Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Уравнения математической физики

Лабораторная работа №2

Факультет	ПМИ
Группа	ПМ-01
Студент	Жигалов П.С.
Преподаватель	Задорожный А.Г.
	Персова М.Г.
Вариант	13

Новосибирск

2013

1. Цель работы

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

2. Задание

Уравнение: $-\operatorname{div}\left(\lambda \operatorname{grad} u\right)+\gamma u=f\left(u\right)$. Базисные функции линейные

3. Анализ

3.1. Вариационная постановка и дискретизация

Для нелинейной задачи $-\operatorname{div}\left(\lambda \operatorname{grad} u\right)+\gamma u=f\left(u\right)$ (1) выполним вариационную постановку Галеркина: скалярно умножим правую и левую часть на пробную функцию $v \in H_0$, H_0 - множество функций, удовлетворяющих первым краевым условиям. $\int\limits_{\Omega} \lambda gradu \cdot gradv \cdot d\Omega - \int\limits_{S_1} \lambda \frac{\partial u}{\partial n} vdS - \int\limits_{S_2} \theta vdS - \int\limits_{S_3} \beta\left(u-u_\beta\right)vdS + \int\limits_{\Omega}\left(\gamma u - f\left(u\right)\right)vd\Omega = 0$ (2).

Перейдем от гильбертова пространства H к конечномерному пространству V^h, которое определим как линейное пространство, натянутое на базисные функции $\psi_i,\ i=1...n$. Заменим в функцию u аппроксимирующей ее функцией u^h, а функцию v - функцией v^h и получим аппроксимацию уравнения Галеркина:

$$\int\limits_{\Omega} \lambda gradu^h \cdot gradv^h \cdot d\Omega + \int\limits_{\Omega} \gamma u^h v^h d\Omega + \int\limits_{S_3} \beta u^h v^h dS = \int\limits_{\Omega} f\left(u^h\right) v^h d\Omega + \int\limits_{S_2} \theta v^h dS + \int\limits_{S_3} \beta u_\beta v^h dS \quad (3).$$

Поскольку любая функция v^h может быть представлена в виде линейной комбинации $v^h = \sum\limits_{i=1}^{n} q_i \psi_i$, вариационное уравнение (3) эквивалентно следующему:

$$\int\limits_{\Omega} \lambda gradu^h \cdot grad\psi_i \cdot d\Omega + \int\limits_{\Omega} \gamma u^h \psi_i d\Omega + \int\limits_{S_3} \beta u^h \psi_i dS = \int\limits_{\Omega} f\left(u^h\right) \psi_i d\Omega + \int\limits_{S_2} \theta \psi_i dS + \int\limits_{S_3} \beta u_\beta \psi_i dS \quad (4)$$

Таким образом, МКЭ-решение u^h удовлетворяет (4). Оно может быть представлено в виде: $u^h = \sum\limits_{j=1}^{n} q_j \psi_j$ (5). Подставляя (5) в (4) получим СЛАУ $Aq = b\left(q\right)$, где:

$$A_{ij} = \int\limits_{\Omega} \lambda grad\psi_j \cdot grad\psi_i \cdot d\Omega + \int\limits_{\Omega} \gamma \psi_j \psi_i d\Omega + \int\limits_{S_3} \beta \psi_j \psi_i dS \quad (6)$$

$$b_i\left(q\right) = \int\limits_{\Omega} f\left(u^h\left(q\right)\right) \psi_i d\Omega + \int\limits_{S_2} \theta \psi_i dS + \int\limits_{S_3} \beta u_\beta \psi_i dS \quad (7)$$

3.2. Вычисление компонент матрицы и вектора правой части для метода простой итерации

Поскольку случай одномерный, линейные базисные функции на конечном элементе могут быть записаны в виде: $\psi_1 = \frac{x_{i+1}-x}{h_x}$, $\psi_2 = \frac{x - x_i}{h_x}$, где $h_x = x_{i+1} - x_i$.

Матрица A представляется в виде $A_{i,j} = \int\limits_a^b \lambda \cdot \frac{\partial \psi_i}{\partial x} \cdot \frac{\partial \psi_j}{\partial x} dx + \int\limits_a^b \gamma \psi_i \psi_j dx$ (в случае третьих краевых добавляется добавка из β), ее можно представить в виде сборки из локальных матриц.

Вид локальной матрицы будет следующим: $\hat{A}_{i,j} = \int\limits_a^b \lambda_k \cdot \frac{\partial \hat{\psi}_i}{\partial x} \cdot \frac{\partial \hat{\psi}_j}{\partial x} dx + \int\limits_a^b \gamma_k \hat{\psi}_i \hat{\psi}_j dx = \hat{G}_{i,j} + \hat{M}_{i,j}$ (8).

$$\hat{G}_{i,j}=\int_a^b \lambda_k \cdot \frac{\partial \hat{\psi}_i}{\partial x} \cdot \frac{\partial \hat{\psi}_j}{\partial x}dx \ , \ \hat{G}=\frac{\lambda_k}{h_k}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$ (9) - локальная матрица жесткости,

$$\hat{M}_{i,j}=\int_a^b \gamma_k \hat{\psi}_i \hat{\psi}_j dx \ , \ \hat{M}=\gamma_k h_k \begin{pmatrix} 1/3 & 1/6 \\ 1/6 & 1/3 \end{pmatrix}=\frac{\gamma_k h_k}{6}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$ (10) - локальная матрица массы.

Вектор правых частей также можно представить в виде $b_i(q)=\int_a^b f\left(u^h(q)\right)\psi_i dx$ (в случае первых краевых добавляется добавка из u_g, в случае вторых – из θ, в случае третьих – из βu_β), его также можно представить в виде сборки из локальных векторов.

Вид локального вектора будет следующим: $\hat{b}_i=\int_a^b f\left(u^h(q)\right)\hat{\psi}_i dx$, $\hat{b}=\begin{pmatrix} \int_a^b f\left(u^h\right)\hat{\psi}_1 dx \\ \int_a^b f\left(u^h\right)\hat{\psi}_2 dx \end{pmatrix}$. Заменяя $f\left(u^h(q)\right)$ линейным интерполянтом, получим $f\left(u^h(q)\right)\sim f^k\left(u^h(q,x_k)\right)\hat{\psi}_1(x)+f^k\left(u^h(q,x_{k+1})\right)\hat{\psi}_2(x)$. Тогда:

$$\hat{b}=\begin{pmatrix} \int_a^b \left(f^k\left(u^h(q,x_k)\right)\hat{\psi}_1+f^k\left(u^h(q,x_{k+1})\right)\hat{\psi}_2\right)\hat{\psi}_1 dx \\ \int_a^b \left(f^k\left(u^h(q,x_k)\right)\hat{\psi}_1+f^k\left(u^h(q,x_{k+1})\right)\hat{\psi}_2\right)\hat{\psi}_2 dx \end{pmatrix}=\begin{pmatrix} \frac{h_k}{6}\left(2f^k\left(u^h(q,x_k)\right)+f^k\left(u^h(q,x_{k+1})\right)\right) \\ \frac{h_k}{6}\left(f^k\left(u^h(q,x_k)\right)+2f^k\left(u^h(q,x_{k+1})\right)\right) \end{pmatrix}=$$

$$=\frac{h_k}{6}\begin{pmatrix} 2f^k\left(u^h(q,x_k)\right)+f^k\left(u^h(q,x_{k+1})\right) \\ f^k\left(u^h(q,x_k)\right)+2f^k\left(u^h(q,x_{k+1})\right) \end{pmatrix}$$, (11) так как $\int_a^b \hat{\psi}_1^2 dx=\frac{h_k}{3}$, $\int_a^b \hat{\psi}_1\hat{\psi}_2 dx=\frac{h_k}{6}$, $\int_a^b \hat{\psi}_2^2 dx=\frac{h_k}{3}$.

