Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет

Уравнения математической физики
Лабораторная работа №2

Факультет:	ФПМИ
Группа:	ПМ-63
Студент:	Утюганов Д.С.
Вариант:	5

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 5: Базисные функции линейные.

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

3. Анализ

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

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

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

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

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

$$G_{0,0} = \sum_{k=0}^{1}\int_\Omega \lambda_k\psi_k grad\psi_0 grad\psi_0 d\Omega =$$

$$= \int_\Omega \lambda_0\psi_0 grad\psi_0 grad\psi_0 d\Omega + \int_\Omega \lambda_1\psi_1 grad\psi_0 grad\psi_0 d\Omega =$$

$$= \frac{1}{h}\Big[\lambda_0\int_\Omega \psi_0 d\Omega + \lambda_1\int_\Omega \psi_1 d\Omega\Big] = \frac{1}{h}\Big[\lambda_0\int_0^1 \xi d\xi + \lambda_1\int_0^1 (1-\xi)d\xi\Big] =$$

$$= \frac{1}{h}\Big[\lambda_0\frac{\xi^2}{2}\Big|_0^1 + \lambda_1(\xi - \frac{\xi^2}{2})\Big|_0^1\Big] = \frac{\lambda_0+\lambda_1}{2h} = G_{1,1}$$

$$G_{0,1} = \sum_{k=0}^{1}\int_\Omega \lambda_k\psi_k grad\psi_1 grad\psi_1 d\Omega =$$

$$= \int_\Omega \lambda_0\psi_0 grad\psi_1 grad\psi_1 d\Omega + \int_\Omega \lambda_1\psi_1 grad\psi_1 grad\psi_1 d\Omega =$$

$$= -\frac{1}{h}\Big[\lambda_0\int_\Omega \psi_0 d\Omega + \lambda_1\int_\Omega \psi_1 d\Omega\Big] = -\frac{1}{h}\Big[\lambda_0\int_0^1 \xi d\xi + \lambda_1\int_0^1 (1-\xi)d\xi\Big] =$$

$$= -\frac{1}{h}\Big[\lambda_0\frac{\xi^2}{2}\Big|_0^1 + \lambda_1(\xi - \frac{\xi^2}{2})\Big|_0^1\Big] = -\frac{\lambda_0+\lambda_1}{2h} = G_{1,0}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

---

4. Точность для разных функций u и λ

В ходе следующего исследования использовались следующие параметры:
$\varepsilon = 1e-7$
$\sigma = 1$
$maxiter = 10000$
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	u^2	u^2+1	u^3	u^4	e^u	$\sin u$
$3x+t$	2.62e-02	1.13e-02	5.63e-04	4.37e-04	4.81e-01	1.52e+00	1.14e-02	-nan(ind)
$2x^2+t$	2.62e-02	1.11e-02	7.76e-03	5.42e-03	1.03e+00	2.14e+00	1.51e-02	-nan(ind)
x^3+t	2.36e-02	1.59e-02	7.90e-03	3.30e-03	7.52e-03	3.00e-02	2.36e-03	-nan(ind)
x^4+t	2.10e-02	1.26e-02	6.28e-03	4.68e-03	2.31e-02	4.02e-01	8.20e-03	-nan(ind)
e^x+t	2.55e-02	7.74e-03	8.97e-04	8.72e-04	6.53e-03	1.29e-02	6.44e-03	-nan(ind)
$3x+t$	2.62e-02	1.13e-02	5.63e-04	4.37e-04	4.81e-01	1.52e+00	1.14e-02	-nan(ind)
$3x+t^2$	5.25e-02	2.24e-02	3.32e-02	4.77e-03	1.18e-01	5.07e+00	8.48e-03	-nan(ind)
$3x+t^3$	7.87e-02	3.33e-02	6.52e-02	9.03e-03	1.91e+00	2.90e-01	5.68e-03	-nan(ind)
$3x+e^t$	7.13e-02	1.72e-02	2.93e-03	2.74e-03	3.35e-03	9.49e-03	1.29e-02	-nan(ind)
$3x+sin(t)$	1.42e-02	6.66e-03	2.45e-03	1.79e-03	2.00e-01	1.63e+00	1.25e-02	-nan(ind)
e^x+t^2	5.17e-02	1.75e-02	3.36e-03	2.79e-03	4.98e-03	1.22e-02	4.54e-03	inf
e^x+t^3	7.80e-02	2.72e-02	7.16e-03	6.08e-03	3.48e-03	1.15e-02	2.79e-03	inf
e^x+e^t	7.06e-02	1.46e-02	1.04e-03	9.48e-04	3.52e-03	6.19e-03	7.44e-03	-nan(ind)
$e^x+sin(t)$	1.34e-02	3.44e-03	2.51e-03	2.27e-03	7.81e-03	1.47e-02	7.15e-03	-nan(ind)

