x

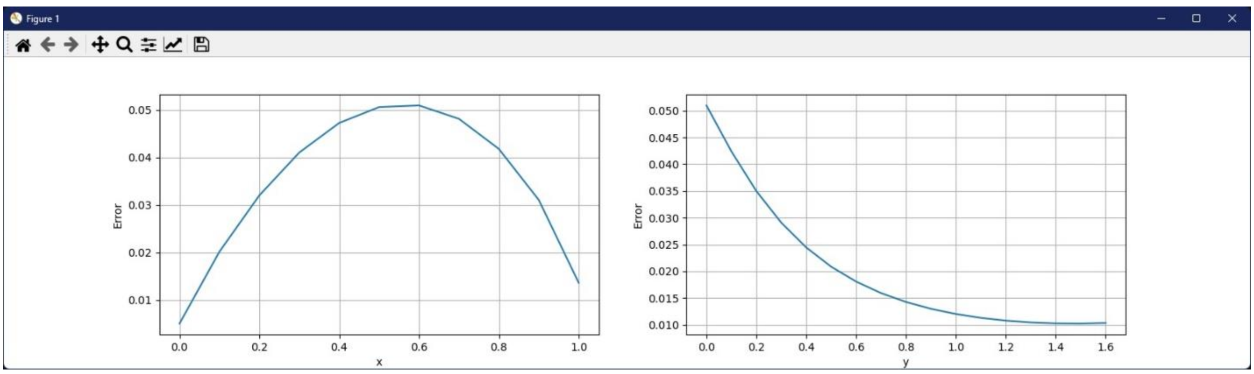
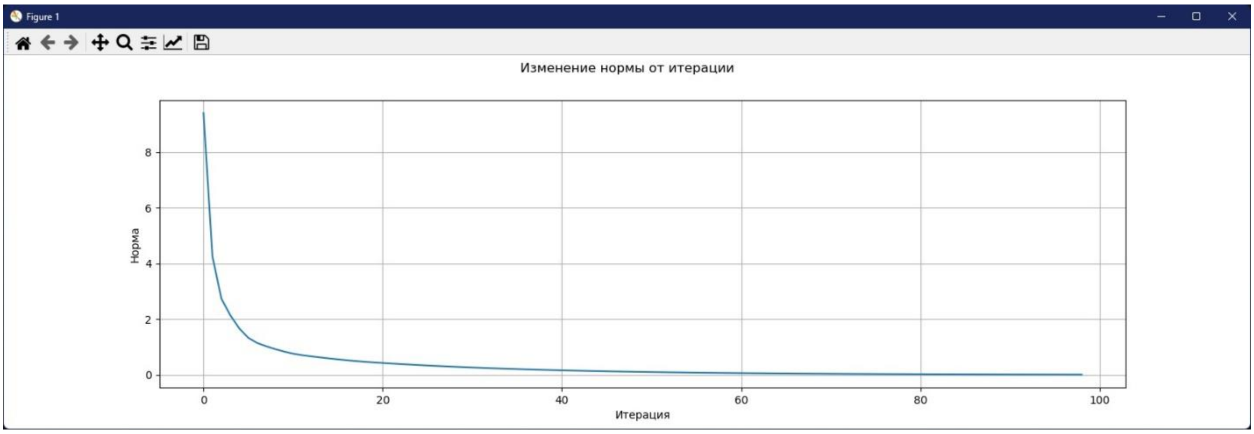