3.3. Вычисление компонент матрицы и вектора правой части для метода Ньютона

Так как исходно левая часть уравнения линейна, формирование локальных линеаризованных матрицы и вектора в методе Ньютона происходит по следующим формулам:

$$\hat{A}^L_{i,j}=\hat{A}_{i,j}\left(\hat{q}^0\right)-\left.\frac{\partial \hat{b}_i(\hat{q})}{\partial \hat{q}_j}\right|_{\hat{q}=\hat{q}^0} \quad (12), \qquad \hat{b}^L_i=\hat{b}_i\left(\hat{q}^0\right)-\sum_{r=1}^{\hat{n}}\left.\frac{\partial \hat{b}_i(\hat{q})}{\partial \hat{q}_r}\right|_{\hat{q}=\hat{q}^0}\hat{q}^0_r \quad (13).$$

Так как $\hat{f}^k$ можно представить в виде $\hat{f}^k\left(u^h(\hat{q},\hat{x})\right)=\hat{f}^k\left(u^h(\hat{q},\hat{x}_1)\right)\psi_1+\hat{f}^k\left(u^h(\hat{q},\hat{x}_2)\right)\psi_2$, то $\hat{b}_i(\hat{q})$ можно представить в виде $\hat{b}_i(\hat{q})=\int_a^b\left(\hat{f}^k\left(u^h(\hat{q},\hat{x}_1)\right)\psi_1+\hat{f}^k\left(u^h(\hat{q},\hat{x}_2)\right)\psi_2\right)\psi_i dx$, откуда

$$\left.\frac{\partial \hat{b}_i(\hat{q})}{\partial \hat{q}_r}\right|_{\hat{q}=\hat{q}^0}=\int_a^b\left(\left.\frac{\partial \hat{f}^k\left(u^h(\hat{q},\hat{x}_1)\right)}{\partial \hat{q}_r}\right|_{\hat{q}=\hat{q}^0}\psi_1+\left.\frac{\partial \hat{f}^k\left(u^h(\hat{q},\hat{x}_2)\right)}{\partial \hat{q}_r}\right|_{\hat{q}=\hat{q}^0}\psi_2\right)\psi_i dx \quad (14).$$

Найдем $\left.\frac{\partial \hat{f}^k\left(u^h(\hat{q},\hat{x}_l)\right)}{\partial \hat{q}_r}\right|_{\hat{q}=\hat{q}^0}$ как производную сложной функции:

$$\left.\frac{\partial \hat{f}^k\left(u^h(\hat{q},\hat{x}_l)\right)}{\partial \hat{q}_r}\right|_{\hat{q}=\hat{q}^0}=\left.\frac{\partial \hat{f}^k\left(u^h(\hat{q},\hat{x}_l)\right)}{\partial u}\right|_{u=u^h(\hat{q},\hat{x}_l)}\cdot\left.\frac{\partial u^h(\hat{q},\hat{x}_l)}{\partial \hat{q}_r}\right|_{\hat{q}=\hat{q}^0}, \text{ где } \left.\frac{\partial u^h(\hat{q},\hat{x}_l)}{\partial \hat{q}_r}\right|_{\hat{q}=\hat{q}^0}=\begin{cases} 1, l=r \\ 0, l\neq r \end{cases}$$

Тогда части добавок будут выглядеть следующим образом:

$$\left.\frac{\partial \hat{b}_1(\hat{q})}{\partial \hat{q}_1}\right|_{\hat{q}=\hat{q}^0} = \frac{h_k}{3}\left(\left.\frac{\partial \hat{f}^k\left(u^h(\hat{q},\hat{x}_1)\right)}{\partial u}\right|_{u=u^h(\hat{q},\hat{x}_1)}\right) \qquad \left.\frac{\partial \hat{b}_1(\hat{q})}{\partial \hat{q}_2}\right|_{\hat{q}=\hat{q}^0} = \frac{h_k}{6}\left(\left.\frac{\partial \hat{f}^k\left(u^h(\hat{q},\hat{x}_2)\right)}{\partial u}\right|_{u=u^h(\hat{q},\hat{x}_2)}\right)$$

$$\left.\frac{\partial \hat{b}_2(\hat{q})}{\partial \hat{q}_1}\right|_{\hat{q}=\hat{q}^0} = \frac{h_k}{6}\left(\left.\frac{\partial \hat{f}^k\left(u^h(\hat{q},\hat{x}_1)\right)}{\partial u}\right|_{u=u^h(\hat{q},\hat{x}_1)}\right) \qquad \left.\frac{\partial \hat{b}_2(\hat{q})}{\partial \hat{q}_2}\right|_{\hat{q}=\hat{q}^0} = \frac{h_k}{3}\left(\left.\frac{\partial \hat{f}^k\left(u^h(\hat{q},\hat{x}_2)\right)}{\partial u}\right|_{u=u^h(\hat{q},\hat{x}_2)}\right)$$

Их необходимо подставить в формулы (12) и (13).

4. Исследования и тесты

4.1. Решение – полином первой степени

Уравнение: $-\operatorname{div}\operatorname{grad} u + u = u$, решение: $u = x + 2$, сетка: $[0,1]$ с шагом $h = 0.1$, краевые условия: первые-первые, коэффициент релаксации $\omega = 1$, целевая невязка 10^{-10} .

x	$u*$	Простой итерации	Ньютона	$\left\|u*-u^{si}\right\|$	$\left\|u*-u^{N}\right\|$
0.00	2.00000000000000	2.00000000000000	2.00000000000000	0.000E+00	0.000E+00
0.10	2.10000000000000	2.09999999999634	2.09999999999738	3.660E-12	2.620E-12
0.20	2.20000000000000	2.19999999999305	2.19999999999510	6.950E-12	4.900E-12
0.30	2.30000000000000	2.29999999999043	2.29999999999337	9.570E-12	6.630E-12
0.40	2.40000000000000	2.39999999998875	2.39999999999233	1.125E-11	7.670E-12
0.50	2.50000000000000	2.49999999998817	2.49999999999207	1.183E-11	7.930E-12
0.60	2.60000000000000	2.59999999998875	2.59999999999258	1.125E-11	7.420E-12
0.70	2.70000000000000	2.69999999999043	2.69999999999379	9.570E-12	6.210E-12
0.80	2.80000000000000	2.79999999999304	2.79999999999556	6.960E-12	4.440E-12
0.90	2.90000000000000	2.89999999999634	2.89999999999770	3.660E-12	2.300E-12
1.00	3.00000000000000	3.00000000000000	3.00000000000000	0.000E+00	0.000E+00
Итераций		11	9		
Невязка		7.688E-11	9.955E-11		
Погрешность		3.165E-12	2.124E-12		

