Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет

Уравнения математической физики
Лабораторная работа №3, 4

Факультет: ФПМИ
Группа: ПМ-63
Студент: Кожекин М.В.
Вариант: 1, 4

Новосибирск
2019

1. Цель работы

Разработать программу решения гармонической задачи методом конечных элементов. Сравнить прямой и итерационные методы решения получаемой в результате конечноэлементной аппроксимации СЛАУ.

2. Задание

1. Выполнить конечноэлементную аппроксимацию исходного уравнения в соответствии с заданием. Получить формулы для вычисления компонент матрицы **A** и вектора правой части **b**.

2. Реализовать программу решения гармонической задачи с учетом следующих требований:

- язык программирования C++ или Фортран;
- предусмотреть возможность задания неравномерной сетки по пространству, разрывность параметров уравнения по подобластям, учет краевых условий;
- матрицу хранить в разреженном строчно-столбцовом формате с возможностью перегенерации ее в профильный формат;
- реализовать (или воспользоваться реализованными в курсе «Численные методы») методы решения СЛАУ: итерационный – локально-оптимальную схему или метод сопряженных градиентов для несимметричных матриц с предобусловливанием и прямой – LU-разложение или его модификации [2, с. 871; 3].

3. Протестировать разработанную программу на полиномах первой степени.

4. Исследовать реализованные методы для сеток с небольшим количеством узлов 500 – 1000 и большим количеством узлов – примерно 20 000 – 50 000 для различных значений параметров $10^{-4} \leq \omega \leq 10^9$, $10^2 \leq \lambda \leq 8 \cdot 10^5$, $0 \leq \sigma \leq 10^8$, $8.81 \cdot 10^{-12} \leq \chi \leq 10^{-10}$. Для всех решенных задач сравнить вычислительные затраты, требуемые для решения СЛАУ итерационным и прямым методом.

Вариант 1: Решить одномерную гармоническую задачу в декартовых координатах, базисные функции - линейные.

Вариант 4: Решить СЛАУ методом BSG без предобуславливания.

3. Анализ

Рассмотрим задачу для уравнения:

$$\chi\frac{d^2u}{dt^2} + \sigma\frac{du}{dt} - div(\lambda grad(u)) = f$$

Решение данного уравния u и его правая часть f представимы в виде:

$$u(x,y,t) = u^s sin\omega t + u^c cos\omega t$$

$$f(x,y,t) = f^s sin\omega t + f^c cos\omega t$$

Значит исходное уравнение можно привести к системе уравнений

$$-div(\lambda grad(u^s)) - \omega\sigma u^c - \omega^2\chi u^s = f$$

$$-div(\lambda grad(u^c)) - \omega\sigma u^s - \omega^2\chi u^c = f$$

$$p_{ij}(q_s) = \int_\Omega \Big(\lambda grad\psi_i grad\psi_j - \omega^2 \chi \psi_i \psi_j\Big) d\Omega$$

$$c_{ij}(q_s) = \omega \int_\Omega \sigma \psi_i \psi_j d\Omega$$

Матрица конечноэлементной СЛАУ будет иметь следующую структуру:

$$\begin{pmatrix} p_{00} & -c_{00} & p_{01} & -c_{01} \\ c_{00} & p_{00} & c_{01} & p_{01} \\ p_{10} & -c_{10} & p_{11} & -c_{11} \\ c_{10} & p_{10} & c_{11} & p_{11} \end{pmatrix}$$

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

$u_s(x,t)$ \ $u_c(x,t)$	1	x	x^2	x^3	x^4	x^5	$sin(x)$	$\sin u$
1	2.66e-01	1.55e-01	1.41e-01	1.33e-01	1.29e-01	1.26e-01	1.47e-01	3.30e-01
x	1.84e-01	1.03e-01	8.74e-02	7.99e-02	7.55e-02	7.26e-02	9.44e-02	2.52e-01
x^2	1.80e-01	9.93e-02	8.30e-02	7.50e-02	7.03e-02	6.73e-02	9.02e-02	2.49e-01
x^3	1.78e-01	9.78e-02	8.11e-02	7.29e-02	6.80e-02	6.49e-02	8.85e-02	2.48e-01
x^4	1.77e-01	9.70e-02	8.01e-02	7.18e-02	6.69e-02	6.36e-02	8.76e-02	2.47e-01
x^5	1.76e-01	9.67e-02	7.97e-02	7.13e-02	6.63e-02	6.30e-02	8.72e-02	2.47e-01
$sin(x)$	1.82e-01	1.01e-01	8.54e-02	7.77e-02	7.32e-02	7.02e-02	9.25e-02	2.51e-01
e^x	2.87e-01	1.76e-01	1.64e-01	1.57e-01	1.54e-01	1.51e-01	1.70e-01	3.48e-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:
$\varepsilon = 1e - 22$
$\sigma = 1$
$maxiter = 1000$, 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

$u_s = x, u_c = -2 \cdot x$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	21	0	2.655e-01	0	21	0	2.705e-01
	1	41	0	1.887e-01	1	41	0	1.925e-01
	2	81	0	1.338e-01	2	81	2	1.365e-01
	3	161	0	9.471e-02	3	161	2	9.667e-02
	4	321	0	6.702e-02	4	321	3	6.841e-02
	5	641	0	4.740e-02	5	641	3	4.839e-02
	6	1281	0	3.352e-02	6	1281	3	3.423e-02
	7	2561	0	2.371e-02	7	2561	1	-nan(ind)
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	21	0	2.655e-01	0	21	0	2.705e-01
	1	41	0	1.887e-01	1	41	0	1.925e-01
	2	81	0	1.338e-01	2	81	2	1.365e-01
	3	161	0	9.471e-02	3	161	2	9.667e-02
	4	321	0	6.702e-02	4	321	3	6.841e-02
	5	641	0	4.740e-02	5	641	3	4.839e-02
	6	1281	0	3.352e-02	6	1281	3	3.423e-02
	7	2561	0	2.371e-02	7	2561	1	-nan(ind)

$u_s = x, u_c = -2 \cdot x$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	21	0	1.032e-01	0	21	0	7.039e-02
	1	41	0	7.255e-02	1	41	0	2.947e-02
	2	81	0	5.116e-02	2	81	0	1.620e-02
	3	161	0	3.613e-02	3	161	0	8.461e-03
	4	321	0	2.553e-02	4	321	0	4.336e-03
	5	641	0	1.805e-02	5	641	0	2.202e-03
	6	1281	0	1.276e-02	6	1281	1	-nan(ind)
	7	2561	0	9.022e-03	7	2561	1	-nan(ind)
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	21	0	1.032e-01	0	21	0	7.039e-02
	1	41	0	7.255e-02	1	41	0	2.947e-02
	2	81	0	5.116e-02	2	81	0	1.620e-02
	3	161	0	3.613e-02	3	161	0	8.461e-03
	4	321	0	2.553e-02	4	321	0	4.336e-03
	5	641	0	1.805e-02	5	641	0	2.202e-03
	6	1281	0	1.276e-02	6	1281	1	-nan(ind)
	7	2561	0	9.022e-03	7	2561	1	-nan(ind)

