Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет

Уравнения математической физики
Лабораторная работа №2

Факультет: ФПМИ
Группа: ПМ-63
Студент: Кожекин М.В.
Вариант: 7

Новосибирск
2019

1. Цель работы

Разработать программу решения нелинейной одномерной краевой задачи методом конечных элементов. Сравнить метод простой итерации и метод Ньютона для решения данной задачи.

2. Задание

1. Выполнить конечноэлементную аппроксимацию исходного уравнения в соответствии с заданием. Получить формулы для вычисления компонент матрицы и вектора правой части для метода простой итерации.

2. Реализовать программу решения нелинейной задачи методом простой итерации с учетом следующих требований:

- язык программирования C++ или Фортран;
- предусмотреть возможность задания неравномерных сеток по пространству и по времени, разрывность параметров уравнения по подобластям, учет краевых условий;
- матрицу хранить в ленточном формате, для решения СЛАУ использовать метод -разложения;
- предусмотреть возможность использования параметра релаксации.

3. Выполнить линеаризацию нелинейной системы алгебраических уравнений с использованием метода Ньютона. Получить формулы для вычисления компонент линеаризованных матрицы и вектора правой части

4. Реализовать программу решения нелинейной задачи методом Ньютона.

5. Протестировать разработанные программы.

6. Исследовать реализованные методы на различных зависимостях коэффициента от решения (или производной решения) в соответствии с заданием. На одних и тех же задачах сравнить по количеству итераций метод простой итерации и метод Ньютона. Исследовать скорость сходимости от параметра релаксации.

Вариант 7: Базисные функции линейные.

$$-div(\lambda(\frac{du}{dx})grad(u) + \sigma\frac{du}{dt} = f$$

3. Анализ

Произведя временную аппроркимацию по двуслойной неявной схеме исходное уравнение примет вид:

$$-div(\lambda(\frac{du}{dx})grad(u) + \frac{\sigma}{\Delta t_s}u_s = f + \frac{\sigma}{\Delta t_s}u_{s-1}$$

В ходе конечноэлементной аппроксимации нелинейной начально-краевой задачи получается система нелинейных уравнений

$$\mathbf{A}(\mathbf{q_s})\mathbf{q_s} = \mathbf{b}(\mathbf{q_s})$$

у которой компоненты матрицы $\mathbf{A}(\mathbf{q_s})\mathbf{q_s}$ и вектора правой части $\mathbf{b}(\mathbf{q_s})$ вычисляются следующим образом:

$$A_{ij}(q_s) = \int_\Omega \lambda_s(u^h(q_s)) grad\psi_i grad\psi_j d\Omega + \frac{1}{\Delta t_s}\int_\Omega \sigma_s(u^h(q_s))\psi_i\psi_j d\Omega + \int_{S_3} \beta_s(u^h(q_s))\psi_i\psi_j dS$$

$$b_i(q_s) = \int_\Omega f_s(u^h(q_s))\psi_i d\Omega + \frac{1}{\Delta t_s}\int_\Omega (u^h(q_s))(u^h(q_{s-1}))+$$

$$+ \int_{S_2} \Theta_s(u^h(q_s))\psi_i dS + \int_{S_2} \beta_s(u^h(q_s))u_{\beta,s}(u^h(q_s))\psi_i dS$$

где

$$u^h(q_s) = \sum_k q_{k,s}\psi_k \qquad u^h(q_{s-1}) = \sum_k q_{k,s-1}\psi_k$$

Выведем формулы для локальных матриц массы, жёсткости и вектора правой части.

$$\frac{du}{dx} = \frac{d\sum_{k=0}^{1} q_k\psi_k}{dx} = q_0\frac{d\psi_0}{dx} + q_1\frac{d\psi_1}{dx} = -\frac{1}{h}q_0 + \frac{1}{h}q_1 = \frac{q_1 - q_0}{h}$$

$$G_{i,j} = \int_\Omega \lambda(\frac{du}{dx}) grad\psi_i grad\psi_j d\Omega$$

$$G_{0,0} = \sum_{k=0}^{1}\int_\Omega \lambda(\frac{q_1 - q_0}{h})\psi_k grad\psi_0 grad\psi_0 d\Omega =$$

$$= \frac{1}{h}\sum_{k=0}^{1}\int_\Omega \lambda(\frac{q_1 - q_0}{h})\psi_k d\Omega =$$

$$= \frac{\lambda_0(\frac{q_1-q_0}{h}) + \lambda_1(\frac{q_1-q_0}{h})}{2h} = G_{1,1}$$

$$G_{0,1} = \sum_{k=0}^{1}\int_\Omega \lambda(\frac{q_1 - q_0}{h})\psi_k grad\psi_0 grad\psi_1 d\Omega =$$

$$= -\frac{1}{h}\sum_{k=0}^{1}\int_\Omega \lambda(\frac{q_1 - q_0}{h})\psi_k d\Omega =$$

$$= -\frac{\lambda_0(\frac{q_1-q_0}{h}) + \lambda_1(\frac{q_1-q_0}{h})}{2h} = G_{1,0}$$

$$G = \frac{\lambda_0(\frac{q_1-q_0}{h}) + \lambda_1(\frac{q_1-q_0}{h})}{2h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

$$M_{i,j} = \frac{\sigma}{\Delta t_s}\int_\Omega \psi_i\psi_j d\Omega$$

$$M_{0,0} = \frac{\sigma}{\Delta t_s}\int_\Omega \psi_0\psi_0 d\Omega = \frac{\sigma h}{\Delta t_s}\int_0^1 \xi^2 d\xi = \frac{\sigma h}{\Delta t_s}\frac{\xi^3}{3}\Big|_0^1 = \frac{\sigma h}{3\Delta t_s} = M_{1,1}$$

$$M_{0,1} = \frac{\sigma}{\Delta t_s}\int_\Omega \psi_0\psi_1 d\Omega = \frac{\sigma h}{\Delta t_s}\int_0^1 \xi(1-\xi) d\xi = \frac{\sigma h}{\Delta t_s}(\frac{\xi^2}{2} - \frac{\xi^3}{3})\Big|_0^1 = \frac{\sigma h}{6\Delta t_s} = M_{1,0}$$

$$M = \frac{\sigma h}{6\Delta t_s}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

$$b_i = \int_\Omega f_s \psi_i d\Omega + \frac{1}{\Delta t_s}\int_\Omega \sigma u^h_{q-1}\psi_i d\Omega \,\Big|\, u_{q-1}h = \sum_{k=0}^{1} q_{k,s-1}\psi_k \Big|$$

