Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 805 «Математическая кибернетика»
Дисциплина: «Численные методы»

Лабораторная работа №3
Тема: Решение начально-краевой задачи для дифференциального уравнения эллиптического типа

Студент:	Хахин Максим
Группа:	80-403
Преподаватель:	Иванов И. Э.
Дата:	
Оценка:	

Москва, 2021

1. **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. Исследовать зависимость погрешности от сеточных параметров.

2. **Вариант 9:**

9.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2\frac{\partial u}{\partial y} - 3u,$$

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

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

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

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

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

3. **Теория:**
   Рассмотрим уравнение Пуассона, которое является классическим примером эллиптического уравнения:

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

Или уравнение Лапласа при $f(x, y) = 0$. Первая краевая задача для уравнения Лапласа или Пуассона называется задачей Дирихле:

$$\begin{cases} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y),\ (x, y) \in \Omega; \\ u(x, y)|_1 = \phi(x, y),\ (x, y) \in \Gamma. \end{cases}$$

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

$$\begin{cases} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y),\ (x, y) \in \Omega; \\ \frac{\partial u(x,y)}{\partial n}\Big|_\Gamma = \phi(x, y),\ (x, y) \in \Gamma. \end{cases}$$

При этом n – направление внешней к границе Γ нормали.
Для решения – строим сетку по x и y. И на ней аппроксимируем задачу во внутренних узлах с помощью отношения конечных разностей по следующей схеме:

$$\frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{h_x^2} + \frac{u_i^{j+1} - 2u_i^j + u_i^{j-1}}{h_y^2} + O(h_x^2 + h_y^2) = f(x_i, y_j)$$

$i = 1...N_x - 1,\quad j = 1...N_y - 1$ В результате получаем СЛАУ, которую можно решить разными итерационными методами.

1)Метод Либмана:

$$\frac{u_{i-1j}^k - 2u_{ij}^{k+1} + u_{i+1j}^k}{h_x^2} + \frac{u_{ij-1}^k - 2u_{ij}^{k+1} + u_{ij+1}^k}{h_y^2} = -2\frac{u_{ij+1}^k - u_{ij-1}^k}{2h_y} - 3u_{ij}^k$$

$$-u_{ij}^{k+1}\left(\frac{2}{h_x^2} + \frac{2}{h_y^2}\right) = -\frac{h_y^2(u_{i-1j}^k + u_{i+1j}^k) - h_x^2(u_{ij-1}^k + u_{ij+1}^k) - h_y h_x^2(u_{ij+1}^k - u_{ij-1}^k) - 3h_x^2 h_y^2 u_{ij}^k}{h_x^2 h_y^2}$$

$$u_{ij}^{k+1} = \frac{(3u_{ij}^k h_x^2 - u_{i-1j}^k + u_{i+1j}^k)h_y^2 + u_{ij-1}^k + u_{ij+1}^k h_x^2(h_y + 1)}{2h_y^2 + 2h_x^2}$$

2)Метод Зейделя:

$$\frac{u_{i-1j}^{k+1} - 2u_{ij}^{k+1} + u_{i+1j}^k}{h_x^2} + \frac{u_{ij-1}^{k+1} - 2u_{ij}^{k+1} + u_{ij+1}^k}{h_y^2} = -2\frac{u_{ij+1}^{k+1} - u_{ij-1}^{k+1}}{2h_y} - 3u_{ij}^k$$

$$u_{ij}^{k+1} = \frac{(6u_{ij}^k h_y^2 + (u_{ij+1}^{k+1} - u_{ij-1}^{k+1})h_y + 2u_{ij-1}^{k+1} + 2u_{ij+1}^k)h_x^2 + 2(u_{i-1j}^{k+1} + u_{i+1j}^k)h_y^2}{4h_x^2 + 4h_y^2}$$

