МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ НЭТИ | Факультет прикладной математики и информатики

Кафедра прикладной математики

Лабораторная работа №2

по дисциплине «Уравнения математической физики»

РЕШЕНИЕ НЕЛИНЕЙНЫХ НАЧАЛЬНО-КРАЕВЫХ ЗАДАЧ

Группа ПМ-92 АРТЮХОВ РОМАН

Вариант 3

Преподаватели ЗАДОРОЖНЫЙ А. Г.

ПАТРУШЕВ И. И.

Новосибирск, 2022

Цель работы

Разработать программу решения нелинейной одномерной краевой задачи методом конечных элементов. Провести сравнение метода простой итерации и метода Ньютона для решения данной задачи.

Задача (вариант 3)

Уравнение:

$$-div\left(\lambda\left(\frac{\partial u}{\partial x}\right)gradu\right)+\gamma u=f$$

Базисные функции – линейные. Краевые условия всех типов.

Анализ

Система нелинейных уравнений:

$A(q)q=b$, где:

$$A_{ij}=\int_{\Omega}\lambda\left(\frac{\partial u(q)}{\partial x}\right)grad\psi_j grad\psi_i d\Omega+\int_{\Omega}\gamma\psi_j\psi_i d\Omega+\int_{S_3}\beta\psi_j\psi_i dS$$

$$b_i=\int_{\Omega}f\psi_i d\Omega+\int_{S_2}\theta\psi_i dS+\int_{S_3}\beta u_\beta\psi_i dS$$

Метод простой итерации

$$u=\sum_{i=1}^{n}q_i\psi_i(x)$$

Поскольку случай одномерный, линейные базисные функции на конечном элементе могут быть записаны в виде:

$$\psi_k=\frac{x_{k+1}-x}{h_k};\ \psi_{k+1}=\frac{x-x_k}{h_k};\ h_k=x_{k+1}-x_k$$

$$\frac{\partial u}{\partial x}=q_1\frac{\partial\psi_1(x)}{\partial x}+q_2\frac{\partial\psi_2(x)}{\partial x}=\frac{q_2-q_1}{h}$$

Получаем: $\lambda\left(\frac{\partial u}{\partial x}\right)=\lambda\left(\frac{q_2-q_1}{h}\right)$

Вид локальной матрицы будет следующим:

$$A_{ij} = \int_{x_1}^{x_2} \lambda\left(\frac{\partial u(q)}{\partial x}\right)\frac{\partial \psi_i}{\partial x}\frac{\partial \psi_j}{\partial x}dx + \int_{x_1}^{x_2} \gamma_k \psi_i \psi_j dx = G_{ij} + M_{ij}$$

Матрица жесткости (2x2):

$$G_{ij} = \int_{x_1}^{x_2} \lambda\left(\frac{q_2 - q_1}{h}\right)\Bigg|_{x=x_1} \psi_1(x)\frac{\partial \psi_i}{\partial x}\frac{\partial \psi_j}{\partial x}dx + \int_{x_1}^{x_2} \lambda\left(\frac{q_2 - q_1}{h}\right)\Bigg|_{x=x_2} \psi_2(x)\frac{\partial \psi_i}{\partial x}\frac{\partial \psi_j}{\partial x}dx$$

$$\lambda_i = \lambda\left(\frac{q_2 - q_1}{h}\right)\Bigg|_{x=x_i}$$

На диагонали матрицы жесткости: $\frac{\partial \psi_i}{\partial x}\frac{\partial \psi_j}{\partial x} = \frac{1}{h^2}$

Вне диагонали матрицы жесткости: $\frac{\partial \psi_i}{\partial x}\frac{\partial \psi_j}{\partial x} = -\frac{1}{h^2}$

Остальная часть: $\int_{x_1}^{x_2} \lambda_1 \psi_1(x)dx + \int_{x_1}^{x_2} \lambda_2 \psi_2(x)dx = \frac{h}{2}(\lambda_1 + \lambda_2)$

Следовательно матрица жесткости выглядит следующим образом:

$$G = \left(\frac{\lambda_1 + \lambda_2}{2h}\right)\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Матрица масс:

$$M = \frac{\gamma h}{6}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Вектор правой части:

$$b = \frac{h}{6}\begin{pmatrix} 2f(x_k) + f(x_{k+1}) \\ 2f(x_{k+1}) + f(x_k) \end{pmatrix}$$

Метод Ньютона

Метод Ньютона основан на линеаризации нелинейных уравнений нашей системы с использованием разложения в ряд Тейлора. Каждый нелинейный член уравнения представляется в виде разложения в ряд Тейлора в окрестности вектора весов q^0:

$$A_{ij}(q)\cdot q_j \approx A_{ij}\left(q^0\right)\cdot q_j^0 + \sum_r \frac{\partial\left(A_{ij}\left(q^0\right)\cdot q_j\right)}{\partial q_r}\left(q_r - q_r^0\right)$$

$$b_i(q) \approx b_i\left(q^0\right) + \sum_r \frac{\partial\left(b_i q^0\right)}{\partial q_r}\left(q - q_r^0\right)$$

В результате получаем новую СЛАУ $A^L q = b^L$:

Для нашей задачи матрица и правая часть выглядит следующим образом:

$$A_{ij}^L = A_{ij}\left(q^0\right) + \sum_{r=1}^{n} \frac{\partial\left(A_{ir}\left(q^0\right)\right)}{\partial q_j} q_r^0$$

$$b_i^L = b_i\left(q^0\right) + \sum_{j=1}^{n} q_j^0 \sum_{r=1}^{n} \frac{\partial\left(A_{ij}\left(q^0\right)\right)}{\partial q_r} q_r^0$$