$$b_0 = \sum_{k=0}^{1}\int_\Omega f_k\psi_k\psi_0 d\Omega + \frac{\sigma}{\Delta t_s}\sum_{k=0}^{1}\int_\Omega q_{k,q-1}\psi_k\psi_0 d\Omega$$

$$= \Big[f_0\int_\Omega \psi_0\psi_0 d\Omega + f_1\int_\Omega \psi_1\psi_0 d\Omega\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\int_\Omega \psi_0\psi_0 d\Omega + q_{1,s-1}\int_\Omega \psi_1\psi_0 d\Omega\Big]$$

$$= h\Big[f_0\int_0^1 \xi^2 d\xi + f_1\int_0^1 (1-\xi)\xi d\xi\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\int_0^1 \xi^2 d\xi + q_{1,s-1}\int_0^1 (1-\xi)\xi d\xi\Big]$$

$$= h\Big[f_0\frac{\xi^3}{3}\Big|_0^1 + f_1\big(\frac{\xi^2}{2} - \frac{\xi^3}{3}\big)\Big|_0^1\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\frac{\xi^3}{3}\Big|_0^1 + q_{1,s-1}\big(\frac{\xi^2}{2} - \frac{\xi^3}{3}\big)\Big|_0^1\Big]$$

$$= h\Big[f_0\frac{1}{3} + f_1\frac{1}{6}\Big] + \frac{\sigma}{\Delta t_s}\Big[\frac{1}{3}q_{0,s-1} + \frac{1}{6}q_{1,s-1}\Big]$$

$$= \frac{h}{6}\Big[2f_0 + f_1\Big] + \frac{\sigma}{6\Delta t_s}\Big[2q_{0,s-1} + q_{1,s-1}\Big]$$

$$b_1 = \sum_{k=0}^{1}\int_\Omega f_k\psi_k\psi_1 d\Omega + \frac{\sigma}{\Delta t_s}\sum_{k=0}^{1}\int_\Omega q_{k,q-1}\psi_0\psi_1 d\Omega =$$

$$= \Big[f_0\int_\Omega \psi_0\psi_1 d\Omega + f_1\int_\Omega \psi_1\psi_1 d\Omega\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\int_\Omega \psi_0\psi_1 d\Omega + q_{1,s-1}\int_\Omega \psi_1\psi_1 d\Omega\Big] =$$

$$= h\Big[f_0\int_0^1 \xi(1-\xi) d\xi + f_1\int_0^1 (1-\xi)^2 d\xi\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\int_0^1 \xi(1-\xi) d\xi + q_{1,s-1}\int_0^1 (1-\xi)^2 d\xi\Big] =$$

$$= h\Big[f_0\big(\frac{\xi^2}{2} - \frac{\xi^3}{3}\big)\Big|_0^1 + f_1(1-\xi)^3\Big|_0^1\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\big(\frac{\xi^2}{2} - \frac{\xi^3}{3}\big)\Big|_0^1 + q_{1,s-1}(1-\xi)^3\Big|_0^1\Big] =$$

$$= \frac{h}{6}\Big[f_0\frac{1}{6} + f_1\frac{1}{3}\Big] + \frac{\sigma}{\Delta t_s}\Big[\frac{1}{6}q_{0,s-1} + \frac{1}{3}q_{1,s-1}\Big]$$

$$= \frac{h}{6}\Big[f_0 + 2f_1\Big] + \frac{\sigma}{6\Delta t_s}\Big[q_{0,s-1} + 2q_{1,s-1}\Big]$$

$$b = \frac{hx}{6}\begin{pmatrix}2f_0 + f_1\\ f_0 + 2f_1\end{pmatrix} + \frac{\sigma}{6\Delta t_s}\begin{pmatrix}2q_{0,s-1} + q_{1,s-1}\\ q_{0,s-1} + 2q_{1,s-1}\end{pmatrix}$$

В итоге:

$$G = \frac{\lambda_0\big(\frac{q_1 - q_0}{h}\big) + \lambda_1\big(\frac{q_1 - q_0}{h}\big)}{2h}\begin{pmatrix}1 & -1\\ -1 & 1\end{pmatrix}$$

$$M = \frac{\sigma h}{6\Delta t_s}\begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix}$$

$$b = \frac{hx}{6}\begin{pmatrix}2f_0 + f_1\\ f_0 + 2f_1\end{pmatrix} + \frac{\sigma}{6\Delta t_s}\begin{pmatrix}2q_{0,s-1} + q_{1,s-1}\\ q_{0,s-1} + 2q_{1,s-1}\end{pmatrix}$$

4. Точность для разных функций u и λ

В ходе следующего исследования использовались следующие параметры:
$\varepsilon = 1e - 7$
$\sigma = 1$
$maxiter = 1000$
Область пространства $\Omega = [0, 1]$
Время задано на отрезке [0, 1]
Первоначальное число узлов 11, а конечных элементов 10
Для неравномерных сеток по времени и пространству коэффициент k=1.1

$u(x,t)$ \ $\lambda(u)$	1	$u+x+1$	u^2	u^2+1	u^3	u^4	e^u	$\sin u$
$3x+t$	1.31e-02	7.00e-03	4.15e-01	9.85e-02	3.99e-01	4.16e-01	2.29e-01	8.98e-01
$2x^2+t$	6.56e-02	5.93e-02	2.06e+01	1.35e-01	4.00e-01	4.00e-01	-nan(ind)	3.26e-01
x^3+t	5.14e-02	4.32e-02	8.69e+00	6.26e-02	1.82e-01	1.45e-01	1.64e-01	9.94e-02
x^4+t	5.82e-02	4.68e-02	5.63e+00	9.83e-02	1.50e-01	2.15e-01	-nan(ind)	1.06e-01
e^x+t	3.49e-02	2.85e-02	1.54e+01	6.12e-02	6.12e-01	9.52e-01	1.81e-01	4.49e+00
$3x+t$	1.31e-02	7.00e-03	4.15e-01	9.85e-02	3.99e-01	4.16e-01	2.29e-01	8.98e-01
$3x+t^2$	2.62e-02	8.75e-03	9.31e-02	9.87e-02	9.78e-02	9.40e-02	1.72e-01	7.20e-01
$3x+t^3$	3.94e-02	1.05e-02	6.31e-02	9.93e-02	1.04e-01	1.72e-01	1.72e-01	7.19e+00
$3x+e^t$	3.57e-02	1.00e-02	5.22e-01	9.88e-02	4.75e-01	4.81e-01	1.40e-01	2.01e+00
$3x+sin(t)$	7.09e-03	6.20e-03	4.47e-01	9.84e-02	2.22e-01	4.08e-01	2.29e-01	2.58e-01
e^x+t^2	4.80e-02	3.11e-02	1.60e+01	6.09e-02	2.96e-01	9.51e-01	1.79e-01	1.79e-01
e^x+t^3	6.12e-02	3.38e-02	1.61e+01	6.02e-02	2.91e-01	9.44e-01	1.76e-01	7.88e-01
e^x+e^t	5.75e-02	3.30e-02	7.94e+00	6.08e-02	1.35e+01	9.45e-01	1.79e-01	2.34e+00
$e^x+sin(t)$	2.89e-02	2.73e-02	1.54e+01	6.14e-02	6.13e-01	9.54e-01	1.82e-01	2.42e-01

