Санкт-Петербургский государственный политехнический университет
Институт компьютерных наук и кибербезопасности
«Высшая школа программной инженерии»

Курсовая работа

по дисциплине «Разработка программного обеспечения для моделирования физических процессов»

Выполнил

студент гр.5130904/10101 Абраамян А. М.

Руководитель Воскобойников С. П.

«___» ____________202__ г.

Санкт-Петербург
2025

Оглавление

Постановка задачи

Используя интегро-интерполяционный метод, разработать программу для моделирования распределения температуры в брусе, описываемого математической моделью:

$$-\left[\frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right)+\frac{\partial^2 u}{\partial y^2}\right]=f(x,y),$$

$$a\le x\le b,\quad c\le y\le d,\quad 0<c_1\le k(x)\le c_2$$

и граничными условиями вида:

$$u|_{x=a}=g_1(y),\qquad -k_1\frac{\partial u}{\partial x}|_{x=b}=\chi u|_{x=b}-g_2(y),$$

$$u|_{y=c}=g_3(x),\qquad u|_{y=d}=g_4(x)$$

Для построения и тестирования модели будет использоваться язык C++.

Дискретная модель

Введём обозначения:

N - число разбиений интервала [X_a, X_b]

$$h_i = x_i - x_{i-1}$$

$$x_{i-1/2} = (x_i + x_{i-1})/2$$

$$\hbar_i = \begin{cases} \frac{h_i+1}{2},\ i=0 \\ \frac{h_i+h_{i+1}}{2},\ i=1,2,\ldots,N-1 \\ \frac{h_i}{2},\ i=N \end{cases}$$

Проинтегрируем уравнение для промежутка, не включая границы:

$$-\left(\int_{y_{j-0.5}}^{y_{j+0.5}}\int_{x_{i-0.5}}^{x_{i+0.5}}\frac{\partial}{\partial x}\left(k_1(x)\frac{\partial u}{\partial x}\right)dx\,dy+\int_{y_{j-0.5}}^{y_{j+0.5}}\int_{x_{i-0.5}}^{x_{i+0.5}}\frac{\partial^2 u}{\partial y^2}dx\,dy\right)=\int_{y_{j-0.5}}^{y_{j+0.5}}\int_{x_{i-0.5}}^{x_{i+0.5}}f(x,y)\,dx\,dy$$

$$i=1,\ldots,N_x-1\quad j=1,\ldots,N_y-1$$

$$-\left(\int_{y_{i-0.5}}^{y_{j+0.5}}\left(\left(k_1(x_{i+0.5})\frac{\partial u}{\partial x}\Big|_{x=x_{i+0.5}}\right)-\left(k_1(x_{i-0.5})\frac{\partial u}{\partial x}\Big|_{x=x_{i-0.5}}\right)\right)dy+\int_{y_{i-0.5}}^{y_{j+0.5}}\int_{x_{i-0.5}}^{x_{i+0.5}}\frac{\partial^2 u}{\partial y^2}dx\,dy\right)=\int_{y_{i-0.5}}^{y_{j+0.5}}\int_{x_{i-0.5}}^{x_{i+0.5}}f(x,y)\,dx\,dy$$

$$i = 1,\dots,N_x\text{-}1 \quad j = 1,\dots,N_y\text{-}1$$

$$\int_{x_{i-1/2}}^{x_{i+1/2}}\int_{y_{j-1/2}}^{y_{j+1/2}} f(x,y)dxdy=h_x h_y f_{ij} \qquad \int_{x_{i-1/2}}^{x_{i+1/2}}\phi dx=h_x\phi$$

$$k_1(x_{i+0.5})\frac{\partial u}{\partial x}\Big|_{x=x_{i+0.5}\ y=y_j}=k_1(x_{i+0.5})\frac{v_{i+1,j}-v_{i,j}}{h_x}$$

$$\frac{\partial u}{\partial y}\Big|_{x=x_i\ y=y_{j+0.5}}=\frac{v_{i,j+1}-v_{i,j}}{h_y}$$

$$-k\frac{\partial u}{\partial x}\Big|_{x=b}=\chi u|_{x=b}-g_2(y) \quad \rightarrow \quad k_1(x_{i+0.5})\frac{v_{i+1,j}-v_{i,j}}{h_x}=\chi u|_{x=b}-g_2(y) \qquad i=N_x-1$$

По формуле центральных разностей и по формуле **средних** прямоугольников получаем разностную схему:

1. $$-\left[h_y k_1(x_{i+1/2})\frac{v_{i+1,j}-v_{i,j}}{h_x}-h_y k_1(x_{i-1/2})\frac{v_{i,j}-v_{i-1,j}}{h_x}+h_x\frac{v_{i,j+1}-v_{i,j}}{h_y}-h_x\frac{v_{i,j}-v_{i,j-1}}{h_y}\right]=h_x h_y f_{ij}$$

$$i = 1,\dots,N_x\text{-}1 \quad j = 1,\dots,N_y\text{-}1$$

2. $v_{i,j}=\mu_1(y_j) \qquad i=0;\quad j=1,\dots,N_y-1;$

3. Подставляя граниченое условие 3 рода получаем

$$-\left[-h_y\left(\chi_2 v_{i,j}-\mu_2(y_j)\right)-h_y k_1(x_{i-1/2})\frac{v_{i,j}-v_{i-1,j}}{h_x}+\frac{h_x}{2}\frac{v_{i,j+1}-v_{i,j}}{h_y}-\frac{h_x}{2}\frac{v_{i,j}-v_{i,j-1}}{h_y}\right]=\frac{h_x}{2}h_y f_{ij}$$

$$i=N_x;\quad j=1,\dots,N_y-1;$$

4. $v_{i,j}=\mu_3(x_i) \qquad j=0;\quad i=1,\dots,N_x-1;$
5. $v_{i,j}=\mu_4(x_i) \qquad j=N_y;\quad i=1,\dots,N_x-1;$

Домножим первое уравнение на h_y и разделим на h_x, получим:

1. $$-\left[h_y^2 k_1(x_{i+1/2})\frac{v_{i+1,j}-v_{i,j}}{h_x^2}-h_y^2 k_1(x_{i-1/2})\frac{v_{i,j}-v_{i-1,j}}{h_x^2}+v_{i,j+1}-2v_{i,j}+v_{i,j-1}\right]=h_y^2 f_{ij}$$
2. $v_{i,j}=\mu_1(y_j) \qquad i=0;\quad j=1,\dots,N_y-1;$

Домножим третье уравнение на $2h_y$ и разделим на h_x, получим

3. $$-\left[-2h_y^2\left(\chi_2 v_{i,j}-\mu_2(y_j)\right)-2h_y^2 k_1(x_{i-1/2})\frac{v_{i,j}-v_{i-1,j}}{h_x^2}+v_{i,j+1}-2v_{i,j}+v_{i,j-1}\right]=h_y^2 f_{ij}$$
4. $v_{i,j}=\mu_3(x_i) \qquad j=0;\quad i=1,\dots,N_x-1;$
5. $v_{i,j}=\mu_4(x_i) \qquad j=N_y;\quad i=1,\dots,N_x-1;$

Теперь заменим двухразмерную v на одноразмерную w с помощью следующего преобразования

$$j=0,1,\dots,N_y;\qquad i=0,1,\dots,N_x;\qquad m=jL+i+1,\qquad L=N_x+1$$

$$v_{i,j-1}\rightarrow w_{m-L}\qquad v_{i-1,j}\rightarrow w_{m-1}\qquad v_{i,j}\rightarrow w_m\qquad v_{i+1,j}\rightarrow w_{m+1}\qquad v_{i,j+1}\rightarrow w_{m+L}$$

Получаем для первого:

$$-[h_y^2 k_1(x_{i+1/2})\frac{w_{m+1}-w_m}{h_x^2}-h_y^2 k_1(x_{i-1/2})\frac{w_m-w_{m-1}}{h_x^2}+w_{m-L}-2w_m+w_{m+L}]=h_y^2 f_m$$

$$a_m w_{m-L}+b_m w_{m-1}+c_m w_m+d_m w_{m+1}+e_m w_{m+L}=g_m,$$

$$a_m=-1,$$

$$b_m=-\frac{h_y^2}{h_x^2}k_1(x_{i-1/2})$$

$$c_m=2+\frac{h_y^2}{h_x^2}k_1(x_{i-1/2})+\frac{h_y^2}{h_x^2}k_1(x_{i+1/2}) \quad d_m=-\frac{h_y^2}{h_x^2}k_1(x_{i+1/2}) \quad e_m=-1 \quad g_m=h_y^2 f_m$$

Для второго:

$$w_m=\mu_1(y_j) \quad c_m w_m=g_m, \quad a_m=0, \quad b_m=0 \quad c_m=1 \quad d_m=0 \quad d_m=0 \quad g_m=\mu_1(y_j)$$

Для третьего:

$$-[-2h_y^2(\chi_2 w_m-\mu_2(y_j))-2h_y^2 k_1(x_{i-1/2})\frac{w_m-w_{m-1}}{h_x^2}+w_{m+L}-2w_m+w_{m-L}]=h_y^2 f_m$$

$$-w_{m-L}-\frac{2h_y^2}{h_x^2}k_1(x_{i-1/2})w_{m-1}+\left(\frac{2h_y^2}{h_x^2}k_1(x_{i-1/2})+2+2h_y^2\chi_2\right)w_m-w_{m+L}=h_y^2 f_{ij}+2h_y^2\mu_2(y_j)$$

$$a_m=-1,$$

$$b_m=-\frac{2h_y^2}{h_x^2}k_1(x_{i-1/2}) \quad c_m=2+\frac{h_y^2}{h_x^2}k_1(x_{i-1/2})+2h_y^2\chi_2 \quad d_m=0 \quad e_m=-1$$

$$g_m=h_y^2 f_m+2h_y^2\mu_2(y_j)$$

Для четвертого:

$$w_m=\mu_3(x_i) \quad c_m w_m=g_m, \quad a_m=0, \quad b_m=0 \quad c_m=1 \quad d_m=0 \quad e_m=0 \quad g_m=\mu_3(x_i)$$

Для пятого:

$$w_m=\mu_4(x_i) \quad c_m w_m=g_m, \quad a_m=0, \quad b_m=0 \quad c_m=1 \quad d_m=0 \quad e_m=0 \quad g_m=\mu_4(x_i)$$

Построение главной матрицы

$$Aw = g \qquad A \in R^{N\times N}, \quad w, g \in R^N, \quad N = (N_x + 1)(N_y + 1)$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1																								
	1																							
		1																						
			1																					
				1																				
					1																			
	*				*	*	*				*													
		*				*	*	*				*												
			*				*	*	*				*											
				*				*	*					*										
										1														
						*				*	*	*				*								
							*				*	*	*				*							
								*				*	*	*				*						
									*				*	*					*					
															1									
											*				*	*	*				*			
												*				*	*	*				*		
													*				*	*	*				*	
														*				*	*					*
																				1				
																					1			
																						1		
																							1	
																								1

С помощью формул выше построим матрицу А. Её структуру приблизительно отображает рисунок сверху. Избавимся от элементов помеченных красным квадратом для упрощения вычислений с помощью домножения и вычитания соответствующих рядов.