4.1. Вывод

Как видно из таблицы метод начинает сходиться хуже при степени полинома выше двух. Метод не работает если функция λ гармоническая (в нашем случае sin(u). Это происходит даже при численном взятии интегралов при построении локальной матрицы A и вектора b.

5. Точность решения при дроблении сетки

В ходе следующего исследования использовались следующие параметры:
$\varepsilon = 1e-7$
$\sigma = 1$
$maxiter = 10000$
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 $\lambda(u) = u$

$3x+t$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	111	0.011339	0	11	96	0.135681
	1	21	109	0.008403	1	21	114	0.248637
	2	41	111	0.006087	2	41	120	0.215011
	3	81	109	0.004358	3	81	126	0.181014
	4	161	108	0.003100	4	161	126	0.141060
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	123	0.011339	0	11	94	0.135681
	1	21	122	0.008403	1	21	119	0.248637
	2	41	119	0.006087	2	41	125	0.215011
	3	81	119	0.004358	3	81	132	0.181014
	4	161	125	0.003100	4	161	136	0.141060

$2x^2 + t$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	206	0.011104	0	11	89	0.161428
	1	21	206	0.010649	1	21	107	0.212827
	2	41	205	0.008589	2	41	110	0.168092
	3	81	200	0.006461	3	81	115	0.129943
	4	161	202	0.004708	4	161	115	0.096639
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	247	0.011104	0	11	92	0.161428
	1	21	233	0.010649	1	21	113	0.212827
	2	41	233	0.008589	2	41	117	0.168092
	3	81	230	0.006461	3	81	123	0.129943
	4	161	232	0.004708	4	161	123	0.096639

$x^3 + t$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	261	0.015859	0	11	108	0.096806
	1	21	257	0.014021	1	21	92	0.111232
	2	41	250	0.010968	2	41	96	0.085477
	3	81	245	0.008141	3	81	97	0.064514
	4	161	245	0.005895	4	161	98	0.047343
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	314	0.015859	0	11	118	0.096806
	1	21	289	0.014021	1	21	97	0.111232
	2	41	288	0.010968	2	41	103	0.085477
	3	81	289	0.008141	3	81	106	0.064514
	4	161	288	0.005895	4	161	106	0.047343

$x^4 + t$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	280	0.012571	0	11	108	0.108551
	1	21	271	0.013159	1	21	92	0.116517
	2	41	258	0.010911	2	41	96	0.088139
	3	81	262	0.008302	3	81	97	0.065598
	4	161	258	0.006081	4	161	97	0.047756
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	329	0.012571	0	11	121	0.108551
	1	21	311	0.013159	1	21	97	0.116517
	2	41	304	0.010911	2	41	103	0.088139
	3	81	299	0.008302	3	81	106	0.065598
	4	161	305	0.006081	4	161	106	0.047756

$e^x + t$

пространство / время	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	127	0.007737	0	11	76	0.091407
	1	21	130	0.006543	1	21	72	0.145897
	2	41	127	0.005032	2	41	71	0.122113
	3	81	126	0.003707	3	81	72	0.099997
	4	161	125	0.002675	4	161	72	0.076895
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	136	0.007737	0	11	79	0.091407
	1	21	137	0.006543	1	21	73	0.145897
	2	41	136	0.005032	2	41	73	0.122113
	3	81	136	0.003707	3	81	73	0.099997
	4	161	135	0.002675	4	161	73	0.076895