4.1. Вывод

Как видно из таблицы метод начинает сходиться хуже при повышении степени полинома. Если же функция λ будет зависеть не от $\frac{du}{dx}$, а просто от u, то сходимость будет куда выше . Если функция λ гармоническая (в нашем случае sin(u)), то метод работает хуже, хотя вообще он не должен сходиться.

Также стоит отметить, что и скорость программы в варианте 7 заметно ниже варианта 5. Рещение сходится медленно.

5. Точность решения при дроблении сетки

В ходе следующего исследования использовались следующие параметры:
$\varepsilon = 1e - 7$
$\sigma = 1$
$maxiter = 1000$, т.к. повышение этого числа не приводит к должному результату, а лишь занимает процессорное время
Область пространства $\Omega = [0, 1]$
Время задано на отрезке [0, 1]
Первоначальное число узлов 11, а конечных элементов 10
Для неравномерных сеток по времени и пространству коэффициент k=1.1
Функция $\lambda(u) = u + x + 1$

$3x + t$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	276	0.007001	0	11	400	0.080084
	1	21	278	0.005186	1	21	1005	0.184592
	2	41	279	0.003757	2	41	1010	0.169350
	3	81	279	0.002689	3	81	1010	0.139427
	4	161	279	0.001913	4	161	1010	0.114218
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	277	0.007001	0	11	397	0.080084
	1	21	279	0.005186	1	21	1005	0.184592
	2	41	280	0.003757	2	41	1010	0.169350
	3	81	281	0.002689	3	81	1010	0.139427
	4	161	281	0.001913	4	161	1010	0.114218

$2x^2 + t$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	183	0.059324	0	11	296	0.145360
	1	21	184	0.044568	1	21	1002	0.179469
	2	41	184	0.032504	2	41	1010	0.144185
	3	81	184	0.023346	3	81	1010	0.107327
	4	161	184	0.016638	4	161	1010	0.081423
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	185	0.059324	0	11	295	0.145360
	1	21	185	0.044568	1	21	1002	0.179469
	2	41	185	0.032504	2	41	1010	0.144185
	3	81	186	0.023346	3	81	1010	0.107327
	4	161	186	0.016638	4	161	1010	0.081423

$x^3 + t$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	148	0.043223	0	11	188	0.096061
	1	21	148	0.033076	1	21	687	0.100743
	2	41	148	0.024328	2	41	1005	0.077695
	3	81	148	0.017545	3	81	1010	0.056877
	4	161	148	0.012529	4	161	1010	0.041608
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	145	0.043223	0	11	191	0.096061
	1	21	145	0.033076	1	21	694	0.100742
	2	41	145	0.024328	2	41	1005	0.077694
	3	81	145	0.017545	3	81	1010	0.056877
	4	161	145	0.012529	4	161	1010	0.041608

$x^4 + t$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	173	0.046792	0	11	190	0.107873
	1	21	175	0.036626	1	21	687	0.106266
	2	41	175	0.027195	2	41	1005	0.080517
	3	81	173	0.019699	3	81	1010	0.058058
	4	161	173	0.014098	4	161	1010	0.042060
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	176	0.046792	0	11	192	0.107873
	1	21	177	0.036626	1	21	694	0.106266
	2	41	177	0.027195	2	41	1005	0.080517
	3	81	177	0.019699	3	81	1010	0.058058
	4	161	177	0.014098	4	161	1010	0.042060

$e^x + t$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	174	0.028486	0	11	264	0.085422
	1	21	178	0.021398	1	21	937	0.128210
	2	41	182	0.015605	2	41	1008	0.109120
	3	81	183	0.011208	3	81	1010	0.085939
	4	161	180	0.007988	4	161	1010	0.067383
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	176	0.028486	0	11	266	0.085422
	1	21	176	0.021398	1	21	940	0.128210
	2	41	176	0.015605	2	41	1009	0.109120
	3	81	178	0.011208	3	81	1010	0.085939
	4	161	182	0.007988	4	161	1010	0.067383

$3x + t$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	276	0.007001	0	11	400	0.080084
	1	21	278	0.005186	1	21	1005	0.184592
	2	41	279	0.003757	2	41	1010	0.169350
	3	81	279	0.002689	3	81	1010	0.139427
	4	161	279	0.001913	4	161	1010	0.114218
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	277	0.007001	0	11	397	0.080084
	1	21	279	0.005186	1	21	1005	0.184592
	2	41	280	0.003757	2	41	1010	0.169350
	3	81	281	0.002689	3	81	1010	0.139427
	4	161	281	0.001913	4	161	1010	0.114218

$3x + t^2$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	273	0.008750	0	11	398	0.080553
	1	21	276	0.006482	1	21	1004	0.184590
	2	41	276	0.004695	2	41	1010	0.169350
	3	81	276	0.003361	3	81	1010	0.139427
	4	161	276	0.002391	4	161	1010	0.114218
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	268	0.008750	0	11	384	0.080553
	1	21	272	0.006482	1	21	1004	0.184590
	2	41	272	0.004695	2	41	1010	0.169350
	3	81	272	0.003361	3	81	1010	0.139427
	4	161	272	0.002391	4	161	1010	0.114218

$3x + t^3$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	263	0.010499	0	11	380	0.081022
	1	21	266	0.007778	1	21	1003	0.184593
	2	41	266	0.005634	2	41	1010	0.169350
	3	81	266	0.004033	3	81	1010	0.139427
	4	161	266	0.002869	4	161	1010	0.114218
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	250	0.010499	0	11	357	0.081022
	1	21	252	0.007778	1	21	1002	0.184588
	2	41	253	0.005634	2	41	1010	0.169350
	3	81	253	0.004033	3	81	1010	0.139427
	4	161	253	0.002869	4	161	1010	0.114218