4.2. Решение – полином второй степени

Уравнение: $-\operatorname{div}\operatorname{grad} u + u = u - 4$, решение: $u = 2x^2$, сетка: $[0,1]$ с шагом $h = 0.1$, краевые условия: первые-первые, коэффициент релаксации $\omega = 1$, целевая невязка 10^{-10} .

x	$u*$	Простой итерации	Ньютона	$\left\|u*-u^{si}\right\|$	$\left\|u*-u^{N}\right\|$
0.00	0.00000000000000	0.00000000000000	0.00000000000000	0.000E+00	0.000E+00
0.10	0.02000000000000	0.01999999999913	0.01999999999936	8.739E-13	6.412E-13
0.20	0.08000000000000	0.07999999999834	0.07999999999880	1.662E-12	1.200E-12
0.30	0.18000000000000	0.17999999999771	0.17999999999838	2.288E-12	1.625E-12
0.40	0.32000000000000	0.31999999999731	0.31999999999812	2.690E-12	1.879E-12
0.50	0.50000000000000	0.49999999999717	0.49999999999806	2.829E-12	1.944E-12
0.60	0.72000000000000	0.71999999999731	0.71999999999818	2.691E-12	1.819E-12
0.70	0.98000000000000	0.97999999999771	0.97999999999848	2.289E-12	1.523E-12
0.80	1.28000000000000	1.27999999999833	1.27999999999891	1.670E-12	1.090E-12
0.90	1.62000000000000	1.61999999999912	1.61999999999943	8.802E-13	5.702E-13
1.00	2.00000000000000	2.00000000000000	2.00000000000000	0.000E+00	0.000E+00
Итераций		11	9		
Невязка		3.017E-11	3.999E-11		
Погрешность		1.988E-12	1.367E-12		

4.3. Решение – полином третьей степени

Уравнение: $-\operatorname{div}\operatorname{grad} u + u = -6\sqrt[3]{u} + u$, решение: $u = x^3$, сетка: $\left[0,1\right]$ с шагом $h = 0.1$, краевые условия: первые-первые, коэффициент релаксации $\omega = 1$, целевая невязка 10^{-10} .

x	u^*	Простой итерации	Ньютона	$\lvert u^* - u^{si}\rvert$	$\lvert u^* - u^N\rvert$
0.00	0.00000000000000	0.00000000000000	0.00000000000000	0.000E+00	0.000E+00
0.10	0.00100000000000	0.13078411087609	0.00099999999889	1.298E-01	1.111E-12
0.20	0.00800000000000	0.24260922847276	0.00799999999902	2.346E-01	9.814E-13
0.30	0.02700000000000	0.33002143332642	0.02699999999908	3.030E-01	9.218E-13
0.40	0.06400000000000	0.39330433888995	0.06399999999918	3.293E-01	8.199E-13
0.50	0.12500000000000	0.43647689268848	0.12499999999931	3.115E-01	6.950E-13
0.60	0.21600000000000	0.47446580699731	0.21599999999944	2.585E-01	5.580E-13
0.70	0.34300000000000	0.53830872171540	0.34299999999958	1.953E-01	4.179E-13
0.80	0.51200000000000	0.64239062670151	0.51199999999972	1.304E-01	2.771E-13
0.90	0.72900000000000	0.79415635148477	0.72899999999986	6.516E-02	1.381E-13
1.00	1.00000000000000	1.00000000000000	1.00000000000000	0.000E+00	0.000E+00
Итераций		100001	35		
Невязка		9.798E-01	5.181E-11		
Погрешность		5.004E-01	1.555E-12		

4.4. Решение – не полином (синус)

Уравнение: $-\operatorname{div}\operatorname{grad} u + u = 2u$, решение: $u = \sin\left(x\right)$, сетка: $\left[0,2\right]$ с шагом $h = 0.2$, краевые условия: первые-первые, коэффициент релаксации $\omega = 1$, целевая невязка 10^{-10} .

x	u^*	Простой итерации	Ньютона	$\lvert u^* - u^{si}\rvert$	$\lvert u^* - u^N\rvert$
0.00	0.00000000000000	0.00000000000000	0.00000000000000	0.000E+00	0.000E+00
0.20	0.19866933079506	0.19804432410100	0.19804432411017	6.250E-04	6.250E-04
0.40	0.38941834230865	0.38821933731183	0.38821933732937	1.199E-03	1.199E-03
0.60	0.56464247339504	0.56296841659214	0.56296841661631	1.674E-03	1.674E-03
0.80	0.71735609089952	0.71534788991844	0.71534788994673	2.008E-03	2.008E-03
1.00	0.84147098480790	0.83930294378166	0.83930294381106	2.168E-03	2.168E-03
1.20	0.93203908596723	0.92990821180332	0.92990821183064	2.131E-03	2.131E-03
1.40	0.98544972998846	0.98356348465787	0.98356348468006	1.886E-03	1.886E-03
1.60	0.99957360304151	0.99813676474786	0.99813676476236	1.437E-03	1.437E-03
1.80	0.97384763087820	0.97304898134033	0.97304898134714	7.986E-04	7.986E-04
2.00	0.90929742682568	0.90929700000000	0.90929700000000	4.268E-07	4.268E-07
Итераций		42	29		
Невязка		6.145E-11	5.188E-11		
Погрешность		1.951E-03	1.951E-03		

4.5. Решение – не полином (экспонента)

Уравнение: $-\operatorname{div}\operatorname{grad} u + u = 0$, решение: $u = e^x$, сетка: $\left[0,10\right]$ с шагом $h = 1$, краевые условия: первые-первые, коэффициент релаксации $\omega = 1$, целевая невязка 10^{-10} .

x	u^*	Простой итерации	Ньютона	$\lvert u^* - u^{si}\rvert$	$\lvert u^* - u^N\rvert$
0.00	1.000000000000E+00	1.000000000000E+00	1.000000000000E+00	0.000E+00	0.000E+00
1.00	2.718281828459E+00	1.912756574388E+00	1.912756574388E+00	8.055E-01	8.055E-01
2.00	7.389056098931E+00	5.120821038041E+00	5.120821038041E+00	2.268E+00	2.268E+00
3.00	2.008553692319E+01	1.447387074734E+01	1.447387074734E+01	5.612E+00	5.612E+00
4.00	5.459815003314E+01	4.119556535346E+01	4.119556535346E+01	1.340E+01	1.340E+01
5.00	1.484131591026E+02	1.173519383837E+02	1.173519383837E+02	3.106E+01	3.106E+01
6.00	4.034287934927E+02	3.343306374745E+02	3.343306374745E+02	6.910E+01	6.910E+01
7.00	1.096633158428E+03	9.525061015346E+02	9.525061015346E+02	1.441E+02	1.441E+02
8.00	2.980957987042E+03	2.713688887436E+03	2.713688887436E+03	2.673E+02	2.673E+02
9.00	8.103083927575E+03	7.731298338261E+03	7.731298338261E+03	3.718E+02	3.718E+02
10.00	2.202646579481E+04	2.202646579500E+04	2.202646579500E+04	1.933E-07	1.933E-07
Итераций		2	2		
Невязка		6.145E-11	5.188E-11		
Погрешность		2.053E-02	2.053E-02		

4.6. Исследование на вложенных сетках

Уравнение: $-\operatorname{div}\operatorname{grad} u + u = 2u$, решение: $u = \cos(x)$, сетка: $[0,2]$ с шагом $h = 0.2 \quad h/2 = 0.1 \; h/4 = 0.05$, краевые условия: первые-первые, коэффициент релаксации $\omega = 1$, целевая невязка 10^{-10} .