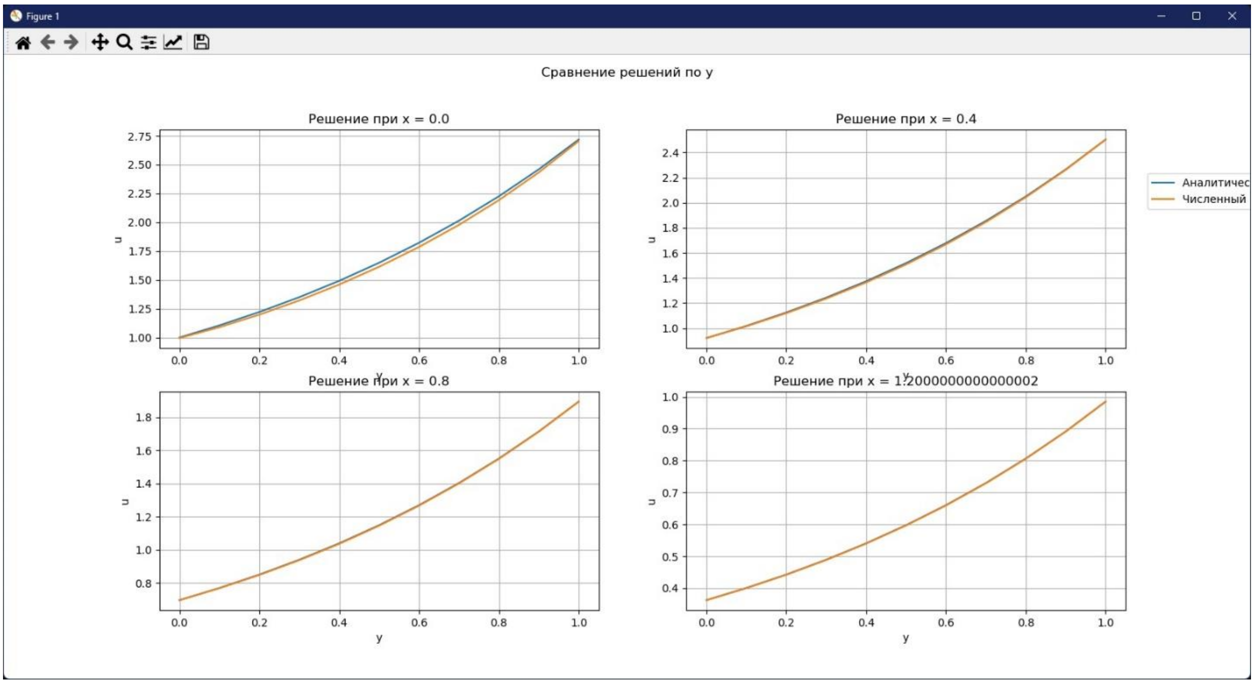
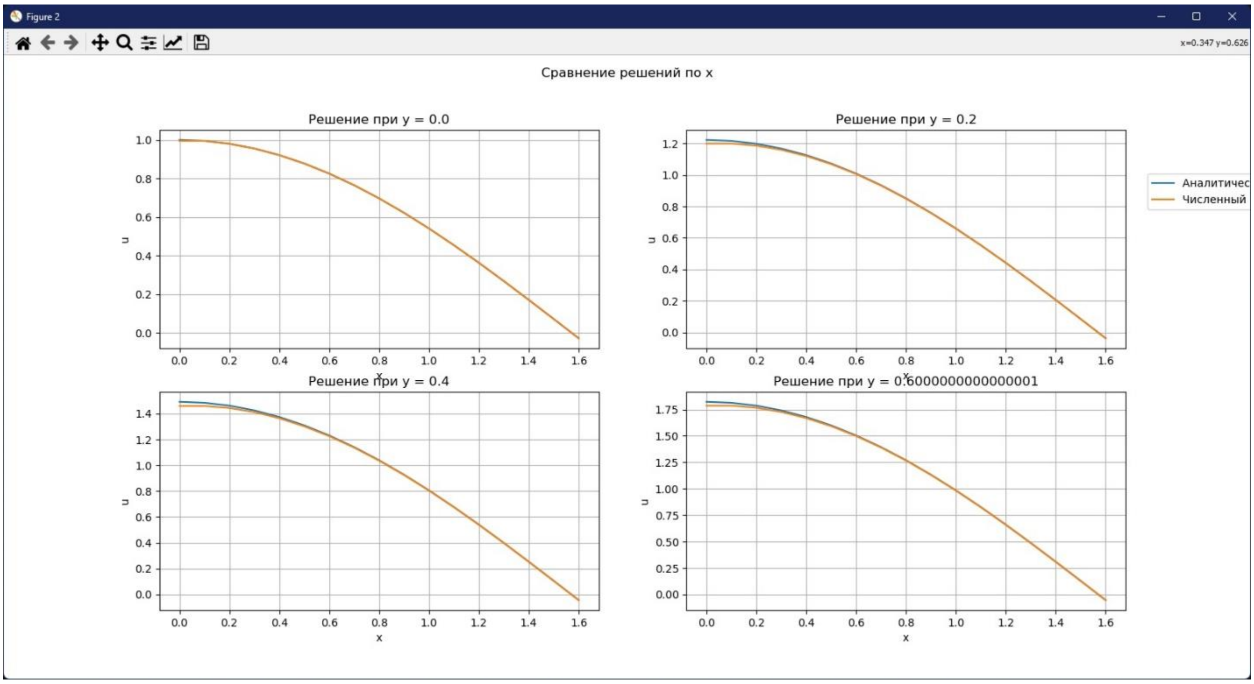