$3x + e^t$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	287	0.010006	0	11	411	0.080890
	1	21	289	0.007413	1	21	1004	0.184592
	2	41	289	0.005369	2	41	1010	0.169350
	3	81	290	0.003844	3	81	1010	0.139427
	4	161	290	0.002735	4	161	1010	0.114218
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	287	0.010006	0	11	412	0.080890
	1	21	290	0.007413	1	21	1004	0.184587
	2	41	292	0.005369	2	41	1010	0.169350
	3	81	294	0.003844	3	81	1010	0.139427
	4	161	295	0.002735	4	161	1010	0.114218

$3x + sin(t)$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	270	0.006197	0	11	393	0.079868
	1	21	273	0.004591	1	21	1005	0.184592
	2	41	273	0.003325	2	41	1010	0.169350
	3	81	275	0.002380	3	81	1010	0.139427
	4	161	275	0.001694	4	161	1010	0.114218
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	274	0.006197	0	11	392	0.079868
	1	21	276	0.004591	1	21	1005	0.184592
	2	41	276	0.003325	2	41	1010	0.169350
	3	81	277	0.002380	3	81	1010	0.139427
	4	161	277	0.001694	4	161	1010	0.114218

$e^x + t^2$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	174	0.031125	0	11	262	0.086157
	1	21	176	0.023353	1	21	924	0.128212
	2	41	178	0.017021	2	41	1007	0.109120
	3	81	178	0.012221	3	81	1010	0.085939
	4	161	179	0.008709	4	161	1010	0.067383
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	171	0.031125	0	11	257	0.086157
	1	21	171	0.023353	1	21	889	0.128212
	2	41	168	0.017021	2	41	1009	0.109120
	3	81	175	0.012221	3	81	1010	0.085939
	4	161	172	0.008709	4	161	1010	0.067383

$e^x + t^3$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	170	0.033758	0	11	253	0.086892
	1	21	172	0.025303	1	21	861	0.128214
	2	41	176	0.018433	2	41	1005	0.109120
	3	81	173	0.013232	3	81	1010	0.085939
	4	161	174	0.009428	4	161	1010	0.067383
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	160	0.033758	0	11	241	0.086892
	1	21	162	0.025303	1	21	792	0.128214
	2	41	162	0.018433	2	41	1005	0.109120
	3	81	167	0.013232	3	81	1010	0.085939
	4	161	166	0.009428	4	161	1010	0.067383

$e^x + e^t$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	179	0.033017	0	11	269	0.086685
	1	21	181	0.024754	1	21	945	0.128214
	2	41	184	0.018036	2	41	1009	0.109120
	3	81	182	0.012948	3	81	1010	0.085939
	4	161	184	0.009226	4	161	1010	0.067383
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	181	0.033017	0	11	270	0.086685
	1	21	179	0.024754	1	21	934	0.128214
	2	41	182	0.018036	2	41	1009	0.109120
	3	81	178	0.012948	3	81	1010	0.085939
	4	161	180	0.009226	4	161	1010	0.067383

$e^x + sin(t)$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	171	0.027270	0	11	261	0.085085
	1	21	175	0.020498	1	21	915	0.128209
	2	41	179	0.014953	2	41	1008	0.109120
	3	81	176	0.010742	3	81	1010	0.085939
	4	161	180	0.007656	4	161	1010	0.067383
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	11	172	0.027270	0	11	263	0.085085
	1	21	170	0.020498	1	21	924	0.128209
	2	41	173	0.014953	2	41	1009	0.109120
	3	81	180	0.010742	3	81	1010	0.085939
	4	161	175	0.007656	4	161	1010	0.067383

5.1. Вывод

Т.к. **порядок сходимости** - это степень того, насколько сильно увеличивается точность при дроблении сетки. Он определяется из степени x.

Исходя из исследований можно заметить, что порядок сходимости $\frac{1}{3}$

6. Исходный код программы

head.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
#include <functional>
#include <cmath>

using namespace std;


typedef std::function<double(double)> function1D;
typedef std::function<double(double, double)> function2D;

typedef vector <double> vector1D;
typedef vector <vector <double>> matrix2D;


// Сравнение векторов
```

```
inline bool operator==(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    for (int i = 0; i < a.size(); ++i)
        if (a[i] != b[i])
            return false;

    return true;
}

// Сложение векторов
inline vector1D operator+(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    vector1D result = a;
    for (int i = 0; i < b.size(); i++)
        result[i] += b[i];
    return result;
}
// Сложение матриц
inline matrix2D operator+(const matrix2D& a, const matrix2D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    matrix2D result = a;
    for (int i = 0; i < b.size(); i++)
        for (int j = 0; j < b.size(); j++)
            result[i][j] += b[i][j];
    return result;
}


// Деление матрицы на число
inline matrix2D operator/(const matrix2D& a, const double& b) {

    matrix2D result = a;
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < a.size(); j++)
            result[i][j] /= b;
    return result;
}


// Вычитание векторов
inline vector1D operator-(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    vector1D result = a;
    for (int i = 0; i < b.size(); i++)
        result[i] -= b[i];
    return result;
}
// Обратный знак вектора
```

```
inline vector1D operator-(const vector1D& a) {
    vector1D result = a;
    for (int i = 0; i < a.size(); i++)
        result[i] = -result[i];
    return result;
}



// Умножение матрицы на вектор
inline vector1D operator*(const matrix2D& a, const vector1D& b) {
    vector1D result = { 0.0, 0.0 };
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < a.size(); j++)
            result[i] += a[i][j] * b[j];
    return result;
}



// Умножение на число
inline vector1D operator*(const vector1D& a, double b) {
    vector1D result = a;
    for (int i = 0; i < result.size(); i++)
        result[i] *= b;
    return result;
}
// Умножение на число
inline vector1D operator*(double b, const vector1D& a) {
    return operator*(a, b);
}



// Деление на число
inline vector1D operator/(const vector1D& a, double b) {
    vector1D result = a;
    for (int i = 0; i < result.size(); i++)
        result[i] /= b;
    return result;
}
// Деление на число
inline vector1D operator/(double b, const vector1D& a) {
    return operator/(a, b);
}



// Скалярное произведение
inline double operator*(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    double sum = 0;
    for (int i = 0; i < a.size(); i++)
        sum += a[i] * b[i];
    return sum;
}

```