Метод простой итерации:

x	u^*	u^h	$u^{h/2}$	$u^{h/4}$	$\lvert u^*-u^h\rvert$	$\lvert u^*-u^{h/2}\rvert$	$\lvert u^*-u^{h/4}\rvert$
0.00	1.00000000	1.00000000	1.00000000	1.00000000	0.00E+00	0.00E+00	0.00E+00
0.20	0.98006658	0.97947473	0.97991780	0.98002931	5.92E-04	1.49E-04	3.73E-05
0.40	0.92106099	0.92002993	0.92080178	0.92099605	1.03E-03	2.59E-04	6.49E-05
0.60	0.82533561	0.82402765	0.82500674	0.82525321	1.31E-03	3.29E-04	8.24E-05
0.80	0.69670671	0.69528255	0.69634857	0.69661695	1.42E-03	3.58E-04	8.98E-05
1.00	0.54030231	0.53891032	0.53995221	0.54021454	1.39E-03	3.50E-04	8.78E-05
1.20	0.36235775	0.36112445	0.36204751	0.36227995	1.23E-03	3.10E-04	7.78E-05
1.40	0.16996714	0.16898926	0.16972109	0.16990541	9.78E-04	2.46E-04	6.17E-05
1.60	-0.02919952	-0.02986074	-0.02936596	-0.02924133	6.61E-04	1.66E-04	4.18E-05
1.80	-0.22720209	-0.22752422	-0.22728326	-0.22722255	3.22E-04	8.12E-05	2.05E-05
2.00	-0.41614684	-0.41614700	-0.41614700	-0.41614700	1.63E-07	1.63E-07	1.63E-07
Итераций		41	41	40			
Невязка		6.829E-11	6.222E-11	8.393E-11			
Погреш-ность		1.474E-03	3.707E-04	9.293E-05			

$\log_2 \dfrac{\lVert u^*-u_h\rVert}{\lVert u^*-u_{h/2}\rVert} \approx 1.99$, $\qquad \log_2 \dfrac{\lVert u^*-u_{h/2}\rVert}{\lVert u^*-u_{h/4}\rVert} \approx 2.00$

Метод Ньютона:

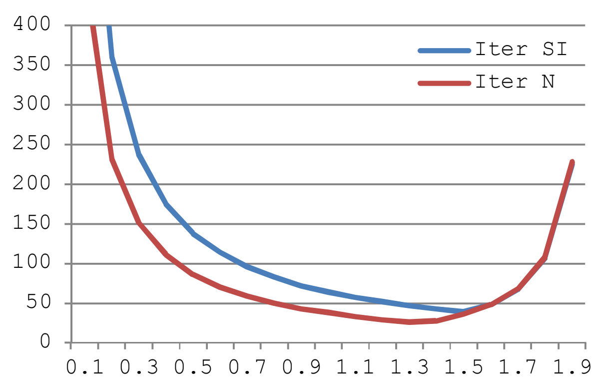
x	u^*	u^h	$u^{h/2}$	$u^{h/4}$	$\lvert u^*-u^h\rvert$	$\lvert u^*-u^{h/2}\rvert$	$\lvert u^*-u^{h/4}\rvert$
0.00	1.00000000	1.00000000	1.00000000	1.00000000	0.00E+00	0.00E+00	0.00E+00
0.20	0.98006658	0.97947473	0.97991780	0.98002931	5.92E-04	1.49E-04	3.73E-05
0.40	0.92106099	0.92002993	0.92080178	0.92099605	1.03E-03	2.59E-04	6.49E-05
0.60	0.82533561	0.82402765	0.82500674	0.82525321	1.31E-03	3.29E-04	8.24E-05
0.80	0.69670671	0.69528255	0.69634857	0.69661695	1.42E-03	3.58E-04	8.98E-05
1.00	0.54030231	0.53891032	0.53995221	0.54021454	1.39E-03	3.50E-04	8.78E-05
1.20	0.36235775	0.36112445	0.36204751	0.36227995	1.23E-03	3.10E-04	7.78E-05
1.40	0.16996714	0.16898926	0.16972109	0.16990541	9.78E-04	2.46E-04	6.17E-05
1.60	-0.02919952	-0.02986074	-0.02936596	-0.02924133	6.61E-04	1.66E-04	4.18E-05
1.80	-0.22720209	-0.22752422	-0.22728326	-0.22722255	3.22E-04	8.12E-05	2.05E-05
2.00	-0.41614684	-0.41614700	-0.41614700	-0.41614700	1.63E-07	1.63E-07	1.63E-07
Итераций		55	55	55			
Невязка		6.829E-11	8.890E-11	6.916E-11			
Погреш-ность		1.474E-03	3.707E-04	9.293E-05			

$\log_2 \dfrac{\lVert u^*-u_h\rVert}{\lVert u^*-u_{h/2}\rVert} \approx 1.99$, $\qquad \log_2 \dfrac{\lVert u^*-u_{h/2}\rVert}{\lVert u^*-u_{h/4}\rVert} \approx 2.00$

4.7. Исследование зависимости сходимости от параметра релаксации

Уравнение: $-\operatorname{div}\operatorname{grad} u + u = u - 4$, решение: $u = 2x^2$, сетка: $[0,5]$ с шагом $h = 1$, краевые условия: первые-первые, целевая невязка 10^{-10} .

ω	Итер.SI	Итер.N	ω	Итер.SI	Итер.N
0.20	360	231	1.10	57	33
0.30	237	151	1.20	52	29
0.40	175	111	1.30	47	26
0.50	138	86	1.40	43	28
0.60	114	70	1.50	39	36
0.70	96	59	1.60	48	48
0.80	83	50	1.70	67	68
0.90	72	43	1.80	107	108
1.00	64	38	1.90	225	228



4.8. Неравномерная сетка с краевыми условиями третьего рода

Уравнение: $-\operatorname{div}\operatorname{grad} u + u = 2u$, решение: $u = \sin(x)$, сетка: $[0,2]$ с начальным шагом $h_0 = 0.05$ и коэффициентом разрядки $k = 1.3$, краевые условия: третьи-первые, коэффициент релаксации $\omega = 1$, целевая невязка 10^{-10} .

Третьи краевые условия: $\left.\frac{\partial u}{\partial n}\right|_{x=0} + u\big|_{x=0} - 1 = 0$

x	u^*	Простой итерации	Ньютона	$\lvert u^* - u^{si}\rvert$	$\lvert u^* - u^N\rvert$
0.00	0.00000000000000	-0.01399160235724	-0.01399160213121	1.399E-02	1.399E-02
0.05	0.04997916927068	0.03530576645776	0.03530576669467	1.467E-02	1.467E-02
0.12	0.11474668839366	0.09924207154255	0.09924207179240	1.550E-02	1.550E-02
0.20	0.19817927269289	0.18169282653996	0.18169282680452	1.649E-02	1.649E-02
0.31	0.30443955522297	0.28685535241621	0.28685535269613	1.758E-02	1.758E-02
0.45	0.43690498611986	0.41822016523857	0.41822016553170	1.868E-02	1.868E-02
0.64	0.59543099372021	0.57591875905608	0.57591875935445	1.951E-02	1.951E-02
0.88	0.77019191958923	0.75073264183311	0.75073264211781	1.946E-02	1.946E-02
1.19	0.92943734098401	0.91217757522179	0.91217757545569	1.726E-02	1.726E-02
1.60	0.99955142227177	0.98912975320628	0.98912975332818	1.042E-02	1.042E-02
2.00	0.90929742682568	0.90929700000000	0.90929700000000	4.268E-07	4.268E-07
Итераций		139	76		
Невязка		9.255E-11	7.635E-11		
Погрешность		2.629E-02	2.629E-02		

5. Выводы

По результатам исследований, метод Ньютона, как и следовало ожидать, оказался лучше, так как сходился в среднем за меньшее число итераций, к тому же позволил решить задачу, решением которой был полином третьей степени. С точки зрения получения формул и построения матриц метод Ньютона сложнее метода простой итерации, однако позволяет учесть особенности нелинейности конкретной задачи.

