



МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



**НГТУ
НЭТИ** | **Факультет прикладной
математики и информатики**

Кафедра прикладной математики

Практическое задание № 2

по дисциплине «Численные методы»

ИТЕРАЦИОННЫЕ МЕТОДЫ РЕШЕНИЯ СЛАУ

Группа ПМ-21

Место для ввода
текста

ПОРСИН ДАНИЛ

Преподаватели

ЛЕОНОВИЧ Д.А.

ЗАДОРОВИЧ А.Г.

Новосибирск, 2025

1. Цель работы

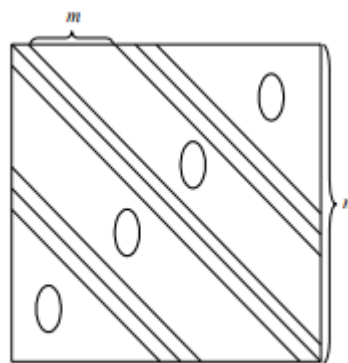
Разработать программы решения СЛАУ методами Якоби, Гаусса-Зейделя, блочной релаксации с хранением матрицы в диагональном формате. Исследовать сходимость методов для различных тестовых матриц и её зависимость от параметра релаксации. Изучить возможность оценки порядка числа обусловленности матрицы путем вычислительного эксперимента.

2. Теория

Дана СЛАУ

$$Ax = F$$

Матрица коэффициентов A – квадратная 9-ти диагональная матрица размерности n с 3 блоками ненулевых диагоналей по 3 диагонали в каждом блоке, один из которых состоит из главной диагонали и диагоналей сверху и снизу прилегающих к главной, а оставшиеся находятся выше и ниже упомянутого блока на расстоянии m .



Решение данной СЛАУ производится исходя из начального приближения в виде нулевого вектора следующими методами:

- Метод Якоби с параметром релаксации

Итерационный процесс имеет следующий вид

$$x_i^{(k+1)} = x_i^k + \frac{\omega}{a_{ii}} \left(f_i - \sum_{j=1}^n a_{ij} x_j^{(k)} \right), 0 < \omega < 1$$

- Метод Гаусса-Зейделя с параметром релаксации

Итерационный процесс имеет следующий вид

$$x_i^{(k+1)} = x_i^k + \frac{\omega}{a_{ii}} \left(f_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^n a_{ij} x_j^{(k)} \right), 0 < \omega < 2$$

- Метод блочной релаксации

Матрица A разбивается на квадратные блоки равной размерности, векторы x, f разбиваются на блок-векторы X, F размерности, равной размерности блока матрицы A .

Итерационный процесс для блоков матрицы A и блок-векторов X, F представляет собой решение следующего СЛАУ для каждого блока блок-вектора X

$$A_{ii}Y_i^{(k)} = \omega R_i^{(k)}, 0 < \omega < 2$$

где

$$R_i^{(k)} = \left(F_i - \sum_{j=1}^{i-1} A_{ij}X_j^{(k+1)} - \sum_{j=i}^n A_{ij}X_j^{(k)} \right)$$

$$Y_i^{(k)} = X_i^{(k+1)} - X_i^{(k)}$$

Решение данных СЛАУ производится с предварительной LDU -факторизацией блока A_{ii} 1 раз перед первой итерацией.

Условия выхода из итерационного процесса для данных методов следующие:

- Выход по относительной невязке

$$\frac{\|F - Ax^{(k)}\|}{\|F\|} < \epsilon$$

где ϵ – заданная точность. Для данной работы была выбрана точность $\epsilon = 1e - 9$

- Аварийный выход при достижении заданного максимального количества итераций

Для данной работы было выбрано максимальное количество итераций, равное $1e + 6$

Оценка числа обусловленности получаемого решения производится по формуле

$$V_A \leq \frac{\frac{\|x-x^*\|}{\|x^*\|}}{\frac{\|F-Ax\|}{\|F\|}}$$

где x^* – аналитическое решение СЛАУ.

3. Входные и выходные данные

Входные данные представляют собой 4 текстовых файла:

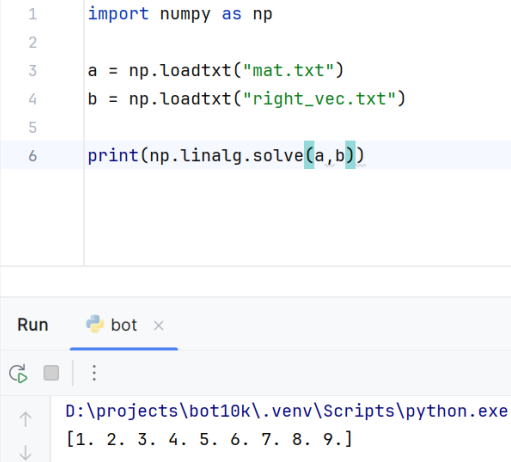
- *sys_mat.txt* – матрица коэффициентов. Элементы строки разделены пробелом, строки разделены знаком перехода на новую строку
- *sys_params.txt* – параметры СЛАУ. Для методов Якоби с параметром релаксации и Гаусса-Зейделя с параметром релаксации – 4 числа, разделенные пробелом: размерность матрицы коэффициентов, расстояние m между блоками ненулевых диагоналей матрицы коэффициентов, точность ϵ , значение максимального числа итераций. Для метода блочной релаксации – 3 числа: размерность матрицы коэффициентов, точность ϵ , значение максимального числа итераций.
- *sys_sol_right_vec.txt* – вектор правой части СЛАУ. Элементы вектора разделены либо пробелом, либо знаком перехода на новую строку.
- *sys_sol_begin_aprox.txt* – начальное приближение вектора решений СЛАУ. Элементы вектора разделены либо пробелом, либо знаком перехода на новую строку

Выходные данные представляют собой 1 текстовый файл:

- *sys_sol.txt* – вектор решений СЛАУ. Элементы вектора разделены знаком перехода на новую строку.

4. Тестирование

Метод Якоби с параметром релаксации

Входные данные				Результат	Ожидаемый результат
sys_mat.txt	sys_right_vec.txt	sys_params.txt	sys_sol_begin_aprox.txt		
1 1 0 0 1 1 1 0 0	21	9 2 1e-10	0	1.000000152727802	 <pre>1 import numpy as np 2 3 a = np.loadtxt("mat.txt") 4 b = np.loadtxt("right_vec.txt") 5 6 print(np.linalg.solve(a,b))</pre> <p>Run bot x</p> <p>D:\projects\bot10k\.venv\Scripts\python.exe</p> <p>[1. 2. 3. 4. 5. 6. 7. 8. 9.]</p>
1 2 1 0 0 1 1 1 0	29	1000000	0	1.9999999902917909	
0 1 3 1 0 0 1 1 1	39		0	3.0000000037893773	
0 0 1 4 1 0 0 1 1	41		0	3.999999995702647	
1 0 0 1 5 1 0 0 1	45		0	4.9999999969384756	
1 1 0 0 1 6 1 0 0	51		0	5.999999998313864	
1 1 1 0 0 1 7 1 0	69		0	6.9999999984562589	
0 1 1 1 0 0 1 8 1	89		0	8.0000000010614018	
0 0 1 1 1 0 0 1 9	101		0	8.9999999998406949	

Метод Гаусса-Зейделя с параметром релаксации

Входные данные				Результат	Ожидаемый результат
sys_mat.txt	sys_right_vec.txt	sys_params.txt	sys_sol_begin_aprox.txt		
1 1 0 0 1 1 1 0 0	21	9 2 1e-10	0	1.000000172321535	<pre>1 import numpy as np 2 3 a = np.loadtxt("mat.txt") 4 b = np.loadtxt("right_vec.txt") 5 6 print(np.linalg.solve(a,b))</pre>
1 2 1 0 0 1 1 1 0	29	1000000	0	1.999999892904444	
0 1 3 1 0 0 1 1 1	39		0	3.000000041107514	
0 0 1 4 1 0 0 1 1	41		0	3.999999995820144	
1 0 0 1 5 1 0 0 1	45		0	4.999999965999559	
1 1 0 0 1 6 1 0 0	51		0	5.999999997676412	
1 1 1 0 0 1 7 1 0	69		0	6.999999982829772	
0 1 1 1 0 0 1 8 1	89		0	8.000000011539521	
0 0 1 1 1 0 0 1 9	101		0	8.999999998333871	

Метод Гаусса-Зейделя с параметром релаксации

Входные данные				Результат	Ожидаемый результат
sys_mat.txt	sys_right_vec.txt	sys_params.txt	sys_sol_begin_aprox.txt		
1 1 0 0 1 1 1 0 0	21	9 2 1e-10	0	1.000000172321535	<pre>1 import numpy as np 2 3 a = np.loadtxt("mat.txt") 4 b = np.loadtxt("right_vec.txt") 5 6 print(np.linalg.solve(a,b))</pre>
1 2 1 0 0 1 1 1 0	29	1000000	0	1.999999892904444	
0 1 3 1 0 0 1 1 1	39		0	3.000000041107514	
0 0 1 4 1 0 0 1 1	41		0	3.999999995820144	
1 0 0 1 5 1 0 0 1	45		0	4.999999965999559	
1 1 0 0 1 6 1 0 0	51		0	5.999999997676412	
1 1 1 0 0 1 7 1 0	69		0	6.999999982829772	
0 1 1 1 0 0 1 8 1	89		0	8.000000011539521	
0 0 1 1 1 0 0 1 9	101		0	8.999999998333871	