$u_s = x^2, u_c = -2 \cdot x^2$

время \ пространство	равномерное				не равномерное			
	i	nodes	iters	norm	i	nodes	iters	norm
равномерное	0	21	0	8.298e-02	0	21	0	5.701e-02
	1	41	0	5.728e-02	1	41	0	2.580e-02
	2	81	0	4.001e-02	2	81	0	1.364e-02
	3	161	0	2.812e-02	3	161	0	7.019e-03
	4	321	0	1.982e-02	4	321	0	3.569e-03
	5	641	0	1.400e-02	5	641	0	1.803e-03
	6	1281	0	9.890e-03	6	1281	1	-nan(ind)
	7	2561	0	6.990e-03	7	2561	1	-nan(ind)
	i	nodes	iters	norm	i	nodes	iters	norm
не равномерное	0	21	0	8.298e-02	0	21	0	5.701e-02
	1	41	0	5.728e-02	1	41	0	2.580e-02
	2	81	0	4.001e-02	2	81	0	1.364e-02
	3	161	0	2.812e-02	3	161	0	7.019e-03
	4	321	0	1.982e-02	4	321	0	3.569e-03
	5	641	0	1.400e-02	5	641	0	1.803e-03
	6	1281	0	9.890e-03	6	1281	1	-nan(ind)
	7	2561	0	6.990e-03	7	2561	1	-nan(ind)

$u_s = x^3, u_c = -2 \cdot x^3$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	0	7.289e-02	0	21	0	5.230e-02
	1	41	0	4.935e-02	1	41	0	2.489e-02
	2	81	0	3.414e-02	2	81	0	1.289e-02
	3	161	0	2.388e-02	3	161	0	6.569e-03
	4	321	0	1.680e-02	4	321	0	3.323e-03
	5	641	0	1.184e-02	5	641	0	1.673e-03
	6	1281	0	8.363e-03	6	1281	1	-nan(ind)
	7	2561	0	5.910e-03	7	2561	1	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	0	7.289e-02	0	21	0	5.230e-02
	1	41	0	4.935e-02	1	41	0	2.489e-02
	2	81	0	3.414e-02	2	81	0	1.289e-02
	3	161	0	2.388e-02	3	161	0	6.569e-03
	4	321	0	1.680e-02	4	321	0	3.323e-03
	5	641	0	1.184e-02	5	641	0	1.673e-03
	6	1281	0	8.363e-03	6	1281	1	-nan(ind)
	7	2561	0	5.910e-03	7	2561	1	-nan(ind)

$u_s = x^4, u_c = -2 \cdot x^4$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	0	6.687e-02	0	21	0	5.017e-02
	1	41	0	4.438e-02	1	41	0	2.459e-02
	2	81	0	3.041e-02	2	81	0	1.260e-02
	3	161	0	2.116e-02	3	161	0	6.387e-03
	4	321	0	1.485e-02	4	321	0	3.220e-03
	5	641	0	1.046e-02	5	641	0	1.618e-03
	6	1281	0	7.380e-03	6	1281	1	-nan(ind)
	7	2561	0	5.213e-03	7	2561	1	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	0	6.687e-02	0	21	0	5.017e-02
	1	41	0	4.438e-02	1	41	0	2.459e-02
	2	81	0	3.041e-02	2	81	0	1.260e-02
	3	161	0	2.116e-02	3	161	0	6.387e-03
	4	321	0	1.485e-02	4	321	0	3.220e-03
	5	641	0	1.046e-02	5	641	0	1.618e-03
	6	1281	0	7.380e-03	6	1281	1	-nan(ind)
	7	2561	0	5.213e-03	7	2561	1	-nan(ind)

$u_s = x^5, u_c = -2 \cdot x^5$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	0	6.296e-02	0	21	0	4.909e-02
	1	41	0	4.095e-02	1	41	0	2.448e-02
	2	81	0	2.777e-02	2	81	0	1.247e-02
	3	161	0	1.924e-02	3	161	0	6.304e-03
	4	321	0	1.346e-02	4	321	0	3.172e-03
	5	641	0	9.471e-03	5	641	0	1.592e-03
	6	1281	0	6.680e-03	6	1281	1	-nan(ind)
	7	2561	0	4.717e-03	7	2561	1	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	0	6.296e-02	0	21	0	4.909e-02
	1	41	0	4.095e-02	1	41	0	2.448e-02
	2	81	0	2.777e-02	2	81	0	1.247e-02
	3	161	0	1.924e-02	3	161	0	6.304e-03
	4	321	0	1.346e-02	4	321	0	3.172e-03
	5	641	0	9.471e-03	5	641	0	1.592e-03
	6	1281	0	6.680e-03	6	1281	1	-nan(ind)
	7	2561	0	4.717e-03	7	2561	1	-nan(ind)

$u_s = sin(x), u_c = -2 \cdot sin(x)$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	0	9.254e-02	0	21	0	6.313e-02
	1	41	0	6.536e-02	1	41	0	2.593e-02
	2	81	0	4.619e-02	2	81	0	1.439e-02
	3	161	0	3.265e-02	3	161	0	7.547e-03
	4	321	0	2.308e-02	4	321	0	3.874e-03
	5	641	0	1.632e-02	5	641	0	1.970e-03
	6	1281	0	1.154e-02	6	1281	1	-nan(ind)
	7	2561	0	8.159e-03	7	2561	1	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	0	9.254e-02	0	21	0	6.313e-02
	1	41	0	6.536e-02	1	41	0	2.593e-02
	2	81	0	4.619e-02	2	81	0	1.439e-02
	3	161	0	3.265e-02	3	161	0	7.547e-03
	4	321	0	2.308e-02	4	321	0	3.874e-03
	5	641	0	1.632e-02	5	641	0	1.970e-03
	6	1281	0	1.154e-02	6	1281	1	-nan(ind)
	7	2561	0	8.159e-03	7	2561	1	-nan(ind)

$u_s = e^x, u_c = -2 \cdot e^x$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	0	3.480e-01	0	21	0	3.184e-01
	1	41	0	2.461e-01	1	41	0	2.059e-01
	2	81	0	1.740e-01	2	81	2	1.424e-01
	3	161	0	1.231e-01	3	161	2	9.901e-02
	4	321	0	8.706e-02	4	321	3	6.929e-02
	5	641	0	6.156e-02	5	641	3	4.872e-02
	6	1281	0	4.353e-02	6	1281	3	3.434e-02
	7	2561	0	3.078e-02	7	2561	1	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	0	3.480e-01	0	21	0	3.184e-01
	1	41	0	2.461e-01	1	41	0	2.059e-01
	2	81	0	1.740e-01	2	81	2	1.424e-01
	3	161	0	1.231e-01	3	161	2	9.901e-02
	4	321	0	8.706e-02	4	321	3	6.929e-02
	5	641	0	6.156e-02	5	641	3	4.872e-02
	6	1281	0	4.353e-02	6	1281	3	3.434e-02
	7	2561	0	3.078e-02	7	2561	1	-nan(ind)

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. Решатель из 4 лабораторной работы

Классическая схема **метода бисопряжённых градиентов** выглядит следующим образом:

Выбирается начальное приближение x^0 и полагается

$r^0 = b - Ax^0$
$p^0 = r^0$
$z^0 = r^0$
$s^0 = r^0$

Далее для k=1,2,... производятся следующие вычисления:

$$\alpha_k = \frac{(p^{k-1}, r^{k-1})}{(s^{k-1}, Az^{k-1})}$$

$$x^k = x^{k-1} + \alpha_k z^{k-1}$$
$$r^k = r^{k-1} - \alpha_k A z^{k-1}$$
$$p^k = p^{k-1} - \alpha_k A^T s^{k-1}$$

$$\beta_k = \frac{(p^k, r^k)}{(p^{k-1}, r^{k-1})}$$

$$z^k = x^{k-1} + \beta_k z^{k-1}$$
$$s^k = p^{k-1} + \beta_k s^{k-1}$$

7. 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

$u_s(x,t)$ \ $u_c(x,t)$	1	x	x^2	x^3	x^4	x^5	$sin(x)$	$\sin u$
1	2.66e-01	1.55e-01	1.41e-01	1.33e-01	1.29e-01	1.26e-01	1.47e-01	3.30e-01
x	1.84e-01	1.03e-01	8.74e-02	7.99e-02	7.55e-02	7.26e-02	9.44e-02	2.52e-01
x^2	1.80e-01	9.93e-02	8.30e-02	7.50e-02	7.03e-02	6.73e-02	9.02e-02	2.49e-01
x^3	1.78e-01	9.78e-02	8.11e-02	7.29e-02	6.80e-02	6.49e-02	8.85e-02	2.48e-01
x^4	1.77e-01	9.70e-02	8.01e-02	7.18e-02	6.69e-02	6.36e-02	8.76e-02	2.47e-01
x^5	1.76e-01	9.67e-02	7.97e-02	7.13e-02	6.63e-02	6.30e-02	8.72e-02	2.47e-01
$sin(x)$	1.82e-01	1.01e-01	8.54e-02	7.77e-02	7.32e-02	7.02e-02	9.25e-02	2.51e-01
e^x	2.87e-01	1.76e-01	1.64e-01	1.57e-01	1.54e-01	1.51e-01	1.70e-01	3.48e-01

7.1. 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.

8. Точность решения при дроблении сетки

В ходе следующего исследования использовались следующие параметры:
$\varepsilon = 1e - 22$
$\sigma = 1$

$maxiter = 1000$, 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

$u_s = x, u_c = -2 \cdot x$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	2.655e-01	0	21	40	2.705e-01
	1	41	80	1.887e-01	1	41	1	2.343e-01
	2	81	160	1.338e-01	2	81	1	1.667e-01
	3	161	320	9.471e-02	3	161	1	1.182e-01
	4	321	643	6.702e-02	4	321	1	8.372e-02
	5	641	1001	4.681e-02	5	641	1	5.925e-02
	6	1281	1001	4.307e-02	6	1281	1	4.191e-02
	7	2561	1001	3.023e-02	7	2561	1001	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	2.655e-01	0	21	40	2.705e-01
	1	41	80	1.887e-01	1	41	1	2.343e-01
	2	81	160	1.338e-01	2	81	1	1.667e-01
	3	161	320	9.471e-02	3	161	1	1.182e-01
	4	321	643	6.702e-02	4	321	1	8.372e-02
	5	641	1001	4.681e-02	5	641	1	5.925e-02
	6	1281	1001	4.307e-02	6	1281	1	4.191e-02
	7	2561	1001	3.023e-02	7	2561	1001	-nan(ind)

$u_s = x, u_c = -2 \cdot x$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	1.032e-01	0	21	40	7.039e-02
	1	41	82	7.255e-02	1	41	1	1.175e-01
	2	81	161	5.116e-02	2	81	1	8.160e-02
	3	161	322	3.613e-02	3	161	1	5.690e-02
	4	321	887	2.553e-02	4	321	1	3.990e-02
	5	641	1001	1.805e-02	5	641	1	2.809e-02
	6	1281	1001	1.397e-02	6	1281	1	1.981e-02
	7	2561	1001	1.871e-02	7	2561	1001	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	1.032e-01	0	21	40	7.039e-02
	1	41	82	7.255e-02	1	41	1	1.175e-01
	2	81	161	5.116e-02	2	81	1	8.160e-02
	3	161	322	3.613e-02	3	161	1	5.690e-02
	4	321	887	2.553e-02	4	321	1	3.990e-02
	5	641	1001	1.805e-02	5	641	1	2.809e-02
	6	1281	1001	1.397e-02	6	1281	1	1.981e-02
	7	2561	1001	1.871e-02	7	2561	1001	-nan(ind)

$u_s = x^2, u_c = -2 \cdot x^2$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	8.298e-02	0	21	40	5.701e-02
	1	41	97	5.728e-02	1	41	1	1.160e-01
	2	81	160	4.001e-02	2	81	1	8.076e-02
	3	161	404	2.812e-02	3	161	1	5.655e-02
	4	321	808	1.982e-02	4	321	1	3.977e-02
	5	641	1001	1.400e-02	5	641	1	2.804e-02
	6	1281	1001	1.102e-02	6	1281	1	1.980e-02
	7	2561	1001	1.726e-02	7	2561	1001	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	8.298e-02	0	21	40	5.701e-02
	1	41	97	5.728e-02	1	41	1	1.160e-01
	2	81	160	4.001e-02	2	81	1	8.076e-02
	3	161	404	2.812e-02	3	161	1	5.655e-02
	4	321	808	1.982e-02	4	321	1	3.977e-02
	5	641	1001	1.400e-02	5	641	1	2.804e-02
	6	1281	1001	1.102e-02	6	1281	1	1.980e-02
	7	2561	1001	1.726e-02	7	2561	1001	-nan(ind)

$u_s = x^3, u_c = -2 \cdot x^3$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	41	7.289e-02	0	21	40	5.230e-02
	1	41	83	4.935e-02	1	41	1	1.156e-01
	2	81	193	3.414e-02	2	81	1	8.054e-02
	3	161	320	2.388e-02	3	161	1	5.645e-02
	4	321	1001	1.680e-02	4	321	1	3.973e-02
	5	641	1001	1.184e-02	5	641	1	2.802e-02
	6	1281	1001	9.392e-03	6	1281	1	1.979e-02
	7	2561	1001	1.656e-02	7	2561	1001	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	41	7.289e-02	0	21	40	5.230e-02
	1	41	83	4.935e-02	1	41	1	1.156e-01
	2	81	193	3.414e-02	2	81	1	8.054e-02
	3	161	320	2.388e-02	3	161	1	5.645e-02
	4	321	1001	1.680e-02	4	321	1	3.973e-02
	5	641	1001	1.184e-02	5	641	1	2.802e-02
	6	1281	1001	9.392e-03	6	1281	1	1.979e-02
	7	2561	1001	1.656e-02	7	2561	1001	-nan(ind)

$u_s = x^4, u_c = -2 \cdot x^4$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	6.687e-02	0	21	57	5.017e-02
	1	41	93	4.438e-02	1	41	1	1.156e-01
	2	81	160	3.041e-02	2	81	1	8.046e-02
	3	161	432	2.116e-02	3	161	1	5.642e-02
	4	321	871	1.485e-02	4	321	1	3.971e-02
	5	641	1001	1.046e-02	5	641	1	2.802e-02
	6	1281	1001	8.379e-03	6	1281	1	1.979e-02
	7	2561	1001	1.614e-02	7	2561	1001	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	6.687e-02	0	21	57	5.017e-02
	1	41	93	4.438e-02	1	41	1	1.156e-01
	2	81	160	3.041e-02	2	81	1	8.046e-02
	3	161	432	2.116e-02	3	161	1	5.642e-02
	4	321	871	1.485e-02	4	321	1	3.971e-02
	5	641	1001	1.046e-02	5	641	1	2.802e-02
	6	1281	1001	8.379e-03	6	1281	1	1.979e-02
	7	2561	1001	1.614e-02	7	2561	1001	-nan(ind)

$u_s = x^5, u_c = -2 \cdot x^5$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	6.296e-02	0	21	40	4.909e-02
	1	41	80	4.095e-02	1	41	1	1.156e-01
	2	81	176	2.777e-02	2	81	1	8.044e-02
	3	161	431	1.924e-02	3	161	1	5.640e-02
	4	321	785	1.346e-02	4	321	1	3.971e-02
	5	641	1001	9.471e-03	5	641	1	2.802e-02
	6	1281	1001	7.723e-03	6	1281	1	1.979e-02
	7	2561	1001	1.586e-02	7	2561	1001	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	6.296e-02	0	21	40	4.909e-02
	1	41	80	4.095e-02	1	41	1	1.156e-01
	2	81	176	2.777e-02	2	81	1	8.044e-02
	3	161	431	1.924e-02	3	161	1	5.640e-02
	4	321	785	1.346e-02	4	321	1	3.971e-02
	5	641	1001	9.471e-03	5	641	1	2.802e-02
	6	1281	1001	7.723e-03	6	1281	1	1.979e-02
	7	2561	1001	1.586e-02	7	2561	1001	-nan(ind)