Получим более упрощенный вариант главной матрицы:

7	8	9	10	12	13	14	15	17	18	19	20

1	2	3	4	5	6	7	8	9	10	11	12

*	*			*							
*	*	*			*						
	*	*	*			*					
		*	*				*				
*				*	*			*			
	*			*	*	*			*		
		*			*	*	*			*	
			*			*	*				*
				*				*	*		
					*			*	*	*	
						*			*	*	*
							*			*	*

$j = 1,2,\ldots,N_y - 1;$

$i = 1,2,\ldots,N_x;$ $\quad m = (j-1)L + i,$ $\quad L = N_x$

Решение Aw=g

Для решения заданной системы уравнений воспользуемся методом полной редукции. Приведем основные формулы

$$\begin{cases} V_0 = F_0, \\ -V_{j-1} + CV_j - V_{j+1} = F_j, \quad j = 1,2,\ \ ,N_y - 1 \\ V_{N_y} = F_{N_y}, \end{cases} \qquad N_y = 2^n$$

$$C \in R^{(N_x+1)\times(N_x+1)}, \qquad F,\ V_j \in R^{(N_x+1)},$$

$$C^{(0)} = C, \qquad F_j^{(0)} = F_j, \quad j = 1,2,\ldots,N_y - 1$$

$$\begin{cases} V_0 = F_0, \\ -V_{j-1} + C^{(0)}V_j - V_{j+1} = F_j^{(0)}, \quad j = 1,2,\ \ ,N_y - 1 \\ V_{N_y} = F_{N_y}, \end{cases}$$

$$-V_{j-2}+C^{(0)}V_{j-1}\quad -V_{j}\quad =F^{(0)}_{j-1},$$
$$-V_{j-1}+C^{(0)}V_{j}\quad -V_{j+1}\quad =F^{(0)}_{j},\quad j=2,4,6,\ldots,N_y-2$$
$$-V_{j}+C^{(0)}V_{j+1}-V_{j}=F^{(0)}_{j+1}$$

$$\frac{N_y}{2}-1$$

$$-V_{j-2}+C^{(1)}V_{j}\quad -V_{j+2}\quad =F^{(1)}_{j},\quad j=2,4,6,\ldots,N_y-2$$

$$C^{(1)}=\left[C^{(0)}\right]^2-2E$$

$$F^{(1)}_{j}=F^{(0)}_{j-1}+C^{(0)}F^{(0)}_{j}+F^{(0)}_{j+1},\quad j=2,4,6,\ldots,N_y-2$$

$$C^{(0)}V_{j}=F^{(0)}_{j}+V_{j-1}+V_{j+1},\quad j=1,3,5,\ldots,N_y-1$$

$$-V_{j-4}+C^{(1)}V_{j-2}\quad -V_{j}\quad =F^{(1)}_{j-2},$$
$$-V_{j-2}+C^{(1)}V_{j}\quad -V_{j+2}\quad =F^{(1)}_{j},\quad j=4,8,12,\ldots,N_y-4$$
$$-V_{j}+C^{(1)}V_{j+4}-V_{j}=F^{(1)}_{j+2}$$

$$\frac{N_y}{4}-1$$

$$-V_{j-4}+C^{(2)}V_{j}\quad -V_{j+4}\quad =F^{(2)}_{j},\quad j=4,8,12,\ldots,N_y-4$$

$$C^{(2)}=\left[C^{(1)}\right]^2-2E$$

$$F^{(2)}_{j}=F^{(1)}_{j-2}+C^{(1)}F^{(1)}_{j}+F^{(1)}_{j+2},\quad j=4,8,12,\ldots,N_y-4$$

$$-V_{j-4}+C^{(1)}V_{j-2}\qquad -V_j\qquad\qquad =F^{(1)}_{j-2},$$

$$-V_{j-2}+C^{(1)}V_j\qquad -V_{j+2}\qquad =F^{(1)}_j,\qquad j=4,8,12,\ldots,N_y-4$$

$$-V_j+C^{(1)}V_{j+4}-V_j=F^{(1)}_{j+2}$$

$$\frac{N_y}{4}-1$$

$$-V_{j-4}+C^{(2)}V_j\qquad -V_{j+4}\qquad =F^{(2)}_j,\qquad j=4,8,12,\ldots,N_y-4$$

$$C^{(2)}=\left[C^{(1)}\right]^2-2E$$

$$F^{(2)}_j=F^{(1)}_{j-2}+C^{(1)}F^{(1)}_j+F^{(1)}_{j+2},\qquad j=4,8,12,\ldots,N_y-4$$

Метод полной редукции. Прямой ход.

$$k=1,2,\ldots,n-1$$

$$-V_{j-2^k}+C^{(k)}V_j\qquad -V_{j+2^k}\qquad =F^{(k)}_j,\qquad j=2^k,\ 2\cdot 2^k,\ 3\cdot 2^k,\ldots,N_y-2^k$$

$$V_0=F_0,\qquad\qquad V_{N_y}=F_{N_y},$$

$$C^{(k)}=\left[C^{(k-1)}\right]^2-2E$$

$$F^{(k)}_j=F^{(k-1)}_{j-2^{k-1}}+C^{(k-1)}F^{(k-1)}_j+F^{(k-1)}_{j+2^{k-1}},\qquad j=2^k,2\cdot 2^k,3\cdot 2^k,\ldots,N_y-2^k$$

$$k=n-1$$

$$j=2^{n-1},N_y-2^{n-1}\qquad N_y-2^{n-1}=2^n-2^{n-1}=2^{n-1}(2-1)=2^{n-1}\qquad j=2^{n-1}=\frac{N_y}{2}$$

$$-V_0+C^{(n-1)}V_j-V_{N_y}=F^{(n-1)}_j,\qquad\qquad C^{(n-1)}V_j=F^{(n-1)}_j+V_0+V_{N_y},$$

Метод полной редукции. Прямой ход.

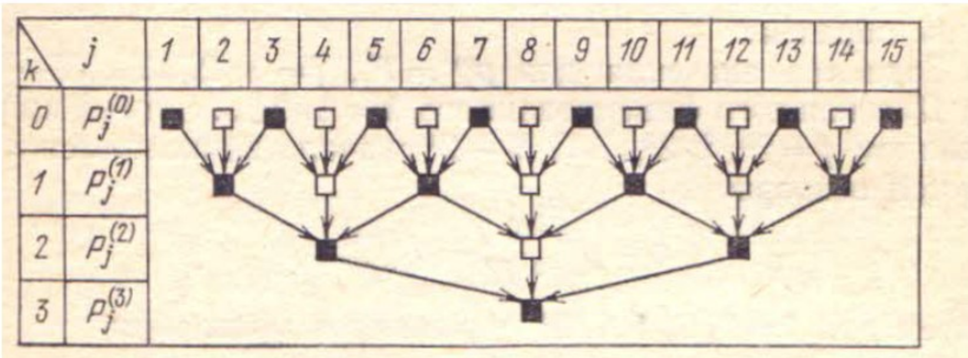


Рис. 1.

Метод полной редукции. Обратный ход.

$$k = n,\ n-1, \ldots, 1$$

$$C^{(k-1)} V_j = F_j^{(k-1)} + V_{j-2^{k-1}} + V_{j+2^{k-1}}, \qquad j = 2^{k-1},\ 3\cdot 2^{k-1},\ 5\cdot 2^{k-1} \ldots,\ N_y - 2^{k-1}$$

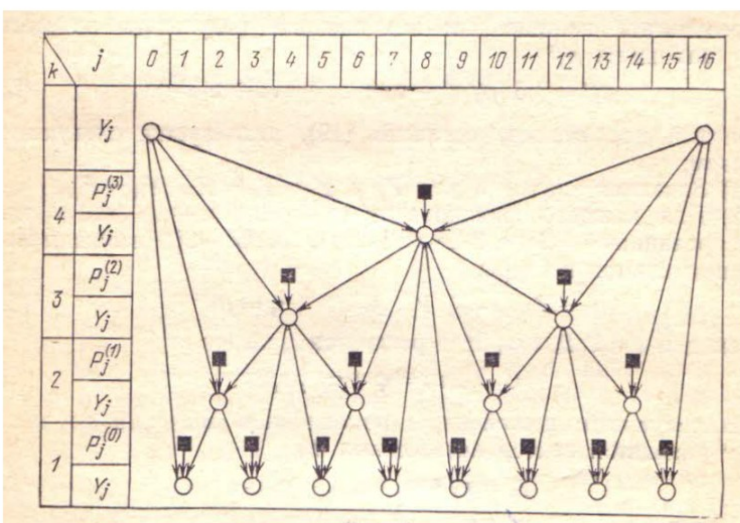


Рис. 2.