4. **Код:**

```
        \\Define the initial boundary conditions


double u0y(double y)
{
    return exp(-y)*cos(y);
}
double u1y(double y)
{
    return 0;
}
double u0x(double x)
{
    return cos(x);
}
double u1x(double x)
{
    return 0;
}


        \\liebmann_method


void liebmann_method(int N, int K, double Lx, double Ly, double eps,
```

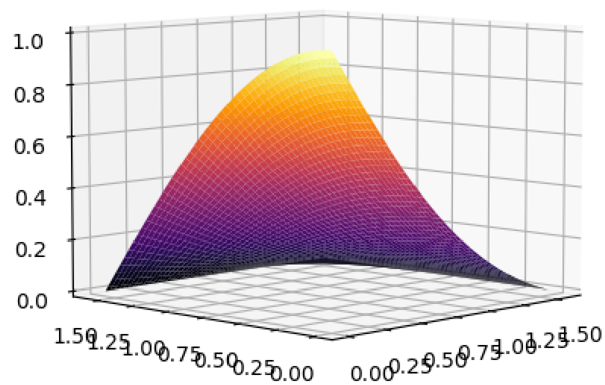
```
double omg, double *args, matrix *grid, double(*f_args[4])(double))
{
    double hx = Lx/(N-1), hy = Ly/(K-1);
    double delta = 1/(2/pow(hx, 2) + 2/pow(hy,2) + args[2]);
    double hhx = 1/pow(hx,2);
    double ahx = args[0]/2/hx;
    double hhy = 1/pow(hy,2);
    double bhy = args[1]/2/hy;
    for(int i=0; i<K; i++)
    {
            *get_element(grid, i, 0) = f_args[2](hx*i);
            *get_element(grid, i, N-1) = f_args[3](hx*i);
    }
    for(int i=0; i<N; i++)
    {
            *get_element(grid, 0, i) = f_args[0](hy*i);
            *get_element(grid, K-1, i) = f_args[1](hy*i);
    }
    for(int i=1; i<K-1; i++)
        for(int j=1; j<N-1; j++)
        {
            double alpha = (j * hy) / Ly;
            *get_element(grid, i, j) = f_args[2](i * hx)*(1 - alpha) +
            f_args[3](i * hx) * alpha;
        }
    int n = 0;
    double err = eps*100;
    double *U2;
    do
    {
        n++;
        U2=malloc(N*K*sizeof(double));
        for(int i=0; i<grid->rows; i++)
            for(int j=0; j<grid->columns; j++)
                U2[i*(grid->columns)+ j] = *get_element(grid, i, j);

        for(int i=1; i<K-1; i++)
            for(int j=1; j<N-1; j++)
                *get_element(grid, i, j) = delta * ((hhx + ahx) *
                U2[(i - 1)*N+ j] + (hhx - ahx) * U2[(i + 1)*N+ j] +
                (hhy + bhy) * U2[i*N+ (j -1)] + (hhy - bhy) * U2[i*N+ (j +1)]);
        err = eror(grid, U2);
        free(U2);
    } while((err > eps) && (n<10000));
    printf("%d", n);
}
```

```
            \\seidel_method


void seidel_method(int N, int K, double Lx, double Ly, double eps,
double omg, double *args, matrix *grid, double(*f_args[4])(double))
{
    double hx = Lx/(N-1), hy = Ly/(K-1);
    double delta = 1/(2/pow(hx, 2) + 2/pow(hy,2) + args[2]);
    double hhx = 1/pow(hx,2);
    double ahx = args[0]/2/hx;
    double hhy = 1/pow(hy,2);
    double bhy = args[1]/2/hy;
    for(int i=0; i<K; i++)
    {
            *get_element(grid, i, 0) = f_args[2](hx*i);
            *get_element(grid, i, N-1) = f_args[3](hx*i);
    }
    for(int i=0; i<N; i++)
    {
            *get_element(grid, 0, i) = f_args[0](hy*i);
            *get_element(grid, K-1, i) = f_args[1](hy*i);
    }
    for(int i=1; i<K-1; i++)
        for(int j=1; j<N-1; j++)
        {
            double alpha = (j * hy) / Ly;
            *get_element(grid, i, j) = f_args[2](i * hx)*(1 - alpha) +
            f_args[3](i * hx) * alpha;
        }
    int n = 0;
    double err = eps*100;
    double *U2;
    do
    {
        n++;
        U2=malloc(N*K*sizeof(double));
        for(int i=0; i<grid->rows; i++)
            for(int j=0; j<grid->columns; j++)
                U2[i*(grid->columns)+ j] = *get_element(grid, i, j);

        for(int i=1; i<K-1; i++)
            for(int j=1; j<N-1; j++)
                *get_element(grid, i, j) = delta * ((hhx + ahx) *
                (*get_element(grid, i-1, j)) + (hhx - ahx) *
                (*get_element(grid, i+1, j)) + (hhy + bhy) *
                (*get_element(grid, i, j-1)) + (hhy - bhy) *
                (*get_element(grid, i, j+1)));
```