Выпишем производные:

$$\frac{\partial A_{11}}{\partial q_1} = \frac{1}{2h}\left(\frac{\partial \lambda_1}{\partial q_1} + \frac{\partial \lambda_2}{\partial q_1}\right); \qquad \frac{\partial A_{12}}{\partial q_1} = -\frac{1}{2h}\left(\frac{\partial \lambda_1}{\partial q_1} + \frac{\partial \lambda_2}{\partial q_1}\right)$$

$$\frac{\partial A_{11}}{\partial q_2} = \frac{1}{2h}\left(\frac{\partial \lambda_1}{\partial q_2} + \frac{\partial \lambda_2}{\partial q_2}\right); \qquad \frac{\partial A_{12}}{\partial q_2} = -\frac{1}{2h}\left(\frac{\partial \lambda_1}{\partial q_2} + \frac{\partial \lambda_2}{\partial q_2}\right)$$

$$\frac{\partial A_{21}}{\partial q_1} = \frac{\partial A_{12}}{\partial q_1}; \quad \frac{\partial A_{21}}{\partial q_2} = \frac{\partial A_{12}}{\partial q_2}$$

$$\frac{\partial A_{22}}{\partial q_1} = \frac{\partial A_{11}}{\partial q_1}; \quad \frac{\partial A_{22}}{\partial q_2} = \frac{\partial A_{11}}{\partial q_2}$$

Для выхода из итерационного процесса воспользуемся формулой:

$$\frac{\left\| A\left(q^k\right)q^k - b\left(q^k\right)\right\|}{\|b\|} < \varepsilon$$

Для ускорения процесса сходимости нелинейной задачи, ищем следующие приближения по формуле:

$$q^k = \omega^k \bar{q}^k + \left(1-\omega^k\right) q^{(k-1)}$$

Учет краевых условий

Для учета первых краевых условий нужно сделать следующее: соответствующий диагональный элемент заменяют единицей, а соответствующую координату вектора правой части значением q_i, где q_i – есть первое краевое условие в узле.

$\lambda \frac{\partial u}{\partial n}\Big|_{s_2} = \theta$ При учете вторых краевых условий идет добавка значения θ к глобальному вектору.

$\left(\lambda \frac{\partial u}{\partial n} + \beta(u - u_\beta \right)\Big|_{S_3} = 0$ При учете третьих краевых условий идет добавка значения β к диагональному элементу глобальной матрицы и значения $\beta \cdot u_\beta$ к глобальному вектору.

Helper.cs

```
namespace NonLinear.helper;

public struct Elem     /// Структура конечного элемента
{
    public double x1;  /// Координата начала КЭ
    public double x2;  /// Координата конца КЭ
    public double h;   /// Длина КЭ

    public Elem(double _x1, double _x2, double _h) {
        x1 = _x1; x2 = _x2; h = _h;
    }

    public override string ToString() => $"{x1,20} {x2,24} {h,24}";
}

public struct SLAU                 /// Структура СЛАУ
{
    public double[] di, dl, du;    /// Матрица
    public double[] q, q1;         /// Вектора решений
    public double[] f;             /// Правая часть
    public int MaxIter;            /// Максимальное количество итераций
    public double EPS;             /// Точность

    //* Обнуление СЛАУ (левой и правой частей)
    public void Clear() {
        Array.Clear(di, 0, di.Length);
        Array.Clear(dl, 0, dl.Length);
        Array.Clear(du, 0, du.Length);
        Array.Clear(f , 0, f.Length);
    }

    //* Перемножение матрицы на вектор
    public void Mult(double[] vec, out double[] res_vec) {
        res_vec = new double[vec.Length];
        res_vec[0] = di[0] * q[0] + du[0] * q[1];
        for (int i = 1; i < vec.Length - 1; i++) {
            res_vec[i]  = di[i] * q[i];
            res_vec[i] += du[i] * q[i + 1];
            res_vec[i] += dl[i] * q[i - 1];
        }
        res_vec[vec.Length - 1] = di[vec.Length - 1] * q[vec.Length - 1] + dl[vec.Length - 1] * q[vec.Length - 2];
    }

    //* Метод прогонки
    public double[] Progonka() {
        var alpha = new double[q.Length];
        var betta = new double[q.Length];

        // Подсчет альф и бетт
        alpha[0] = -du[0] / di[0];
        betta[0] = f[0] / di[0];
        for (int i = 1; i < q.Length; i++) {
            alpha[i] = -du[i] / (di[i] + dl[i] * alpha[i - 1]);
            betta[i] = (f[i] - dl[i] * betta[i - 1]) / (di[i] + dl[i] * alpha[i - 1]);
        }

        q[q.Length - 1] = betta[q.Length - 1];
        for (int i = q.Length - 2; i >= 0; i--)
            q[i] = alpha[i] * q[i + 1] + betta[i];
        return q;
    }
}

public static class Helper
```

```
{
    //: ***************** Перечисления ***************** :\\
    public enum Method {
        Iteration,
        Newton
    }
    //: ***************** Перечисления ***************** :\\

    //* Вычисление нормы вектора
    public static double Norm(double[] array) {
        double norm = 0;
        for (int i = 0; i < array.Count(); i++)
            norm += array[i] * array[i];
        return Sqrt(norm);
    }

    //* Окно помощи при запуске (если нет аргументов или по команде)
    public static void ShowHelp() {
        WriteLine("----Команды----                          \n" +
        "-help              - показать справку              \n" +
        "-i                 - входной файл                  \n" +
        "-m iteration       - метод простой итерации        \n" +
        "-m newton          - метод Ньютона                 \n" +
        "-func 'number'     - номер функции                 \n");
    }
}
```

Data.cs

```
namespace NonLinear;
public class Data
{
    //* Данные задачи
    public uint     countEl { get; set; }   /// Количество КЭ/отрезков
    public double   begin   { get; set; }   /// Начало отрезка
    public double   end     { get; set; }   /// Конец отрезка
    public double   k       { get; set; }   /// Коэффициент разрядки
    public double   gamma   { get; set; }   /// Коэффициенты гамма
    public double   betta   { get; set; }   /// Коэффициенты бетта
    public double[] kraev   { get; set; }   /// Краевые условия
    public double[] init_q  { get; set; }   /// Начальное приближение
}
```