Метод Блочной релаксации

Входные данные					Результат	Ожидаемый результат
sys_mat.txt	sys_right_vec.txt	sys_params.txt	sys_sol_begin_aprox.txt	Раз- мер- ность блока		
1 1 0 0 1 1 1 0 0 1 2 1 0 0 1 1 1 0 0 1 3 1 0 0 1 1 1 0 0 1 4 1 0 0 1 1 1 0 0 1 5 1 0 0 1 1 1 0 0 1 6 1 0 0 1 1 1 0 0 1 7 1 0 0 1 1 1 0 0 1 8 1 0 0 1 1 1 0 0 1 9	21 29 39 41 45 51 69 89 101	9 1e-10 1000000	0 0 0 0 0 0 0 0 0	3	1.0000000001073905 1.9999999993958783 3.0000000000655850 3.9999999997366018 4.999999999487611 6.0000000000836425 7.000000000109208 8.0000000000834586 8.999999999512267	<pre> 1 import numpy as np 2 3 a = np.loadtxt("mat.txt") 4 b = np.loadtxt("right_vec.txt") 5 6 print(np.linalg.solve(a,b)) </pre> <div> Run bot × </div> <div> D:\projects\bot10k\.env\Scripts\python.exe [1. 2. 3. 4. 5. 6. 7. 8. 9.] </div>

5. Исследования

Цель исследования заключается в поиске оптимального веса ω для каждого использованного метода, т.е. значения ω , при котором данный метод сходится до заданной точности за наименьшее количество итераций, и оценке числа обусловленности полученного решения.

Исследования произведены на СЛАУ, построенных путём умножения составленной заданным образом матрицы коэффициентов A размерности n на вектор $x = (1, 2, \dots, n)^T$

- Матрица коэффициентов с диагональным преобладанием

Для данного исследования была составлена следующая матрица размерностью $n = 12$

$$A = \begin{pmatrix} 11 & -1 & 0 & 0 & 0 & -2 & -3 & -4 & 0 & 0 & 0 & 0 \\ -4 & 14 & -2 & 0 & 0 & 0 & -3 & -4 & -1 & 0 & 0 & 0 \\ 0 & -3 & 13 & -3 & 0 & 0 & 0 & -4 & -1 & -2 & 0 & 0 \\ 0 & 0 & -2 & 12 & -4 & 0 & 0 & 0 & -1 & -2 & -3 & 0 \\ 0 & 0 & 0 & -1 & 11 & -1 & 0 & 0 & 0 & -2 & -3 & -4 \\ -3 & 0 & 0 & 0 & -4 & 16 & -2 & 0 & 0 & 0 & -3 & -4 \\ -2 & -2 & 0 & 0 & 0 & -3 & 14 & -3 & 0 & 0 & 0 & -4 \\ -1 & -1 & -1 & 0 & 0 & 0 & -2 & 9 & -4 & 0 & 0 & 0 \\ 0 & -4 & -4 & -4 & 0 & 0 & 0 & -1 & 14 & -1 & 0 & 0 \\ 0 & 0 & -3 & -3 & -3 & 0 & 0 & 0 & -4 & 15 & -2 & 0 \\ 0 & 0 & 0 & -2 & -2 & -2 & 0 & 0 & 0 & -3 & 12 & -3 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & -2 & 5 \end{pmatrix}$$

- Метод Якоби с параметром релаксации

Результаты вычислений

ω	x	$x^* - x$	(количество итераций)
1.17	9.99999935525640e-01	6.44743597399966e-08	3244
	1.99999993060611e+00	6.93938861928700e-08	
	2.99999992843957e+00	7.15604264733827e-08	
	3.99999992726179e+00	7.27382123422160e-08	
	4.99999992728652e+00	7.27134841227439e-08	
	5.99999992888633e+00	7.11136740605411e-08	
	6.99999992950323e+00	7.04967693110348e-08	
	7.99999992933095e+00	7.06690537199961e-08	
	8.99999992836922e+00	7.16307759773827e-08	
	9.99999992745858e+00	7.25414182056738e-08	
	1.09999999273505e+01	7.26494953084966e-08	
	1.19999999277259e+01	7.22741422265472e-08	
1.18	9.99999935735522e-01	6.42644779613732e-08	3217
	1.99999993083201e+00	6.91679900022280e-08	
	2.99999992867252e+00	7.13274768138206e-08	
	3.99999992749857e+00	7.25014284164160e-08	
	4.99999992752322e+00	7.24767792448233e-08	
	5.99999992911782e+00	7.08821792372305e-08	
	6.99999992973272e+00	7.02672808827742e-08	
	7.99999992956099e+00	7.04390057393312e-08	
	8.99999992860240e+00	7.13975971677883e-08	
	9.99999992769473e+00	7.23052746565145e-08	
	1.09999999275870e+01	7.24130000406831e-08	
	1.19999999279611e+01	7.20388708685960e-08	
1.19	9.99999935776727e-01	6.42232732550596e-08	3190
	1.99999993087636e+00	6.91236421435093e-08	
	2.99999992871825e+00	7.12817453951686e-08	
	3.99999992754506e+00	7.24549447106426e-08	
	4.99999992756969e+00	7.24303141907967e-08	

	5.99999992916327e+00 6.99999992977777e+00 7.99999992960616e+00 8.99999992864818e+00 9.99999992774108e+00 1.09999999276334e+01 1.19999999280073e+01	7.08367329238513e-08 7.02222298087918e-08 7.03938427548678e-08 7.13518204520369e-08 7.22589170720767e-08 7.23665749546853e-08 7.19926838144147e-08	
1.2	9.99999935649577e-01 1.9999993073951e+00 2.99999992857713e+00 3.99999992740161e+00 4.99999992742629e+00 5.99999992902302e+00 6.99999992963874e+00 7.99999992946679e+00 8.99999992850691e+00 9.99999992759803e+00 1.09999999274902e+01 1.19999999278648e+01	6.43504234343339e-08 6.92604944507735e-08 7.14228693965424e-08 7.25983930749408e-08 7.25737123730141e-08 7.09769762963219e-08 7.03612554886490e-08 7.05332121597735e-08 7.14930870060471e-08 7.24019741937809e-08 7.25098452392103e-08 7.21352169108513e-08	3163
1.21	9.99999935731381e-01 1.99999993082755e+00 2.99999992866793e+00 3.99999992749390e+00 4.99999992751855e+00 5.99999992911325e+00 6.99999992972819e+00 7.99999992955645e+00 8.99999992859780e+00 9.99999992769007e+00 1.09999999275823e+01 1.19999999279565e+01	6.42686192042774e-08 6.91724473256272e-08 7.13320749134994e-08 7.25061006789929e-08 7.24814528396678e-08 7.08867462506646e-08 7.02718105927147e-08 7.04435461074127e-08 7.14021979320023e-08 7.23099340405042e-08 7.24176683064570e-08 7.20435142653741e-08	3137
1.22	9.99999935650971e-01 1.99999993074101e+00 2.99999992857868e+00 3.99999992740318e+00	6.43490288831927e-08 6.92589914308428e-08 7.14213208574677e-08 7.25968156700674e-08	3111

	4.99999992742786e+00 5.99999992902456e+00 6.99999992964027e+00 7.99999992946832e+00 8.99999992850847e+00 9.99999992759960e+00 1.09999999274917e+01 1.19999999278663e+01	7.25721367444976e-08 7.09754353067638e-08 7.03597304863024e-08 7.05316791638211e-08 7.14915326938126e-08 7.24004038943349e-08 7.25082731634075e-08 7.21336519404758e-08	
1.23	9.99999935791995e-01 1.99999993089279e+00 2.99999992873520e+00 3.99999992756228e+00 4.99999992758691e+00 5.99999992918011e+00 6.99999992979447e+00 7.99999992962289e+00 8.99999992866514e+00 9.99999992775826e+00 1.09999999276506e+01 1.19999999280244e+01	6.42080054680250e-08 6.91072088443434e-08 7.12647967304747e-08 7.24377198224602e-08 7.24130915230603e-08 7.08198921728354e-08 7.02055329426798e-08 7.03771112497975e-08 7.13348562442206e-08 7.22417397014397e-08 7.23493691623389e-08 7.19755686162671e-08	3086
1.24	9.99999935777115e-01 1.99999993087678e+00 2.99999992871868e+00 3.99999992754550e+00 4.99999992757012e+00 5.99999992916370e+00 6.99999992977819e+00 7.99999992960658e+00 8.99999992864861e+00 9.99999992774152e+00 1.09999999276339e+01 1.19999999280078e+01	6.42228854541571e-08 6.91232238114736e-08 7.12813164049919e-08 7.24545037300572e-08 7.24298754306574e-08 7.08363039336746e-08 7.02218079240424e-08 7.03934164292264e-08 7.13513905736818e-08 7.22584765355805e-08 7.23661379709029e-08 7.19922432779185e-08	3061
1.25	9.99999935606232e-01 1.99999993069285e+00 2.99999992852902e+00	6.43937682065499e-08 6.93071455781791e-08 7.14709766924670e-08	3036