6. Код программы

Метод простой итерации

```
#define class type
    module simple_iter_module
        implicit none

        type, private :: finite_element
            double precision :: begin_, end_, lambda_, gamma_
        end type

        type, private :: area
            type(finite_element), allocatable :: fe(:)
            integer :: fe_num
            ! Для первых bound_val1_x = ug, bound_val2_x = undefined
            ! Для вторых bound_val1_x = theta, bound_val2_x = undefined
            ! Для третьих bound_val1_x = beta, bound_val2_x = ub
            double precision :: bound_val1_l, bound_val2_l, bound_val1_r, bound_val2_r
            integer :: bound_type_l, bound_type_r
        end type

        type, private :: slae
            double precision, allocatable :: di(:), dl(:), du(:), f(:), q(:), q_old(:)
            integer :: n
        contains
            procedure :: solve
        end type

        type :: simple_iter_solver
            double precision :: omega_ = 1.0d0
            integer :: maxiter_ = 100000
            double precision :: epsilon_ = 1d-10
            type(area), private :: area
            type(slae), private :: slae
            double precision, private :: & ! матрица массы
                m_x(2,2)=reshape(source=(/2d0,1d0,1d0,2d0/),shape=(/2,2/))
            double precision, private :: & ! матрица жесткости
                g_x(2,2)=reshape(source=(/1d0,-1d0,-1d0,1d0/),shape=(/2,2/))
        contains
            procedure :: read_
            procedure :: solve_
            procedure :: clean_
            procedure :: write_
            procedure, private :: get_matrix
```

```
        procedure, private :: f_
        procedure, private :: psi1
        procedure, private :: psi2
        procedure, private :: residual
        procedure, private, nopass :: norm_2
    end type

contains

    function f_(this, q, x, num_fe)
        implicit none
        class(simple_iter_solver) :: this
        double precision :: q(*), u, f_, x
        integer :: num_fe
        u = q(num_fe) * psi1(this, x, num_fe) + &
            q(num_fe+1) * psi2(this, x, num_fe)
        ! # Test 4.1.
        f_ = u
        ! # Test 4.2.
        f_ = -4d0 + u
        ! # Test 4.3.
        f_ = -6d0 * sign(abs(u)**(1d0/3d0), u) + u
        ! # Test 4.4.
        f_ = 2d0 * u
        ! # Test 4.5.
        f_ = 0d0
        ! # Test 4.6.
        f_ = 2d0 * u
        ! # Test 4.7.
        f_ = -4d0 + u
        ! # Test 4.8.
        f_ = 2d0 * u
        !f_ = - 4d0 * this%area%fe(num_fe)%lambda_ + &
        !    this%area%fe(num_fe)%gamma_ * u
    end function

    function psi1(this, x, num_fe)
        implicit none
        class(simple_iter_solver) :: this
        double precision :: x, psi1, x1, x2, hx
        integer :: num_fe
        x1 = this%area%fe(num_fe)%begin_
        x2 = this%area%fe(num_fe)%end_
        hx = x2 - x1
        psi1 = (x2 - x) / hx
    end function

    function psi2(this, x, num_fe)
        implicit none
        class(simple_iter_solver) :: this
        double precision :: x, psi2, x1, x2, hx
        integer :: num_fe
        x1 = this%area%fe(num_fe)%begin_
        x2 = this%area%fe(num_fe)%end_
        hx = x2 - x1
        psi2 = (x - x1) / hx
    end function

    subroutine read_(this)
        implicit none
        class(simple_iter_solver) :: this
        integer :: i
        open(10,file='../area.txt',status='old')
        read(10,*) this%area%fe_num
        allocate(this%area%fe(this%area%fe_num))
        do i=1,this%area%fe_num
            read(10,*) this%area%fe(i)%begin_, &
                this%area%fe(i)%end_, &
                this%area%fe(i)%lambda_, &
                this%area%fe(i)%gamma_
        end do
        close(10)
        open(10,file='../bound.txt',status='old')
        read(10,*) this%area%bound_type_l
        if(this%area%bound_type_l.eq.3) then
            read(10,*) this%area%bound_val1_l,this%area%bound_val2_l
        else
            read(10,*) this%area%bound_val1_l
            this%area%bound_val2_l = 0d0
        end if
        read(10,*) this%area%bound_type_r
        if(this%area%bound_type_r.eq.3) then
            read(10,*) this%area%bound_val1_r,this%area%bound_val2_r
        else
            read(10,*) this%area%bound_val1_r
            this%area%bound_val2_r = 0d0
        end if
        close(10)
    end subroutine

    function norm_2(x, n)
        implicit none
        double precision :: x(*), norm_2
        integer :: n, i
        norm_2 = 0d0
        do i = 1, n
```

```
        norm_2 = norm_2 + x(i)**2
    end do
end function

function residual(this)
    implicit none
    class(simple_iter_solver) :: this
    double precision :: residual
    integer :: i
    double precision, allocatable :: dq(:)
    allocate(dq(this%slae%n))
    dq(1) = this%slae%di(1) * this%slae%q_old(1) + &
        this%slae%du(1) * this%slae%q_old(2)
    do i = 2, this%slae%n - 1
        dq(i) = this%slae%di(i) * this%slae%q_old(i) + &
            this%slae%du(i) * this%slae%q_old(i+1) + &
            this%slae%dl(i-1) * this%slae%q_old(i-1)
    end do
    dq(this%slae%n) = this%slae%di(this%slae%n) * this%slae%q_old(this%slae%n) + &
        this%slae%dl(this%slae%n-1) * this%slae%q_old(this%slae%n-1)
    dq = dq - this%slae%f