Тестовый пример

$$A=\begin{bmatrix} 40 & 0.1 & 0 & 0 & 0 \\ 0.2 & 50 & 0.1 & 0 & 0 \\ 0 & 0.3 & 60 & 0.1 & 0 \\ 0 & 0 & 0.4 & 70 & 0.1 \\ 0 & 0 & 0 & 0.5 & 80 \end{bmatrix} \qquad b=\begin{bmatrix} 6 \\ 14 \\ 22 \\ 30 \\ 38 \end{bmatrix}$$

Ожидаемое решение:

$$b=\begin{bmatrix} 0.14930332 \\ 0.27867366 \\ 0.36456361 \\ 0.42581344 \\ 0.47233867 \end{bmatrix}$$

Вектор невязки

Для оценки погрешности примененного метода найдем формулу для вектора невязки

$$\varepsilon=f(x,y)+\left[\frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right)+\frac{\partial^2 u}{\partial y^2}\right]$$

$$\varepsilon=f(x,y)+\left[\frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right)+\frac{\partial^2 u}{\partial y^2}\right]$$

$$\varepsilon=[h_y k_1(x_{i+1/2})\frac{v_{i+1,j}-v_{i,j}}{h_x}-h_y k_1(x_{i-1/2})\frac{v_{i,j}-v_{i-1,j}}{h_x}+h_x\frac{v_{i,j+1}-v_{i,j}}{h_y}-h_x\frac{v_{i,j}-v_{i,j-1}}{h_y}]+h_x h_y f_{ij}$$

$$i = 1,\ldots,N_x\text{-}1 \quad j = 1,\ldots,N_y\text{-}1$$

$$v_{i+1,j}=v(x_i+h_x,y_j)=v_{i,j}+h_x\frac{\partial u_{i,j}}{\partial x}+h_x^2\frac{1}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}+h_x^3\frac{1}{6}\frac{\partial^3 u_{i,j}}{\partial x^3}+h_x^4\frac{1}{24}\frac{\partial^4 u_{i,j}}{\partial x^4}+O(h_x^5)$$

$$\frac{v_{i+1,j}-v_{i,j}}{h_x}=\frac{\partial u_{i,j}}{\partial x}+h_x\frac{1}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}+h_x^2\frac{1}{6}\frac{\partial^3 u_{i,j}}{\partial x^3}+h_x^3\frac{1}{24}\frac{\partial^4 u_{i,j}}{\partial x^4}+O(h_x^4)$$

$$k_{i+0.5}=k(x_i+\frac{h_x}{2})=k_i+\frac{h_x}{2}\frac{dk_i}{dx}+\frac{h_x^2}{8}\frac{d^2 k_i}{dx^2}+\frac{h_x^3}{48}\frac{d^3 k_i}{dx^3}+O(h_x^4)$$

$$k_{i+0.5}\frac{v_{i+1,j}-v_{i,j}}{h_x}=k_i\frac{\partial u_{i,j}}{\partial x}+\frac{h_x}{2}\frac{dk_i}{dx}\frac{\partial u_{i,j}}{\partial x}+\frac{h_x}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}k_i+\frac{h_x^2}{8}\frac{d^2 k_i}{dx^2}\frac{\partial u_{i,j}}{\partial x}+\frac{h_x^2}{4}\frac{\partial^2 u_{i,j}}{\partial x^2}\frac{dk_i}{dx}+\frac{h_x^2}{6}k_i\frac{\partial^3 u_{i,j}}{\partial x^3}$$
$$+\frac{h_x^3}{24}k_i\frac{\partial^4 u_{i,j}}{\partial x^4}+\frac{h_x^3}{12}\frac{dk_i}{dx}\frac{\partial^3 u_{i,j}}{\partial x^3}+\frac{h_x^3}{16}\frac{d^2 k_i}{dx^2}\frac{\partial^2 u_{i,j}}{\partial x^2}+\frac{h_x^3}{48}\frac{d^3 k_i}{dx^3}\frac{\partial u_{i,j}}{\partial x}+O(h_x^4)$$

$$v_{i-1,j}=v(x_i-h_x,y_j)=v_{i,j}-h_x\frac{\partial u_{i,j}}{\partial x}+h_x^2\frac{1}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}-h_x^3\frac{1}{6}\frac{\partial^3 u_{i,j}}{\partial x^3}+h_x^4\frac{1}{24}\frac{\partial^4 u_{i,j}}{\partial x^4}+O(h_x^5)$$

$$\frac{v_{i,j}-v_{i-1,j}}{h_x}=\frac{\partial u_{i,j}}{\partial x}-\frac{h_x}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}+h_x^2\frac{1}{6}\frac{\partial^3 u_{i,j}}{\partial x^3}-h_x^3\frac{1}{24}\frac{\partial^4 u_{i,j}}{\partial x^4}+O(h_x^4)$$

$$k_{i+0.5}=k(x_i+\frac{h_x}{2})=k_i-\frac{h_x}{2}\frac{dk_i}{dx}+\frac{h_x^2}{8}\frac{d^2k_i}{dx^2}-\frac{h_x^3}{48}\frac{d^3k_i}{dx^3}+O(h_x^4)$$

$$k_{i+0.5}\frac{v_{i-1,j}-v_{i,j}}{h_x}=k_i\frac{\partial u_{i,j}}{\partial x}-\frac{h_x}{2}\frac{dk_i}{dx}\frac{\partial u_{i,j}}{\partial x}-\frac{h_x}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}k_i+\frac{h_x^2}{8}\frac{d^2k_i}{dx^2}\frac{\partial u_{i,j}}{\partial x}+\frac{h_x^2}{4}\frac{\partial^2 u_{i,j}}{\partial x^2}\frac{dk_i}{dx}+\frac{h_x^2}{6}k_i\frac{\partial^3 u_{i,j}}{\partial x^3}$$
$$\frac{-h_x^3}{24}k_i\frac{\partial^4 u_{i,j}}{\partial x^4}-\frac{h_x^3}{12}\frac{dk_i}{dx}\frac{\partial^3 u_{i,j}}{\partial x^3}-\frac{h_x^3}{16}\frac{d^2k_i}{dx^2}\frac{\partial^2 u_{i,j}}{\partial x^2}-\frac{h_x^3}{48}\frac{d^3k_i}{dx^3}\frac{\partial u_{i,j}}{\partial x}+O(h_x^4)$$

$$v_{i,j+1}=v(x_i,y_j+h_y)=v_{i,j}+h_y\frac{\partial u_{i,j}}{\partial y}+h_y^2\frac{1}{2}\frac{\partial^2 u_{i,j}}{\partial y^2}+h_y^3\frac{1}{6}\frac{\partial^3 u_{i,j}}{\partial y^3}+h_y^4\frac{1}{24}\frac{\partial^4 u_{i,j}}{\partial y^4}+O(h_y^5)$$

$$\frac{v_{i,j+1}-v_{i,j}}{h_y}=\frac{\partial u_{i,j}}{\partial y}+\frac{h_y}{2}\frac{\partial^2 u_{i,j}}{\partial y^2}+\frac{h_y^2}{6}\frac{\partial^3 u_{i,j}}{\partial y^3}+\frac{h_y^3}{24}\frac{\partial^4 u_{i,j}}{\partial y^4}+O(h_y^4)$$

$$v_{i,j-1}=v(x_i,y_j-h_y)=v_{i,j}-h_y\frac{\partial u_{i,j}}{\partial y}+h_y^2\frac{1}{2}\frac{\partial^2 u_{i,j}}{\partial y^2}-h_y^3\frac{1}{6}\frac{\partial^3 u_{i,j}}{\partial y^3}+h_y^4\frac{1}{24}\frac{\partial^4 u_{i,j}}{\partial y^4}+O(h_y^5)$$

$$\frac{v_{i,j}-v_{i,j-1}}{h_y}=\frac{\partial u_{i,j}}{\partial y}-\frac{h_y}{2}\frac{\partial^2 u_{i,j}}{\partial y^2}+\frac{h_y^2}{6}\frac{\partial^3 u_{i,j}}{\partial y^3}-\frac{h_y^3}{24}\frac{\partial^4 u_{i,j}}{\partial y^4}+O(h_y^4)$$

Подставим все выведенные формулы в изначальное уравнение

$$\varepsilon=h_y k_{i+0.5}\frac{v_{i+1,j}-v_{i,j}}{h_x}-h_y k_{i+0.5}\frac{v_{i,j}-v_{i-1,j}}{h_x}+h_x\frac{v_{i,j+1}-v_{i,j}}{h_y}-h_x\frac{v_{i,j-1}-v_{i,j}}{h_y}+h_x h_y f_{i,j}$$

$\rightarrow$

$$\varepsilon=h_y k_i\frac{\partial u_{i,j}}{\partial x}+\frac{h_x}{2}\frac{dk_i}{dx}\frac{\partial u_{i,j}}{\partial x}+\frac{h_x}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}k_i+\frac{h_x^2}{8}\frac{d^2k_i}{dx^2}\frac{\partial u_{i,j}}{\partial x}+\frac{h_x^2}{4}\frac{\partial^2 u_{i,j}}{\partial x^2}\frac{dk_i}{dx}+\frac{h_x^2}{6}k_i\frac{\partial^3 u_{i,j}}{\partial x^3}$$
$$+\frac{h_x^3}{24}k_i\frac{\partial^4 u_{i,j}}{\partial x^4}+\frac{h_x^3}{12}\frac{dk_i}{dx}\frac{\partial^3 u_{i,j}}{\partial x^3}+\frac{h_x^3}{16}\frac{d^2k_i}{dx^2}\frac{\partial^2 u_{i,j}}{\partial x^2}+\frac{h_x^3}{48}\frac{d^3k_i}{dx^3}\frac{\partial u_{i,j}}{\partial x}$$
$$-h_y k_i\frac{\partial u_{i,j}}{\partial x}+\frac{h_x}{2}\frac{dk_i}{dx}\frac{\partial u_{i,j}}{\partial x}+\frac{h_x}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}k_i-\frac{h_x^2}{8}\frac{d^2k_i}{dx^2}\frac{\partial u_{i,j}}{\partial x}-\frac{h_x^2}{4}\frac{\partial^2 u_{i,j}}{\partial x^2}\frac{dk_i}{dx}-\frac{h_x^2}{6}k_i\frac{\partial^3 u_{i,j}}{\partial x^3}$$
$$+\frac{h_x^3}{24}k_i\frac{\partial^4 u_{i,j}}{\partial x^4}+\frac{h_x^3}{12}\frac{dk_i}{dx}\frac{\partial^3 u_{i,j}}{\partial x^3}+\frac{h_x^3}{16}\frac{d^2k_i}{dx^2}\frac{\partial^2 u_{i,j}}{\partial x^2}+\frac{h_x^3}{48}\frac{d^3k_i}{dx^3}\frac{\partial u_{i,j}}{\partial x}$$
$$+h_x\frac{\partial u_{i,j}}{\partial y}+\frac{h_y}{2}\frac{\partial^2 u_{i,j}}{\partial y^2}+\frac{h_y^2}{6}\frac{\partial^3 u_{i,j}}{\partial y^3}+\frac{h_y^3}{24}\frac{\partial^4 u_{i,j}}{\partial y^4}$$
$$-h_x\frac{\partial u_{i,j}}{\partial y}+\frac{h_y}{2}\frac{\partial^2 u_{i,j}}{\partial y^2}-\frac{h_y^2}{6}\frac{\partial^3 u_{i,j}}{\partial y^3}+\frac{h_y^3}{24}\frac{\partial^4 u_{i,j}}{\partial y^4}+h_x h_y f_{i,j}$$