Function.cs

```
namespace NonLinear;
public static class Function
{
    public static uint   NumberFunc;     /// Номер задачи

    //* Функция u(x,y)
    public static double Absolut(double x)
    {
        switch(NumberFunc)
        {
            case 1: /// test1-evenly (const)
            return 2.5;

            case 2: /// test2-polynom1
            return x;

            case 3: /// test3-polynom2
```

```
        return x*x;

        case 4: /// test4-polynom3
        return Pow(x, 3);

        case 5: /// test5_nopolynom
        return Sin(2*x);
    }
    return 0;
}

//* Функция f(x,y)
public static double Func(double x)
{
    switch(NumberFunc)
    {
        case 1: /// test1-evenly (const)
        return 5;

        case 2: /// test2-polynom1
        return 2*x;

        case 3: /// test3-polynom2
        return 2*x*x - 8*x - 8;

        case 4: /// test4-polynom3
        return -34*Pow(x, 3) - 24*x;

        case 5: /// test5_nopolynom
        return 16*Sin(2*x)*Cos(2*x);
    }
    return 0;
}

//* Лямбда от производной l(u`)
public static double Lambda(double diff, double x)
{
    switch(NumberFunc)
    {
        case 1: /// test1-evenly (const)
        return diff + 4;

        case 2: /// test2-polynom1
        return diff + 4;

        case 3: /// test3-polynom2
        return diff + 4;

        case 4: /// test4-polynom3
        return diff + 4;

        case 5: /// test5_nopolynom
        return diff + 4;
    }
    return 0;
}

//* Производная от Лямбды (в зависимости почему берем)
public static double DiffLambda(double diff, double x, double h,  uint num_q)
{
    switch(NumberFunc)
    {
        case 1: /// test1-evenly (const)
        return num_q switch
        {
            1 => -1/h,
            2 =>  1/h,
            _ => 0
```

```
            };

            case 2: /// test2-polynom1
            return num_q switch
            {
                1 => -1/h,
                2 =>  1/h,
                _ => 0
            };

            case 3: /// test3-polynom2
            return num_q switch
            {
                1 => -1/h,
                2 =>  1/h,
                _ => 0
            };

            case 4: /// test4-polynom3
            return num_q switch
            {
                1 => -1/h,
                2 =>  1/h,
                _ => 0
            };

            case 5: /// test5_nopolynom
            return num_q switch
            {
                1 => -1/h,
                2 =>  1/h,
                _ => 0
            };
        }
        return 0;
    }

    //* Производная абсолютной функции
    public static double Diff(double x)
    {
        switch(NumberFunc)
        {
            case 1: /// test1-evenly (const)
            return 0;

            case 2: /// test2-polynom1
            return 1;

            case 3: /// test3-polynom2
            return 2*x;

            case 4: /// test4-polynom3
            return 3*Pow(x, 2);

            case 5: /// test5_nopolynom
            return Cos(2*x);
        }
        return 0;
    }

}
```

PSolve.cs

```
public partial class Solve
{
    //* Генерация сетки
    private Elem[] Generate() {
        Elem[] elems = new Elem[data.countEl];

        // Подсчет начального шага
        double _h = data.k == 1
                    ? (data.end - data.begin) / data.countEl
                    : (1 - data.k) * (data.end - data.begin) / (1 - Pow(data.k, data.countEl));

        double x = data.begin; // Текущая позиция
        for (int i = 0; i < data.countEl && x <= data.end; i++) {
            elems[i] = new Elem(x, x + _h, _h);
            x  += _h;
            _h *= data.k;
        }

        StringBuilder grid = new StringBuilder();
        grid.Append($"{"x1",20} {"x2",24} {"h",24}\n");
        grid.Append(String.Join("\n", elems));
        File.WriteAllText($"{path}\\grid.txt", grid.ToString());
        return elems;
    }

    //* Путь к папке, где наход. решение задачи
    public void SetPath(string _path) {
        path = _path;
    }

    public partial void solve();
    private partial void memory();
}
```

Solve.cs

```
public partial class Solve
{
    protected Data    data;                    /// Данные задачи
    protected Method  method;                  /// Метод решения (Итерация или Ньютон)
    protected Elem[]  elems;                   /// Конечные элементы
    protected SLAU    slau;                    /// Структура СЛАУ
    protected string  path  { get; set; }      /// Путь к папке с задачей

    public Solve(Data _data, Method _method, uint _funcNumber) {
        Function.NumberFunc = _funcNumber;
        method              = _method    ;
        data                = _data      ;
    }

    public partial void solve() {
        elems = Generate();              //? Генерация сетки
        memory();                        //? Выделение памяти под матрица и вектора

        IMethods task = method switch
                    {
                        Method.Iteration => new Iteration(slau, data, elems),
                        Method.Newton    => new Newton(slau, data, elems),
                        _                => new Iteration(slau, data, elems)
                    };
        Directory.CreateDirectory($"{path}\\output");
```

```
        File.WriteAllText($"{path}\\output\\table_" + $"{task.GetName()}" + ".txt", task.solve());
    }

    private partial void memory() {
        slau.di   = new double[data.countEl + 1];
        slau.dl   = new double[data.countEl + 1];
        slau.du   = new double[data.countEl + 1];
        slau.q    = new double[data.countEl + 1];
        slau.q1   = new double[data.countEl + 1];
        slau.f    = new double[data.countEl + 1];
        // Записываем начальное приближение
        Array.Copy(data.init_q, slau.q, slau.q.Length);
    }
}
```

IMethods.cs

```
namespace NonLinear.methods;
public interface IMethods
{
    public string GetName();                                                    /// Возвращает имя метода
    public string solve();                                                      /// Главная функция решения
    public void global();                                                       /// Генерация глобальной матрицы
    public (double[][], double[]) local(int _i);                                /// Генерация локальной матрицы
    public void add_to_global(double[][] _mat, double[] _vec,  int _i);         /// Занесение локальной в глоабльную
    public void kraev();                                                        /// Учет краевых условий
    public void firstKraev(int _i);                                             /// Учет первого краевого
    public void secondKraev(int _i);                                            /// Учет второго краевого
    public void thirdKraev(int _i);                                             /// Учет третьего краевого
    public double[] newApprox();                                                /// Получение нового приближения (релаксация)
    public StringBuilder output();                                              /// Вывод решения
}
```