$3x + t$

пространство / время	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	111	0.011339	0	11	96	0.135681
	1	21	109	0.008403	1	21	114	0.248637
	2	41	111	0.006087	2	41	120	0.215011
	3	81	109	0.004358	3	81	126	0.181014
	4	161	108	0.003100	4	161	126	0.141060
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	123	0.011339	0	11	94	0.135681
	1	21	122	0.008403	1	21	119	0.248637
	2	41	119	0.006087	2	41	125	0.215011
	3	81	119	0.004358	3	81	132	0.181014
	4	161	125	0.003100	4	161	136	0.141060

$3x + t^2$

пространство / время	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	118	0.022432	0	11	98	0.139644
	1	21	118	0.016625	1	21	120	0.248699
	2	41	120	0.012043	2	41	130	0.215013
	3	81	121	0.008621	3	81	140	0.181014
	4	161	122	0.006134	4	161	146	0.141060
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	125	0.022432	0	11	98	0.139644
	1	21	123	0.016625	1	21	121	0.248699
	2	41	126	0.012043	2	41	132	0.215013
	3	81	125	0.008621	3	81	145	0.181014
	4	161	138	0.006134	4	161	152	0.141060

$3x + t^3$

пространство / время	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	120	0.033296	0	11	98	0.143583
	1	21	124	0.024678	1	21	120	0.248761
	2	41	128	0.017876	2	41	132	0.215015
	3	81	131	0.012796	3	81	142	0.181014
	4	161	137	0.009105	4	161	152	0.141060
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	122	0.033296	0	11	95	0.143583
	1	21	125	0.024678	1	21	118	0.248761
	2	41	131	0.017876	2	41	131	0.215015
	3	81	132	0.012796	3	81	142	0.181014
	4	161	137	0.009105	4	161	157	0.141060

$3x + e^t$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	90	0.017242	0	11	78	0.114386
	1	21	90	0.012774	1	21	83	0.223047
	2	41	90	0.009253	2	41	83	0.197016
	3	81	90	0.006624	3	81	83	0.168909
	4	161	91	0.004713	4	161	83	0.132925
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	107	0.017242	0	11	78	0.114386
	1	21	103	0.012774	1	21	84	0.223047
	2	41	102	0.009253	2	41	86	0.197016
	3	81	106	0.006624	3	81	85	0.168909
	4	161	102	0.004713	4	161	85	0.132925

$3x + sin(t)$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	112	0.006664	0	11	96	0.138054
	1	21	109	0.004940	1	21	114	0.252773
	2	41	113	0.003578	2	41	120	0.217918
	3	81	110	0.002562	3	81	126	0.182964
	4	161	110	0.001823	4	161	127	0.142370
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	123	0.006664	0	11	94	0.138054
	1	21	122	0.004940	1	21	119	0.252773
	2	41	119	0.003578	2	41	125	0.217918
	3	81	123	0.002562	3	81	133	0.182964
	4	161	125	0.001823	4	161	137	0.142370

$e^x + t^2$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	133	0.017536	0	11	78	0.094965
	1	21	131	0.013792	1	21	74	0.145953
	2	41	132	0.010279	2	41	76	0.122114
	3	81	133	0.007461	3	81	76	0.099997
	4	161	137	0.005345	4	161	76	0.076895
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	143	0.017536	0	11	81	0.094965
	1	21	145	0.013792	1	21	74	0.145953
	2	41	141	0.010279	2	41	74	0.122114
	3	81	142	0.007461	3	81	75	0.099997
	4	161	145	0.005345	4	161	75	0.076895

$e^x + t^3$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	133	0.027190	0	11	79	0.098510
	1	21	133	0.020932	1	21	73	0.146009
	2	41	132	0.015447	2	41	75	0.122116
	3	81	132	0.011159	3	81	75	0.099997
	4	161	135	0.007976	4	161	75	0.076895
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	142	0.027190	0	11	78	0.098510
	1	21	138	0.020932	1	21	73	0.146009
	2	41	141	0.015447	2	41	75	0.122116
	3	81	142	0.011159	3	81	75	0.099997
	4	161	141	0.007976	4	161	75	0.076895