Упрощаем

$$\varepsilon=\frac{h_x}{2}\frac{dk_i}{dx}\frac{\partial u_{i,j}}{\partial x}+\frac{h_x}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}k_i$$

$$+\frac{h_x^3}{24}k_i\frac{\partial^4 u_{i,j}}{\partial x^4}+\frac{h_x^3}{12}\frac{dk_i}{dx}\frac{\partial^3 u_{i,j}}{\partial x^3}+\frac{h_x^3}{16}\frac{d^2k_i}{dx^2}\frac{\partial^2 u_{i,j}}{\partial x^2}+\frac{h_x^3}{48}\frac{d^3k_i}{dx^3}\frac{\partial u_{i,j}}{\partial x}$$

$$+\frac{h_x}{2}\frac{dk_i}{dx}\frac{\partial u_{i,j}}{\partial x}+\frac{h_x}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}k_i$$

$$+\frac{h_x^3}{24}k_i\frac{\partial^4 u_{i,j}}{\partial x^4}+\frac{h_x^3}{12}\frac{dk_i}{dx}\frac{\partial^3 u_{i,j}}{\partial x^3}+\frac{h_x^3}{16}\frac{d^2k_i}{dx^2}\frac{\partial^2 u_{i,j}}{\partial x^2}+\frac{h_x^3}{48}\frac{d^3k_i}{dx^3}\frac{\partial u_{i,j}}{\partial x}$$

$$+\frac{h_y}{2}\frac{\partial^2 u_{i,j}}{\partial y^2}+\frac{h_y^3}{24}\frac{\partial^4 u_{i,j}}{\partial y^4}$$

$$+\frac{h_y}{2}\frac{\partial^2 u_{i,j}}{\partial y^2}+\frac{h_y^3}{24}\frac{\partial^4 u_{i,j}}{\partial y^4}+h_x h_y f_{i,j}$$

Еще упрощаем

$$\varepsilon=h_x\frac{dk_i}{dx}\frac{\partial u_{i,j}}{\partial x}+h_x\frac{\partial^2 u_{i,j}}{\partial x^2}k_i$$

$$+\frac{h_x^3}{12}k_i\frac{\partial^4 u_{i,j}}{\partial x^4}+\frac{h_x^3}{6}\frac{dk_i}{dx}\frac{\partial^3 u_{i,j}}{\partial x^3}+\frac{h_x^3}{8}\frac{d^2k_i}{dx^2}\frac{\partial^2 u_{i,j}}{\partial x^2}+\frac{h_x^3}{24}\frac{d^3k_i}{dx^3}\frac{\partial u_{i,j}}{\partial x}$$

$$+h_y\frac{\partial^2 u_{i,j}}{\partial y^2}+\frac{h_y^3}{12}\frac{\partial^4 u_{i,j}}{\partial y^4}+h_x h_y f_{i,j}$$

В конце концов имеем:

$$\varepsilon=h_x\frac{\partial}{\partial x}\left(k_i\frac{\partial u_{i,j}}{\partial x}\right)$$

$$+h_x^3\left(\frac{+k_i}{12}\frac{\partial^4 u_{i,j}}{\partial x^4}+\frac{1}{6}\frac{dk_i}{dx}\frac{\partial^3 u_{i,j}}{\partial x^3}+\frac{1}{8}\frac{d^2k_i}{dx^2}\frac{\partial^2 u_{i,j}}{\partial x^2}+\frac{1}{24}\frac{d^3k_i}{dx^3}\frac{\partial u_{i,j}}{\partial x}\right)$$

$$+h_y\frac{\partial^2 u_{i,j}}{\partial y^2}+\frac{h_y^3}{12}\frac{\partial^4 u_{i,j}}{\partial y^4}+h_x h_y f_{i,j}$$

Видим, что при уменьшении размера шага h_x в два раза — первое и последнее слагаемые уменьшатся соответственно в два раза. Второе слагаемое в 8 раз. Аналогично можно сказать про h_y .

Также следует обратить внимание, что если у нас u(x, y) будет зависеть от x линейно и от y хотя бы квадратично, и при этом k будет линейным, то погрешность аппроксимации будет стремиться к нулю.

Благодаря этим выводам можно будет оценить полученную погрешность и убедиться в корректности работы программы.

Теперь рассмотрим для границы третьего рода:

$$\varepsilon=[-h_y\left(\chi_2 v_{i,j}-\mu_2\left(y_j\right)\right)-h_y k_1\left(x_{i-1/2}\right)\frac{v_{i,j}-v_{i-1,j}}{h_x}+\frac{h_x}{2}\frac{v_{i,j+1}-v_{i,j}}{h_y}-\frac{h_x}{2}\frac{v_{i,j}-v_{i,j-1}}{h_y}]+\frac{h_x}{2}h_y f_{ij}$$

Подставим вычисленные выше формулы

$$\varepsilon=-h_y((x_2 v_{i,j}-\mu_2(y_j))-k_i\frac{\partial u_{i,j}}{\partial x})+\frac{h_x}{2}\frac{dk_i}{dx}\frac{\partial u_{i,j}}{\partial x}+\frac{h_x}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}k_i-\frac{h_x^2}{8}\frac{d^2 k_i}{dx^2}\frac{\partial u_{i,j}}{\partial x}-\frac{h_x^2}{4}\frac{\partial^2 u_{i,j}}{\partial x^2}\frac{dk_i}{dx}-\frac{h_x^2}{6}k_i\frac{\partial^3 u_{i,j}}{\partial x^3}$$

$$+\frac{h_x^3}{24}k_i\frac{\partial^4 u_{i,j}}{\partial x^4}+\frac{h_x^3}{12}\frac{dk_i}{dx}\frac{\partial^3 u_{i,j}}{\partial x^3}+\frac{h_x^3}{16}\frac{d^2 k_i}{dx^2}\frac{\partial^2 u_{i,j}}{\partial x^2}+\frac{h_x^3}{48}\frac{d^3 k_i}{dx^3}\frac{\partial u_{i,j}}{\partial x}$$

$$+h_x\frac{\partial u_{i,j}}{\partial y}+\frac{h_y}{2}\frac{\partial^2 u_{i,j}}{\partial y^2}+\frac{h_y^2}{6}\frac{\partial^3 u_{i,j}}{\partial y^3}+\frac{h_y^3}{24}\frac{\partial^4 u_{i,j}}{\partial y^4}$$

$$-h_x\frac{\partial u_{i,j}}{\partial y}+\frac{h_y}{2}\frac{\partial^2 u_{i,j}}{\partial y^2}-\frac{h_y^2}{6}\frac{\partial^3 u_{i,j}}{\partial y^3}+\frac{h_y^3}{24}\frac{\partial^4 u_{i,j}}{\partial y^4}+h_x h_y f_{i,j}$$

$i=N_x-1$

Упрощаем

$$\varepsilon=-h_y((x_2 v_{i,j}-\mu_2(y_j))-k_i\frac{\partial u_{i,j}}{\partial x})+\frac{h_x}{2}\frac{dk_i}{dx}\frac{\partial u_{i,j}}{\partial x}+\frac{h_x}{2}\frac{\partial^2 u_{i,j}}{\partial x^2}k_i-\frac{h_x^2}{8}\frac{d^2 k_i}{dx^2}\frac{\partial u_{i,j}}{\partial x}-\frac{h_x^2}{4}\frac{\partial^2 u_{i,j}}{\partial x^2}\frac{dk_i}{dx}-\frac{h_x^2}{6}k_i\frac{\partial^3 u_{i,j}}{\partial x^3}$$

$$+\frac{h_x^3}{24}k_i\frac{\partial^4 u_{i,j}}{\partial x^4}+\frac{h_x^3}{12}\frac{dk_i}{dx}\frac{\partial^3 u_{i,j}}{\partial x^3}+\frac{h_x^3}{16}\frac{d^2 k_i}{dx^2}\frac{\partial^2 u_{i,j}}{\partial x^2}+\frac{h_x^3}{48}\frac{d^3 k_i}{dx^3}\frac{\partial u_{i,j}}{\partial x}$$

$$+h_y\frac{\partial^2 u_{i,j}}{\partial y^2}+\frac{h_y^3}{12}\frac{\partial^4 u_{i,j}}{\partial y^4}+h_x h_y f_{i,j}$$

Видим что в полученной аппроксимации погрешности границы 3 рода стремление к нулю получить так просто не удастся. Однако по аналогии с предыдущим выводом — при уменьшении размера h_x в два раза — второе и третье слагаемые уменьшаются в два раза, третье, четвертое и пятое в 4 раза, а оставшиеся около h_x^3 в 8 раз. Аналогично с h_y. Опять же, эти выводы позволят нам оценить правильность полученной погрешности

Пример 1

$u = 3$

$a = 1$

$b = 2$

$c = 1$

$d = 2$

$\mu_1 = 3$

$\mu_3 = 3$

$\mu_4 = 3$

$\kappa_1 = 2$

$\kappa_2 = 5$

$\mu_2 = 15$

$f = 0$

Данный пример нужен в основном чтобы удостовериться что в программе не допущена какая либо нелепая ошибка. Если всё написано корректно, то мы увидим достаточно хорошую точность, поскольку все входные данные здесь константы. Погрешность аппроксимации в данном случае в соответствии с выведенным вектором невязки должна быть равна нулю. Лишь граничное условие 3 рода будет давать небольшую погрешность аппроксимацию

x	y	Inaccuracy
5	5	6.68037543430942e-15
5	10	5.78876585363819e-15
5	20	6.56619844751721e-15
5	50	6.59688006695215e-15
10	5	1.18232947158736e-14
10	10	1.03951027385868e-14
10	20	1.06704455114298e-14
10	50	1.15364591896238e-14
20	5	2.05555494216064e-14
20	10	2.11079399115909e-14
20	20	2.09547941037243e-14
20	50	2.12577418495238e-14
50	5	5.07295689455837e-14
50	10	5.10268018203913e-14
50	20	5.19044594503495e-14
50	50	5.18323901360074e-14