Iteration.cs

```
namespace NonLinear.methods;
public class Iteration : IMethods
{
    double w;                 /// Параметр релаксации
    private SLAU    slau;     /// структура СЛАУ
    private Data    data;     /// Данные задачи
    private Elem[] elems;     /// Конечные элементы

    public string GetName() => "Iteration";

    public Iteration(SLAU _slau, Data _data, Elem[] _elems, double _w = 1) {
        slau         = _slau;
        data         = _data;
        elems        = _elems;
        w            = _w;
        slau.MaxIter = 1000;
        slau.EPS     = 1e-10;
    }

    private bool Check(int iter, bool log = true) {
        global();
        slau.Mult(slau.q, out double[] vec_nev);
        vec_nev = slau.f.Zip(vec_nev, (f, s) => f - s).ToArray();
        double value = Norm(vec_nev) / Norm(slau.f);

        if (log)
```

```
        WriteLine($"{iter,5} : {value,10}");

    if (iter  > slau.MaxIter) return false;
    if (value < slau.EPS)     return false;

    return true;
}


//: ***** Реализация функций интерфейса ***** :\\
public string solve() {
    int Iter = 0; //? Количество итераций

    do {
        Array.Copy(slau.q, slau.q1, slau.q.Length);     //? q1 = q
        global();                                       //? Генерация глобальной матрицы
        slau.q = slau.Progonka();                       //? Решаем СЛАУ методом Прогонки
        slau.q = newApprox();                           //? Получаем новое приближение (релаксация)
    } while (Check(++Iter));

    return output().ToString();
}

public void global() {
    slau.Clear();                                               // Очищаем СЛАУ
    for (int index = 0; index < data.countEl; index++) {        // Проход по всем КЭ
        (double[][] mat, double[] vec) = local(index);          // Генерируем локальную матрицу и вектор
        add_to_global(mat, vec, index);                         // Заносим локальное в глобальное
    }
    kraev();                                                    // Учет краевых
}

public (double[][], double[]) local(int i) {
    var local_vec = new double[2];
    var local_mat = new double[2][];
    for (int j = 0; j < 2; j++) local_mat[j] = new double[2];

    double lam_arg = (slau.q[i + 1] - slau.q[i]) / elems[i].h;

    // + Матрица жесткости
    double value = (Lambda(lam_arg, elems[i].x1) + Lambda(lam_arg, elems[i].x2)) / (2*elems[i].h);
    for (int j = 0; j < 2; j++)
        for (int k = 0; k < 2; k++)
            local_mat[j][k] = j == k
                            ?  value
                            : -value;

    // + Матрица масс
    value = data.gamma * elems[i].h / 6.0;
    for (int j = 0; j < 2; j++)
        for (int k = 0; k < 2; k++)
            local_mat[j][k] += j == k
                            ?  2*value
                            :    value;

    // Правая часть
    var f = new double[2] { Func(elems[i].x1), Func(elems[i].x2) };
    for (int j = 0, k = 1; j < 2; j++, k--)
        local_vec[j] = (elems[i].h / 6.0) * (2*f[j] + f[k]);

    return (local_mat, local_vec);
}

public void add_to_global(double[][] mat, double[] vec, int i) {
    slau.di[i]     += mat[0][0];
    slau.di[i + 1] += mat[1][1];
    slau.du[i]     += mat[0][1];
    slau.dl[i + 1] += mat[1][0];
```

```
        slau.f[i]      += vec[0];
        slau.f[i + 1]  += vec[1];
    }

    public void kraev() {
        for (int i = 0; i < data.kraev.Length; i++) {
            switch (data.kraev[i])
            {
                case 1: firstKraev(i);  break;
                case 2: secondKraev(i); break;
                case 3: thirdKraev(i);  break;
            }
        }
    }

    public void firstKraev(int i) {
        switch(i) {
            // Левая граница
            case 0:
                slau.di[0] = 1;
                slau.du[0] = 0;
                slau.f[0]  = Absolut(elems[0].x1);
            break;

            // Правая граница
            case 1:
                slau.di[data.countEl] = 1;
                slau.dl[data.countEl] = 0;
                slau.f [data.countEl] = Absolut(elems[data.countEl - 1].x2);
            break;
        }
    }

    public void secondKraev(int i) {
        switch(i) {
            // Левая граница
            case 0:
                double lam_arg = (slau.q[1] - slau.q[0]) / elems[0].h;
                slau.f[0] -= Lambda(lam_arg, elems[0].x1) * Diff(elems[0].x1);
            break;

            // Правая граница
            case 1:
                lam_arg = (slau.q[data.countEl] - slau.q[data.countEl - 1]) / elems[data.countEl - 1].h;
                slau.f[data.countEl] += Lambda(lam_arg, elems[data.countEl - 1].x2) * Diff(elems[data.countEl - 1].x2);
            break;
        }
    }

    public void thirdKraev(int i)
    {
        switch(i) {
            // Левая граница
            case 0:
                slau.di[0] += data.betta;
                double lam_arg = (slau.q[1] - slau.q[0]) / elems[0].h;
                double res = Lambda(lam_arg, elems[0].x1) * (-1) * Diff(elems[0].x1);
                slau.f[0] += data.betta * (res + data.betta * Absolut(elems[0].x1));
            break;

            // Правая граница
            case 1:
                slau.di[data.countEl] += data.betta;
                lam_arg = (slau.q[data.countEl] - slau.q[data.countEl - 1]) / elems[data.countEl - 1].h;
                res = Lambda(lam_arg, elems[data.countEl - 1].x2) * Diff(elems[data.countEl - 1].x2);
                slau.f[data.countEl] += data.betta * (res + data.betta * Absolut(elems[data.countEl - 1].x2));
            break;
        }
```

```
    }

    public double[] newApprox() {
        return slau.q.Zip(slau.q1, (f, s) => w*f + (1 - w)*s).ToArray();
    }

    public StringBuilder output() {
        var table = new StringBuilder();

        string margin = String.Join("", Enumerable.Repeat("-", 16));

        table.Append(String.Join("", Enumerable.Repeat("-", 52)) + "\n");
        table.Append($"|U{" ",-14} | U`{" ",-12} | |U`- U| {" ",-7}|\n");
        table.Append($"|" + margin + "|" + margin + "|" + margin + "|\n");

        for (int i = 0; i < data.countEl; i++)
            table.Append($"|{String.Format("{0,16}", slau.q[i].ToString("E6"))}" +
                        $"|{String.Format("{0,16}", Absolut(elems[i].x1).ToString("E6"))}"   +
                        $"|{String.Format("{0,16}", Abs(Absolut(elems[i].x1) - slau.q[i]).ToString("E6"))}|\n");


        table.Append($"|{String.Format("{0,16}", slau.q[data.countEl].ToString("E6"))}" +
                    $"|{String.Format("{0,16}", Absolut(elems[data.countEl - 1].x2).ToString("E6"))}"   +
                    $"|{String.Format("{0,16}", Abs(Absolut(elems[data.countEl - 1].x2) - slau.q[data.countEl]).ToString("

        table.Append(String.Join("", Enumerable.Repeat("-", 52)) + "\n");

        return table;
    }
}
```