	3.99999992735271e+00 4.99999992737741e+00 5.99999992897522e+00 6.99999992959135e+00 7.99999992941928e+00 8.99999992845876e+00 9.99999992754926e+00 1.09999999274413e+01 1.19999999278162e+01	7.26472935141942e-08 7.26225941605207e-08 7.10247842761191e-08 7.04086486891242e-08 7.05807199352648e-08 7.15412387108927e-08 7.24507422944498e-08 7.25586843941528e-08 7.21838038231226e-08	
1.26	9.99999935672545e-01 1.99999993076409e+00 2.99999992860282e+00 3.99999992742725e+00 4.99999992745229e+00 5.99999992904822e+00 6.99999992966404e+00 7.99999992949173e+00 8.99999992853246e+00 9.99999992762367e+00 1.09999999275162e+01 1.19999999278902e+01	6.43274552514228e-08 6.92359125586961e-08 7.13971775034850e-08 7.25727509198748e-08 7.25477091734206e-08 7.09517848918040e-08 7.03359566145423e-08 7.05082694452130e-08 7.14675447710533e-08 7.23763271537337e-08 7.24837985188742e-08 7.21097705991269e-08	3012
При последующих значениях ω метод расходится			

Вывод

Как видно по полученным результатам, оптимальным весом для метода Якоби с параметром релаксации является $\omega = 1.26$, при котором данный метод на данной СЛАУ сходится за 3012 итераций.

Число обусловленностей

Для данного метода на данной СЛАУ имеем следующую оценку числа обусловленностей

$$V_A \leq 9.6599324951206e + 01$$

- Метод Гаусса-Зейделя с параметром релаксации

Результаты вычислений

В качестве результатов представлены последние 10 строк итоговой таблицы

ω	x	$x^* - x$	(количество итераций)
1.66	9.99999984168308e-01	1.58316924014912e-08	351
	1.99999998305670e+00	1.69432994301388e-08	
	2.99999998257432e+00	1.74256831186881e-08	
	3.99999998246618e+00	1.75338246144463e-08	
	4.99999998272111e+00	1.72788876540153e-08	
	5.99999998319207e+00	1.68079274942556e-08	
	6.99999998329506e+00	1.67049369892425e-08	
	7.99999998314746e+00	1.68525371435635e-08	
	8.99999998309655e+00	1.69034546360081e-08	
	9.99999998308075e+00	1.69192517773809e-08	
1.67	1.09999999832051e+01	1.67948606133450e-08	335
	1.19999999835256e+01	1.64744218267288e-08	
	9.99999984282741e-01	1.57172586057186e-08	
	1.99999998318389e+00	1.68161127245270e-08	
	2.99999998270749e+00	1.72925109787059e-08	
	3.99999998260917e+00	1.73908336620343e-08	
	4.99999998287433e+00	1.71256724357249e-08	
	5.99999998334550e+00	1.66544982249661e-08	
	6.99999998344526e+00	1.65547362485086e-08	
	7.99999998329370e+00	1.67063003431167e-08	
1.68	8.99999998325222e+00	1.67477782753167e-08	320
	9.99999998324642e+00	1.67535816331110e-08	
	1.09999999833770e+01	1.66230051945604e-08	
	1.19999999837056e+01	1.62943596393461e-08	
	9.99999985347098e-01	1.46529022249098e-08	
	1.99999998432750e+00	1.56724988542578e-08	

	2.99999998388593e+00 3.99999998380352e+00 4.99999998406305e+00 5.99999998450602e+00 6.99999998459649e+00 7.99999998445001e+00 8.99999998442071e+00 9.99999998442538e+00 1.09999999845543e+01 1.19999999848714e+01	1.61140696341988e-08 1.61964770484246e-08 1.59369513141883e-08 1.54939812091470e-08 1.54035069144243e-08 1.55499879639365e-08 1.55792907463592e-08 1.55746242569421e-08 1.54456838430406e-08 1.51285792782119e-08	
1.69	9.99999985435046e-01 1.99999998442493e+00 2.99999998398997e+00 3.99999998391957e+00 4.99999998419558e+00 5.99999998463699e+00 6.99999998472257e+00 7.99999998457594e+00 8.99999998454774e+00 9.99999998456487e+00 1.09999999847040e+01 1.19999999850317e+01	1.45649542426796e-08 1.55750741193117e-08 1.60100337431857e-08 1.60804267679282e-08 1.58044191067575e-08 1.53630104193780e-08 1.52774317641047e-08 1.54240593630561e-08 1.54522581397032e-08 1.54351340597714e-08 1.52959902521843e-08 1.49683252459454e-08	304
1.7	9.99999993295047e-01 1.99999999302316e+00 2.99999999281508e+00 3.99999999291540e+00 4.99999999323990e+00 5.99999999361827e+00 6.99999999286360e+00 7.99999999335034e+00 8.99999999294151e+00 9.99999999287208e+00 1.09999999931854e+01 1.19999999935236e+01	6.70495325927334e-09 6.97684465755799e-09 7.18492021434258e-09 7.08459513276694e-09 6.76009559441582e-09 6.38173425215882e-09 7.13640435634488e-09 6.64965593699662e-09 7.05848890447669e-09 7.12792491697201e-09 6.81455958329025e-09 6.47638387363259e-09	299

1.71	9.99999999755759e-01 2.00000000013046e+00 3.00000000000715e+00 4.00000000004078e+00 4.9999999985600e+00 6.00000000032304e+00 6.9999999955765e+00 7.9999999977808e+00 9.00000000016093e+00 9.9999999994054e+00 1.0999999999421e+01 1.20000000000891e+01	2.44241293856362e-10 - 1.30460531266863e-10 - 7.15427717068451e-12 - 4.07824884973707e-11 1.43996814472303e-10 - 3.23038484850713e-10 4.42346603790611e-10 2.21919371767854e-10 - 1.60927271508626e-10 5.94617688420840e-11 5.78808112550178e-11 - 8.90896245664408e-11	339
1.72	9.9999999828433e-01 2.00000000020328e+00 3.00000000008343e+00 4.00000000012651e+00 4.9999999994875e+00 6.00000000040803e+00 6.9999999963106e+00 7.9999999986701e+00 9.00000000022727e+00 1.00000000000098e+01 1.10000000000182e+01 1.20000000001709e+01	1.71566871820517e-10 - 2.03275174470718e-10 - 8.34319280329510e-11 - 1.26508581388407e-10 5.12496711735366e-11 - 4.08030054188657e-10 3.68936881045556e-10 1.32987842960119e-10 - 2.27267094032868e-10 - 9.83035874924099e-12 - 1.82041048901738e-11 - 1.70906844232377e-10	404
1.73	1.00000000013152e+00 1.99999999968395e+00 2.9999999984519e+00 3.9999999983347e+00 5.00000000015878e+00 5.9999999959927e+00 7.00000000035436e+00 8.00000000024748e+00 8.9999999958792e+00 9.9999999985338e+00	-1.31519239943145e-10 3.16047410464648e-10 1.54805945840053e-10 1.66525904177206e-10 - 1.58778767911372e-10 4.00728339400303e-10 - 3.54360096821438e-10 - 2.47482034865243e-10 4.12082812317749e-10 1.46618717167257e-10	649

	1.0999999999439e+01 1.19999999998083e+01	5.60866908472235e-11 1.91681337469163e-10	
1.74	9.9999999908166e-01 2.00000000036230e+00 3.00000000018685e+00 4.00000000017871e+00 4.9999999978391e+00 6.00000000035747e+00 6.9999999968680e+00 7.9999999969949e+00 9.00000000049866e+00 1.00000000002266e+01 1.10000000000769e+01 1.20000000001890e+01	9.18344289502215e-11 - 3.62297303269088e-10 - 1.86854087758093e-10 - 1.78710379827862e-10 2.16091144977781e-10 - 3.57472274004067e-10 3.13199244317275e-10 3.00511615591859e-10 - 4.98655339242760e-10 - 2.26640040068560e-10 - 7.69215802165490e-11 - 1.89041671205814e-10	1219
1.75	1.00000000000639e+00 1.9999999958003e+00 2.9999999976564e+00 3.9999999981914e+00 5.00000000030983e+00 5.9999999976268e+00 7.00000000020093e+00 8.00000000037504e+00 8.9999999937771e+00 9.9999999963750e+00 1.0999999998885e+01 1.1999999998394e+01	-6.39333030960643e-12 4.19966283971007e-10 2.34357422357334e-10 1.80864656584845e-10 - 3.09829495392933e-10 2.37315056494936e-10 - 2.00925498461402e-10 - 3.75038666788896e-10 6.22286222551338e-10 3.62497587502730e-10 1.11462838958687e-10 1.60593316422819e-10	6606
При последующих значениях ω метод расходится			

Вывод

Как видно по полученным результатам, оптимальным весом для метода Гаусса-Зейделя с параметром релаксации является $\omega = 1.7$, при котором данный метод на данной СЛАУ сходится за 299 итераций.