$e^x + e^t$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	105	0.014611	0	11	64	0.086740
	1	21	107	0.011423	1	21	61	0.139089
	2	41	106	0.008490	2	41	62	0.117296
	3	81	106	0.006155	3	81	62	0.096715
	4	161	105	0.004407	4	161	62	0.074665
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	114	0.014611	0	11	70	0.086740
	1	21	110	0.011423	1	21	63	0.139089
	2	41	111	0.008490	2	41	63	0.117296
	3	81	112	0.006155	3	81	63	0.096715
	4	161	113	0.004407	4	161	63	0.074665

$e^x + sin(t)$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	11	127	0.003444	0	11	75	0.091008
	1	21	129	0.003399	1	21	72	0.146931
	2	41	127	0.002769	2	41	71	0.122855
	3	81	126	0.002092	3	81	71	0.100502
	4	161	128	0.001527	4	161	71	0.077238
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	11	136	0.003444	0	11	79	0.091008
	1	21	137	0.003399	1	21	72	0.146931
	2	41	136	0.002769	2	41	73	0.122855
	3	81	136	0.002092	3	81	73	0.100502
	4	161	135	0.001527	4	161	73	0.077238

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 0.5

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
```

```cpp
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;
}

```

```cpp

// 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

```cpp
#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   10
    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 function1D &
    _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;
    function1D 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 function1D &
    _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));
```

```cpp
}


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

// Строим глобальную матрицу системы нелинейных уравнений
void FEM::buildGlobalMatrixA(double _dt)
{
    dt = _dt;
    A.clear();
    di.clear();
    au.clear();
    al.clear();
    A.resize(nodesCount);
    for (size_t i = 0; i < nodesCount; i++)
        A[i].resize(nodesCount, 0);

    di.resize(nodesCount, 0);
    al.resize(nodesCount - 1, 0);
    au.resize(nodesCount - 1, 0);
    for (size_t elemNumber = 0; elemNumber < finiteElementsCount; elemNumber++)
    {
        buildLocalmatrixA(elemNumber);

        //cout << ALocal << endl;
        di[elemNumber] += ALocal[0][0];     au[elemNumber] += ALocal[0][1];
        al[elemNumber] += ALocal[1][0];     di[elemNumber + 1] += ALocal[1][1];

        A[elemNumber][elemNumber] += ALocal[0][0];      A[elemNumber][elemNumber + 1] += ALocal[0][1];
        A[elemNumber + 1][elemNumber] += ALocal[1][0];  A[elemNumber + 1][elemNumber + 1] += ALocal[1][1];
    }

    // Первые краевые условия
    A[0][0] = 1; A[0][1] = 0;
    A[nodesCount - 1][nodesCount - 1] = 1; A[nodesCount - 1][nodesCount - 2] = 0;
    di[0] = 1;
    au[0] = 0;
    di[nodesCount - 1] = 1;
    al[al.size() - 1] = 0;
}



// Строим глобальный вектор правой части системы нелинейных уравнений
void FEM::buildGlobalVectorb()
{
    b.clear();
    b.resize(nodesCount, 0);

    for (size_t elemNumber = 0; elemNumber < finiteElementsCount; elemNumber++)
    {
        buildLocalVectorb(elemNumber);
        b[elemNumber] += bLocal[0];
        b[elemNumber + 1] += bLocal[1];
    }
```

```

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


// Вывод матрицы А 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 А 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)
{
    lambda0 = lambda(q[elemNumber]);
    lambda1 = lambda(q[elemNumber + 1]);
    double numerator = (lambda0 + lambda1) / (2 * 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++)
    {
```

```cpp
        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;
```

```cpp
    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

```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 function1D& 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);
            //return lambda(duBydt(t)) * duBydx(x);
        };
        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 <function1D>  lambda(8);
    lambda[0] = { [](double u) -> double {return 1; } };
    lambda[1] = { [](double u) -> double {return u; } };
    lambda[2] = { [](double u) -> double {return u * u; } };
    lambda[3] = { [](double u) -> double {return u * u + 1; } };
    lambda[4] = { [](double u) -> double {return u * u*u; } };
    lambda[5] = { [](double u) -> double {return u * u*u*u; } };
    lambda[6] = { [](double u) -> double {return exp(u); } };
    lambda[7] = { [](double u) -> 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$u$\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;
    }
    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;
}
```