Newton.cs

```
namespace NonLinear.methods;
public class Newton : IMethods
{
    double w;                 /// Параметр релаксации
    private SLAU    slau;     /// структура СЛАУ
    private Data    data;     /// Данные задачи
    private Elem[] elems;     /// Конечные элементы

    public string GetName() => "Newton";

    public Newton(SLAU _slau, Data _data, Elem[] _elems, double _w = 1) {
        slau         = _slau;
        data         = _data;
        elems        = _elems;
        w            = _w;
        slau.MaxIter = 1000;
        slau.EPS     = 1e-10;
    }

    private bool Check(int iter, bool log = true) {
        global();
        slau.Mult(slau.q, out double[] vec_nev);
        vec_nev = slau.f.Zip(vec_nev, (f, s) => f - s).ToArray();
        double value = Norm(vec_nev) / Norm(slau.f);

        if (log)
            WriteLine($"{iter,5} : {value,10}");

        if (iter  > slau.MaxIter) return false;
        if (value < slau.EPS)     return false;

        return true;
```

```
}

void AddNewton(ref double[] mat_newton, int i) {
    double lam_arg   = (slau.q[i + 1] - slau.q[i]) / elems[i].h;
    double diff_l_q1_x1 = DiffLambda(lam_arg, elems[i].x1, elems[i].h, 1);
    double diff_l_q1_x2 = DiffLambda(lam_arg, elems[i].x2, elems[i].h, 1);
    double diff_l_q2_x1 = DiffLambda(lam_arg, elems[i].x1, elems[i].h, 2);
    double diff_l_q2_x2 = DiffLambda(lam_arg, elems[i].x2, elems[i].h, 2);

    // Матрица 1x4
    // A11_q1=A22_q1
    // A11_q2=A22_q2
    // A12_q1=A21_q1
    // A12_q2=A21_q2
    mat_newton[0] =  (diff_l_q1_x1 + diff_l_q1_x2) / (2 * elems[i].h);
    mat_newton[1] =  (diff_l_q2_x1 + diff_l_q2_x2) / (2 * elems[i].h);
    mat_newton[2] = -(diff_l_q1_x1 + diff_l_q1_x2) / (2 * elems[i].h);
    mat_newton[3] = -(diff_l_q2_x1 + diff_l_q2_x2) / (2 * elems[i].h);
}

//: ***** Реализация функций интерфейса ***** :\\
public string solve() {
    int Iter = 0; //? Количество итераций

    do {
        Array.Copy(slau.q, slau.q1, slau.q.Length);     //? q1 = q
        global();                                       //? Генерация глобальной матрицы
        slau.q = slau.Progonka();                       //? Решаем СЛАУ методом Прогонки
        slau.q = newApprox();                           //? Получаем новое приближение (релаксация)
    } while (Check(++Iter));

    return output().ToString();
}

public void global() {
    slau.Clear();                                               // Очищаем СЛАУ
    for (int index = 0; index < data.countEl; index++) {        // Проход по всем КЭ
        (double[][] mat, double[] vec) = local(index);          // Генерируем локальную матрицу и вектор
        add_to_global(mat, vec, index);                         // Заносим локальное в глобальное
    }
    kraev();                                                    // Учет краевых
}

public (double[][], double[]) local(int i) {
    var local_vec  = new double[2];
    var local_mat  = new double[2][];
    var mat_newton = new double[4];
    for (int j = 0; j < 2; j++) local_mat[j] = new double[2];

    double lam_arg = (slau.q[i + 1] - slau.q[i]) / elems[i].h;
    AddNewton(ref mat_newton, i);

    // + Матрица жесткости
    double value = (Lambda(lam_arg, elems[i].x1) + Lambda(lam_arg, elems[i].x2)) / (2*elems[i].h);
    local_mat[0][0] =  value + (mat_newton[0]*slau.q[i] + mat_newton[2]*slau.q[i + 1]);
    local_mat[0][1] = -value + (mat_newton[1]*slau.q[i] + mat_newton[3]*slau.q[i + 1]);
    local_mat[1][0] = -value + (mat_newton[2]*slau.q[i] + mat_newton[0]*slau.q[i + 1]);
    local_mat[1][1] =  value + (mat_newton[3]*slau.q[i] + mat_newton[1]*slau.q[i + 1]);

    // + Матрица масс
    value = (data.gamma * elems[i].h) / 6.0;
    for (int j = 0; j < 2; j++)
        for (int k = 0; k < 2; k++)
            local_mat[j][k] += j == k
                            ?  2*value
                            :    value;

    // Правая часть
```

```
        var f = new double[2] { Func(elems[i].x1), Func(elems[i].x2) };
        double add_newton_b0 = slau.q[i]     * (mat_newton[0]*slau.q[i] + mat_newton[1]*slau.q[i + 1]) +
                               slau.q[i + 1] * (mat_newton[2]*slau.q[i] + mat_newton[3]*slau.q[i + 1]);