Число обусловленностей

Для данного метода на данной СЛАУ имеем следующую оценку числа обусловленностей

$$V_A \leq 1.0966799872586e + 01$$

- Метод блочной релаксации

Результаты вычислений

В качестве результатов представлены последние 10 строк итоговой таблицы

ω	x	$x^* - x$	(количество итераций)
1.47	9.99999972452783e-01	2.75472165078838e-08	611
	1.99999997042136e+00	2.95786404258536e-08	
	2.99999996958801e+00	3.04119907035272e-08	
	3.99999996915999e+00	3.08400101012296e-08	
	4.99999996945732e+00	3.05426839375400e-08	
	5.99999997018499e+00	2.98150091282423e-08	
	6.99999997044506e+00	2.95549371642778e-08	
	7.99999997020460e+00	2.97953963723785e-08	
	8.99999997006054e+00	2.99394571356970e-08	
	9.99999996984555e+00	3.01544513803265e-08	
1.48	1.09999999699695e+01	3.00304989764300e-08	598
	1.19999999701937e+01	2.98062641235219e-08	
	9.99999972393435e-01	2.76065650339774e-08	
	1.99999997035816e+00	2.96418414258426e-08	
	2.99999996952485e+00	3.04751530677549e-08	

	3.99999996909834e+00 4.99999996940237e+00 5.99999997013163e+00 6.99999997039215e+00 7.99999997014912e+00 8.99999997001094e+00 9.99999996979789e+00 1.09999999699247e+01 1.19999999701509e+01	3.09016550126273e-08 3.05976310954748e-08 2.98683691113411e-08 2.96078459527394e-08 2.98508808782572e-08 2.99890601240804e-08 3.02021092579707e-08 3.00752667214965e-08 2.98491116268451e-08	
1.49	9.99999973111286e-01 1.99999997112946e+00 2.99999997031966e+00 3.99999996990669e+00 4.99999997020890e+00 5.99999997091922e+00 6.99999997117286e+00 7.99999997093407e+00 8.99999997080571e+00 9.99999997060062e+00 1.09999999707268e+01 1.19999999709485e+01	2.68887144772023e-08 2.88705415130153e-08 2.96803435162474e-08 3.00933069574683e-08 2.97910975888271e-08 2.90807751213151e-08 2.88271397863582e-08 2.90659292190298e-08 2.91942896524233e-08 2.93993842603868e-08 2.92732025286568e-08 2.90514847733903e-08	586
1.5	9.99999973698606e-01 1.99999997176060e+00 2.99999997097034e+00 3.99999997056887e+00 4.99999997087063e+00 5.99999997156545e+00 6.99999997181344e+00 7.99999997157777e+00 8.99999997145850e+00 9.99999997126032e+00 1.09999999713864e+01 1.19999999716047e+01	2.63013940626777e-08 2.82394021500210e-08 2.90296582328153e-08 2.94311286452853e-08 2.91293726917274e-08 2.84345462731039e-08 2.81865615292531e-08 2.84222272384227e-08 2.85415016065826e-08 2.87396790810135e-08 2.86135968252665e-08 2.83952665824927e-08	574
1.51	9.99999974158465e-01 1.99999997225488e+00	2.58415349119190e-08 2.77451210894952e-08	562

	2.99999997148033e+00 3.99999997108839e+00 4.99999997139110e+00 5.99999997207380e+00 6.99999997231734e+00 7.99999997208367e+00 8.99999997197285e+00 9.99999997178061e+00 1.09999999719072e+01 1.19999999721232e+01	2.85196741778293e-08 2.89116148799451e-08 2.86088983614263e-08 2.79261973545886e-08 2.76826614964421e-08 2.79163261396320e-08 2.80271486019501e-08 2.82193912681805e-08 2.80928187379459e-08 2.78768297334864e-08	
1.52	9.99999974492262e-01 1.99999997261381e+00 2.99999997185119e+00 3.99999997146687e+00 4.99999997177204e+00 5.99999997244594e+00 6.99999997268622e+00 7.99999997245341e+00 8.99999997235052e+00 9.99999997216327e+00 1.09999999722910e+01 1.19999999725057e+01	2.55077380328927e-08 2.73861862076785e-08 2.81488099496130e-08 2.85331260840849e-08 2.82279630781090e-08 2.75540559258047e-08 2.73137796824585e-08 2.75465916743656e-08 2.76494844797526e-08 2.78367302541938e-08 2.77090457245777e-08 2.74943428024699e-08	550
1.53	9.99999975628312e-01 1.99999997383369e+00 2.99999997310679e+00 3.99999997274015e+00 4.99999997303639e+00 5.99999997368270e+00 6.99999997391265e+00 7.99999997368377e+00 8.99999997359846e+00 9.99999997341985e+00 1.09999999735433e+01 1.19999999737502e+01	2.43716875569788e-08 2.61663075651342e-08 2.68932121016974e-08 2.72598525974388e-08 2.69636126759565e-08 2.63173047798659e-08 2.60873509461135e-08 2.63162309721565e-08 2.64015351802982e-08 2.65801478604999e-08 2.64567301400120e-08 2.62497739100809e-08	539

1.54	9.99999999433338e-01 1.99999999932422e+00 2.99999999928192e+00 3.99999999892368e+00 4.99999999864165e+00 5.99999999908548e+00 6.99999999917896e+00 7.99999999840512e+00 8.99999999958362e+00 9.99999999918480e+00 1.09999999989571e+01 1.19999999990232e+01	5.66661939593871e-10 6.75779654457642e-10 7.18075821026787e-10 1.07631548118547e-09 1.35834810066626e-09 9.14518238914752e-10 8.21043677490252e-10 1.59487534290292e-09 4.16379819512258e-10 8.15203904380724e-10 1.04285113877722e-09 9.76768887994695e-10	611
1.55	1.00000000041920e+00 2.00000000038130e+00 3.00000000036897e+00 4.00000000002200e+00 4.99999999972220e+00 6.00000000014816e+00 7.00000000023397e+00 7.99999999945628e+00 9.00000000065974e+00 1.00000000002587e+01 1.10000000000220e+01 1.20000000000824e+01	-4.19195123058103e-10 - 3.81301212826202e-10 - 3.68974628628393e-10 - 2.19975149207130e-11 2.77797340686448e-10 - 1.48164147617535e-10 - 2.33966623852666e-10 5.43721512258344e-10 - 6.59738930153253e-10 - 2.58690846521858e-10 - 2.19930740286145e-11 - 8.239098292506	1055
1.56	9.99999999578929e-01 1.99999999961767e+00 2.99999999962915e+00 3.9999999997765e+00 5.00000000028144e+00 5.99999999984956e+00 6.99999999976263e+00 8.00000000054933e+00 8.99999999933047e+00 9.99999999973875e+00	4.21071399969719e-10 3.82328391168585e-10 3.70846908737121e-10 2.23483453964946e-11 - 2.81437095850379e-10 1.50437884371968e-10 2.37367459021698e-10 - 5.49327694443491e-10 6.69531985408867e-10 2.61252353084274e-10	3790

	1.0999999999781e+01	2.18545181951413e-11	
	1.1999999999153e+01	8.46895886752463e-11	
При последующих значениях ω метод расходится			

Вывод

Как видно по полученным результатам, оптимальным весом для метода блочной релаксации является $\omega = 1.53$, при котором данный метод на данной СЛАУ сходится за 539 итераций.

Число обусловленностей

Для данного метода на данной СЛАУ имеем следующую оценку числа обусловленностей

$$V_A \leq 3.7145001663956e + 01$$

- Матрица коэффициентов с диагональным преобладанием A с обратным знаком внедиагональных элементов

- Метод Якоби с параметром релаксации

Результаты вычислений

В качестве результатов представлены 10 строк итоговой таблицы, включая содержащие оптимальное значение параметра релаксации

ω	x	$x^* - x$	(количество итераций)
0.75	1.00000000069624e+00, 1.99999999867470e+00, 3.00000000366971e+00, 3.99999999677082e+00, 5.00000000219197e+00, 5.99999999863858e+00, 7.00000000347008e+00, 7.99999999729915e+00, 9.00000000102897e+00, 9.99999999788603e+00, 1.10000000031880e+01, 1.19999999963179e+01	-6.96238622310830e-10, 1.32530053598146e-09, - 3.66971475429523e-09, 3.22918314310527e-09, - 2.19197371364999e-09, 1.36141942164159e-09, - 3.47008111134528e-09, 2.70084576925456e-09, - 1.02896535736363e-09, 2.11397477301034e-09, - 3.18799386889168e-09, 3.68205554934775e-09	49
0.76	1.00000000075648e+00, 1.99999999855970e+00, 3.00000000398768e+00, 3.99999999649078e+00, 5.00000000238185e+00, 5.99999999852044e+00, 7.00000000377075e+00, 7.99999999706492e+00, 9.00000000111803e+00, 9.99999999770265e+00, 1.10000000034642e+01, 1.19999999959986e+01	-7.56476881136336e-10, 1.44030276594265e-09, - 3.98768484899392e-09, 3.50922402247988e-09, - 2.38184760803506e-09, 1.47955603324590e-09, - 3.77074815816059e-09, 2.93508417570365e-09, - 1.11803188929116e-09, 2.29734631318479e-09, - 3.46420847563422e-09, 4.00135213851627e-09	48