    residual = dsqrt(norm_2(dq, this%slae%n) / &
        norm_2(this%slae%f, this%slae%n))
    deallocate(dq)
    print*, residual
end function

subroutine get_matrix(this)
    implicit none
    class(simple_iter_solver) :: this
    integer :: i, j, ind
    double precision :: h, g_mul, m_mul, f1, f2, f_mul
    this%slae%di = 0d0
    this%slae%dl = 0d0
    this%slae%du = 0d0
    this%slae%f = 0d0
    ! Цикл по КЭ
    do i = 1, this%area%fe_num
        h = this%area%fe(i)%end_ - this%area%fe(i)%begin_
        ! Добавки в матрицу
        g_mul = this%area%fe(i)%lambda_ / h
        m_mul = this%area%fe(i)%gamma_ * h / 6d0
        do j = 0, 1
            ind = i + j
            this%slae%di(ind) = this%slae%di(ind) + &
                g_mul * this%g_x(j+1, j+1) + &
                m_mul * this%m_x(j+1, j+1)
        end do
        this%slae%dl(i) = this%slae%dl(i) + &
            g_mul * this%g_x(2, 1) + &
            m_mul * this%m_x(2, 1)
        this%slae%du(i) = this%slae%du(i) + &
            g_mul * this%g_x(1, 2) + &
            m_mul * this%m_x(1, 2)
        ! Добавки в правую часть
        f_mul = h / 6d0
        f1 = f_(this, this%slae%q, this%area%fe(i)%begin_, i)
        f2 = f_(this, this%slae%q, this%area%fe(i)%end_, i)
        this%slae%f(i) = this%slae%f(i) + f_mul * (2d0 * f1 + f2)
        this%slae%f(i+1) = this%slae%f(i+1) + f_mul * (f1 + 2d0 * f2)
    end do
    ! Краевые слева
    select case(this%area%bound_type_l)
    case(1)
        this%slae%di(1) = 1d0
        this%slae%du(1) = 0d0
        this%slae%f(1) = this%area%bound_val1_l
    case(2)
        this%slae%f(1) = this%slae%f(1) + this%area%bound_val1_l
    case(3)
        this%slae%di(1) = this%slae%di(1) + this%area%bound_val1_l
        this%slae%f(1) = this%slae%f(1) + this%area%bound_val1_l * &
            this%area%bound_val2_l
    end select
    ! Краевые справа
    ind = this%slae%n
    select case(this%area%bound_type_r)
    case(1)
        this%slae%di(ind) = 1d0
        this%slae%dl(ind-1) = 0d0
        this%slae%f(ind) = this%area%bound_val1_r
    case(2)
        this%slae%f(ind) = this%slae%f(ind) + this%area%bound_val1_r
    case(3)
        this%slae%di(ind) = this%slae%di(ind) + this%area%bound_val1_r
        this%slae%f(ind) = this%slae%f(ind) + this%area%bound_val1_r * &
            this%area%bound_val2_r
    end select
end subroutine

subroutine solve_(this)
    implicit none
    class(simple_iter_solver) :: this
    integer :: i, iter = 0
    this%slae%n = this%area%fe_num + 1
    allocate(this%slae%di(this%slae%n))
```

```
            allocate(this%slae%du(this%slae%n - 1))
            allocate(this%slae%dl(this%slae%n - 1))
            allocate(this%slae%f(this%slae%n))
            allocate(this%slae%q(this%slae%n))
            allocate(this%slae%q_old(this%slae%n))
            this%slae%q = 0d0
            do
                this%slae%q_old = this%slae%q
                call get_matrix(this)
                call this%slae%solve()
                this%slae%q = this%omega_ * this%slae%q + &
                    (1d0 - this%omega_) * this%slae%q_old
                iter = iter + 1
                print*, 'Iteration:', iter
                do i = 1, this%slae%n
                    print*, this%slae%q(i)
                end do
                if(residual(this).lt.this%epsilon_.or.iter.gt.this%maxiter_) exit
            end do
        end subroutine

        subroutine solve(this)
            implicit none
            class(slae) :: this
            integer :: i
            double precision, allocatable :: dl(:), du(:), di(:), f(:)
            allocate(di(this%n))
            allocate(du(this%n - 1))
            allocate(dl(this%n - 1))
            allocate(f(this%n))
            di = this%di
            dl = this%dl
            du = this%du
            f = this%f
            do i = 2, this%n
                dl(i-1) = dl(i-1) / di(i-1)
                di(i) = di(i) - dl(i-1) * du(i-1)
                f(i) = f(i) - f(i-1) * dl(i-1)
            end do
            do i = this%n, 2, -1
                this%q(i) = f(i) / di(i)
                f(i-1) = f(i-1) - this%q(i) * du(i-1)
            end do
            this%q(1) = f(1) / di(1)
            deallocate(di)
            deallocate(du)
            deallocate(dl)
            deallocate(f)
        end subroutine

        subroutine clean_(this)
            implicit none
            class(simple_iter_solver) :: this
            deallocate(this%slae%di)
            deallocate(this%slae%du)
            deallocate(this%slae%dl)
            deallocate(this%slae%f)
            deallocate(this%slae%q)
            deallocate(this%slae%q_old)
            deallocate(this%area%fe)
        end subroutine

        subroutine write_(this)
            implicit none
            class(simple_iter_solver) :: this
            integer :: i
            open(10, file='../simple_iter.txt', status='unknown')
            do i = 1, this%slae%n
                write(10, fmt='( e27.16 )') this%slae%q(i)
            end do
            close(10)
        end subroutine
    end module

    program main
        use simple_iter_module
        implicit none
        type(simple_iter_solver) :: si
        call si%read_()
        call si%solve_()
        call si%write_()
        call si%clean_()
    end program
```