$u_s = sin(x), u_c = -2 \cdot sin(x)$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	55	9.254e-02	0	21	40	6.313e-02
	1	41	81	6.536e-02	1	41	1	1.163e-01
	2	81	162	4.619e-02	2	81	1	8.112e-02
	3	161	472	3.265e-02	3	161	1	5.672e-02
	4	321	642	2.308e-02	4	321	1	3.984e-02
	5	641	1001	1.632e-02	5	641	1	2.806e-02
	6	1281	1001	1.277e-02	6	1281	1	1.981e-02
	7	2561	1001	1.817e-02	7	2561	1001	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	55	9.254e-02	0	21	40	6.313e-02
	1	41	81	6.536e-02	1	41	1	1.163e-01
	2	81	162	4.619e-02	2	81	1	8.112e-02
	3	161	472	3.265e-02	3	161	1	5.672e-02
	4	321	642	2.308e-02	4	321	1	3.984e-02
	5	641	1001	1.632e-02	5	641	1	2.806e-02
	6	1281	1001	1.277e-02	6	1281	1	1.981e-02
	7	2561	1001	1.817e-02	7	2561	1001	-nan(ind)

$u_s = e^x, u_c = -2 \cdot e^x$

время \ пространство	равномерное				не равномерное			
равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	3.480e-01	0	21	40	3.184e-01
	1	41	83	2.461e-01	1	41	1	2.493e-01
	2	81	160	1.740e-01	2	81	1	1.731e-01
	3	161	320	1.231e-01	3	161	1	1.207e-01
	4	321	941	8.706e-02	4	321	1	8.466e-02
	5	641	1001	6.237e-02	5	641	1	5.959e-02
	6	1281	1001	4.743e-02	6	1281	1	4.203e-02
	7	2561	1001	3.913e-02	7	2561	1001	-nan(ind)
не равномерное	i	nodes	iters	norm	i	nodes	iters	norm
	0	21	40	3.480e-01	0	21	40	3.184e-01
	1	41	83	2.461e-01	1	41	1	2.493e-01
	2	81	160	1.740e-01	2	81	1	1.731e-01
	3	161	320	1.231e-01	3	161	1	1.207e-01
	4	321	941	8.706e-02	4	321	1	8.466e-02
	5	641	1001	6.237e-02	5	641	1	5.959e-02
	6	1281	1001	4.743e-02	6	1281	1	4.203e-02
	7	2561	1001	3.913e-02	7	2561	1001	-nan(ind)

8.1. 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 $\frac{1}{3}$

9. 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 <utility>
#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;
}
```

```
58
59
60  // Деление матрицы на число
61  inline matrix2D operator/(const matrix2D& a, const double& b) {
62
63      matrix2D result = a;
64      for (int i = 0; i < a.size(); i++)
65          for (int j = 0; j < a.size(); j++)
66              result[i][j] /= b;
67      return result;
68  }
69
70
71  // Вычитание векторов
72  inline vector1D operator-(const vector1D& a, const vector1D& b) {
73  #ifdef _DEBUG
74      if (a.size() != b.size())
75          throw std::exception();
76  #endif
77      vector1D result = a;
78      for (int i = 0; i < b.size(); i++)
79          result[i] -= b[i];
80      return result;
81  }
82  // Обратный знак вектора
83  inline vector1D operator-(const vector1D& a) {
84      vector1D result = a;
85      for (int i = 0; i < a.size(); i++)
86          result[i] = -result[i];
87      return result;
88  }
89
90
91
92  // Умножение матрицы на вектор
93  inline vector1D operator*(const matrix2D& a, const vector1D& b) {
94      vector1D result = { 0.0, 0.0 };
95      for (int i = 0; i < a.size(); i++)
96          for (int j = 0; j < a.size(); j++)
97              result[i] += a[i][j] * b[j];
98      return result;
99  }
100
101
102
103 // Умножение на число
104 inline vector1D operator*(const vector1D& a, double b) {
105     vector1D result = a;
106     for (int i = 0; i < result.size(); i++)
107         result[i] *= b;
108     return result;
109 }
110 // Умножение на число
111 inline vector1D operator*(double b, const vector1D& a) {
112     return operator*(a, b);
113 }
114
115
116
117 // 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(2 * nodesCount);

    if (isGridUniform) {
```

```
67
68          size_t i, elem;
69          double x;
70
71          // Первый элемент
72          nodes[0].setNodesData(xLeft, 0, 1, coefGrid);
73          i = 1;
74          for (elem = 1; elem < nodesCount - 1; elem++, i++)
75          {
76              x = xLeft + hx * i;
77              nodes[elem].setNodesData(x, i, 0, coefGrid);
78              nodes[elem].border = 0;
79          }
80          // Последний элемент
81          nodes[nodesCount - 1].setNodesData(xRight, width, 1, coefGrid);
82
83      }
84      else {
85
86          double x;
87          size_t i, elem;
88
89          i = 1;
90          dx = hx * kx;
91          x = xLeft + hx;
92          // Первый элемент
93          nodes[0].setNodesData(xLeft, 0, 1, coefGrid);
94          for (elem = 1; elem < width; elem++, i++, dx *= kx)
95          {
96              nodes[elem].setNodesData(x, i, 0, coefGrid);
97              nodes[elem].border = 0;
98              x += dx;
99          }
100         // Последний элемент
101         nodes[nodesCount - 1].setNodesData(xRight, width, 1, coefGrid);
102     }
103 }
104
105 //
106 //void GRID::buildTimeGrid()
107 //{
108 //  //  tFirst         tLast
109 //  //    *--------------*
110 //  //    0    tCount    1
111 //  times.resize(tCount);
112 //
113 //  if (isTimeUniform) {
114 //
115 //      ht = ((tLast - tFirst) / double(tCount - 1)) / pow(2, coefTime);
116 //      if (coefTime != 0)
117 //          width = (width - 1) * pow(2, coefTime) + 1;
118 //
119 //      size_t i, elem;
120 //      double t;
121 //      // Первый элемент
122 //      times[0] = tFirst;
123 //      i = 1;
124 //      for (elem = 1; elem < tCount; elem++, i++)
125 //          times[elem] = tFirst + ht * i;
126 //
```

```
//      // 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(
        function1D _u_s,
        function1D _u_c,
        function1D _f_s,
        function1D _f_c,
        double _lambda,
        double _sigma,
        double _omega,
        double _hi,
        bool _isGridUniform,
        bool _isTimeUniform,
        int _condType,
        int _coefGrid,
        int _coefTime
    );
    pair<int, double> solve(int solver);
    inline int getNodesCount() { return nodesCount; }
    void convAToDense();
    void outputA();
    void outputALocal();


protected:
    double  p00, p01, p10, p11,
            c00, c01, c10, c11;
    function1D u_s, u_c, f_s, f_c;
    double lambda, sigma, omega, hi;

    double calcNormAtMainNodes(const vector1D &x) {
        double tmp = 0;
        for (size_t i = 0; i < x.size(); i++)
            if (i % 2 == 0)
                tmp += pow((x[i] - u_c(nodes[i].x)), 2);
            else
                tmp += pow((x[i] - u_s(nodes[i].x)), 2);
        return sqrt(tmp) / nodes.size();
    }

    void buildGlobalMatrixA();
    void buildGlobalVectorb();
    /*void printGlobalMatrixA();
    void printGlobalVectorb();*/

    void buildLocalmatrixA(int elemNumber);
    void buildLocalVectorb(int elemNumber);
    matrix2D ALocal;
    vector1D bLocal;
};
```

fem.cpp