0.77	1.00000000083207e+00, 1.9999999841974e+00, 3.00000000438063e+00, 3.99999999614779e+00, 5.00000000261717e+00, 5.9999999837667e+00, 7.00000000414237e+00, 7.99999999677825e+00, 9.00000000122927e+00, 9.99999999747863e+00, 1.10000000038058e+01, 1.19999999956073e+01	-8.32070412570829e-10, 1.58025903473913e-09, - 4.38062830454555e-09, 3.85221143872627e-09, - 2.61717314486987e-09, 1.62332725039960e-09, - 4.14237355528257e-09, 3.22175086608922e-09, - 1.22926735457440e-09, 2.52136977962891e-09, - 3.80578413228250e-09, 4.39265868124039e-09	47
0.78	1.00000000062678e+00, 1.9999999882711e+00, 3.00000000327541e+00, 3.99999999713205e+00, 5.00000000195960e+00, 5.9999999879513e+00, 7.00000000309752e+00, 7.99999999760232e+00, 9.00000000092382e+00, 9.99999999812512e+00, 1.10000000028465e+01, 1.19999999967287e+01	-6.26776630596737e-10, 1.17289178369617e-09, - 3.27540750078015e-09, 2.86795387438588e-09, - 1.95959870552542e-09, 1.20487442245576e-09, - 3.09752135052577e-09, 2.39768294107989e-09, - 9.23822796039531e-10, 1.87487714242707e-09, - 2.84654966264952e-09, 3.27128546473432e-09	47
0.79	1.00000000062950e+00, 1.99999999861793e+00, 3.00000000357461e+00, 3.99999999672467e+00, 5.00000000210649e+00, 5.99999999858043e+00, 7.00000000337744e+00, 7.99999999725116e+00, 9.00000000095296e+00, 9.99999999783197e+00,	-6.29500229720747e-10, 1.38206712740896e-09, - 3.57460860911374e-09, 3.27532534427633e-09, - 2.10649453435963e-09, 1.41956935095777e-09, - 3.37744143763530e-09, 2.74884470741199e-09, - 9.52963929989892e-10, 2.16802575891961e-09, -	46

	1.10000000030953e+01, 1.19999999962755e+01	3.09534442521908e-09, 3.72446784524527e-09	
0.80	1.00000000022188e+00, 1.99999999869788e+00, 3.00000000239794e+00, 3.99999999726848e+00, 5.00000000129509e+00, 5.99999999866318e+00, 7.00000000225454e+00, 7.99999999767037e+00, 9.00000000043624e+00, 9.99999999809769e+00, 1.10000000020352e+01, 1.19999999969342e+01	-2.21879403738967e-10, 1.30212041149491e-09, - 2.39793962464319e-09, 2.73151634644364e-09, - 1.29508848090154e-09, 1.33682132030799e-09, - 2.25453522517682e-09, 2.32962715784879e-09, - 4.36237712619914e-10, 1.90230942109793e-09, - 2.03516314911667e-09, 3.06575742570203e-09	46
0.81	9.99999999658612e-01, 1.99999999913169e+00, 3.00000000032048e+00, 3.99999999864563e+00, 4.9999999995461e+00, 5.99999999910938e+00, 7.00000000028069e+00, 7.99999999878953e+00, 8.99999999968239e+00, 9.99999999891642e+00, 1.10000000001956e+01, 1.19999999985400e+01	3.41387917934810e-10, 8.68306981871569e-10, - 3.20481419180396e-10, 1.35437350223810e-09, 4.53894699603552e-11, 8.90618245819041e-10, - 2.80691025977831e-10, 1.21047083467829e-09, 3.17610826527925e-10, 1.08358300110467e-09, - 1.95576888017968e-10, 1.46003920065141e-09	48
0.82	9.99999999637864e-01, 1.99999999954216e+00, 2.99999999969311e+00, 3.99999999945972e+00, 4.99999999963774e+00, 5.99999999953071e+00, 6.99999999969291e+00, 7.99999999948942e+00, 8.99999999960605e+00,	3.62135987863610e-10, 4.57839322010045e-10, 3.06886516199256e-10, 5.40275824079117e-10, 3.62256002972572e-10, 4.69291272509054e-10, 3.07085912254479e-10, 5.10584463597752e-10, 3.93949761701151e-10,	52

	9.99999999949749e+00, 1.09999999996708e+01, 1.19999999994476e+01	5.02510033584258e-10, 3.29194449477654e-10, 5.52359935568347e-10	
0.83	1.00000000054608e+00, 2.00000000057428e+00, 3.00000000062490e+00, 4.00000000059002e+00, 5.00000000062499e+00, 6.00000000058849e+00, 7.00000000061467e+00, 8.00000000057607e+00, 9.00000000060835e+00, 1.0000000005956e+01, 1.1000000006309e+01, 1.2000000005832e+01	-5.46081624364092e-10, - 5.74283287591015e-10, - 6.24899243462096e-10, - 5.90017812385213e-10, - 6.24993390374584e-10, - 5.88494586395427e-10, - 6.14669204423990e-10, - 5.76067193946983e-10, - 6.08352479503083e-10, - 5.95601790109868e-10, - 6.30921093147663e-10, - 5.83179726731942e-10	55
0.84	1.00000000058796e+00, 2.00000000063117e+00, 3.00000000065489e+00, 4.00000000066012e+00, 5.00000000066422e+00, 6.00000000064681e+00, 7.00000000064504e+00, 8.00000000064169e+00, 9.00000000065342e+00, 1.0000000006592e+01, 1.1000000006644e+01, 1.2000000006555e+01	-5.87960125031373e-10, - 6.31168006748339e-10, - 6.54889475981690e-10, - 6.60118182338465e-10, - 6.64217125745381e-10, - 6.46807052362419e-10, - 6.45038689128796e-10, - 6.41687591951268e-10, - 6.53423981589185e-10, - 6.59216681242469e-10, - 6.64428512209270e-10, - 6.55530740800714e-10	59
При последующих значениях ω метод расходится			

Вывод

Как видно по полученным результатам, оптимальными весами для метода Якоби с параметром релаксации является $\omega = 0.79$ и $\omega = 0.80$, при которых данный метод на данной СЛАУ сходится за 46 итераций.

Число обусловленностей

Для данного метода на данной СЛАУ имеем следующую оценку числа обусловленностей

$$V_A \leq 4.5913200292886e + 00$$

- Метод Гаусса-Зейделя с параметром релаксации

Результаты вычислений

В качестве результатов представлены 10 строк итоговой таблицы, включая содержащую оптимальное значение параметра релаксации

ω	x	$x^* - x$	(количество итераций)
0.99	1.00000000142416e+00, 1.99999999725288e+00, 3.00000000488848e+00, 3.99999999705849e+00, 5.00000000143044e+00, 5.99999999824097e+00, 7.00000000282509e+00, 7.99999999810043e+00, 9.00000000069939e+00, 9.99999999839870e+00, 1.10000000018826e+01, 1.19999999987241e+01	-1.42415679249552e-09, 2.74711986492093e-09, - 4.88848383994878e-09, 2.94150792612413e-09, - 1.43043532574438e-09, 1.75903291932400e-09, - 2.82509038385115e-09, 1.89956761431631e-09, - 6.99394320236024e-10, 1.60130220194787e-09, - 1.88261140010582e-09, 1.27591981424757e-09	21
1.00	1.00000000071163e+00, 1.99999999849776e+00, 3.00000000252014e+00, 3.99999999847905e+00, 5.00000000074619e+00, 5.99999999908041e+00, 7.00000000144168e+00, 7.9999999903963e+00, 9.00000000038631e+00, 9.9999999917343e+00, 1.10000000009585e+01, 1.1999999993630e+01	-7.11630088190418e-10, 1.50223899986202e-09, - 2.52014187296368e-09, 1.52095269712049e-09, - 7.46190664813184e-10, 9.19589737691240e-10, - 1.44167877635937e-09, 9.60370449831771e-10, - 3.86311427291730e-10, 8.26569035439206e-10, - 9.58488399760427e-10, 6.37049524243594e-10	21
1.01	1.00000000095822e+00, 1.99999999752246e+00, 3.00000000386669e+00,	-9.58222168279121e-10, 2.47753573212606e-09, - 3.86668830287817e-09,	20