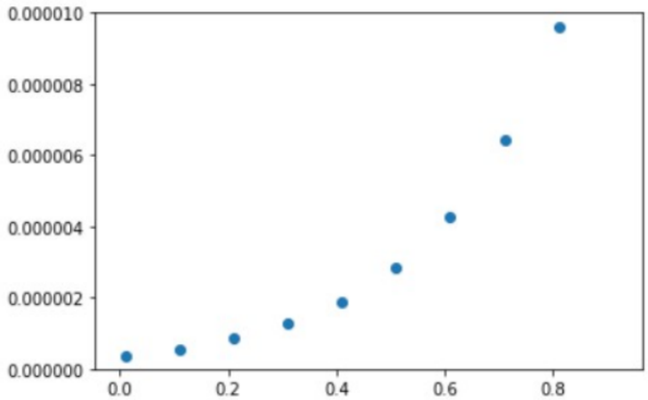
```
        err = eror(grid, U2);
        free(U2);
    } while((err > eps) && (n<10000));
    printf("%d", n);
}


            \\relax_method


void relax_method(int N, int K, double Lx, double Ly, double eps,
double omg, double *args, matrix *grid, double(*f_args[4])(double))
{
    double hx = Lx/(N-1), hy = Ly/(K-1);
    double delta = 1/(2/pow(hx, 2) + 2/pow(hy,2) + args[2]);
    double hhx = 1/pow(hx,2);
    double ahx = args[0]/2/hx;
    double hhy = 1/pow(hy,2);
    double bhy = args[1]/2/hy;
    for(int i=0; i<K; i++)
    {
            *get_element(grid, i, 0) = f_args[2](hx*i);
            *get_element(grid, i, N-1) = f_args[3](hx*i);
    }
    for(int i=0; i<N; i++)
    {
            *get_element(grid, 0, i) = f_args[0](hy*i);
            *get_element(grid, K-1, i) = f_args[1](hy*i);
    }
    for(int i=1; i<K-1; i++)
        for(int j=1; j<N-1; j++)
        {
            double alpha = (j * hy) / Ly;
            *get_element(grid, i, j) = f_args[2](i * hx)*(1 - alpha) +
             f_args[3](i * hx) * alpha;
        }
    int n = 0;
    double err = eps*100;
    double *U2;
    do
    {
        n++;
        U2=malloc(N*K*sizeof(double));
        for(int i=0; i<grid->rows; i++)
            for(int j=0; j<grid->columns; j++)
                U2[i*(grid->columns)+ j] = *get_element(grid, i, j);
```

```
        for(int i=1; i<K-1; i++)
            for(int j=1; j<N-1; j++)
                *get_element(grid, i, j) = (*get_element(grid, i, j)) +
                 omg * (delta * ((hhx + ahx) * (*get_element(grid, i-1, j))+
                (hhx - ahx) * U2[(i + 1)*N+ j] + (hhy + bhy) *
                (*get_element(grid, i, j-1)) +(hhy - bhy) *
                U2[i*N+ (j+1)]) - (*get_element(grid, i, j)));
        err = eror(grid, U2);
        free(U2);
    } while((err > eps) && (n<10000));
    printf("%d", n);
}
```