```
#include "fem.h"




void FEM::outputALocal() {
```

```
    cout << endl;
    for (int i = 0; i < ALocal.size(); ++i) {
        for (int j = 0; j < ALocal.size(); ++j) {
            cout << ALocal[i][j] << "  ";
        }
        cout << endl;
    }
}

void FEM::convAToDense() {

    A.clear();
    A.resize(n);
    for (int i = 0; i < n; ++i) {
        A[i].resize(n, 0);
    }

    for (int i = 0; i < n; ++i) {

        A[i][i] = di[i];
        int i0 = ia[i];
        int i1 = ia[i + 1];

        for (int k = i0; k < i1; ++k) {
            A[i][ja[k]] = al[k];
            A[ja[k]][i] = au[k];
        }
    }
}



void FEM::outputA() {
    cout << endl;
    for (int i = 0; i < A.size(); ++i) {
        for (int j = 0; j < A.size(); ++j) {
            cout << A[i][j] << "\t";
        }
        cout << endl;
    }
}




// Инициализируем модель, задавая функции u, f и тип сетки
void FEM::init(
    function1D _u_s,
    function1D _u_c,
    function1D _f_s,
    function1D _f_c,
    double _lambda,
    double _sigma,
    double _omega,
    double _hi,
    bool _isGridUniform,
    bool _isTimeUniform,
```

```
    int _condType,
    int _coefGrid,
    int _coefTime
) {
    ifstream fin("input/SLAE_parameters.txt");
    fin >> E >> delta >> maxiter;
    fin.close();
    u_s = _u_s;
    u_c = _u_c;
    f_s = _f_s;
    f_c = _f_c;
    lambda = _lambda;
    sigma = _sigma;
    omega = _omega;
    hi = _hi;
    isGridUniform = _isGridUniform;
    isTimeUniform = _isTimeUniform;
    condType = _condType;
    coefGrid = _coefGrid;
    coefTime = _coefTime;
}

pair<int, double> FEM::solve(int solver)
{
    int i;
    n = nodesCount * 2;
    buildGlobalMatrixA();
    buildGlobalVectorb();

    switch (solver)
    {
    case 1:
        i = LOSfactLUsq();
        break;
    case 2:
        i = BiCG();
        break;
    case 3:
        //result = BiCG();
        break;
    default:
        break;
    }
    return make_pair(i, calcNormAtMainNodes(x));
}



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

// Строим глобальную матрицу системы нелинейных уравнений
void FEM::buildGlobalMatrixA()
{
    di.clear();
    au.clear();
    al.clear();

    di.resize(2 * nodesCount, 0);
```

```
    ia.resize(2 * nodesCount + 1);
    ia[0] = 0; ia[1] = 0; ia[2] = 1;
    double iaLast = 1;
    for (size_t i = 3; i < ia.size(); i++)
    {
        if (i % 2 != 0)
            iaLast += 2;
        else
            iaLast += 3;
        ia[i] = iaLast;
    }
    ja.resize(5 * finiteElementsCount + 1, 0);
    ja[0] = 0;
    for (size_t i = 1; i < ja.size(); i += 5)
    {
        ja[i] = ja[i - 1];
        ja[i + 1] = ja[i - 1] + 1;
        ja[i + 2] = ja[i - 1];
        ja[i + 3] = ja[i - 1] + 1;
        ja[i + 4] = ja[i - 1] + 2;
    }
    al.resize(5 * finiteElementsCount + 1, 0);
    au.resize(5 * finiteElementsCount + 1, 0);
    for (size_t elemNumber = 0; elemNumber < 2 * finiteElementsCount; elemNumber
    += 2)
    {
        buildLocalmatrixA(elemNumber);

        int t = 0;
        for (size_t i = elemNumber; i < elemNumber + 4; i++, t++)
            di[i] += ALocal[t][t];

        t = 0;
        for (size_t i = elemNumber; i < elemNumber + 4; i++, t++)
        {
            int i0 = ia[i];
            int i1 = ia[i + 1];
            int tLocal = 0;
            if (t == 0)
                i0 = i1;
            else if (t == 1)
                i0 = i1 - 1;
            for (size_t k = i0; k < i1; k++, tLocal++)
            {
                al[k] += ALocal[t][tLocal];
                au[k] += ALocal[tLocal][t];
            }
        }

        /*outputALocal();
        convAToDense();
        outputA();*/
    }

    // Первые краевые условия
    di[0] = 1; di[1] = 1; al[0] = 0;
    for (size_t i = 0; i < 5; i++)
        au[i] = 0;

    di[di.size() - 1] = 1; di[di.size() - 2] = 1; au[au.size() - 1] = 0;
```

```
    for (size_t i = 1; i < 6; i++)
        al[al.size() - i] = 0;

    /*convAToDense();
    cout << endl << "Added 1st boundary conditions:" << endl;
    outputA();*/
}



// Строим глобальный вектор правой части системы нелинейных уравнений
void FEM::buildGlobalVectorb()
{
    b.clear();
    b.resize(2 * nodesCount, 0);

    for (size_t elemNumber = 0; elemNumber < finiteElementsCount; elemNumber++)
    {
        buildLocalVectorb(elemNumber);
        int k = 0;
        for (size_t i = elemNumber; i < elemNumber + 4; i++, k++)
            b[i] = bLocal[k];
    }

    // Первые краевые условия
    b[0] = u_c(nodes[0].x);
    b[1] = u_s(nodes[0].x);
    b[b.size() - 2] = u_s(nodes[nodes.size() - 1].x);
    b[b.size() - 1] = u_c(nodes[nodes.size() - 1].x);
}


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



// Построение локальной матрицы A
void FEM::buildLocalmatrixA(int elemNumber)
{
    p00 = lambda / (hx*hx) - (omega*omega)*hi / 3;
    p11 = p00;
    p01 = -lambda / (hx*hx) - (omega*omega)*hi / 6;
    p10 = p01;

    c00 = omega * sigma / 3;
    c11 = c00;
    c01 = omega * sigma / 6;
    c10 = c01;

    ALocal = { {p00,    -c00,   p01,    -c01},
               {c00,    p00,    c01,    p01},
               {p10,    -c10,   p11,    -c11},
               {c10,    p10,    c11,    p11} };
}


// Построение локального вектора b
```

```
247 void FEM::buildLocalVectorb(int elemNumber)
248 {
249     bLocal = { 0, 0, 0, 0 };
250     double f0_s = f_s(nodes[elemNumber].x);
251     double f0_c = f_c(nodes[elemNumber].x);
252     double f1_s = f_s(nodes[elemNumber + 1].x);
253     double f1_c = f_c(nodes[elemNumber + 1].x);
254     bLocal[0] = hx * (2 * f0_s + f1_s) / 6;
255     bLocal[1] = hx * (2 * f0_c + f1_c) / 6;
256     bLocal[2] = hx * (f0_s + 2 * f1_s) / 6;
257     bLocal[3] = hx * (f0_c + 2 * f1_c) / 6;
258 }
```

solver.h