Пример 2

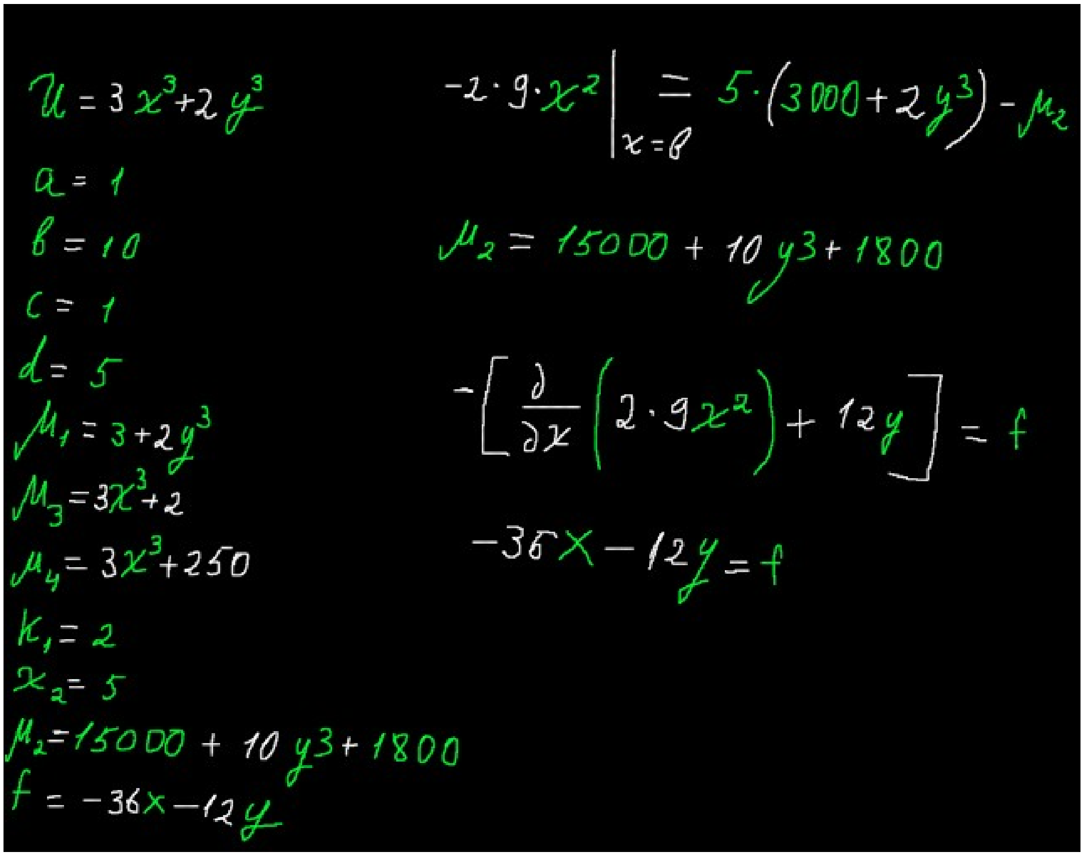
$$u = x + y$$
$$a = 1$$
$$b = 2$$
$$c = 1$$
$$d = 2$$
$$\mu_1 = y + 1$$
$$\mu_3 = x + 1$$
$$\mu_4 = x + 2$$
$$k_1 = 2x$$
$$x_2 = 2$$
$$\mu_2 = 8 + y$$
$$f = -2$$

$$-2 \cdot 2 \cdot 1 = 2(2 + y) - \mu_2$$
$$\mu_2 = 8 + y$$

В данном примере по аналогии с предыдущим погрешность аппроксимации должна стремиться к нулю. Видим, что u и k зависят от x, y линейно. Имеем следующий результат

x	y	Inaccuracy
5	5	7.34381704603011e-15
5	10	6.58489202283941e-15
5	20	4.33593896819791e-15
5	50	6.24224294918813e-15
10	5	1.18887640565603e-14
10	10	1.21856270768979e-14
10	20	1.01899110222863e-14
10	50	1.07998129626126e-14
20	5	1.94489276812106e-14
20	10	2.18311251542592e-14
20	20	2.15169253384621e-14
20	50	2.02461818231255e-14
50	5	5.1339578359735e-14
50	10	5.15289005109281e-14
50	20	5.13370292561121e-14
50	50	5.28811122007937e-14

Пример 3



Подготовим пример для тестирования программы. Исходная функция будет иметь 3 степень x и y, таким образом мы получил f которая зависит линейно от переменных. K возьмем константной для простоты.

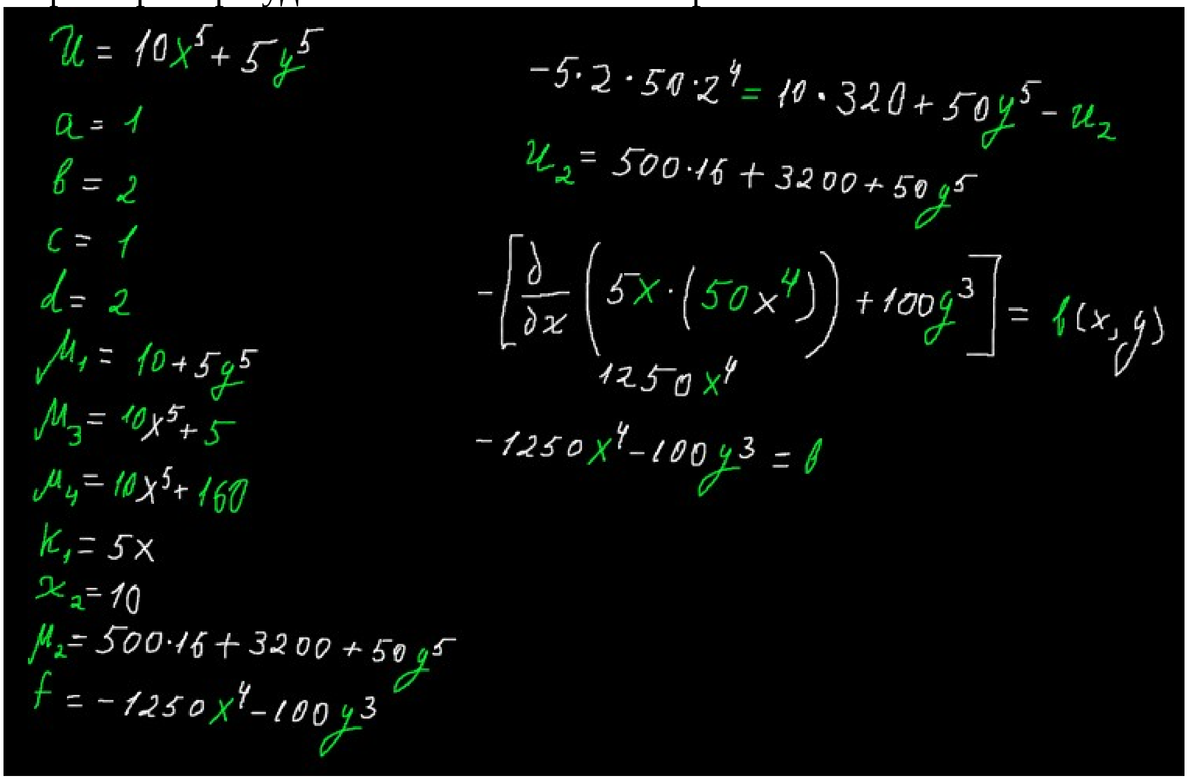
В результате работы программы получаем следующие результаты:

x	y	Inaccuracy
5	5	0.017185863
5	10	0.0061385518
5	20	0.002192399
5	50	0.00056209458
10	5	0.0061386631
10	10	0.0021923828
10	20	0.00078299581
10	50	0.00020074799
20	5	0.0021923942
20	10	0.00078299967
20	20	0.00027964294
20	50	7.1695706e-05
50	5	0.00056209539
50	10	0.00020074762
50	20	7.169567e-05
50	50	1.8381556e-05

Здесь уже хорошо видно погрешность аппроксимации. Действительно, поскольку производная второй степени u не равна нулю ни по какой переменной, и производная k первой степени не равна нулю — то она начинает заметно расти по сравнению с предыдущим примером. Видно что при уменьшении шага примерно в 2 раза погрешность падает примерно в 2.8 раз, что примерно соответствует ожидаемому вектору невязки.

Пример 4

Второй пример будет несколько сложнее первого в плане степеней.



В данном примере можно наблюдать уже довольно большую погрешность апроксимации, которую получается побороть лишь сильно увеличив количество разбиений. Даже с количеством разбиений по x 50 и по y 50 — получаем суммарную погреность ~ 0.028.

x	y	Inaccuracy
5	5	26.318671
5	10	9.3170945
5	20	3.2285166
5	50	0.84579845
10	5	8.7356194
10	10	3.337217
10	20	1.1871234
10	50	0.30186056
20	5	3.1898254
20	10	1.1783316
20	20	0.4185313
20	50	0.10751796
50	5	0.8426628
50	10	0.30164657
50	20	0.10755476
50	50	0.027579884

Вывод

Метод **нечетно-чётной редукции** является эффективным способом решения **пятидиагональных систем**, которые часто встречаются при дискретизации **двумерных дифференциальных уравнений в частных производных** методом конечных разностей. Этот метод значительно снижает вычислительную сложность по сравнению с прямыми методами решения, что делает его особенно полезным при решении **крупных задач**.

Основные преимущества использования редукции

1. **Сведение пентадиагональной системы к трехдиагональной**
   - Вместо того, чтобы решать исходную систему напрямую, выполняется разбиение на **чётные и нечётные узлы**, что приводит к сокращению размерности системы и упрощает её решение.
2. **Снижение вычислительной сложности**
   - Прямое решение пятидиагональной системы с N неизвестными обычно требует **O(N2) операций**,
   - Метод редукции уменьшает сложность до **O(N)** в случае последовательного выполнения.