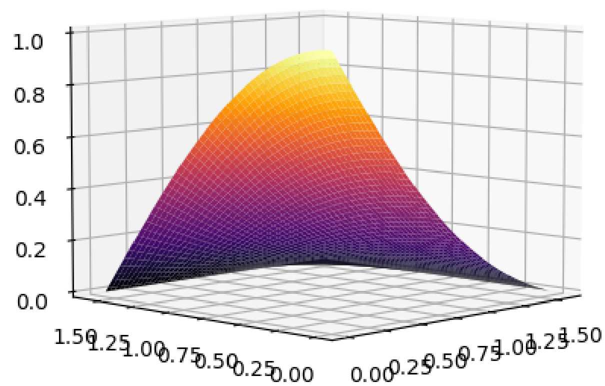
5. **Результат:**
   Метод Либмана:



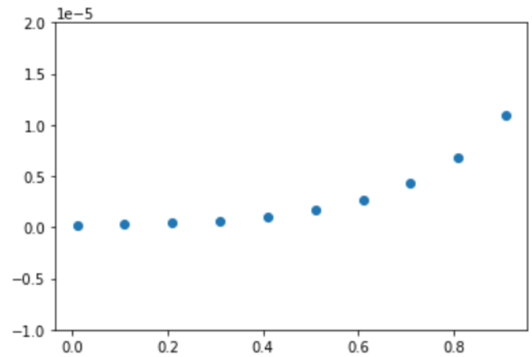
Покажем ошибку в зависимости от длины шага по пространству



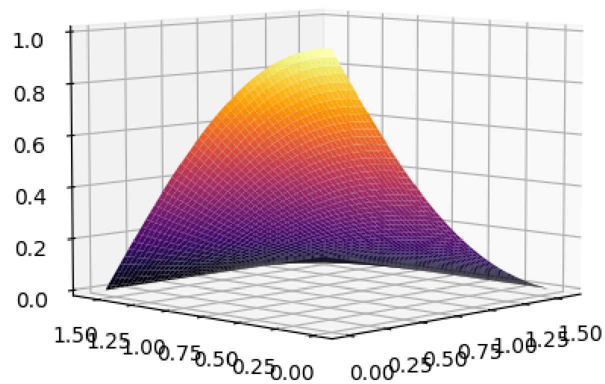
Метод Зейделя:



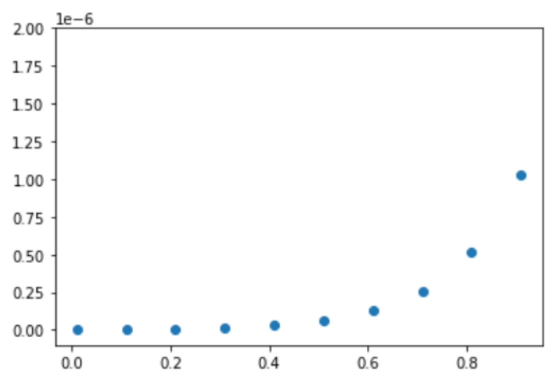
Покажем ошибку в зависимости от длины шага по пространству



Метод Релаксации:



Покажем ошибку в зависимости от длины шага по пространству



6. **Вывод:**

   В лабораторной работе №3 изучил методы Либмана, Зейделя и Релаксации, построил график зависимости ошибки от размера шага по пространству и сравнил количество итераций для выполнения алгоритма в заданой точности:
   1)При 100x100 сетка 0.00001 эпсилон;
   Простые 5000;
   Зейдель 3400;
   Релаксация 1600;
   2)При 50x50 сетка 0.00001 эпсилон;
   Простые 2157;
   Зейдель 1606;
   Релаксация 554;
   3)При 25x25 сетка 0.00001 эпсилон;
   Простые 739;
   Зейдель 425;
   Релаксация 169;