```
1  #pragma once
2  #include "head.h"
3
4
5  // Система линейных алгебраических уравнений
6  class SOLVER {
7
8  public:
9      void initSLAE();
10
11
12
13     void getVectX(vector1D &x) { x = b; };
14     void generateVectX(int size);
15     void writeXToFile(const char *fileName);
16     void writeXToStream(std::ofstream& fout);
17     void writeFToFile(const char *fileName);
18
19
20     int getDimention() { return n; }
21     void decomposionD();
22     void decomposionLUsq();
23     vector1D execDirectTraversal(const vector1D &_F);
24     vector1D execReverseTraversal(const vector1D &_y);
25
26     int LOS();
27     int LOSfactD();
28     int LOSfactLUsq();
29     int BiCG();
30     void clearAll();
31     void setMaxiter(int new_maxiter) { maxiter = new_maxiter; }
32     void setE(double new_E) { E = new_E; }
33
34 protected:
35     vector1D q, qPrev;
36     vector1D di, al, au, di_f, al_f, au_f;
37     vector <int> ia, ja;
38     int n, maxiter;
39     double E, delta;
40     vector1D x, r, z, p, b, bTmp;
41
42
43     vector1D multA(const vector1D&x);
44     vector1D multD(const vector1D&x);
45     double calcRelativeDiscrepancy();
46     double calcNormE(const vector1D &x) { return sqrt(x*x); }
47
```

```

    matrix2D A;

    vector1D xPrev; // решение на k-1 итерации

    vector1D s; // вспомогательный вектор
    double alpha, beta;


    void generateInitialGuess();
    vector1D multAOn(const vector1D &v);
    vector1D multAtOn(const vector1D &v);
    bool doStop(int i);
};
```

solver.cpp

```
#include "solver.h"



// A*x = b, где x – произвольный вектор
vector1D SOLVER::multA(const vector1D& x) {

    vector1D result(n);
    for (int i = 0; i < n; ++i) {

        result[i] = di[i] * x[i];
        int i0 = ia[i];
        int i1 = ia[i + 1];

        for (int k = i0; k < i1; ++k) {

            result[i] += al[k] * x[ja[k]];
            result[ja[k]] += au[k] * x[i];
        }
    }

    return result;
}

// A*x = b, где x – произвольный вектор
vector1D SOLVER::multD(const vector1D&x) {

    vector1D result(n);

    for (int i = 0; i < n; ++i)
        result[i] = di_f[i] * x[i];

    return result;
}


// Создаём вектор x* = (1,2,...n)'
void SOLVER::generateVectX(int size) {

    x.resize(size);
    for (int i = 0; i < size; ++i) {
        x[i] = i + 1;
    }
}

```

```
46
47  // Вывод вектора b в файл
48  void SOLVER::writeFToFile(const char *fileName) {
49
50      std::ofstream fout;
51      fout.open(fileName);
52
53      for (int i = 0; i < b.size(); ++i)
54          fout << b[i] << endl;
55      fout.close();
56  }
57
58
59  // Вывод вектора x в файл
60  void SOLVER::writeXToFile(const char * fileName) {
61
62      std::ofstream fout;
63      fout.open(fileName);
64      for (int i = 0; i < x.size(); ++i)
65          fout << x[i] << " ";
66      fout << " \t";
67      fout.close();
68  }
69
70
71  // Вывод вектора x в поток
72  void SOLVER::writeXToStream(std::ofstream& fout) {
73
74      for (int i = 0; i < x.size(); ++i)
75          fout << x[i] << "\n";
76      fout << "\n";
77  }
78
79
80  // Диагональное предобуславливание M = D
81  void SOLVER::decomposionD() {
82
83      di_f.clear();
84      di_f.resize(n);
85      for (int i = 0; i < n; ++i)
86          di_f[i] = 1.0 / sqrt(di[i]);
87  }
88
89
90  // LU_sq разложение матрицы A
91  void SOLVER::decomposionLUsq() {
92
93      double sum_u, sum_l, sum_d;
94      di_f = di;
95      al_f = al;
96      au_f = au;
97
98      // Идём построчно в верхнем треугольнике, что экививалентно
99      // Обходу нижнего треугольника по столбцам вниз, начиная с первого
100     for (int i = 0; i < n; ++i) {
101
102         int i0 = ia[i];
103         int i1 = ia[i + 1];
104
105         // Рассчёт элементов нижнего треугольника
```

```
106         for (int k = i0; k < i1; ++k) {
107 
108             int j = ja[k]; // текущий j
109             int j0 = ia[j]; // i0 строки j
110             int j1 = ia[j + 1]; // i1 строки j
111             sum_l = 0;
112             sum_u = 0;
113             int ki = i0; // Индекс l_ik
114             int kj = j0; // Индекс u_kj
115 
116             while (ki < k && kj < j1) {
117 
118                 if (ja[ki] == ja[kj]) { // l_ik * u_kj
119                     sum_l += al_f[ki] * au_f[kj];
120                     sum_u += au_f[ki] * al_f[kj];
121                     ki++;
122                     kj++;
123                 }
124                 else { // Ищем следующие элементы i и j строки, которые можем
    перемножить
125                     if (ja[ki] > ja[kj]) kj++;
126                     else ki++;
127                 }
128             }
129 
130             al_f[k] = (al_f[k] - sum_l) / di_f[j];
131             au_f[k] = (au_f[k] - sum_u) / di_f[j];
132         }
133 
134 
135         // Рассчёт диагонального элемента
136         sum_d = 0.0;
137         for (int k = i0; k < i1; ++k)
138             sum_d += al_f[k] * au_f[k];
139         di_f[i] = sqrt(di_f[i] - sum_d);
140     }
141 }
142 
143 
144 // Прямой ход    L y = b    ==>    y = L^-1 b
145 vector1D SOLVER::execDirectTraversal(const vector1D &_F) {
146 
147     vector1D y;
148     y.resize(n, 0);
149 
150     for (int i = 0; i < n; ++i) {
151         double sum = 0;
152         int i0 = ia[i];
153         int i1 = ia[i + 1];
154         for (int k = i0; k < i1; ++k)
155             sum += al_f[k] * y[ja[k]];
156 
157         y[i] = (_F[i] - sum) / di_f[i];
158     }
159     return y;
160 }
161 
162 
163 // Обратный ход    U(sq) x = y    ==>    x = U(sq)^-1 y
164 vector1D SOLVER::execReverseTraversal(const vector1D &_y) {
```

```
165
166     vector1D x, y = _y;
167     x.resize(n);
168     for (int i = n - 1; i >= 0; --i) {
169
170         x[i] = y[i] / di_f[i];
171         int i0 = ia[i];
172         int i1 = ia[i + 1];
173         for (int k = i0; k < i1; ++k)
174             y[ja[k]] -= au_f[k] * x[i];
175     }
176
177     return x;
178 }
179
180
181
182 // Полная очистка СЛАУ
183 void SOLVER::clearAll() {
184
185     n = 0;
186     E = 0.0;
187     maxiter = 0;
188
189     di.clear();
190     ia.clear();
191     ja.clear();
192     al.clear();
193     au.clear();
194
195     di_f.clear();
196     al_f.clear();
197     au_f.clear();
198
199     x.clear();
200     r.clear();
201     z.clear();
202     p.clear();
203     b.clear();
204     bTmp.clear();
205 }
206
207
208
209 // Рассчёт относительной невязки
210 double SOLVER::calcRelativeDiscrepancy() {
211     //return calcNormE(r) / calcNormE(b);
212     return (r*r);
213 }
214
215
216 // Локально - оптимальная схема
217 int SOLVER::LOS() {
218
219     x.clear();           // Задаём начальное приближение
220     x.resize(n, 0);      // x_0 = (0, 0, ...)
221     r.resize(n);
222
223     vector1D xprev = x;
224     r = b - multA(x);    // r_0 = b - A*x_0
```