Хотя **последовательный алгоритм** редукции уже даёт выигрыш по скорости, его можно **существенно ускорить за счёт параллельных вычислений**.

1. **Разбиение системы на независимые подзадачи**
   - На этапе редукции можно **одновременно исключать элементы в разных частях матрицы**.
   - Это позволяет эффективно использовать **многопоточные вычисления** на CPU.
2. **Использование GPU для ускорения решения**
   - Реализация метода на **графических процессорах (GPU)** с помощью **CUDA/OpenCL** позволяет распараллелить вычисления на тысячи потоков, что **даёт прирост производительности в разы**.
3. **Применение MPI для распределённых систем**

- В случае **кластерных вычислений**, можно распределить подзадачи между узлами **MPI**, что ускоряет решение при очень больших размерностях сетки.

Исходный код

```
====================================
File: ./src/main.cc
====================================
#include <iostream>
#include <iomanip>  // For std::setw, std::fixed, std::setprecision, etc.
#include <chrono>
#include <memory>

#include <Eigen/Dense>
#include <Eigen/Sparse>

#include <defines.hpp>
#include <input_parameters.hpp>
#include <default_impl/main_matrix_calculator.hpp>
#include <interval_splitter.hpp>
#include <default_impl/odd_even_reduction.hpp>
#include <utils.hpp>

auto build_main_matrix(DefaultMainMatrixCalculator const& calc) -> Eigen::SparseMatrix<double>
{
  size_t Nx = calc.interiour_x_points().size();  // Interior points in x-direction
  size_t Ny = calc.interiour_y_points().size();  // Interior points in y-direction
  size_t size = Nx * Ny;  // Total unknowns (interior grid points)

  auto result = Eigen::SparseMatrix<double>(size, size);
  result.setZero();  // Initialize with zeros

  // Iterate over the interior grid
  for (size_t i = 0; i < Nx; ++i) {
    for (size_t j = 0; j < Ny; ++j) {
      size_t idx = i * Ny + j;  // Correct 1D index

      // Left neighbor (i-1, j)
      if (i > 0) {
        result.insert(idx, idx - Ny) = calc.calc_a({i, j});
      }

      // Right neighbor (i+1, j)
      if (i < Nx - 1) {
        result.insert(idx, idx + Ny) = calc.calc_b({i, j});
      }

      // Bottom neighbor (i, j-1)
      if (j > 0) {
        result.insert(idx, idx - 1) = calc.calc_d({i, j});
      }

      // Top neighbor (i, j+1)
      if (j < Ny - 1) {
        result.insert(idx, idx + 1) = calc.calc_e({i, j});
      }

      // Center coefficient
      result.insert(idx, idx) = calc.calc_c({i, j});
    }
  }

  return result;
}
```

```
auto build_g_vector(DefaultMainMatrixCalculator const& calc) -> Eigen::VectorXd
{
  size_t Nx = calc.interiour_x_points().size();  // Interior points in x-direction
  size_t Ny = calc.interiour_y_points().size();  // Interior points in y-direction
  size_t size = Nx * Ny;  // Total number of unknowns

  Eigen::VectorXd g(size);  // Initialize vector of correct size

  // Iterate over interior grid points
  for (size_t i = 0; i < Nx; ++i) {
    for (size_t j = 0; j < Ny; ++j) {
      size_t idx = i * Ny + j;  // Convert (i, j) to 1D index
      g(idx) = calc.calc_g({i, j});  // Compute g at (i, j)
    }
  }

  return g;
}

auto reduce_matrix(DefaultMainMatrixCalculator const& calc, Eigen::MatrixXd const& matrix)
  -> Eigen::MatrixXd
{
  Eigen::MatrixXd reduced_matrix(matrix.rows() - 2, matrix.cols());
  for(size_t row = 1; row < calc.x_points().size(); ++row) {
    reduced_matrix.row(
      row + calc.x_points().size()
    ) = reduced_matrix.row(row + calc.x_points().size())
      - matrix.row(row) * (-matrix(row + calc.x_points().size(), row));
  }

  for(size_t row = matrix.rows() - calc.x_points().size(); row < matrix.rows(); ++row) {
    reduced_matrix.row(
      row - calc.x_points().size()
    ) = reduced_matrix.row(row - calc.x_points().size())
      - matrix.row(row) * (-matrix(row - calc.x_points().size(), row));
  }

  return reduced_matrix;
}

auto convert_w_to_v(Eigen::VectorXd const& w, DefaultMainMatrixCalculator const& calc) -> Eigen::MatrixXd {
    size_t Nx = calc.interiour_x_points().size();
    size_t Ny = calc.interiour_y_points().size();

    Eigen::MatrixXd v(Nx + 2, Ny + 2);
    v.setZero();  // Initialize with zeros

    size_t idx = 0;
    for (size_t i = 1; i <= Nx; ++i) {  // Skip first & last row (boundaries)
        for (size_t j = 1; j <= Ny; ++j) {  // Skip first & last column (boundaries)
            v(i, j) = w(idx++);  // Fill interior solution
        }
    }

    auto params = calc.params();

    for (size_t j = 0; j < Ny + 2; ++j) {
        v(0, j) = params->u1(calc.y_points()[j]);
    }

    for (size_t j = 0; j < Ny + 2; ++j) {
        v(Nx + 1, j) = params->u2(calc.y_points()[j]);
    }

    for (size_t i = 0; i < Nx + 2; ++i) {
```

```
    v(i, 0) = params->u3(calc.x_points()[i]);
  }

  for (size_t i = 0; i < Nx + 2; ++i) {
    v(i, Ny + 1) = params->u4(calc.x_points()[i]);
  }

  return v;
}


void print_expected(DefaultMainMatrixCalculator const& calc, X_Y_Function_type expected_func) {
  const auto& x_points = calc.x_points();  // Full x grid
  const auto& y_points = calc.y_points();  // Full y grid
  size_t Nx = x_points.size();
  size_t Ny = y_points.size();

  //      Print table header (y-values at top)
  //      Print computed values in a grid format
  for (size_t i = 0; i < Nx; ++i) {
    for (size_t j = 0; j < Ny; ++j) {
      double expected_value = expected_func(x_points[i], y_points[j]);  // Compute expected value
      std::cout << std::setw(10) << std::fixed << std::setprecision(4) << expected_value;
    }
    std::cout << "\n";
  }
}

void _do_all(std::shared_ptr<InputParameters> params, X_Y_Function_type expected_func)
{
 static constexpr auto x_interval_counts = {20};
 static constexpr auto y_interval_counts = {20};

 for(auto const x_count : x_interval_counts) {
  for(auto const y_count : y_interval_counts) {
   auto x_points = split_interval(params->xl, params->xr, x_count);
   auto y_points = split_interval(params->yl, params->yr, y_count);

   DefaultMainMatrixCalculator calc(params, x_points, y_points);
   auto main_matrix = build_main_matrix(calc);
   std::cout << "Main matrix: \n" << main_matrix << '\n';
   std::cout << "----------------------------------------\n";
   auto g_vector = build_g_vector(calc);
   std::cout << "G vector: \n" << g_vector << '\n';
   std::cout << "----------------------------------------\n";
   std::cout << "Main matrix size: " << main_matrix.rows() << "x" << main_matrix.cols() << '\n';
   std::cout << "G vector size: " << g_vector.size() << '\n';
   // Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
   // solver.compute(main_matrix);
   // Eigen::VectorXd solution = solver.solve(g_vector);
   Eigen::VectorXd solution = odd_even_reduction_solver(main_matrix, g_vector);
   std::cout << "Solution: \n" << solution << '\n';
   auto v_matrix = convert_w_to_v(solution, calc);
   std::cout << "Solution in v coordinates: \n" << v_matrix << '\n';
   std::cout << "----------------------------------------\n";
   std::cout << "Expected:: \n";
   print_expected(calc, expected_func);
  }
 }
}

void basic_example()
{
 std::shared_ptr<InputParameters> params = std::make_shared<InputParameters>();
 params->xl = 1;
```

```
  params->xr = 10;
  params->yl = 1;
  params->yr = 5;

  params->u1 = [](double y) { return 3 + 2 * y * y * y; };
  params->u3 = [](double x) { return 3 * x * x * x + 2; };
  params->u4 = [](double x) { return 3 * x * x * x + 250; };

  params->k1 = [](double) { return 2; };
  params->hi2 = 5;

  params->u2 = [](double y) { return 15'000 + 10 * y * y * y + 1'800; };

  params->f = [](double x, double y) { return -36 * x - 12 * y; };

  auto expected_func = [](double x, double y) { return 3 * x * x * x + 2 * y * y * y; };

  do_all1(params, expected_func);
}

int main()
{
  basic_example();
  return 0;
}


====================================
File: ./src/default_impl/odd_even_reduction.cc
====================================
#include <default_impl/odd_even_reduction.hpp>

Eigen::VectorXd odd_even_reduction_solver(
  Eigen::VectorXd const& a,
  Eigen::VectorXd const& b,
  Eigen::VectorXd const& c,
  Eigen::VectorXd const& rhs
)
{
  int n = rhs.size();

  if(n == 1) {
    return rhs.array() / b.array();
  }

  int n_half = n / 2;
  Eigen::VectorXd a_half(n_half), b_half(n_half), c_half(n_half), rhs_half(n_half);

  for(int i = 0; i < n_half; ++i) {
    int j = 2 * i + 1;
    double denom = b[j] - a[j] * c[j - 1] / b[j - 1];

    b_half[i] = denom;
    rhs_half[i] = rhs[j] - a[j] * rhs[j - 1] / b[j - 1];

    if(j + 1 < n) {
      a_half[i] = -a[j + 1];
      c_half[i] = -c[j - 1] * c[j] / b[j];
    }
  }

  Eigen::VectorXd x_half = odd_even_reduction_solver(a_half, b_half, c_half, rhs_half);

  Eigen::VectorXd x(n);
  for(int i = 0; i < n_half; ++i) {
```

```
    x[2 * i + 1] = x_half[i];
  }

  for(int i = 0; i < n_half; ++i) {
    int j = 2 * i;
    x[j] = (rhs[j] - c[j] * x[j + 1]) / b[j];
  }

  return x;
}

Eigen::VectorXd odd_even_reduction_solver(
  Eigen::VectorXd const& a,
  Eigen::VectorXd const& b,
  Eigen::VectorXd const& c,
  Eigen::VectorXd const& d,
  Eigen::VectorXd const& e,
  Eigen::VectorXd const& rhs
)
{
  int n = rhs.size();

  if(n == 1) {
    return rhs.array() / b.array();
  }

  int n_half = n / 2;
  Eigen::VectorXd a_half(n_half), b_half(n_half), c_half(n_half), rhs_half(n_half);

  for(int i = 0; i < n_half; ++i) {