	3.99999999776449e+00, 5.00000000115013e+00, 5.99999999857956e+00, 7.00000000219971e+00, 7.99999999875062e+00, 9.00000000059963e+00, 9.99999999873394e+00, 1.10000000014224e+01, 1.19999999990643e+01	2.23550955524843e-09, - 1.15012710466544e-09, 1.42044243034434e-09, - 2.19970885950715e-09, 1.24937749035325e-09, - 5.99628791064788e-10, 1.26605748107522e-09, - 1.42243372636131e-09, 9.35722610506673e-10	
1.02	1.00000000016857e+00, 1.99999999850066e+00, 3.000000000162867e+00, 3.99999999891080e+00, 5.00000000061058e+00, 5.99999999932750e+00, 7.00000000095675e+00, 7.99999999946938e+00, 9.00000000045761e+00, 9.99999999938537e+00, 1.10000000006841e+01, 1.19999999995678e+01	-1.68566494096467e-10, 1.49933887527709e-09, - 1.62866875541567e-09, 1.08919673280639e-09, - 6.10582695514950e-10, 6.72496724973826e-10, - 9.56752899128333e-10, 5.30624433281446e-10, - 4.57609061754738e-10, 6.14631900930362e-10, - 6.84101664205627e-10, 4.32216040735511e-10	20
1.03	9.99999999133207e-01, 1.99999999658416e+00, 3.000000000263558e+00, 3.99999999874544e+00, 5.000000000115026e+00, 5.99999999888916e+00, 7.000000000169660e+00, 8.00000000054918e+00, 9.00000000087872e+00, 9.99999999886926e+00, 1.10000000010557e+01, 1.19999999992844e+01	8.66793414822098e-10, 3.41583650254051e-09, - 2.63557842217210e-09, 1.25456134369983e-09, - 1.15025677871472e-09, 1.11084208498369e-09, - 1.69659664095434e-09, - 5.49176704112142e-10, - 8.78719319530319e-10, 1.13073639340655e-09, - 1.05565511887562e-09, 7.15573378329282e-10	19
1.04	1.00000000010959e+00, 1.99999999980415e+00,	-1.09585895913256e-10, 1.95845339945322e-10,	20

	2.9999999982573e+00, 3.9999999958186e+00, 5.0000000009054e+00, 5.9999999993889e+00, 6.9999999988528e+00, 7.9999999931899e+00, 9.00000000026136e+00, 9.9999999998800e+00, 1.10000000001178e+01, 1.1999999999789e+01	1.74272152264621e-10, 4.18139745050894e-10, - 9.05382435689717e-11, 6.11111161674671e-11, 1.14718012866888e-10, 6.81013467840330e-10, - 2.61360710851477e-10, 1.20046195206669e-11, - 1.17752918527003e-10, 2.11386463888630e-11	
1.05	1.00000000088113e+00, 2.00000000061366e+00, 3.0000000000689e+00, 3.9999999956307e+00, 4.9999999988849e+00, 5.9999999997420e+00, 6.9999999986453e+00, 7.9999999887453e+00, 8.9999999991728e+00, 1.00000000001132e+01, 1.10000000000367e+01, 1.20000000000367e+01	-8.81126283047706e-10, - 6.13664230542099e-10, - 6.89448498292222e-12, 4.36934932679378e-10, 1.11512576950190e-10, 2.57953658433507e-11, 1.35469413464762e-10, 1.12547038355615e-09, 8.27160562266727e-11, - 1.13184128736066e-10, - 3.66835450904546e-11, - 3.67279540114396e-11	20
1.06	1.00000000202478e+00, 2.00000000153600e+00, 3.00000000091231e+00, 3.9999999954474e+00, 4.9999999969037e+00, 5.9999999987651e+00, 7.00000000020451e+00, 7.99999999861696e+00, 8.99999999932098e+00, 1.00000000001218e+01, 1.10000000000170e+01, 1.20000000000165e+01	-2.02477834498893e-09, - 1.53599755137179e-09, - 9.12309783274168e-10, 4.55258497567002e-10, 3.09628767070080e-10, 1.23490551118266e-10, - 2.04505745671213e-10, 1.38303857255551e-09, 6.79017730931264e-10, - 1.21756826843011e-10, - 1.69766423141482e-11, - 1.65094604653859e-11	20

1.07	9.99999999736655e-01, 1.9999999974878e+00, 3.00000000041195e+00, 4.00000000023225e+00, 5.00000000007155e+00, 5.9999999995100e+00, 7.00000000027977e+00, 8.00000000069791e+00, 8.9999999992980e+00, 9.9999999986438e+00, 1.10000000000128e+01, 1.1999999999362e+01	2.63345456552599e-10, 2.51216825120082e-10, - 4.11953582357683e-10, - 2.32246222253707e-10, - 7.15525416694618e-11, 4.89999152364362e-11, - 2.79769984956602e-10, - 6.97911062275125e-10, 7.01980695794191e-11, 1.35619515617691e-10, - 1.28324018078274e-11, 6.37978558870600e-11	21
1.08	9.99999998808813e-01, 1.9999999906910e+00, 2.9999999988115e+00, 4.00000000039199e+00, 5.00000000021828e+00, 6.0000000003162e+00, 7.00000000012186e+00, 8.000000000118613e+00, 9.00000000030867e+00, 9.9999999982780e+00, 1.10000000000035e+01, 1.1999999999413e+01	1.19118659380746e-09, 9.30897137152442e-10, 1.18849374786123e-10, - 3.91987775572034e-10, - 2.18276063890244e-10, - 3.16209280981639e-11, - 1.21861631896536e-10, - 1.18612675237273e-09, - 3.08670422555224e-10, 1.72203584725139e-10, - 3.45146133895469e-12, 5.86730664053903e-11	21
При последующих значениях ω метод расходится			

Вывод

Как видно по полученным результатам, оптимальным весом для метода Гаусса-Зейделя с параметром релаксации является $\omega = 1.03$, при котором данный метод на данной СЛАУ сходится за 19 итераций.

Число обусловленностей

Для данного метода на данной СЛАУ имеем следующую оценку числа обусловленностей

$$V_A \leq 2.5165715193487e + 00$$

- Метод блочной релаксации

Результаты вычислений

В качестве результатов представлены 15 строк итоговой таблицы, включая содержащие оптимальное значение параметра релаксации

ω	x	$x^* - x$	(количество итераций)
0.96	1.00000000020409e+00	-2.04091188393818e-10	17
	1.99999999944089e+00	5.59107649067414e-10 -	
	3.000000000214270e+00	2.14269579856818e-09	
	3.99999999735614e+00	2.64386468273869e-09 -	
	5.00000000098102e+00	9.81018821732960e-10	
	5.9999999980906e+00	1.90941484845553e-10 -	
	7.00000000136388e+00	1.36387612315048e-09	
	7.99999999863329e+00	1.36671030048774e-09 -	
	9.00000000047577e+00	4.75765205010248e-10	
	9.9999999967823e+00	3.21769277888961e-10 -	
	1.10000000006253e+01	6.25300700107800e-10	
	1.1999999992601e+01	7.39866834464920e-	
0.97	9.99999999248693e-01	7.51307127622169e-10	16
	1.99999999856577e+00	1.43422718146269e-09 -	
	3.00000000260717e+00	2.60717092359641e-09	
	3.99999999564525e+00	4.35475078219838e-09 -	
	5.00000000147427e+00	1.47426515439975e-09	
	5.9999999987590e+00	1.24095400622082e-10 -	
	7.00000000247109e+00	2.47108822293285e-09	
	7.99999999758643e+00	2.41357245300833e-09 -	
	9.00000000119682e+00	1.19681864418908e-09	
	9.9999999959904e+00	4.00961042146264e-10 -	
	1.10000000009792e+01	9.79229142217264e-10	
	1.1999999987720e+01	1.22801679935947e-09	
0.98	9.9999999909011e-01	9.09887720723646e-11	16
	1.9999999953303e+00	4.66967797763118e-10 -	
	3.00000000131711e+00	1.31711264117484e-09	

	3.99999999841670e+00 5.00000000056430e+00 6.00000000001429e+00 7.000000000116309e+00 7.99999999799404e+00 9.00000000043378e+00 9.9999999979891e+00 1.10000000003600e+01 1.1999999994845e+01	1.58330015764818e-09 - 5.64299718064376e-10 - 1.42872380592962e-11 - 1.16308562780887e-09 2.00595895449851e-09 - 4.33779234754184e-10 2.01092476004305e-10 - 3.59982266218140e-10 5.15532505573901e-10	
0.99	1.00000000114215e+00 2.00000000031824e+00 3.000000000173536e+00 3.9999999928012e+00 5.00000000035987e+00 5.9999999983532e+00 7.00000000040613e+00 7.99999999870634e+00 8.9999999974855e+00 9.9999999977296e+00 1.10000000001959e+01 1.1999999997941e+01	-1.14214615543062e-09 - 3.18235215956975e-10 - 1.73535719127926e-09 7.19881487754037e-10 - 3.59868579380418e-10 1.64683378045538e-10 - 4.06134681441017e-10 1.29365762546740e-09 2.51446863330784e-10 2.27041496714264e-10 - 1.95910843103775e-10 2.05876204972810e-	16
1	1.00000000167326e+00 2.00000000063495e+00 3.000000000186409e+00 3.9999999949415e+00 5.00000000028352e+00 5.9999999968670e+00 6.9999999991374e+00 7.9999999989174e+00 8.9999999945270e+00 9.9999999979774e+00 1.10000000001489e+01 1.1999999999636e+01	-1.67326175137816e-09 - 6.34946317745744e-10 - 1.86409021551981e-09 5.05852693066799e-10 - 2.83522538779835e-10 3.13304049370799e-10 8.62563354075974e-11 1.08260955755668e-10 5.47300871289735e-10 2.02261318804631e-10 - 1.48927981058478e-10 3.63620245025231e-1	16
1.01	1.00000000093421e+00 2.00000000036535e+00	-9.34214927639232e-10 - 3.65349972497597e-10 -	16