```

// Потоковый вывод вектора
inline std::ostream& operator<<(std::ostream& out, const vector1D& v) {
    for (int i = 0; i < v.size() - 1; ++i)
        out << v[i] << ", ";
    out << v.back();
    return out;
}
// Потоковый вывод матрицы
inline std::ostream& operator<<(std::ostream& out, const matrix2D& v) {
    for (int i = 0; i < v.size() - 1; ++i)
        out << v[i] << "  ";
    out << v.back();
    return out;
}


// Потоковый вывод вектора для TeX
inline void printTeXVector(std::ofstream &fout, const vector1D &v, int coefGrid)
    {
    fout << "$(";
    for (int i = 0; i < v.size() - 1; ++i)
        if (i % int(pow(2, coefGrid)) == 0)
            fout << v[i] << ", ";
    fout << v.back() << ")^T$";
}
```

grid.h

```
#pragma once
#include "head.h"


struct NODE {

    bool isFirstNode = false;
    int i;
    double x;
    int type = -9000;       // -9000    начение при инициализации
                            // -1       фиктивный узел
                            // 0        внутренний узел
                            // n        номер границы
    int border;             // номер границы

    void setNodesData(double _x, int _i, int _type, double _coef) {
        x = _x;
        i = _i;
        type = _type;
        if (i % int(pow(2, _coef)) == 0)
            isFirstNode = true;
    }
};


class GRID
{
public:
    void inputGrid();
    void inputTime();
    void buildGrid();
    void buildTimeGrid();
    void showGrid();
```

```
    void saveGridAndBorder(const string &filepathGrid, const string &
    filepathGridBorder);

protected:

    int coefGrid, // Сколько раз дробили сетку по пространству
        coefTime; // Сколько раз дробили сетку по времени

    // Пространство
    bool isGridUniform;
    int width;
    double xLeft, xRight;
    double hx, nx, kx;
    double dx;
    int nodesCount, finiteElementsCount;
    int condType;

    // Время
    bool isTimeUniform;
    int tCount;
    double tFirst, tLast;
    double ht, nt, kt;
    double dt;

    // Узлы
    vector <NODE> nodes;
    vector1D times;
};
```

grid.cpp

```
#include "grid.h"


void GRID::inputGrid()
{
    string filepath;
    if (isGridUniform)
        filepath = "input/uniform_grid.txt";
    else
        filepath = "input/nonuniform_grid.txt";

    std::ifstream fin(filepath);
    fin >> xLeft >> xRight;
    fin >> width;
    if (!isGridUniform) {
        fin >> kx;
        nx = width - 1;
    }
    fin.close();
}


void GRID::inputTime()
{
    string filepath;
    if (isTimeUniform)
        filepath = "input/uniform_time.txt";
    else
        filepath = "input/nonuniform_time.txt";

    std::ifstream fin(filepath);
```

```
    fin >> tFirst >> tLast;
    fin >> tCount;
    if (!isTimeUniform) {
        fin >> kt;
        nt = tCount - 1;
    }
    fin.close();
}


void GRID::buildGrid()
{
    //  xLeft         xRight
    //    *---------------*
    //    0    width    1
    if (isGridUniform) {
        hx = ((xRight - xLeft) / double(width - 1)) / pow(2, coefGrid);
        if (coefGrid != 0)
            width = (width - 1) * pow(2, coefGrid) + 1;
    }
    else {
        if (coefGrid != 0) {
            width = (width - 1) * pow(2, coefGrid) + 1;
            nx *= pow(2, coefGrid);
            kx *= pow(kx, 1.0 / coefGrid);
        }
        hx = (xRight - xLeft) * (1 - kx) / (1 - pow(kx, nx));
    }


    nodesCount = width;
    finiteElementsCount = nodesCount - 1;
    nodes.resize(width);

    if (isGridUniform) {

        size_t i, elem;
        double x;

        // Первый элемент
        nodes[0].setNodesData(xLeft, 0, 1, coefGrid);
        i = 1;
        for (elem = 1; elem < nodesCount - 1; elem++, i++)
        {
            x = xLeft + hx * i;
            nodes[elem].setNodesData(x, i, 0, coefGrid);
            nodes[elem].border = 0;
        }
        // Последний элемент
        nodes[nodesCount - 1].setNodesData(xRight, width, 1, coefGrid);

    }
    else {

        double x;
        size_t i, elem;

        i = 1;
        dx = hx * kx;
        x = xLeft + hx;
```

```
        // Первый элемент
        nodes[0].setNodesData(xLeft, 0, 1, coefGrid);
        for (elem = 1; elem < width; elem++, i++, dx *= kx)
        {
            nodes[elem].setNodesData(x, i, 0, coefGrid);
            nodes[elem].border = 0;
            x += dx;
        }
        // Последний элемент
        nodes[nodesCount - 1].setNodesData(xRight, width, 1, coefGrid);
    }
}


void GRID::buildTimeGrid()
{
    //  tFirst         tLast
    //    *--------------*
    //    0    tCount   1
    times.resize(tCount);

    if (isTimeUniform) {

        ht = ((tLast - tFirst) / double(tCount - 1)) / pow(2, coefTime);
        if (coefTime != 0)
            width = (width - 1) * pow(2, coefTime) + 1;

        size_t i, elem;
        double t;
        // Первый элемент
        times[0] = tFirst;
        i = 1;
        for (elem = 1; elem < tCount; elem++, i++)
            times[elem] = tFirst + ht * i;

        // Последний элемент
        times[tCount - 1] = tLast;
    }

    else {

        if (coefTime != 0) {
            width = (width - 1) * pow(2, coefTime) + 1;
            nt *= pow(2, coefTime);
            kt *= pow(kt, 1.0 / coefTime);
        }

        ht = (tLast - tFirst) * (1 - kt) / (1 - pow(kt, nt));
        double t;
        size_t i, elem;
        i = 1;
        dt = ht * kt;
        t = tFirst + ht;
        // Первый элемент
        times[0] = tFirst;
        for (elem = 1; elem < tCount; elem++, i++, dt *= kt)
        {
            times[elem] = t;
            t += dt;
        }
```