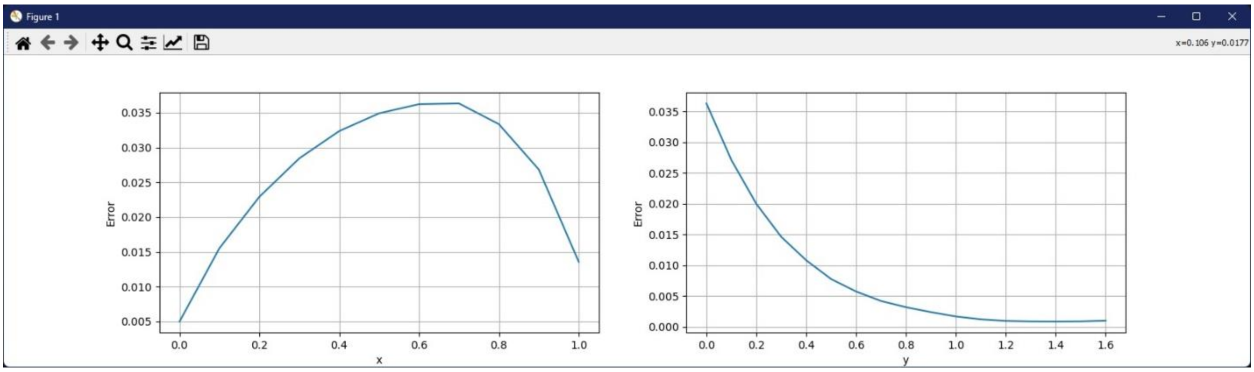
Figure 1
Error
x
y

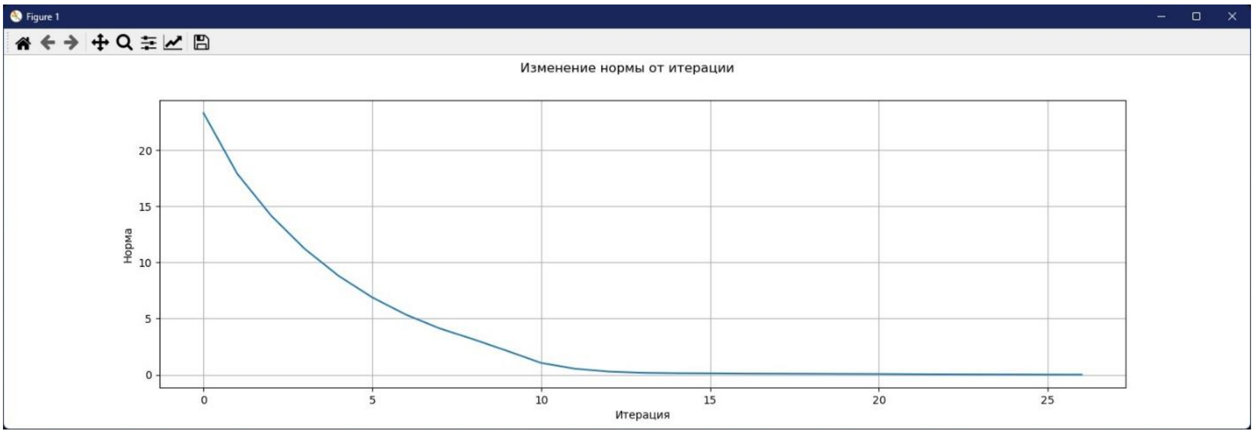


Метод *простых итераций с верхней релаксацией*









Вывод:

В ходе лабораторной работы решена краевая задачу для дифференциального уравнения эллиптического типа. Аппроксимация уравнения произведена с использованием центрально-разностной схемы. Для решения дискретного аналога применены следующие методы: *метод простых итераций (метод Либмана)*, *метод Зейделя*, *метод простых итераций с верхней релаксацией*. Вычислена погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $u\ (x,\ t)$. Исследована зависимость погрешности от сеточных параметров τ и h.