        double add_newton_b1 = slau.q[i]     * (mat_newton[2]*slau.q[i] + mat_newton[3]*slau.q[i + 1]) +
                               slau.q[i + 1] * (mat_newton[0]*slau.q[i] + mat_newton[1]*slau.q[i + 1]);

        local_vec[0] = (elems[i].h / 6.0) * (2*f[0] + f[1]) + add_newton_b0;
        local_vec[1] = (elems[i].h / 6.0) * (2*f[1] + f[0]) + add_newton_b1;



        return (local_mat, local_vec);
    }

    public void add_to_global(double[][] mat, double[] vec, int i) {
        slau.di[i]     += mat[0][0];
        slau.di[i + 1] += mat[1][1];
        slau.du[i]     += mat[0][1];
        slau.dl[i + 1] += mat[1][0];
        slau.f[i]      += vec[0];
        slau.f[i + 1]  += vec[1];
    }

    public void kraev() {
        for (int i = 0; i < data.kraev.Length; i++) {
            switch (data.kraev[i])
            {
                case 1: firstKraev(i);  break;
                case 2: secondKraev(i); break;
                case 3: thirdKraev(i);  break;
            }
        }
    }

    public void firstKraev(int i) {
        switch(i) {
            // Левая граница
            case 0:
                slau.di[0] = 1;
                slau.du[0] = 0;
                slau.f[0]  = Absolut(elems[0].x1);
            break;

            // Правая граница
            case 1:
                slau.di[data.countEl] = 1;
                slau.dl[data.countEl] = 0;
                slau.f [data.countEl] = Absolut(elems[data.countEl - 1].x2);
            break;
        }
    }

    public void secondKraev(int i) {
        switch(i) {
            // Левая граница
            case 0:
                double lam_arg = (slau.q[1] - slau.q[0]) / elems[0].h;
                slau.f[0] -= Lambda(lam_arg, elems[0].x1) * Diff(elems[0].x1);
            break;

            // Правая граница
            case 1:
                lam_arg = (slau.q[data.countEl] - slau.q[data.countEl - 1]) / elems[data.countEl - 1].h;
                slau.f[data.countEl] += Lambda(lam_arg, elems[data.countEl - 1].x2) * Diff(elems[data.countEl - 1].x2);
            break;
        }
    }
```

```
    public void thirdKraev(int i)
    {
        switch(i) {
            // Левая граница
            case 0:
                slau.di[0] += data.betta;
                double lam_arg = (slau.q[1] - slau.q[0]) / elems[0].h;
                double res = Lambda(lam_arg, elems[0].x1) * (-1) * Diff(elems[0].x1);
                slau.f[0] += data.betta * (res + data.betta * Absolut(elems[0].x1));
            break;

            // Правая граница
            case 1:
                slau.di[data.countEl] += data.betta;
                lam_arg = (slau.q[data.countEl] - slau.q[data.countEl - 1]) / elems[data.countEl - 1].h;
                res = Lambda(lam_arg, elems[data.countEl - 1].x2) * Diff(elems[data.countEl - 1].x2);
                slau.f[data.countEl] += data.betta * (res + data.betta * Absolut(elems[data.countEl - 1].x2));
            break;
        }
    }

    public double[] newApprox() {
        return slau.q.Zip(slau.q1, (f, s) => w*f + (1 - w)*s).ToArray();
    }

    public StringBuilder output() {
        var table = new StringBuilder();

        string margin = String.Join("", Enumerable.Repeat("-", 16));

        table.Append(String.Join("", Enumerable.Repeat("-", 52)) + "\n");
        table.Append($"|U{" ",-14} | U`{" ",-12} | |U`- U| {" ",-7}|\n");
        table.Append($"|" + margin + "|" + margin + "|" + margin + "|\n");

        for (int i = 0; i < data.countEl; i++)
            table.Append($"|{String.Format("{0,16}", slau.q[i].ToString("E6"))}" +
                        $"|{String.Format("{0,16}", Absolut(elems[i].x1).ToString("E6"))}"   +
                        $"|{String.Format("{0,16}", Abs(Absolut(elems[i].x1) - slau.q[i]).ToString("E6"))}|\n");

        table.Append($"|{String.Format("{0,16}", slau.q[data.countEl].ToString("E6"))}" +
                    $"|{String.Format("{0,16}", Absolut(elems[data.countEl - 1].x2).ToString("E6"))}"   +
                    $"|{String.Format("{0,16}", Abs(Absolut(elems[data.countEl - 1].x2) - slau.q[data.countEl]).ToString("

        table.Append(String.Join("", Enumerable.Repeat("-", 52)) + "\n");

        return table;
    }
}
```