```
        // Последний элемент
        times[tCount - 1] = tLast;
    }
}


// Отображние сетки на экран
void GRID::showGrid() {

    for (size_t i = 0; i < width; i++)
        cout << nodes[i].x << " ";
}


// Сохранение внутренних и внешних узлов в 2 файлах
void GRID::saveGridAndBorder(const string &filepathGrid, const string &
    filepathGridBorder) {

    ofstream grid(filepathGrid);
    ofstream border(filepathGridBorder);
    for (size_t i = 0; i < nodesCount; i++)
        if (nodes[i].type > 0)
            border << nodes[i].x << endl;
        else
            grid << nodes[i].x << endl;

    border.close();
    grid.close();
}
```

fem.h

```
#pragma once
#include "head.h"
#include "grid.h"
#include "solver.h"


class FEM : public GRID, public SOLVER {
public:
    void init(const function2D &_u, const function2D &_f, const function2D &
    _lambda, double _sigma, bool _isGridUniform, bool _isTimeUniform, int
    _condType, int _coefGrid, int _coefTime);
    pair<int, double> solve();
    inline int getNodesCount() { return nodesCount; }


protected:
    double lambda0, lambda1;
    double sigma;
    double t;
    function2D f, u, lambda;
    double calcNormAtMainNodes(const vector1D &x) {
        double tmp = 0;
        for (size_t i = 0; i < x.size(); i++)
            tmp += pow((x[i] - u(nodes[i].x, t)), 2);
        return sqrt(tmp) / nodes.size();
    }

    void buildGlobalMatrixA(double _dt);
    void buildGlobalVectorb();
    void printGlobalMatrixA();
```

```
    void printGlobalVectorb();

    void buildLocalMatrixG(int elemNumber);
    void buildLocalMatrixM(int elemNumber);
    void buildLocalmatrixA(int elemNumber);
    void buildLocalVectorb(int elemNumber);
    matrix2D GLocal, MLocal, ALocal;
    vector1D bLocal;
};
```

fem.cpp

```
#include "fem.h"


// Инициализируем модель, задавая функции u, f и тип сетки
void FEM::init(const function2D & _u, const function2D & _f, const function2D &
    _lambda, double _sigma, bool _isGridUniform, bool _isTimeUniform, int
    _condType, int _coefGrid, int _coefTime)
{
    ifstream fin("input/SLAE_parameters.txt");
    fin >> E >> delta >> maxiter;
    fin.close();
    u = _u;
    f = _f;
    lambda = _lambda;
    sigma = _sigma;
    isGridUniform = _isGridUniform;
    isTimeUniform = _isTimeUniform;
    condType = _condType;
    coefGrid = _coefGrid;
    coefTime = _coefTime;
}


// Решаем во всех узлах по времени
pair<int, double> FEM::solve()
{
    // Задаём начальные условия
    q.resize(nodesCount, 0);
    qPrev.resize(nodesCount, 0);
    vector1D qExact(nodesCount);
    for (size_t i = 0; i < nodesCount; i++)
        qExact[i] = u(nodes[i].x, times[0]);
    qPrev = qExact;

    int count = 0;
    // Решаем в каждый момент временной сетки
    for (size_t i = 1; i < times.size(); i++)
    {
        dt = times[i] - times[i - 1];
        t = times[i];
        do {
            qPrev = q;
            buildGlobalMatrixA(dt);
            buildGlobalVectorb();
            calcWithLUDecomposition();
            count++;
        } while (shouldCalc(count));
    }
    return make_pair(count, calcNormAtMainNodes(q));
}
```

```
49
50
51  //------------------------------------------------------------------------------
52  //------------------------------------------------------------------------------
53  //------------------------------------------------------------------------------
54
55  // Строим глобальную матрицу системы нелинейных уравнений
56  void FEM::buildGlobalMatrixA(double _dt)
57  {
58      dt = _dt;
59      A.clear();
60      di.clear();
61      au.clear();
62      al.clear();
63      A.resize(nodesCount);
64      for (size_t i = 0; i < nodesCount; i++)
65          A[i].resize(nodesCount, 0);
66
67      di.resize(nodesCount, 0);
68      al.resize(nodesCount - 1, 0);
69      au.resize(nodesCount - 1, 0);
70      for (size_t elemNumber = 0; elemNumber < finiteElementsCount; elemNumber++)
71      {
72          buildLocalmatrixA(elemNumber);
73
74          //cout << ALocal << endl;
75          di[elemNumber] += ALocal[0][0];     au[elemNumber] += ALocal[0][1];
76          al[elemNumber] += ALocal[1][0];     di[elemNumber + 1] += ALocal[1][1];
77
78          A[elemNumber][elemNumber] += ALocal[0][0];      A[elemNumber][elemNumber
      + 1] += ALocal[0][1];
79          A[elemNumber + 1][elemNumber] += ALocal[1][0];  A[elemNumber + 1][
     elemNumber + 1] += ALocal[1][1];
80      }
81
82      // Первые краевые условия
83      A[0][0] = 1; A[0][1] = 0;
84      A[nodesCount - 1][nodesCount - 1] = 1; A[nodesCount - 1][nodesCount - 2] =
     0;
85      di[0] = 1;
86      au[0] = 0;
87      di[nodesCount - 1] = 1;
88      al[al.size() - 1] = 0;
89  }
90
91
92
93  // Строим глобальный вектор правой части системы нелинейных уравнений
94  void FEM::buildGlobalVectorb()
95  {
96      b.clear();
97      b.resize(nodesCount, 0);
98
99      for (size_t elemNumber = 0; elemNumber < finiteElementsCount; elemNumber++)
100     {
101         buildLocalVectorb(elemNumber);
102         b[elemNumber] += bLocal[0];
103         b[elemNumber + 1] += bLocal[1];
104     }
105
```

```
    b[0] = u(nodes[0].x, t);
    b[nodesCount - 1] = u(nodes[nodesCount - 1].x, t);
}


// Вывод матрицы A в консоль
void FEM::printGlobalMatrixA()
{
    ofstream fout("output/A.txt");
    cout << fixed << setprecision(3);
    fout << "A = [ ";
    for (size_t i = 0; i < nodesCount; i++)
    {
        for (size_t j = 0; j < nodesCount; j++)
        {
            fout << A[i][j] << "\t";
            cout << A[i][j] << "\t";
        }
        fout << ";" << endl;
        cout << endl;
    }
    fout << "]";

    fout << endl;
    fout << "di = {" << di << "};" << endl;
    fout << "al = {" << al << "};" << endl;
    fout << "au = {" << au << "};" << endl;
    fout.close();
}


// Вывод матрицы A в консоль
void FEM::printGlobalVectorb()
{
    ofstream fout("output/b.txt");
    cout << endl << fixed << setprecision(3);
    fout << "b = [ ";
    for (size_t i = 0; i < nodesCount; i++)
    {
        fout << b[i] << ";" << endl;
        cout << b[i] << endl;
    }
    fout << "]";
    fout << endl;
    fout << "b = {" << b << "};" << endl;
    fout.close();
}