Метод Ньютона

```
#define class type
    module newton_module
        implicit none

        type, private :: finite_element
            double precision :: begin_, end_, lambda_, gamma_
        end type
```

```
    type, private :: area
        type(finite_element), allocatable :: fe(:)
        integer :: fe_num
        ! Для первых bound_val1_x = ug, bound_val2_x = undefined
        ! Для вторых bound_val1_x = theta, bound_val2_x = undefined
        ! Для третьих bound_val1_x = beta, bound_val2_x = ub
        double precision :: bound_val1_l, bound_val2_l, bound_val1_r, bound_val2_r
        integer :: bound_type_l, bound_type_r
    end type

    type, private :: slae
        double precision, allocatable :: di(:), dl(:), du(:), f(:), q(:), q_old(:)
        integer :: n
    contains
        procedure :: solve
    end type

    type, private :: temp_matrix
        double precision, allocatable :: dl(:), du(:), di(:), f(:)
    contains
        procedure :: alloc
        procedure :: dealloc
    end type

    type :: newton_solver
        double precision :: omega_ = 1.0d0
        integer :: maxiter_ = 10000
        double precision :: epsilon_ = 1d-10
        type(area), private :: area
        type(slae), private :: slae
        type(temp_matrix), private :: tmpmtr
        double precision, private :: & ! матрица массы
            m_x(2,2)=reshape(source=(/2d0,1d0,1d0,2d0/),shape=(/2,2/))
        double precision, private :: & ! матрица жесткости
            g_x(2,2)=reshape(source=(/1d0,-1d0,-1d0,1d0/),shape=(/2,2/))
    contains
        procedure :: read_
        procedure :: solve_
        procedure :: clean_
        procedure :: write_
        procedure, private :: get_matrix
        procedure, private :: f_
        procedure, private :: df_dq
        procedure, private :: dbi_dqr
        procedure, private, nopass :: f_u
        procedure, private :: psi1
        procedure, private :: psi2
        procedure, private :: residual
        procedure, private, nopass :: norm_2
    end type

contains

    function f_u(u_, lambda_, gamma_)
        implicit none
        double precision :: u_, lambda_, gamma_, f_u
        ! # Test 4.1.
        f_u = u_
        ! # Test 4.2.
        f_u = -4d0 + u_
        ! # Test 4.3.
        f_u = -6d0 * sign(abs(u_)**(1d0/3d0), u_) + u_
        ! # Test 4.4.
        f_u = 2d0 * u_
        ! # Test 4.5.
        f_u = 0d0
        ! # Test 4.6.
        f_u = 2d0 * u_
        ! # Test 4.7.
        f_u = -4d0 + u_
        ! # Test 4.8.
        f_u = 2d0 * u_
        !f_u = - 4d0 * lambda_ + gamma_ * u_
    end function

    function f_(this, q, x, num_fe)
        implicit none
        class(newton_solver) :: this
        double precision :: q(*), u, f_, x
        integer :: num_fe
        u = q(num_fe) * psi1(this, x, num_fe) + &
            q(num_fe+1) * psi2(this, x, num_fe)
        f_ = f_u(u, this%area%fe(num_fe)%lambda_, &
            this%area%fe(num_fe)%gamma_)
    end function

    function df_dq(this, q, l, r, num_fe)
        implicit none
        class(newton_solver) :: this
        double precision :: q(*), df_dq, df_du, u1, u2, u, h, x, x1, x2
        integer :: num_fe, l, r
        if(l.eq.1) then
            x = this%area%fe(num_fe)%begin_
        else
            x = this%area%fe(num_fe)%end_
        end if
```

```
    h = (this%area%fe(num_fe)%end_ - this%area%fe(num_fe)%begin_) / 1d6
    x1 = x - h
    x2 = x + h
    u1 = q(num_fe) * psi1(this, x1, num_fe) + &
        q(num_fe+1) * psi2(this, x1, num_fe)
    u2 = q(num_fe) * psi1(this, x2, num_fe) + &
        q(num_fe+1) * psi2(this, x2, num_fe)
    if(dabs(u2 - u1) .gt. this%epsilon_) then
        h = dabs(u2 - u1)
        df_du = (f_u(u2, this%area%fe(num_fe)%lambda_, &
            this%area%fe(num_fe)%gamma_) &
            - f_u(u1, this%area%fe(num_fe)%lambda_, &
            this%area%fe(num_fe)%gamma_)) / h
    else
        df_du = 0d0
    end if
    if(r.eq.1) then
        df_dq = df_du * psi1(this, x, num_fe)
    else
        df_dq = df_du * psi2(this, x, num_fe)
    end if
end function

function dbi_dqr(this, q, i, r, num_fe)
    implicit none
    class(newton_solver) :: this
    integer :: i, r, num_fe
    double precision :: q(*), dbi_dqr, h, df1, df2
    h = this%area%fe(num_fe)%end_ - this%area%fe(num_fe)%begin_
    df1 = df_dq(this, q, 1, r, num_fe)
    df2 = df_dq(this, q, 2, r, num_fe)
    if(dabs(df1).le.this%epsilon_.and.dabs(df1).le.this%epsilon_) then
        dbi_dqr = 0d0
    else
        if(i.eq.1) then
            dbi_dqr = h / 6d0 * (2d0 * df1 + df2)
        else
            dbi_dqr = h / 6d0 * (df1 + 2d0 * df2)
        end if
    end if
end function

function psi1(this, x, num_fe)
    implicit none
    class(newton_solver) :: this
    double precision :: x, psi1, x1, x2, hx
    integer :: num_fe
    x1 = this%area%fe(num_fe)%begin_
    x2 = this%area%fe(num_fe)%end_
    hx = x2 - x1
    psi1 = (x2 - x) / hx
end function

function psi2(this, x, num_fe)
    implicit none
    class(newton_solver) :: this
    double precision :: x, psi2, x1, x2, hx
    integer :: num_fe
    x1 = this%area%fe(num_fe)%begin_
    x2 = this%area%fe(num_fe)%end_
    hx = x2 - x1
    psi2 = (x - x1) / hx
end function

subroutine read_(this)
    implicit none
    class(newton_solver) :: this
    integer :: i
    open(10,file='../area.txt',status='old')
    read(10,*) this%area%fe_num
    allocate(this%area%fe(this%area%fe_num))
    do i=1,this%area%fe_num
        read(10,*) this%area%fe(i)%begin_, &
            this%area%fe(i)%end_, &
            this%area%fe(i)%lambda_, &
            this%area%fe(i)%gamma_
    end do
    close(10)
    open(10,file='../bound.txt',status='old')
    read(10,*) this%area%bound_type_l
    if(this%area%bound_type_l.eq.3) then
        read(10,*) this%area%bound_val1_l,this%area%bound_val2_l
    else
        read(10,*) this%area%bound_val1_l
        this%area%bound_val2_l = 0d0
    end if
    read(10,*) this%area%bound_type_r
    if(this%area%bound_type_r.eq.3) then
        read(10,*) this%area%bound_val1_r,this%area%bound_val2_r
    else
        read(10,*) this%area%bound_val1_r
        this%area%bound_val2_r = 0d0
    end if
    close(10)
end subroutine
```

```
function norm_2(x, n)
    implicit none
    double precision :: x(*), norm_2
    integer :: n, i
    norm_2 = 0d0
    do i = 1, n
        norm_2 = norm_2 + x(i)**2
    end do
end function

function residual(this)
    implicit none
    class(newton_solver) :: this
    double precision :: residual
    integer :: i
    double precision, allocatable :: dq(:)
    allocate(dq(this%slae%n))
    dq(1) = this%tmpmtr%di(1) * this%slae%q_old(1) + &
        this%tmpmtr%du(1) * this%slae%q_old(2)
    do i = 2, this%slae%n - 1
        dq(i) = this%tmpmtr%di(i) * this%slae%q_old(i) + &
            this%tmpmtr%du(i) * this%slae%q_old(i+1) + &
            this%tmpmtr%dl(i-1) * this%slae%q_old(i-1)
    end do
    dq(this%slae%n) = this%tmpmtr%di(this%slae%n) * this%slae%q_old(this%slae%n) + &
        this%tmpmtr%dl(this%slae%n-1) * this%slae%q_old(this%slae%n-1)