Program.cs

```
try {

    if (args.Length == 0) throw new ArgumentException("Not found arguments!");
    if (args[0] == "-help") {
        ShowHelp(); return;
    }

    string json = File.ReadAllText(args[1]);
    Data data = JsonConvert.DeserializeObject<Data>(json)!;
    if (data is null) throw new FileNotFoundException("File uncorrected!");

    Solve task = args[3] switch
    {
        "iteration" => new Solve(data, Method.Iteration, uint.Parse(args[5])),
```

```
        "newton"    => new Solve(data, Method.Newton,    uint.Parse(args[5])),
        _           => new Solve(data, Method.Iteration, uint.Parse(args[5]))
    };
    task.SetPath(Path.GetDirectoryName(args[1])!);
    task.solve();
}
catch (FileNotFoundException ex) {
    WriteLine(ex.Message);
}
catch (ArgumentException ex) {
    ShowHelp();
    WriteLine(ex.Message);
}
```

Данные задачи (тест с константой):

$$
\begin{cases}
u = 2.5 \\
f = 5 \\
\gamma = 2,\ \beta = 1 \\
\lambda(u') = u' + 4 \\
Кол.\ КЭ : 5 \\
Сетка : [0;1] \\
Коеф.\quad k = 1 \\
Краевые\ условия : [1,1]
\end{cases}
$$

Метод простой итерации:

U	U`	|U`- U|
2,500000E+000	2,500000E+000	0,000000E+000
2,500000E+000	2,500000E+000	8,881784E-016
2,500000E+000	2,500000E+000	8,881784E-016
2,500000E+000	2,500000E+000	8,881784E-016
2,500000E+000	2,500000E+000	4,440892E-016
2,500000E+000	2,500000E+000	0,000000E+000

Количество итераций: 1

Метод Ньютона:

U	U`	|U`- U|
2,500000E+000	2,500000E+000	0,000000E+000
2,500000E+000	2,500000E+000	8,881784E-016
2,500000E+000	2,500000E+000	8,881784E-016
2,500000E+000	2,500000E+000	8,881784E-016
2,500000E+000	2,500000E+000	4,440892E-016
2,500000E+000	2,500000E+000	0,000000E+000

Количество итераций: 1

Данные задачи (Полином первой степени):

$$
\begin{cases}
u = x \\
f = 2x \\
\gamma = 2,\ \beta = 1 \\
\lambda(u') = u' + 4 \\
Кол.\ КЭ : 5 \\
Сетка : [0;1] \\
Коеф.\quad k = 1 \\
Краевые\ условия : [1,1]
\end{cases}
$$

Метод простой итерации:

U	U`	|U`- U|
0,000000E+000	0,000000E+000	0,000000E+000
2,000000E-001	2,000000E-001	5,551115E-017
4,000000E-001	4,000000E-001	1,110223E-016
6,000000E-001	6,000000E-001	0,000000E+000
8,000000E-001	8,000000E-001	0,000000E+000
1,000000E+000	1,000000E+000	0,000000E+000

Количество итераций: 1

Метод Ньютона:

U	U`	|U`- U|
0,000000E+000	0,000000E+000	0,000000E+000
2,000000E-001	2,000000E-001	5,551115E-017
4,000000E-001	4,000000E-001	1,110223E-016
6,000000E-001	6,000000E-001	0,000000E+000
8,000000E-001	8,000000E-001	0,000000E+000
1,000000E+000	1,000000E+000	0,000000E+000

Количество итераций: 1

Данные задачи (Полином второй степени):

$$
\begin{cases}
u = x^2 \\
f = 2x^2 - 8x - 8 \\
\gamma = 2,\ \beta = 1 \\
\lambda(u') = u' + 4 \\
Кол.\ КЭ : 5 \\
Сетка : [0;1] \\
Коеф.\quad k = 1 \\
Краевые\ условия : [1,1]
\end{cases}
$$

Метод простой итерации:

U	U`	|U`- U|
0,000000E+000	0,000000E+000	0,000000E+000
4,000000E-002	4,000000E-002	3,030076E-013
1,600000E-001	1,600000E-001	7,352452E-013
3,600000E-001	3,600000E-001	1,496525E-012
6,400000E-001	6,400000E-001	3,664513E-012
1,000000E+000	1,000000E+000	0,000000E+000

Количество итераций: 20

Метод Ньютона:

U	U`	|U`- U|
0,000000E+000	0,000000E+000	0,000000E+000
4,000000E-002	4,000000E-002	1,517494E-012
1,600000E-001	1,600000E-001	1,706774E-012
3,600000E-001	3,600000E-001	6,171286E-012
6,400000E-001	6,400000E-001	1,045175E-011
1,000000E+000	1,000000E+000	0,000000E+000

Количество итераций: 4

Данные задачи (Полином третьей степени):

$$
\begin{cases}
u = x^3 \\
f = -34x^3 - 24x \\
\gamma = 2,\ \beta = 1 \\
\lambda(u') = u' + 4 \\
Кол.\ КЭ : 5 \\
Сетка : [0;1] \\
Коеф.\quad k = 1 \\
Краевые\ условия : [1,1]
\end{cases}
$$

Метод простой итерации:

U	U`	|U`- U|
0,000000E+000	0,000000E+000	0,000000E+000
2,454818E-003	8,000000E-003	5,545182E-003
5,510813E-002	6,400000E-002	8,891870E-003
2,070152E-001	2,160000E-001	8,984788E-003
5,061477E-001	5,120000E-001	5,852274E-003
1,000000E+000	1,000000E+000	0,000000E+000

Количество итераций: 23

Метод Ньютона:

U	U`	|U`- U|
0,000000E+000	0,000000E+000	0,000000E+000
2,454818E-003	8,000000E-003	5,545182E-003
5,510813E-002	6,400000E-002	8,891870E-003
2,070152E-001	2,160000E-001	8,984788E-003
5,061477E-001	5,120000E-001	5,852274E-003
1,000000E+000	1,000000E+000	0,000000E+000

Количество итераций: 5

Данные задачи (Не полиноминальная функция):

$u = \sin(2x)$ $f = 16\sin(2x)\cos(2x)$ $\gamma = -16,\ \beta = 1$ $\lambda(u') = u' + 4$ *Кол. КЭ* : 5 *Сетка* : $[0;1]$ *Коеф.* $k = 1$ *Краевые условия* : $[1,1]$	$u = \sin(2x)$ $f = 16\sin(2x)\cos(2x)$ $\gamma = -16,\ \beta = 1$ $\lambda(u') = u' + 4$ *Кол. КЭ* : 10 *Сетка* : $[0;1]$ *Коеф.* $k = 1$ *Краевые условия* : $[1,1]$	$u = \sin(2x)$ $f = 16\sin(2x)\cos(2x)$ $\gamma = -16,\ \beta = 1$ $\lambda(u') = u' + 4$ *Кол. КЭ* : 20 *Сетка* : $[0;1]$ *Коеф.* $k = 1$ *Краевые условия* : $[1,1]$