//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

// Построение локальной матрицы жёсткости
void FEM::buildLocalMatrixG(int elemNumber)
{
    double arg = (q[elemNumber + 1] - q[elemNumber]) / hx;
    lambda0 = lambda(arg, nodes[elemNumber].x);
    lambda1 = lambda(arg, nodes[elemNumber+1].x);
    double numerator = (lambda0 + lambda1) / hx;
```

```
    GLocal[0][0] = GLocal[1][1] = numerator;
    GLocal[0][1] = GLocal[1][0] = -numerator;
}


// Построение локальной матрицы масс
void FEM::buildLocalMatrixM(int elemNumber)
{
    double numerator = (sigma * hx) / (6 * dt);
    MLocal[0][0] = MLocal[1][1] = 2 * numerator;
    MLocal[0][1] = MLocal[1][0] = numerator;
}


// Построение локальной матрицы A
void FEM::buildLocalmatrixA(int elemNumber)
{
    ALocal = GLocal = MLocal = { {0,0}, {0,0} };
    buildLocalMatrixG(elemNumber);
    buildLocalMatrixM(elemNumber);
    for (size_t i = 0; i < 2; i++)
    {
        for (size_t j = 0; j < 2; j++)
        {
            ALocal[i][j] = GLocal[i][j] + MLocal[i][j];
        }
    }
}


// Построение локального вектора b
void FEM::buildLocalVectorb(int elemNumber)
{
    bLocal = { 0, 0 };
    bLocal[0] = hx * (2 * f(nodes[elemNumber].x, t) + f(nodes[elemNumber + 1].x,
    t)) / 6
        + sigma * hx * (2 * qPrev[elemNumber] + qPrev[elemNumber + 1]) / (6 * dt
    );
    bLocal[1] = hx * (f(nodes[elemNumber + 1].x, t) + 2 * f(nodes[elemNumber +
    1].x, t)) / 6
        + sigma * hx * (qPrev[elemNumber] + 2 * qPrev[elemNumber + 1]) / (6 * dt
    );
}
```

solver.h

```
#pragma once
#include "head.h"

class SOLVER
{
public:
    void calcWithLUDecomposition();
    bool shouldCalc(int i);
    void LUdecomposition();
    void executeDirectTraversal();
    void executeReverseTraversal();
    void testSLAE();

protected:
    matrix2D A;
    vector1D di, al, au;
```

```
    vector1D b;
    vector1D q, qPrev;
    int maxiter;
    double E, delta;

    double calcNormE(const vector1D &x) { return sqrt(x*x); }
    vector1D multAonQ();
};
```

solver.cpp

```
#include "solver.h"


// Решение СЛАУ
void SOLVER::calcWithLUDecomposition()
{
    // LU разложение
    LUdecomposition();
    // Прямой ход
    executeDirectTraversal();
    // Обратный ход
    executeReverseTraversal();
}


// Проверяем условие выхода
bool SOLVER::shouldCalc(int i)
{
    // Выход по числу итераций
    if (i > maxiter) {
        //cout << endl << "STOP: maxiter" << endl
        //  << "Iter = " << i << endl << endl;
        return false;
    }
    // Выход шагу
    if (calcNormE(q - qPrev) / calcNormE(q) < delta) {
        //cout << endl << "STOP: step" << endl
        //  << "Iter = " << i << endl << endl;
        return false;
    }
    // Выход по относительной невязке
    if (calcNormE(multAonQ() - b) / calcNormE(b) < E) {
        //cout << endl << "STOP: E = " << (calcNormE(multAonQ() - b) / calcNormE
   (b)) << endl
        //  << "Iter = " << i << endl << endl;
        return false;
    }
    return true;
}


// LU разложение
void SOLVER::LUdecomposition()
{
    // 1 0 0 0   1 2 0 0   1 2 0 0
    // 3 2 0 0 * 0 1 2 0 = 3 8 4 0
    // 0 3 3 0   0 0 1 2   0 3 9 6
    // 0 0 3 4   0 0 0 1   0 0 3 10
    int lIndex = di.size();
    for (size_t i = 1; i < lIndex; i++)
    {
```

```
        au[i - 1] = au[i - 1] / di[i - 1];
        di[i] = di[i] - al[i - 1] * au[i - 1];
    }
}


// Прямой ход
// LUq=b, y=Uq
// Ly=b
void SOLVER::executeDirectTraversal()
{
    q[0] = b[0] / di[0];

    for (size_t i = 1; i < di.size(); i++)
        q[i] = (b[i] - al[i - 1] * q[i - 1]) / di[i];

    b = q;
}


// Обратный ход
// LUq=b, y=Uq
// Uq = y
void SOLVER::executeReverseTraversal()
{
    int lIndex = di.size() - 1;
    q[lIndex] = b[lIndex];

    for (int i = lIndex - 1; i >= 0; i--)
        q[i] = (b[i] - q[i + 1] * au[i]);
}


// Проверка решения СЛАУ
void SOLVER::testSLAE()
{
    /*di = { 1, 8, 9, 10 };
    al = { 3, 3, 3 };
    au = { 2, 4, 6 };
    b = { 5, 31, 57, 49 };
    q.resize(4, 0);
    calcWithLUDecomposition();
    cout << q << endl;*/
    di = { 1, 2.66667, 2.66667, 2.66667, 2.66667, 2.66667, 2.66667, 2.66667, 2.66667, 1 };
    al = { -0.833333, -0.833333, -0.833333, -0.833333, -0.833333, -0.833333, -0.833333, -0.833333, 0 };
    au = { 0, -0.833333, -0.833333, -0.833333, -0.833333, -0.833333, -0.833333, -0.833333, -0.833333 };
    b = { 1, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 10 };
    q.resize(10, 0);
    calcWithLUDecomposition();
    cout << q << endl;
}


// Умножение матрицы A на вектор q
vector1D SOLVER::multAonQ()
{
    vector1D tmp;
```

```
    tmp.resize(di.size());

    if (di.size() >= 2)
        tmp[0] = di[0] * q[0] + au[0] * q[1];

    if (di.size() >= 3)
        for (size_t i = 1; i < di.size() - 1; i++)
            tmp[i] = al[i - 1] * q[i - 1] + di[i] * q[i] + au[i] * q[i + 1];

    int lIndex = di.size() - 1;
    tmp[lIndex] = al[lIndex - 1] * q[lIndex - 1] + di[lIndex] * q[lIndex];
    return tmp;
}
```

main.cpp