    int j = 2 * i + 1;
    double denom = b[j] - a[j] * c[j - 1] / b[j - 1];

    b_half[i] = denom;
    rhs_half[i] = rhs[j] - a[j] * rhs[j - 1] / b[j - 1];

    if(j + 1 < n) {
      a_half[i] = -a[j + 1];
      c_half[i] = -c[j - 1] * c[j] / b[j];
    }
  }

  Eigen::VectorXd x_half = odd_even_reduction_solver(a_half, b_half, c_half, rhs_half);

  Eigen::VectorXd x(n);
  for(int i = 0; i < n_half; ++i) {
    x[2 * i + 1] = x_half[i];
  }

  for(int i = 0; i < n_half; ++i) {
    int j = 2 * i;
    x[j] = (rhs[j] - c[j] * x[j + 1]) / b[j];
  }

  return x;
}

Eigen::VectorXd
  odd_even_reduction_solver(Eigen::MatrixXd const& main_matrix, Eigen::VectorXd const& b)
{
  return Eigen::VectorXd::Zero(b.size());
}

Eigen::VectorXd odd_even_reduction_solver(
  Eigen::SparseMatrix<double> const& main_matrix,
```

```
  Eigen::VectorXd const& b
)
{
  return Eigen::VectorXd::Zero(b.size());
}


====================================
File: ./src/default_impl/main_matrix_calculator.cc
====================================
#include <default_impl/main_matrix_calculator.hpp>

#include <cassert>

#include <contract/contract.hpp>

#include <interval_splitter.hpp>

auto DefaultMainMatrixCalculator::calc_a(Index index) const -> double
{
  contract(fun) {
    precondition(index.i > 0, "index out of range");  // Prevent accessing invalid left neighbor
  };

  double dx = calc_h(interiour_x_points(), index.i);
  double k_left = m_input_p->k1(middle_point(interiour_x_points(), index.i - 1));  // Use i-1

  return - k_left / (dx * dx);
}

auto DefaultMainMatrixCalculator::calc_b(Index index) const -> double
{
  contract(fun) {
    precondition(index.i < interiour_x_points().size() - 1, "index out of range");
  };

  double dx = calc_h(interiour_x_points(), index.i);  // Use index.i instead of index.i + 1
  double k_right = m_input_p->k1(middle_point(interiour_x_points(), index.i + 1));

  return - k_right / (dx * dx);
}

auto DefaultMainMatrixCalculator::calc_c(Index index) const -> double
{
  contract(fun) {
    // precondition(index.i < interiour_x_points().size() - 1, "index out of range");  // Fix boundary condition
    precondition(index.j < interiour_y_points().size(), "index out of range");
  };

  double dx = calc_h(interiour_x_points(), index.i);
  double dy = calc_h(interiour_y_points(), index.j);

  double k_left = m_input_p->k1(middle_point(interiour_x_points(), index.i - 1));
  double k_right = m_input_p->k1(middle_point(interiour_x_points(), index.i + 1));

  return (k_right + k_left) / (dx * dx) + 2.0 / (dy * dy);
}

auto DefaultMainMatrixCalculator::calc_d(Index index) const -> double
{
  contract(fun) {
    precondition(index.j > 0, "index out of range");
  };

  double dy = calc_h(interiour_y_points(), index.j);
```

```
  return - 1.0 / (dy * dy);
}

auto DefaultMainMatrixCalculator::calc_e(Index index) const -> double
{
  contract(fun) {
    precondition(index.j < interiour_y_points().size() - 1, "index out of range");  // Add check
  };

  double dy = calc_h(interiour_y_points(), index.j);

  return - 1.0 / (dy * dy);
}

auto DefaultMainMatrixCalculator::calc_g(Index index) const -> double
{
  contract(fun) {
    precondition(index.i < interiour_x_points().size(), "index out of range");
    precondition(index.j < interiour_y_points().size(), "index out of range");
  };

  double dx = calc_h(interiour_x_points(), index.i);
  double dy = calc_h(interiour_y_points(), index.j);

  if (index.i == 0) {  // Dirichlet at x = a
    return m_input_p->u1(interiour_y_points()[index.j]);
  }
  else if (index.j == 0) {  // Dirichlet at y = c
    return m_input_p->u3(interiour_x_points()[index.i]);
  }
  else if (index.j == interiour_y_points().size() - 1) {  // Dirichlet at y = d
    return m_input_p->u4(interiour_x_points()[index.i]);
  }
  else if (index.i == interiour_x_points().size() - 1) {  // Robin at x = b
    return (2.0 / dx) * (m_input_p->f(interiour_x_points()[index.i], interiour_y_points()[index.j])
                    + m_input_p->u2(interiour_y_points()[index.j]));
  }
  else {  // Interior points
    return dx * dy * m_input_p->f(interiour_x_points()[index.i], interiour_y_points()[index.j]);
  }
}

// auto DefaultMainMatrixCalculator::calc_a(Index index) const -> double
// {
//   // clang-format off
//   contract(fun) {
//     precondition(index.i != 0, "index out of range");
//     precondition(index.i < m_x_points.size(), "index out of range");
//     precondition(index.j < m_y_points.size(), "index out of range");
//   };
//   // clang-format on

//   if(index.i == 0 and index.j < m_y_points.size() - 1) {                          // i == 0
//     return 0;
//   }
//   else if(index.j == 0 and index.i < m_x_points.size() - 1) {                     // j == 0
//     return 0;
//   }
//   else if(index.i == m_x_points.size() - 1 and index.j < m_y_points.size() - 1) {  // i == Nx
//     return -1;
//   }
//   else if(index.i < m_x_points.size() - 1 and index.j == m_y_points.size() - 1) {  // j == Nx
//     return 0;
//   }
```

```
//   else /*if(index.i == m_x_points.size() - 1 and index.j == m_y_points.size() - 1)*/ {
//     return 1;
//   }
// }

// auto DefaultMainMatrixCalculator::calc_b(Index index) const -> double
// {
//   // clang-format off
//   contract(fun) {
//     precondition(index.i != m_x_points.size() - 1, "index out of range");
//     precondition(index.i < m_x_points.size(), "index out of range");
//     precondition(index.j < m_y_points.size(), "index out of range");
//   };
//   // clang-format on

//   auto sq = [](auto x) { return x * x; };

//   if(index.i == 0 and index.j == 0) {                          // i == 0 and j == 0
//     return 0;                                                  // Not too sure
//   }
//   else if(index.i == 0 and index.j < m_y_points.size() - 1) {  // i == 0
//     return 0;
//   }
//   else if(index.j == 0 and index.i < m_x_points.size() - 1) {  // j == 0
//     return 0;
//   }
//   else if(index.i == m_x_points.size() - 1 and index.j < m_y_points.size() - 1) {  // i == Nx
//     return -2 * sq(calc_h(m_y_points, index.j)) / sq(calc_h(m_x_points, index.i))
//          * m_input_p->k1(middle_point(m_x_points, index.i));
//   }
//   else if(index.i < m_x_points.size() - 1 and index.j == m_y_points.size() - 1) {  // j == Nx
//     return 0;
//   }
//   else /*if(index.i == m_x_points.size() - 1 and index.j == m_y_points.size() - 1)*/ {
//     return sq(calc_h(m_y_points, index.j)) / sq(calc_h(m_x_points, index.i))
//          * m_input_p->k1(middle_point(m_x_points, index.i));
//   }
// }

// auto DefaultMainMatrixCalculator::calc_c(Index index) const -> double
// {
//   // clang-format off
//   contract(fun) {
//     precondition(index.i < m_x_points.size(), "index out of range");
//     precondition(index.j < m_y_points.size(), "index out of range");
//   };
//   // clang-format on

//   auto sq = [](auto x) { return x * x; };

//   if(index.i == 0 and index.j == 0) {                          // i == 0 and j == 0
//     return 1;                                                  // Not too sure
//   }
//   else if(index.i == 0 and index.j < m_y_points.size() - 1) {  // i == 0
//     return 1;
//   }
//   else if(index.j == 0 and index.i < m_x_points.size() - 1) {  // j == 0
//     return 1;
//   }
//   else if(index.i == m_x_points.size() - 1 and index.j < m_y_points.size() - 1) {  // i == Nx
//     return 2
//          + sq(calc_h(m_y_points, index.j)) / sq(calc_h(m_x_points, index.i))
//            * m_input_p->k1(middle_point(m_x_points, index.i))
//          + 2 * sq(calc_h(m_y_points, index.j)) * m_input_p->hi2;
//   }
```

```
//   else if(index.i < m_x_points.size() - 1 and index.j == m_y_points.size() - 1) {  // j == Nx
//     return 1;
//   }
//   else /*if(index.i == m_x_points.size() - 1 and index.j == m_y_points.size() - 1)*/ {
//     return 2
//          + sq(calc_h(m_y_points, index.j)) / sq(calc_h(m_x_points, index.i))
//              * m_input_p->k1(middle_point(m_x_points, index.i))
//          + sq(calc_h(m_y_points, index.j)) / sq(calc_h(m_x_points, index.i))
//              * m_input_p->k1(middle_point(m_x_points, index.i));
//   }
// }

// auto DefaultMainMatrixCalculator::calc_g(Index index) const -> double
// {
//   // clang-format off
//   contract(fun) {
//     precondition(index.i < m_x_points.size(), "index out of range");
//     precondition(index.j < m_y_points.size(), "index out of range");
//   };
//   // clang-format on

//   auto sq = [](auto x) { return x * x; };

//   if(index.i == 0 and index.j == 0) {                        // i == 0 and j == 0
//     return m_input_p->u1(m_y_points[index.j]);               // Not too sure
//   }
//   else if(index.i == 0 and index.j < m_y_points.size() - 1) {  // i == 0
//     return m_input_p->u1(m_y_points[index.j]);
//   }
//   else if(index.j == 0 and index.i < m_x_points.size() - 1) {  // j == 0
//     return m_input_p->u3(m_x_points[index.i]);
//   }
//   else if(index.i == m_x_points.size() - 1 and index.j < m_y_points.size() - 1) {  // i == Nx
//     return 2 * sq(calc_h(m_y_points, index.j))
//            * m_input_p->f(m_x_points[index.i], m_y_points[index.j])
//          + 2 * sq(calc_h(m_y_points, index.j)) * m_input_p->u2(m_y_points[index.j]);
//   }
//   else if(index.i < m_x_points.size() - 1 and index.j == m_y_points.size() - 1) {  // j == Nx