```
    z = r;              // z_0 = r_0
    p = multA(z);       // p_0 = A*z_0


    for (int i = 0; i < maxiter; ++i) {

        double pp = (p * p);
        double alpha = (p * r) / pp;

        x = x + alpha * z;
        r = r - alpha * p;

        bTmp = multA(r);
        double beta = -(p * bTmp) / pp;

        z = r + beta * z;
        p = bTmp + beta * p;


        double relativeDiscrepancy = calcRelativeDiscrepancy();
        if (x == xprev || relativeDiscrepancy < E) {
            return i;
        }
        xprev = x;
    }
}


// Локально - оптимальная схема с неполной диагональной факторизацией
int SOLVER::LOSfactD() {

    x.clear();          // Задаём начальное приближение
    x.resize(n, 0);     // x_0 = (0, 0, ...)
    vector1D xprev = x;
    decomposionD();

    r = b - multA(x);   // r_0 = b - A*x_0
    r = multD(r);
    z = multD(r);       // z = U^-1 r
    p = multA(z);       // p = A*z
    p = multD(p);       // p = L^-1 A*z


    for (int i = 0; i < maxiter; ++i) {

        double pp = p * p;
        double alpha = (p*r) / pp;
        x = x + alpha * z;
        r = r - alpha * p;

        vector1D tmp = multD(r);
        tmp = multA(tmp);
        tmp = multD(tmp);
        double beta = -(p * tmp) / pp;
        p = tmp + beta * p;

        tmp = multD(r);
        z = tmp + beta * z;


```

```
        double relativeDiscrepancy = calcRelativeDiscrepancy();
        if (x == xprev || relativeDiscrepancy < E) {
            return i;
        }
        xprev = x;
    }
}




// Локально – оптимальная схема с неполной факторизацией LU(sq)
int SOLVER::LOSfactLUsq() {

    x.clear();                          // Задаём начальное приближение
    x.resize(n, 0);                     // x_0 = (0, 0, ...)
    vector1D xprev = x;
    decomposionLUsq();

    r = b – multA(x);                   // r_0 = b – A*x_0
    r = execDirectTraversal(r);         // r = L^–1 (b – A*x_0)
    z = execReverseTraversal(r);        // z = U^–1 r
    p = multA(z);                       // p = A*z
    p = execDirectTraversal(p);         // p = L^–1 A*z


    for (int i = 0; i < maxiter; ++i) {

        double pp = p * p;
        double alpha = (p*r) / pp;
        x = x + alpha * z;
        r = r – alpha * p;

        vector1D tmp = execReverseTraversal(r);
        tmp = multA(tmp);
        tmp = execDirectTraversal(tmp);
        double beta = –(p * tmp) / pp;
        p = tmp + beta * p;

        tmp = execReverseTraversal(r);
        z = tmp + beta * z;


        double relativeDiscrepancy = calcRelativeDiscrepancy();
        if (x == xprev || relativeDiscrepancy < E) {
            return i;
        }
        xprev = x;
    }
}




// Считываем все СЛАУ и её параметры из файлов
void SOLVER::initSLAE()
{
    ifstream finMatrix("input/matrix.txt");
    ifstream finVector("input/vector.txt");
```

```
    ifstream finParams("input/SLAE_parameters.txt");

    finParams >> maxiter >> E >> delta;
    finMatrix >> n;
    di.resize(n);
    ia.resize(n + 1);

    for (size_t i = 0; i < n; i++)
        finMatrix >> di[i];
    for (size_t i = 0; i < n + 1; i++)
        finMatrix >> ia[i];

    ja.resize(ia[n]);
    al.resize(ia[n]);
    au.resize(ia[n]);

    for (size_t i = 0; i < ja.size(); i++)
        finMatrix >> ja[i];
    for (size_t i = 0; i < al.size(); i++)
        finMatrix >> al[i];
    for (size_t i = 0; i < au.size(); i++)
        finMatrix >> au[i];

    b.resize(n);
    for (size_t i = 0; i < n; i++)
        finVector >> b[i];

    finMatrix.close();
    finVector.close();
    finParams.close();
}



// Метод бисопряжённых градиентов
int SOLVER::BiCG()
{
    ofstream fout("output/result.txt");
    int i = 0;
    double prPrev, pr;
    vector1D Az, Ats;
    generateInitialGuess();
    r = b - multAOn(x);
    p = z = s = r;
    do {
        xPrev = x;
        prPrev = (p*r);
        Az = multAOn(z);
        Ats = multAtOn(s);

        alpha = prPrev / (s*Az);

        x = x + alpha * z;
        r = r - alpha * Az;
        p = p - alpha * Ats;

        beta = (p*r) / prPrev;

        z = r + beta * z;
        s = p + beta * s;
```

```
405
406         i++;
407     } while (!doStop(i));
408
409     fout.close();
410     return i;
411 }
412
413
414
415 // Создание начального приближения x0 = (0,...,0)'
416 void SOLVER::generateInitialGuess()
417 {
418     x.resize(n, 1);
419 }
420
421
422
423 // Умножение матрицы A на вектор
424 vector1D SOLVER::multAOn(const vector1D &v)
425 {
426     vector1D result(n);
427     for (size_t i = 0; i < n; i++)
428     {
429         result[i] = di[i] * v[i];
430         int i0 = ia[i];
431         int i1 = ia[i + 1];
432
433         for (size_t k = i0; k < i1; k++)
434         {
435             int j = ja[k];
436             result[i] += al[k] * v[j];
437             result[j] += au[k] * v[i];
438         }
439     }
440     return result;
441 }
442
443
444
445 // Умножение транспонированной матрицы A на вектор
446 vector1D SOLVER::multAtOn(const vector1D &v)
447 {
448     vector1D result(n);
449     for (size_t i = 0; i < n; i++)
450     {
451         result[i] = di[i] * v[i];
452         int i0 = ia[i];
453         int i1 = ia[i + 1];
454         for (size_t k = i0; k < i1; k++)
455         {
456             int j = ja[k];
457             result[j] += al[k] * v[i];
458             result[i] += au[k] * v[j];
459         }
460     }
461     return result;
462 }
463
464
```

```

// Проверяем условие выхода
bool SOLVER::doStop(int i)
{
    // Выход по числу итераций
    if (i > maxiter)
        return true;

    // Выход шагу
    if (calcNormE(x – xPrev) / calcNormE(x) < delta)
        return true;

    // Выход по относительной невязке
    if (calcNormE(multAOn(x) – b) / calcNormE(b) < E)
        return true;