```
#include "fem.h"
#include <thread>


function1D calcFirstDerivative(const function1D& f) {
    return [f](double x) -> double {
        const double h = 0.00001;
        return (-f(x + 2 * h) + 8 * f(x + h) - 8 * f(x - h) + f(x - 2 * h)) /
    (12 * h);
    };
}


function2D calcRightPart(const function2D& lambda, const function2D& u, double
    sigma) {
    return [=](double x, double t) -> double {
        using namespace std::placeholders;
        auto duBydt = calcFirstDerivative(std::bind(u, x, _1));
        auto duBydx = calcFirstDerivative(std::bind(u, _1, t));
        auto lambda_grad = [=](double x, double t) -> double {
            //return lambda(u(x, t)) * duBydx(x); // var 5
            return lambda(duBydx(x), x) * duBydx(x); // var 7
        };
        auto div = calcFirstDerivative(std::bind(lambda_grad, _1, t));
        return -div(x) + sigma * duBydt(t);
    };
}



void main() {

    vector <function2D> u(14), f(14);
    u[0] = { [](double x, double t) -> double { return 3 * x + t; } };
    u[1] = { [](double x, double t) -> double { return 2 * x*x + t; } };
    u[2] = { [](double x, double t) -> double { return x * x*x + t; } };
    u[3] = { [](double x, double t) -> double { return x * x*x*x + t; } };
    u[4] = { [](double x, double t) -> double { return exp(x) + t; } };
    u[5] = { [](double x, double t) -> double { return 3 * x + t; } };
    u[6] = { [](double x, double t) -> double { return 3 * x + t * t; } };
    u[7] = { [](double x, double t) -> double { return 3 * x + t * t*t; } };
    u[8] = { [](double x, double t) -> double { return 3 * x + exp(t); } };
    u[9] = { [](double x, double t) -> double { return 3 * x + sin(t); } };
    u[10] = { [](double x, double t) -> double { return exp(x) + t * t; } };
    u[11] = { [](double x, double t) -> double { return exp(x) + t * t*t; } };
    u[12] = { [](double x, double t) -> double { return exp(x) + exp(t); } };
```

```
    u[13] = { [](double x, double t) -> double { return exp(x) + sin(t); } };

    vector <function2D>  lambda(8);
    lambda[0] = { [](double u, double x) -> double {return 1; } };
    lambda[1] = { [](double u, double x) -> double {return u + x + 1; } };
    lambda[2] = { [](double u, double x) -> double {return u * u; } };
    lambda[3] = { [](double u, double x) -> double {return u * u + 1; } };
    lambda[4] = { [](double u, double x) -> double {return u * u*u; } };
    lambda[5] = { [](double u, double x) -> double {return u * u*u*u; } };
    lambda[6] = { [](double u, double x) -> double {return exp(u); } };
    lambda[7] = { [](double u, double x) -> double {return sin(u); } };

    vector <string> u_names = {
        "$ 3x + t $",
        "$ 2x ^ 2 + t $",
        "$ x ^ 3 + t $",
        "$ x ^ 4 + t $",
        "$ e^x + t $",
        "$ 3x + t $",
        "$ 3x + t ^ 2 $",
        "$ 3x + t ^ 3 $",
        "$ 3x + e ^ t $",
        "$ 3x + sin(t) $",
        "$ e^x + t ^ 2 $",
        "$ e^x + t ^ 3 $",
        "$ e^x + e ^ t $",
        "$ e^x + sin(t) $"
    };

    double sigma = 1;
    int coefGrid = 0;
    bool isGridUniform = true;
    bool isTimeUniform = true;

    ofstream foutTable("report/table.txt");
    foutTable << scientific << setprecision(2);
    foutTable << "a\t$1$\t$u+x+1$\t$u^2$\t$u^2+1$\t$u^3$\t$u^4$\t$e^u$\tsinu" <<
     endl;
    cout << "Research 1: convergence with diferent u and lambda" << endl;
    for (size_t i = 0; i < u.size(); i++)
    {
        foutTable << u_names[i] << "\t";
        for (size_t j = 0; j < lambda.size(); j++)
        {
            std::cout << int(float(i*lambda.size() + j) * 100.0 / (u.size()*
    lambda.size())) << " %\r";
            f[i] = calcRightPart(lambda[j], u[i], sigma);
            FEM fem;
            fem.init(u[i], f[i], lambda[j], sigma, isGridUniform, isTimeUniform,
     1, coefGrid, 0);
            fem.inputGrid();
            fem.buildGrid();
            fem.inputTime();
            fem.buildTimeGrid();
            foutTable << fem.solve().second;
            if (j + 1 < lambda.size())
                foutTable << "\t";
        }
        foutTable << endl;
    }
```

```
    foutTable.close();
    cout << endl;


    // Исследование точности при дроблении сетки
    auto reseacrhConvergence = [&](bool isGridUniform, bool isTimeUniform) {
        for (int i = 0; i < u.size(); i++) {
            f[i] = calcRightPart(lambda[1], u[i], sigma);

            string prefix = "";
            if (!isGridUniform)
                prefix = "Non";

            ofstream fout("report/file_u" + to_string(i) + "." + to_string(
    isGridUniform) + to_string(isTimeUniform) + ".txt");
            fout << scientific << fixed;
            fout << "i\tnodes\titers\tnorm\n";
            for (int coefGrid = 0; coefGrid < 5; coefGrid++)
            {
                string gridFile = "grids/" + prefix + "Uniform_" + to_string(
    coefGrid) + ".txt";
                string gridBorderFile = "grids/Border" + prefix + "Uniform_" +
    to_string(coefGrid) + ".txt";
                FEM fem;
                fem.init(u[i], f[i], lambda[1], sigma, isGridUniform,
    isTimeUniform, 1, coefGrid, 0);
                fem.inputGrid();
                fem.buildGrid();
                fem.inputTime();
                fem.buildTimeGrid();
                auto result = fem.solve();
                fout << coefGrid << "\t"
                    << fem.getNodesCount() << "\t"
                    << result.first << "\t"
                    << result.second << endl;
            }
            fout.close();
        }
    };


    thread R21(reseacrhConvergence, true, true);
    thread R22(reseacrhConvergence, true, false);
    thread R23(reseacrhConvergence, false, true);
    thread R24(reseacrhConvergence, false, false);

    R21.join();
    cout << "Research 2.1: convergence with grid crushing" << endl;
    R22.join();
    cout << "Research 2.2: convergence with grid crushing" << endl;
    R23.join();
    cout << "Research 2.3: convergence with grid crushing" << endl;
    R24.join();
    cout << "Research 2.4: convergence with grid crushing" << endl;
}
```