//     return m_input_p->u4(m_x_points[index.i]);
//   }
//   else /*if(index.i == m_x_points.size() - 1 and index.j == m_y_points.size() - 1)*/ {
//     return sq(calc_h(m_y_points, index.j)) * m_input_p->f(m_x_points[index.i], m_y_points[index.j]);
//   }
// }

// auto DefaultMainMatrixCalculator::calc_d(Index index) const -> double
// {
//   // clang-format off
//   contract(fun) {
//     precondition(index.i < m_x_points.size(), "index out of range");
//     precondition(index.j < m_y_points.size(), "index out of range");
//   };
//   // clang-format on

//   auto sq = [](auto x) { return x * x; };

//   if(index.i == 0 and index.j == 0) {                        // i == 0 and j == 0
//     return 0;                                                // Not too sure
//   }
//   else if(index.i == 0 and index.j < m_y_points.size() - 1) {  // i == 0
//     return 0;
//   }
//   else if(index.j == 0 and index.i < m_x_points.size() - 1) {  // j == 0
//     return 0;
//   }
```

```
//   else if(index.i == m_x_points.size() - 1 and index.j < m_y_points.size() - 1) {  // i == Nx
//     return 0;
//   }
//   else if(index.i < m_x_points.size() - 1 and index.j == m_y_points.size() - 1) {  // j == Nx
//     return 0;
//   }
//   else /*if(index.i == m_x_points.size() - 1 and index.j == m_y_points.size() - 1)*/ {
//     return -sq(calc_h(m_y_points, index.j)) / sq(calc_h(m_x_points, index.i))
//           * m_input_p->k1(middle_point(m_x_points, index.i + 1));
//   }
// }

// auto DefaultMainMatrixCalculator::calc_e(Index index) const -> double
// {
//   // clang-format off
//   contract(fun) {
//     precondition(index.i < m_x_points.size(), "index out of range");
//     precondition(index.j < m_y_points.size(), "index out of range");
//   };
//   // clang-format on

//   auto sq = [](auto x) { return x * x; };

//   if(index.i == 0 and index.j == 0) {                          // i == 0 and j == 0
//     return 0;                                                  // Not too sure
//   }
//   else if(index.i == 0 and index.j < m_y_points.size() - 1) {  // i == 0
//     return 0;
//   }
//   else if(index.j == 0 and index.i < m_x_points.size() - 1) {  // j == 0
//     return 0;
//   }
//   else if(index.i == m_x_points.size() - 1 and index.j < m_y_points.size() - 1) {  // i == Nx
//     return -1;
//   }
//   else if(index.i < m_x_points.size() - 1 and index.j == m_y_points.size() - 1) {  // j == Nx
//     return 0;
//   }
//   else /*if(index.i == m_x_points.size() - 1 and index.j == m_y_points.size() - 1)*/ {
//     return -1;
//   }
// }


====================================
File: ./src/interval_splitter.cc
====================================
#include <interval_splitter.hpp>

auto split_interval(double const& left, double const& right, size_t num_intervals) -> std::vector<double>
{
  std::vector<double> intervals;

  contract(fun)
  {
    precondition(num_intervals > 0, "invalid number of intervals");
  };

  auto interval_size = (right - left) / num_intervals;
  for(size_t i = 0; i < num_intervals; ++i) {
    intervals.push_back(left + interval_size * i);
  }
  intervals.push_back(right);
  return intervals;
}
```

```
// Calculate the length of an interval `index-1` to `index`
auto calc_h(std::span<const double> points, size_t index) -> double
{
  contract(fun)
  {
    precondition(index < points.size(), "index out of range");
  };

  if(index == 0) {
    return points[1] - points[0];
  }

  return (points[index] - points[index - 1]);
}

// Calculate the cross h of an interval
auto calc_cross_h(std::span<const double> points, size_t index) -> double
{
  if(index == 0) {
    return calc_h(points, 1) / 2;
  }
  else if(index == points.size() - 1) {
    return calc_h(points, index) / 2;
  }
  else {
    return (calc_h(points, index) + calc_h(points, index + 1)) / 2;
  }
}

/// @return middle point between `index` and `index - 1`
auto middle_point(std::span<const double> points, size_t index) -> double
{
  return (points[index] + points[index - 1]) / 2;
}

====================================
File: ./include/public/interface/i_main_matrix_calculator.hpp
====================================
#pragma once

#include <cstdio>
#include <vector>
#include <span>

struct Index
{
  size_t i = -1;
  size_t j = -1;
};

class IMainMatrixCalculator
{
 public:
  virtual auto calc_a(Index index) const -> double = 0;
  virtual auto calc_b(Index index) const -> double = 0;
  virtual auto calc_c(Index index) const -> double = 0;
  virtual auto calc_d(Index index) const -> double = 0;
  virtual auto calc_e(Index index) const -> double = 0;
  virtual auto calc_g(Index index) const -> double = 0;

  virtual auto x_points() const -> std::vector<double> const& = 0;
  virtual auto y_points() const -> std::vector<double> const& = 0;

  virtual auto interiour_x_points() const -> std::span<const double> = 0;
```

```
  virtual auto interiour_y_points() const -> std::span<const double> = 0;
};


===================================
File: ./include/public/default_impl/odd_even_reduction.hpp
===================================
#pragma once

#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <vector>

// Function to solve a tridiagonal system using Odd-Even Reduction
Eigen::VectorXd odd_even_reduction_solver(
  Eigen::SparseMatrix<double> const& main_matrix,
  Eigen::VectorXd const& b
);

Eigen::VectorXd odd_even_reduction_solver(
  Eigen::VectorXd const& a,
  Eigen::VectorXd const& b,
  Eigen::VectorXd const& c,
  Eigen::VectorXd const& rhs
);

Eigen::VectorXd odd_even_reduction_solver(
  Eigen::VectorXd const& a,
  Eigen::VectorXd const& b,
  Eigen::VectorXd const& c,
  Eigen::VectorXd const& d,
  Eigen::VectorXd const& e,
  Eigen::VectorXd const& rhs
);


===================================
File: ./include/public/default_impl/main_matrix_calculator.hpp
===================================
#pragma once

#include <vector>
#include <memory>

#include <Eigen/Dense>

#include <interface/i_main_matrix_calculator.hpp>
#include <input_parameters.hpp>

class DefaultMainMatrixCalculator : public IMainMatrixCalculator
{
 public:
  explicit DefaultMainMatrixCalculator(
    std::shared_ptr<InputParameters> params,
    std::vector<double> x_points,
    std::vector<double> y_points
  )
    : m_input_p(std::move(params))
    , m_x_points(std::move(x_points))
    , m_y_points(std::move(y_points))
  {}

  auto calc_a(Index index) const -> double override;
  auto calc_b(Index index) const -> double override;
  auto calc_c(Index index) const -> double override;
```

```
  auto calc_g(Index index) const -> double override;
  auto calc_d(Index index) const -> double override;
  auto calc_e(Index index) const -> double override;

  auto params() const -> std::shared_ptr<InputParameters> const& { return m_input_p; }

  auto x_points() const -> std::vector<double> const& override { return m_x_points; }

  auto y_points() const -> std::vector<double> const& override { return m_y_points; }

  auto interiour_x_points() const -> std::span<double const> override
  {
    return {m_x_points.data() + 1, m_x_points.size() - 2};
  }

  auto interiour_y_points() const -> std::span<double const> override
  {
    return {m_y_points.data() + 1, m_y_points.size() - 2};
  }

 protected:
  std::shared_ptr<InputParameters> m_input_p;

  std::vector<double> m_x_points;
  std::vector<double> m_y_points;
};


====================================
File: ./include/public/input_parameters.hpp
====================================
#pragma once

#include <defines.hpp>

struct InputParameters {
  double xl;
  double xr;
  double yl;
  double yr;

  // First type condition
  Y_Function_type u1;

  // Third type condition
  double hi2;
  X_Function_type k1;
  Y_Function_type u2;

  // First type condition
  X_Function_type u3;

  // First type condition
  X_Function_type u4;

  // Just input functions
  X_Y_Function_type f;
};


====================================
File: ./include/public/interval_splitter.hpp
====================================
#pragma once
```

```
#include <vector>
#include <cstdio>
#include <span>

#include <contract/contract.hpp>

#include <defines.hpp>

auto split_interval(const double& left, const double& right, size_t num_intervals) -> std::vector<double>;

// Calculate the length of an interval `index-1` to `index`
auto calc_h(std::span<const double> intervals, size_t index) -> double;

// Calculate the cross h of an interval
auto calc_cross_h(std::span<const double> intervals, size_t index) -> double;

/// @return middle point between `index` and `index - 1`
auto middle_point(std::span<const double> intervals, size_t index) -> double;


====================================
File: ./include/public/defines.hpp
====================================
#pragma once

#include <functional>

// First argument is X, second is Y
using X_Y_Function_type = std::function<double(double, double)>;

using X_Function_type = std::function<double(double)>;
using Y_Function_type = std::function<double(double)>;


====================================
File: ./include/public/utils.hpp
====================================
#pragma once

#include <memory>

#include <defines.hpp>
#include <input_parameters.hpp>

void do_all(std::shared_ptr<InputParameters> params, X_Y_Function_type expected_func);
void do_all1(std::shared_ptr<InputParameters> params, X_Y_Function_type expected_func);
```