	3.00000000086680e+00 3.99999999969270e+00 5.00000000010186e+00 5.99999999976853e+00 6.99999999969171e+00 8.00000000102016e+00 8.99999999968949e+00 9.9999999995623e+00 1.10000000000661e+01 1.20000000000610e+01	8.66798188781104e-10 3.07304848234935e-10 - 1.01856301171210e-10 2.31469954314889e-10 3.08285841299494e-10 - 1.02015995651072e-09 3.10505399170324e-10 4.37694325228222e-11 - 6.60875798530469e-11 - 6.09894357239682e	
1.02	9.99999999209284e-01 1.99999999967491e+00 2.99999999900585e+00 4.00000000006939e+00 4.99999999983361e+00 6.00000000008448e+00 6.99999999979876e+00 8.00000000134056e+00 9.00000000030460e+00 1.00000000001733e+01 1.0999999999314e+01 1.20000000000757e+01	7.90715715126566e-10 3.25094173803109e-10 9.94154092381905e-10 - 6.93853863253935e-11 1.66386016076103e-10 - 8.44790903897774e-11 2.01243466335654e-10 - 1.34055966327651e-09 - 3.04599012679319e-10 - 1.73342229459195e-10 6.86064538513165e-11 - 7.57367502046691e-11	16
1.03	9.99999997544414e-01 1.99999999900680e+00 2.99999999739460e+00 4.00000000050055e+00 4.99999999965307e+00 6.00000000044394e+00 7.00000000018549e+00 8.00000000037743e+00 9.00000000086016e+00 1.00000000002829e+01 1.0999999998213e+01 1.1999999999989e+01	2.45558640088461e-09 9.93195969911653e-10 2.60539501084622e-09 - 5.00549823811980e-10 3.46925155270128e-10 - 4.43943548589232e-10 - 1.85493398419112e-10 - 3.77426090381050e-10 - 8.60161719629104e-10 - 2.82851075894541e-10 1.78651760052162e-10 1.13864473405556e-12	16

1.04	1.00000000043994e+00 2.00000000017918e+00 3.00000000052274e+00 3.9999999993141e+00 5.00000000008860e+00 5.9999999992785e+00 7.0000000004331e+00 7.99999999955006e+00 8.99999999981886e+00 9.9999999991714e+00 1.1000000000412e+01 1.1999999999787e+01	-4.39942970942298e-10 - 1.79178893944254e-10 - 5.22737408914509e-10 6.85917989073914e-11 - 8.85993500787663e-11 7.21467330322412e-11 - 4.33129088150963e-11 4.49943193814306e-10 1.81138659627322e-10 8.28563884169853e-11 - 4.11546352552250e-11 2.12594386539422e-	17
1.05	9.9999999903482e-01 1.9999999997568e+00 3.00000000043653e+00 4.00000000022751e+00 5.00000000027421e+00 6.0000000007018e+00 7.00000000072886e+00 7.99999999636807e+00 8.9999999972741e+00 9.9999999967186e+00 1.10000000000905e+01 1.19999999997874e+01	9.65182378465101e-11 2.43189912652042e-11 - 4.36526814695526e-10 - 2.27514895811964e-10 - 2.74205547157180e-10 - 7.01767532973463e-11 - 7.28862303844835e-10 3.63192942387514e-09 2.72585509719647e-10 3.28141069871890e-10 - 9.04663011169760e-11 2.12551753975276e-10	16
1.06	1.00000000120242e+00 2.00000000048024e+00 3.00000000110334e+00 3.9999999967509e+00 5.0000000006695e+00 5.9999999976010e+00 6.9999999966774e+00 8.00000000101470e+00 8.9999999968085e+00 9.9999999998341e+00	-1.20241705481305e-09 - 4.80235406996599e-10 - 1.10333875369406e-09 3.24908544513391e-10 - 6.69473365633166e-11 2.39895214804164e-10 3.32256888668780e-10 - 1.01470298830009e-09 3.19150927907685e-10 1.65858438094801e-11 -	17

	1.10000000000570e+01 1.20000000000790e+01	5.69784219806024e-11 - 7.90230103575595e	
1.07	9.9999999870905e-01 1.9999999993762e+00 2.9999999953045e+00 3.9999999992022e+00 4.9999999977128e+00 6.0000000003497e+00 6.9999999961591e+00 8.00000000212644e+00 9.00000000030238e+00 1.0000000002442e+01 1.0999999999121e+01 1.2000000001159e+01	1.29094956946574e-10 6.23807672184284e-11 4.69545735626298e-10 7.97824029064031e-11 2.28721042105917e-10 - 3.49702489188530e-11 3.84094533956159e-10 - 2.12643769259557e-09 - 3.02380342986908e-10 - 2.44162023932404e-10 8.79047945545608e-11 - 1.15864651206721e-10	17
1.08	9.9999999269660e-01 1.9999999970989e+00 2.9999999935618e+00 4.00000000020803e+00 4.9999999998760e+00 6.00000000013903e+00 7.00000000022809e+00 7.9999999922979e+00 9.00000000015866e+00 9.9999999998141e+00 1.0999999999749e+01 1.1999999999411e+01	7.30339566601401e-10 2.90108159717306e-10 6.43819664247758e-10 - 2.08025596748485e-10 1.23963062037546e-11 - 1.39031008927759e-10 - 2.28094876320029e-10 7.70213226530814e-10 - 1.58662416538391e-10 1.85913506811630e-11 2.50803822154921e-11 5.89110982218699e-11	18
1.09	1.00000000013745e+00 2.00000000006434e+00 3.00000000040745e+00 4.00000000004289e+00 5.00000000020701e+00 5.9999999993801e+00 7.00000000026677e+00 7.99999999841371e+00 8.9999999970610e+00	-1.37453159965162e-10 - 6.43400888122869e-11 - 4.07448741412963e-10 - 4.28892477088993e-11 - 2.07011296993187e-10 6.19939655166490e-11 - 2.66771493784290e-10 1.58629376301178e-09 2.93898239078771e-10	18

	9.99999999979067e+00 1.10000000000839e+01 1.19999999999191e+01	2.09325889954926e-10 - 8.39239788774648e-11 8.09325939599148e	
При последующих значениях ω метод расходится			

Вывод

Как видно по полученным результатам, оптимальным весом для метода Гаусса-Зейделя с параметром релаксации является $\omega \in \{0.97, 0.98, 0.99, 1.00, 1.01, 1.02, 1.03, 1.05\}$, при котором данный метод на данной СЛАУ сходится за 16 итераций.

Число обусловленностей

Для данного метода на данной СЛАУ имеем следующую оценку числа обусловленностей

$$V_A \leq 3.0497200541911e + 00$$

6. Текст программы

```
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <vector>
#include "sys_gen.h"
#include "sys_solve.h"
#include <map>

enum METHOD {
    JACOBI_RELAX_PARAM,
    GAUSS_ZEIDEL_RELAX_PARAM,
};

void input_diag(double**& sys_mat, double*& sys_right_vec, double*& sys_sol_begin_aprox,
    int& n, int& m, double& epsilon, int& max_iter,
    std::string sys_params_file, std::string sys_mat_file, std::string sys_right_vec_file,
    std::string sys_sol_begin_aprox_file) {

    std::ifstream in(sys_params_file);

    in >> n >> m >> epsilon >> max_iter;

    sys_mat = new double* [n];

    in.close();

    in.open(sys_mat_file);

    while (!in.eof()) {

        int first_col = 5;
        int shift = 0;
        int skip_elem;
        int source_mat_row_elem;
        for (int i = 0; i < n; i++) {

            sys_mat[i] = new double[9] {};

            source_mat_row_elem = 0;

            if (i == m + 2)
                shift = 0;

            if (i > 1 && i < m + 2 || i > m + 4)
                shift++;

            if (shift == 0)
                first_col--;
        }
    }
}
```

```

        for (int skipped_elems = 0; skipped_el-
ems < shift; skipped_elems++) {
            in >> skip_elem;
            source_mat_row_elem++;
        }

        for (int j = first_col; source_mat_row_elem < n; j++) {
            int temp; in >> temp;
            sys_mat[i][j] = temp;

            source_mat_row_elem++;

            if (j == 2 or j == 5 or j == 8) {
                for (int skipped_elems = 0; skipped_el-
ems < m and source_mat_row_elem < n; skipped_elems++) {
                    in >> skip_elem;
                    source_mat_row_elem++;
                }
            }
        }
    }

    in.close();
    in.open(sys_right_vec_file);

    sys_right_vec = new double[n];

    for (int i = 0; i < n; i++) {
        in >> sys_right_vec[i];
    }

    in.close();
    in.open(sys_sol_begin_aprox_file);

    sys_sol_begin_aprox = new double[n];

    for (int i = 0; i < n; i++) {
        in >> sys_sol_begin_aprox[i];
    }

    in.close();
}

void input_block(double****& sys_mat, int& n, double**& sys_right_vec, dou-
ble**& sys_sol_begin_aprox,
                double& epsilon, int& max_iter, int block_dim,
                std::string sys_params_file, std::string sys_mat_file, std::string sys_r
            ight_vec_file,
                std::string sys_sol_begin_aprox_file) {