Метод простой итерации:

U(5)	U(10)	U(20)	U`
3,810239E-001	3,872491E-001	3,888715E-001	3,894183E-001

|U` - U(5)|	|U` - U(10)|	|U` - U(20)|
8,394444E-003	2,169228E-003	5,468633E-004

(Количество КЭ -> Количество итераций)
5 -> 24; 10 -> 25; 20 -> 25

Метод Ньютона:

U(5)	U(10)	U(20)	U`
3,810239E-001	3,872491E-001	3,888715E-001	3,894183E-001

|U` - U(5)|	|U` - U(10)|	|U` - U(20)|
8,394444E-003	2,169228E-003	5,468633E-004

(Количество КЭ -> Количество итераций)
5 -> 5; 10 -> 5; 20 -> 5

Прогоним некоторые тесты на неравномерной сетке

Тест с полиномом первой степени

Сетка (k=1.4):

x1	x2	h
0	0,0913609121473469	0,0913609121473469
0,0913609121473469	0,21926618915363255	0,12790527700628565
0,21926618915363255	0,39833357696243243	0,1790673878087999
0,39833357696243243	0,6490279198947523	0,25069434293231985
0,6490279198947523	1	0,3509720801052478

Метод простой итерации:

U	U`	|U`- U|
0,000000E+000	0,000000E+000	0,000000E+000
9,136091E-002	9,136091E-002	2,775558E-017
2,192662E-001	2,192662E-001	2,775558E-017
3,983336E-001	3,983336E-001	0,000000E+000
6,490279E-001	6,490279E-001	0,000000E+000
1,000000E+000	1,000000E+000	0,000000E+000

Количество итераций: 1

Метод Ньютона:

U	U`	|U`- U|
0,000000E+000	0,000000E+000	0,000000E+000
9,136091E-002	9,136091E-002	2,775558E-017
2,192662E-001	2,192662E-001	2,775558E-017
3,983336E-001	3,983336E-001	0,000000E+000
6,490279E-001	6,490279E-001	0,000000E+000
1,000000E+000	1,000000E+000	0,000000E+000

Количество итераций: 1

Тест с полиномом второй степени

Сетка такая же

Метод простой итерации:

U	U`	|U`- U|
0,000000E+000	0,000000E+000	0,000000E+000
7,989164E-003	8,346816E-003	3,576523E-004
4,733860E-002	4,807766E-002	7,390652E-004
1,576307E-001	1,586696E-001	1,038965E-003
4,202423E-001	4,212372E-001	9,949189E-004
1,000000E+000	1,000000E+000	0,000000E+000

Количество итераций: 18

Метод Ньютона:

U	U`	|U`- U|
0,000000E+000	0,000000E+000	0,000000E+000
7,989164E-003	8,346816E-003	3,576523E-004
4,733860E-002	4,807766E-002	7,390652E-004
1,576307E-001	1,586696E-001	1,038965E-003
4,202423E-001	4,212372E-001	9,949189E-004
1,000000E+000	1,000000E+000	0,000000E+000

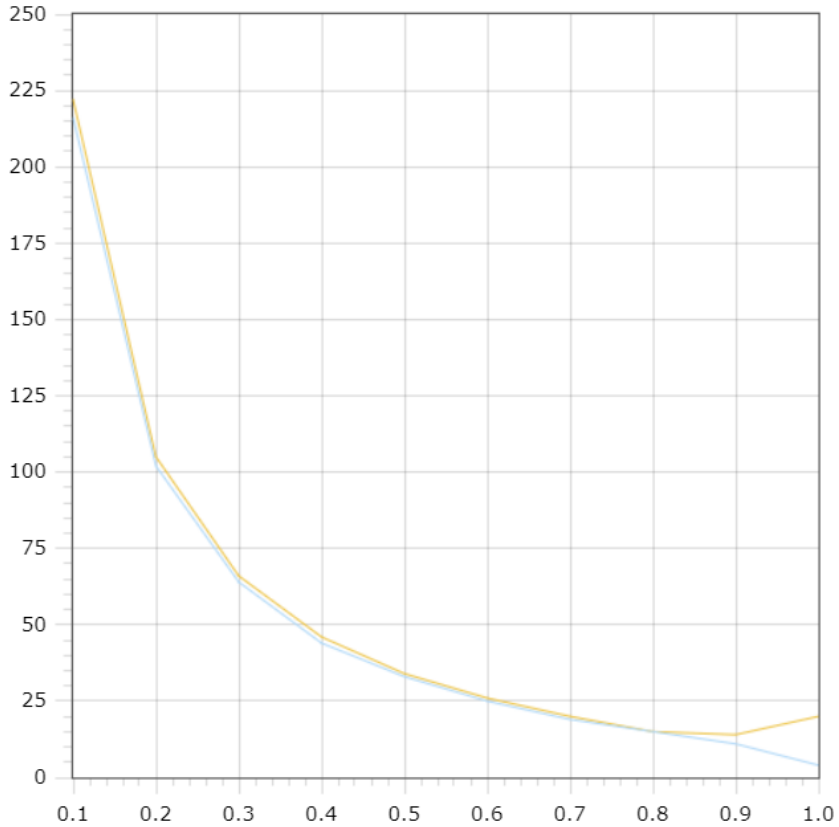
Количество итераций: 4

Проведем сходимость от параметра релаксации

На тесте с полиномом второй степени, на равномерной сетке:

ω	Количество итераций	
	Метод простой итерации	Метод Ньютона
0.1	222	216
0.2	105	102
0.3	66	64
0.4	46	44
0.5	34	33
0.6	26	25
0.7	20	19
0.8	15	15
0.9	14	11
1	20	4

График



Вывод

Метод Ньютона решает быстрее нелинейную систему по количеству итераций чем метод простой итерации. Но построение матрицы и получение формул у метода Ньютона значительно сложнее.