    dq = dq - this%tmpmtr%f
    residual = dsqrt(norm_2(dq, this%slae%n) / &
        norm_2(this%tmpmtr%f, this%slae%n))
    deallocate(dq)
    print*, residual
end function

subroutine get_matrix(this)
    implicit none
    class(newton_solver) :: this
    integer :: i, j, ind
    double precision :: h, g_mul, m_mul, f1, f2, f_mul, add
    this%slae%di = 0d0
    this%slae%dl = 0d0
    this%slae%du = 0d0
    this%slae%f = 0d0
    this%tmpmtr%di = 0d0
    this%tmpmtr%dl = 0d0
    this%tmpmtr%du = 0d0
    this%tmpmtr%f = 0d0
    ! Цикл по КЭ
    do i = 1, this%area%fe_num
        h = this%area%fe(i)%end_ - this%area%fe(i)%begin_
        ! Добавки в матрицу
        g_mul = this%area%fe(i)%lambda_ / h
        m_mul = this%area%fe(i)%gamma_ * h / 6d0
        do j = 0, 1
            ind = i + j
            add = g_mul * this%g_x(j+1, j+1) + &
                m_mul * this%m_x(j+1, j+1)
            this%slae%di(ind) = this%slae%di(ind) + add
            this%tmpmtr%di(ind) = this%tmpmtr%di(ind) + add
        end do
        add = g_mul * this%g_x(2, 1) + &
            m_mul * this%m_x(2, 1)
        this%slae%dl(i) = this%slae%dl(i) + add
        this%tmpmtr%dl(i) = this%tmpmtr%dl(i) + add
        add = g_mul * this%g_x(1, 2) + &
            m_mul * this%m_x(1, 2)
        this%slae%du(i) = this%slae%du(i) + add
        this%tmpmtr%du(i) = this%tmpmtr%du(i) + add
        ! Добавки в правую часть
        f_mul = h / 6d0
        f1 = f_(this, this%slae%q, this%area%fe(i)%begin_, i)
        f2 = f_(this, this%slae%q, this%area%fe(i)%end_, i)
        add = f_mul * (2d0 * f1 + f2)
        this%slae%f(i) = this%slae%f(i) + add
        this%tmpmtr%f(i) = this%tmpmtr%f(i) + add
        add = f_mul * (f1 + 2d0 * f2)
        this%slae%f(i+1) = this%slae%f(i+1) + add
        this%tmpmtr%f(i+1) = this%tmpmtr%f(i+1) + add
    end do
    ! Добавки от линеаризации
    do i = 1, this%area%fe_num
        this%slae%di(i) = this%slae%di(i) - dbi_dqr(this, this%slae%q, 1, 1, i)
        this%slae%di(i+1) = this%slae%di(i+1) - dbi_dqr(this, this%slae%q, 2, 2, i)
        this%slae%dl(i) = this%slae%dl(i) - dbi_dqr(this, this%slae%q, 2, 1, i)
        this%slae%du(i) = this%slae%du(i) - dbi_dqr(this, this%slae%q, 1, 2, i)
        add = dbi_dqr(this, this%slae%q, 1, 1, i) * this%slae%q(i) + &
            dbi_dqr(this, this%slae%q, 1, 2, i) * this%slae%q(i+1)
        this%slae%f(i) = this%slae%f(i) - add
        add = dbi_dqr(this, this%slae%q, 2, 1, i) * this%slae%q(i) + &
            dbi_dqr(this, this%slae%q, 2, 2, i) * this%slae%q(i+1)
        this%slae%f(i+1) = this%slae%f(i+1) - add
    end do
    ! Краевые слева
    select case(this%area%bound_type_l)
    case(1)
        this%slae%di(1) = 1d0
        this%slae%du(1) = 0d0
```

```
        this%slae%f(1) = this%area%bound_val1_l
        this%tmpmtr%di(1) = 1d0
        this%tmpmtr%du(1) = 0d0
        this%tmpmtr%f(1) = this%area%bound_val1_l
    case(2)
        this%slae%f(1) = this%slae%f(1) + this%area%bound_val1_l
        this%tmpmtr%f(1) = this%tmpmtr%f(1) + this%area%bound_val1_l
    case(3)
        this%slae%di(1) = this%slae%di(1) + this%area%bound_val1_l
        this%slae%f(1) = this%slae%f(1) + this%area%bound_val1_l * &
            this%area%bound_val2_l
        this%tmpmtr%di(1) = this%tmpmtr%di(1) + this%area%bound_val1_l
        this%tmpmtr%f(1) = this%tmpmtr%f(1) + this%area%bound_val1_l * &
            this%area%bound_val2_l
    end select
    ! Краевые справа
    ind = this%slae%n
    select case(this%area%bound_type_r)
    case(1)
        this%slae%di(ind) = 1d0
        this%slae%dl(ind-1) = 0d0
        this%slae%f(ind) = this%area%bound_val1_r
        this%tmpmtr%di(ind) = 1d0
        this%tmpmtr%dl(ind-1) = 0d0
        this%tmpmtr%f(ind) = this%area%bound_val1_r
    case(2)
        this%slae%f(ind) = this%slae%f(ind) + this%area%bound_val1_r
        this%tmpmtr%f(ind) = this%tmpmtr%f(ind) + this%area%bound_val1_r
    case(3)
        this%slae%di(ind) = this%slae%di(ind) + this%area%bound_val1_r
        this%slae%f(ind) = this%slae%f(ind) + this%area%bound_val1_r * &
            this%area%bound_val2_r
        this%tmpmtr%di(ind) = this%tmpmtr%di(ind) + this%area%bound_val1_r
        this%tmpmtr%f(ind) = this%tmpmtr%f(ind) + this%area%bound_val1_r * &
            this%area%bound_val2_r
    end select
end subroutine

subroutine alloc(this, n)
    implicit none
    class(temp_matrix) :: this
    integer :: n
    allocate(this%di(n))
    allocate(this%du(n - 1))
    allocate(this%dl(n - 1))
    allocate(this%f(n))
end subroutine

subroutine dealloc(this)
    implicit none
    class(temp_matrix) :: this
    deallocate(this%di)
    deallocate(this%du)
    deallocate(this%dl)
    deallocate(this%f)
end subroutine

subroutine solve_(this)
    implicit none
    class(newton_solver) :: this
    integer :: i, iter = 0
    this%slae%n = this%area%fe_num + 1
    allocate(this%slae%di(this%slae%n))
    allocate(this%slae%du(this%slae%n - 1))
    allocate(this%slae%dl(this%slae%n - 1))
    allocate(this%slae%f(this%slae%n))
    allocate(this%slae%q(this%slae%n))
    allocate(this%slae%q_old(this%slae%n))
    call this%tmpmtr%alloc(this%slae%n)
    this%slae%q = 0d0
    do
        this%slae%q_old = this%slae%q
        call get_matrix(this)
        call this%slae%solve()
        this%slae%q = this%omega_ * this%slae%q + &
            (1d0 - this%omega_) * this%slae%q_old
        iter = iter + 1
        print*, 'Iteration:', iter
        do i = 1, this%slae%n
            print*, this%slae%q(i)
        end do
        if(residual(this).lt.this%epsilon_.or.iter.gt.this%maxiter_) exit
    end do
    call this%tmpmtr%dealloc()
end subroutine

subroutine solve(this)
    implicit none
    class(slae) :: this
    integer :: i
    do i = 2, this%n
        this%dl(i-1) = this%dl(i-1) / this%di(i-1)
        this%di(i) = this%di(i) - this%dl(i-1) * this%du(i-1)
        this%f(i) = this%f(i) - this%f(i-1) * this%dl(i-1)
    end do
    do i = this%n, 2, -1
```

```
            this%q(i) = this%f(i) / this%di(i)
            this%f(i-1) = this%f(i-1) - this%q(i) * this%du(i-1)
        end do
        this%q(1) = this%f(1) / this%di(1)
    end subroutine

    subroutine clean_(this)
        implicit none
        class(newton_solver) :: this
        deallocate(this%slae%di)
        deallocate(this%slae%du)
        deallocate(this%slae%dl)
        deallocate(this%slae%f)
        deallocate(this%slae%q)
        deallocate(this%slae%q_old)
        deallocate(this%area%fe)
    end subroutine

    subroutine write_(this)
        implicit none
        class(newton_solver) :: this
        integer :: i
        open(10, file='../newton.txt', status='unknown')
        do i = 1, this%slae%n
            write(10, fmt='( e27.16 )') this%slae%q(i)
        end do
        close(10)
    end subroutine
end module

program main
    use newton_module
    implicit none
    type(newton_solver) :: n
    call n%read_()
    call n%solve_()
    call n%write_()
    call n%clean_()
end program
```