    return false;
}
```

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);
    };
}

function1D calc_f_s(
    const function1D & u_s,
    const function1D & u_c,
    double lambda,
    double sigma,
    double omega,
    double hi
) {
    // f_s = – lambda * div(grad u_s) – omega * sigma * u_c – omega^2 * hi * u_s
    return [=](double x) -> double {
        using namespace placeholders;
        auto divGrad = calcFirstDerivative(calcFirstDerivative(u_s));
        return -lambda * divGrad(x) – omega * sigma * u_c(x) – omega * omega * hi * u_s(x);
    };
}


function1D calc_f_c(
    const function1D & u_s,
    const function1D & u_c,
    double lambda,
    double sigma,
    double omega,
    double hi
) {
    // f_c = – lambda * div(grad u_c) + omega * sigma * u_s – omega^2 * hi * u_c
    return [=](double x) -> double {
        using namespace placeholders;
        auto divGrad = calcFirstDerivative(calcFirstDerivative(u_c));
```

```
        return -lambda * divGrad(x) + omega * sigma * u_s(x) - omega * omega * hi * u_c(x);
    };
}


void main() {

    pair <int, double> result;
    double lambda = 1;
    double  sigma = 1;
    double omega = 1;
    double hi = 1;

    cout << scientific << setprecision(3);

    vector <function1D> u_s(8), u_c(8), f_s(8), f_c(8);
    u_s[0] = { [](double x) -> double { return 1; } };
    u_s[1] = { [](double x) -> double { return x; } };
    u_s[2] = { [](double x) -> double { return pow(x, 2); } };
    u_s[3] = { [](double x) -> double { return pow(x, 3); } };
    u_s[4] = { [](double x) -> double { return pow(x, 4); } };
    u_s[5] = { [](double x) -> double { return pow(x, 5); } };
    u_s[6] = { [](double x) -> double { return sin(x); } };
    u_s[7] = { [](double x) -> double { return exp(x); } };

    u_c[0] = { [](double x) -> double { return -2; } };
    u_c[1] = { [](double x) -> double { return -2 * x; } };
    u_c[2] = { [](double x) -> double { return -2 * pow(x, 2); } };
    u_c[3] = { [](double x) -> double { return -2 * pow(x, 3); } };
    u_c[4] = { [](double x) -> double { return -2 * pow(x, 4); } };
    u_c[5] = { [](double x) -> double { return -2 * pow(x, 5); } };
    u_c[6] = { [](double x) -> double { return -2 * sin(x); } };
    u_c[7] = { [](double x) -> double { return -2 * exp(x); } };

    vector <string> u_s_names = {
        "$1$",
        "$x$",
        "$x^2$",
        "$x^3$",
        "$x^4$",
        "$x^5$",
        "$sin(x)$",
        "$e^x$",
    };

    ofstream foutTable("report/table1.txt");
    foutTable << scientific << setprecision(2);
    foutTable << "a\t$1$\t$x$\t$x^2$\t$x^3$\t$x^4$\t$x^5$\t$sin(x)$\t$e^x$" << endl;
    cout << "Research 1: convergence with diferent u_s and u_c" << endl;
    for (size_t i = 0; i < u_s.size(); i++)
    {
        foutTable << u_s_names[i] << "\t";
        for (size_t j = 0; j < u_c.size(); j++)
        {
            std::cout << int(float(i*u_c.size() + j) * 100.0 / (u_s.size()*u_c.size())) << " %\r";
            f_s[i] = calc_f_s(u_s[i], u_c[i], lambda, sigma, omega, hi);
            f_c[j] = calc_f_c(u_s[i], u_c[i], lambda, sigma, omega, hi);
```

```
97
98              FEM fem;
99              fem.init(u_s[i], u_c[j], f_s[i], f_c[j], lambda, sigma, omega, hi,
    true, true, 1, 0, 0);
100             fem.inputGrid();
101             fem.buildGrid();
102             result = fem.solve(1);
103
104             if (j + 1 == u_c.size())
105                 foutTable << result.second << endl;
106             else
107                 foutTable << result.second << "\t";
108         }
109     }
110     cout << endl;
111     foutTable.close();
112
113
114     foutTable.open("report/table2.txt");
115     foutTable << scientific << setprecision(2);
116     foutTable << "a\t$1$\t$x$\t$x^2$\t$x^3$\t$x^4$\t$x^5$\t$sin(x)$\t$e^x$" <<
    endl;
117     cout << "Research 1: convergence with diferent u_s and u_c" << endl;
118     for (size_t i = 0; i < u_s.size(); i++)
119     {
120         foutTable << u_s_names[i] << "\t";
121         for (size_t j = 0; j < u_c.size(); j++)
122         {
123             std::cout << int(float(i*u_c.size() + j) * 100.0 / (u_s.size()*u_c.
    size())) << " %\r";
124             f_s[i] = calc_f_s(u_s[i], u_c[i], lambda, sigma, omega, hi);
125             f_c[j] = calc_f_c(u_s[i], u_c[i], lambda, sigma, omega, hi);
126
127             FEM fem;
128             fem.init(u_s[i], u_c[j], f_s[i], f_c[j], lambda, sigma, omega, hi,
    true, true, 1, 0, 0);
129             fem.inputGrid();
130             fem.buildGrid();
131             result = fem.solve(2);
132
133             if (j + 1 == u_c.size())
134                 foutTable << result.second << endl;
135             else
136                 foutTable << result.second << "\t";
137         }
138     }
139     cout << endl;
140     foutTable.close();
141
142
143
144
145     // ▯▯▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯ ▯▯▯▯ ▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯
146     auto reseacrhConvergence = [&](bool isGridUniform, bool isTimeUniform, int
    solver) {
147         for (int i = 0; i < u_s.size(); i++) {
148             f_s[i] = calc_f_s(u_s[i], u_c[i], lambda, sigma, omega, hi);
149             f_c[i] = calc_f_c(u_s[i], u_c[i], lambda, sigma, omega, hi);
150
151             ofstream fout("report/file_u" + to_string(i) + "." + to_string(
```

```
solver) + "." + to_string(isGridUniform) + to_string(isTimeUniform) + ".txt")
;
            fout << scientific << setprecision(3);
            fout << "i\tnodes\titers\tnorm\n";
            for (int coefGrid = 0; coefGrid < 8; coefGrid++)
            {
                FEM fem;
                fem.init(u_s[i], u_c[i], f_s[i], f_c[i], lambda, sigma, omega,
hi, isGridUniform, isTimeUniform, 1, coefGrid, 0);
                fem.inputGrid();
                fem.buildGrid();
                result = fem.solve(solver);
                fout << coefGrid << "\t"
                    << fem.getNodesCount() << "\t"
                    << result.first << "\t"
                    << result.second << endl;
            }
            fout.close();
        }
    };


    reseacrhConvergence(true, true, 1);
    cout << "Research 2.1: convergence with grid crushing" << endl;
    reseacrhConvergence(true, false, 1);
    cout << "Research 2.2: convergence with grid crushing" << endl;
    reseacrhConvergence(false, true, 1);
    cout << "Research 2.3: convergence with grid crushing" << endl;
    reseacrhConvergence(false, false, 1);
    cout << "Research 2.4: convergence with grid crushing" << endl;


    reseacrhConvergence(true, true, 2);
    cout << "Research 3.1: convergence with grid crushing" << endl;
    reseacrhConvergence(true, false, 2);
    cout << "Research 3.2: convergence with grid crushing" << endl;
    reseacrhConvergence(false, true, 2);
    cout << "Research 3.3: convergence with grid crushing" << endl;
    reseacrhConvergence(false, false, 2);
    cout << "Research 3.4: convergence with grid crushing" << endl;


    /*vector1D sigmas = { 0, 1e+1, 1e+2 , 1e+3 , 1e+4, 1e+5, 1e+6, 1e+7, 1e+8 };

    for (double omega = 1e-4; omega <= 1e+9; omega *= 10)
    {
        for (double lambda = 1e+2; lambda <= 8e+5; lambda *= 10)
        {

            for (auto sigma : sigmas)
            {
                for (double chi = 1e-12; chi <= 1e-10; chi*=10)
                {
                    FEM fem;
                    fem.init(u_s[2], u_c[2], f_s[i], f_c[i], lambda, sigma,
omega, hi, isGridUniform, isTimeUniform, 1, coefGrid, 0);
                    fem.inputGrid();
                    fem.buildGrid();
                    result = fem.solve(2);
                    fout << coefGrid << "\t"
```

```
208                         << fem.getNodesCount() << "\t"
209                         << result.first << "\t"
210                         << result.second << endl;
211                 }
212             }
213         }
214     }*/
215
216
217 }
```