```



```

std::ifstream in(sys_params_file);
in >> n >> epsilon >> max_iter;
in.close();

int block_size = block_dim * block_dim;
int block_mat_dim = n / block_dim;

in.open(sys_mat_file);

sys_mat = new double*** [block_mat_dim];

int block;
while (!in.eof()) {
    for (int block_i = 0; block_i < block_mat_dim; block_i++) {
        sys_mat[block_i] = new double** [block_mat_dim];
        for (int i = 0; i < block_dim; i++) {

for (int block_j = 0; block_j < block_mat_dim; block_j++) {

                if (i == 0)

sys_mat[block_i][block_j] = new double* [block_dim];

                sys_mat[block_i][block_j][i] = new dou-
ble[block_dim];

                for (int j = 0; j < block_dim; j++) {
                    int temp; in >> temp;

sys_mat[block_i][block_j][i][j] = temp;
                }

            }

        }

    }

in.close();
in.open(sys_right_vec_file);

sys_right_vec = new double* [block_mat_dim];

for (int block_i = 0; block_i < block_mat_dim; block_i++) {
    sys_right_vec[block_i] = new double[block_dim];

    for (int i = 0; i < block_dim; i++) {
        in >> sys_right_vec[block_i][i];
    }

}

in.close();
in.open(sys_sol_begin_aprox_file);

sys_sol_begin_aprox = new double* [block_mat_dim];

```

```

        for (int block_i = 0; block_i < block_mat_dim; block_i++) {
            sys_sol_begin_aprox[block_i] = new double[block_dim];

            for (int i = 0; i < block_dim; i++) {
                in >> sys_sol_begin_aprox[block_i][i];
            }
        }

        in.close();
    }

    void output_vec(double* vec, int n, std::string output_file) {
        std::ofstream out(output_file);

        out << std::fixed << std::setprecision(15);

        for (int i = 0; i < n; i++)
            out << vec[i] << std::endl;
    }

    void output_block_vec(double** block_vec, int n, int block_dim, std::string output_file) {
        std::ofstream out(output_file);

        out << std::fixed << std::setprecision(15);

        int block_vec_dim = n / block_dim;

        for (int block_i = 0; block_i < block_vec_dim; block_i++) {
            for (int i = 0; i < block_dim; i++) {
                out << block_vec[block_i][i] << std::endl;
            }
        }
    }

    double* mult_mat_diag_vec(double** mat, double* vec, int n, int m) {

        double* res = new double[n] {};

        for (int i = 0, shift = 0; i < n; i++) {
            // достигнут нижний блок ненулевых диагоналей
            if (i == m + 2)
                shift = 0;

            // строки, включающие значения нулевых диагоналей между стредним и нижним блоком или ниже нижнего блока
            if (i > 1 && i < m + 2 || i > m + 4)
                shift++;

            for (int j = 0, vec_i = 0; j < n; j++) {
                if (mat[i][j] != 0) {

                    res[i] += mat[i][j] * vec[vec_i + shift];
                }
            }
        }
    }

```

```

        vec_i++;

        //переход от одного блока к дру-
romy
        if ((j + 1) % 3 == 0)
            vec_i += m;
    }
}

return res;
}

double* vec_substr(double* vec1, double* vec2, int n) {
    double* res = new double[n];

    for (int i = 0; i < n; i++) {
        res[i] = vec1[i] - vec2[i];
    }

    return res;
}

double* vec_sum(double* vec1_elems, double* vec2_elems, int n) {
    double* res = new double[n];

    for (int i = 0; i < n; i++) {
        res[i] = vec1_elems[i] + vec2_elems[i];
    }

    return res;
}

double vec_norm(double* vec, int n) {
    double sum = 0.;

    for (int i = 0; i < n; i++) {
        sum += vec[i] * vec[i];
    }

    return sqrt(sum);
}

double scal_mult(double* vec1, double* vec2, int n) {
    double res = 0.;

    for (int i = 0; i < n; i++) {
        res += vec1[i] * vec2[i];
    }

    return res;
}

```

```

}

double scal_mult_diag(double* vec1_diag, int i, double* vec2, int n, int m) {
    int k = 9;
    int shift = 0;
    double res = 0;

    if (i > 1 and i < 2 + m)
        shift = i - 1;
    else if (i > m + 4)
        shift = i - (m + 4);

    for (int j = 0, vec_j = 0; j < k; j++) {
        if (vec1_diag[j] != 0) {

            res += vec1_diag[j] * vec2[vec_j + shift];

            vec_j++;

            //переход от одного блока к другому
            if ((j + 1) % 3 == 0)
                vec_j += m;

        }
    }

    return res;
}

double* block_mult_mat_row_vec(double*** mat_row, double** vec, int n, int block_dim) {
    int block_mat_dim = n / block_dim;

    double* res = new double[block_dim] {};

    for (int block_j = 0; block_j < block_mat_dim; block_j++) {
        for (int i = 0; i < block_dim; i++) {

            res[i] += scal_mult(mat_row[block_j][i], vec[block_j], block_dim);
        }
    }

    return res;
}

double* mult_num_vec(double num, double* vec, int n) {
    double* mult = new double[n];
    for (int i = 0; i < n; i++) {
        mult[i] = num * vec[i];
    }

    return mult;
}

```

```

double relative_discrepancy(double** sys_mat, int n, int m, double* sys_right_vec, double* sys_sol){
    return vec_norm(vec_sub-
str(sys_right_vec, mult_mat_diag_vec(sys_mat, sys_sol,n,m),n),n) / vec_norm(sys_
right_vec,n);
}

double* mult_mat_vec(double** mat, double* vec, int n) {
    double* mult = new double[n] {};

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            mult[i] += mat[i][j] * vec[j];
        }
    }

    return mult;
}

void gen_conditional(int n, double* vec, int k,
    double* not_diag_elems, int not_diag_elems_size,
    int rand_seed, std::string sys_mat_file, std::string sys_right_vec_file)
{
    std::srand(rand_seed);

    double** sys_mat = new double* [n];

    for (int i = 0; i < n; i++) {
        sys_mat[i] = new double[n] {};
        for (int j = 0; j < n; j++) {
            if (i != j) {
                sys_mat[i][j] = not_diag_elems[std::rand() % not_diag_elems_size];
                sys_mat[i][i] -= sys_mat[i][j];
            }
        }
    }

    sys_mat[0][0] += 1;

    double* sys_right_vec = mult_mat_vec(sys_mat, vec, n);

    std::ofstream out(sys_mat_file);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            out << sys_mat[i][j] << " ";
        }
        out << std::endl;
    }

    out.close();
}

```

```

        out.open(sys_right_vec_file);

        for (int i = 0; i < n; i++) {
            out << sys_right_vec[i] << std::endl;
        }
    }

double* mult_mat_LDU_vec(double** mat, double* vec, int n) {
    double* inter_res = new double[n] {};

    for (int i = 0; i < n; i++) {
        inter_res[i] += vec[i];
        for (int j = i + 1; j < n; j++) {
            inter_res[i] += mat[i][j] * vec[j];
        }
    }

    for (int i = 0; i < n; i++)
        inter_res[i] *= mat[i][i];

    double* res = new double[n] {};
    for (int i = 0; i < n; i++) {
        res[i] += inter_res[i];
        for (int j = 0; j < i; j++) {
            res[i] += mat[i][j] * inter_res[j];
        }
    }
    return res;
}

double block_relative_discrepancy(double**** sys_mat, int n, int block_dim, double** sys_right_vec, double** sys_sol) {
    int block_mat_dim = n / block_dim;
    double** mult_sys_mat_sys_sol = new double* [block_mat_dim];

    for (int block_i=0; block_i < block_mat_dim; block_i++) {
        mult_sys_mat_sys_sol[block_i] = new double[block_dim] {};
        for (int block_j = 0; block_j < block_mat_dim; block_j++) {
            if (block_i == block_j)

                mult_sys_mat_sys_sol[block_i] = vec_sum(mult_sys_mat_sys_sol[block_i], mult_mat_LDU_vec(sys_mat[block_i][block_i], sys_sol[block_i], block_dim), block_dim);

            else {

                mult_sys_mat_sys_sol[block_i] = vec_sum(mult_sys_mat_sys_sol[block_i], mult_mat_vec(sys_mat[block_i][block_j], sys_sol[block_j], block_dim), block_dim);
            }
        }
    }

    double* substr;

```

```

        double sum1 = 0;
        double sum2 = 0;
        for (int block_i = 0; block_i < block_mat_dim; block_i++) {
            substr = vec_substr(sys_right_vec[block_i], mult_sys_mat_sys_sol[block_i], block_dim);
            sum1 += scal_mult(substr, substr, block_dim);

            sum2 += scal_mult(sys_right_vec[block_i], sys_right_vec[block_i], block_dim);
        }

        return sqrt(sum1) / sqrt(sum2);
    }

void iter_step(METHOD method, double** sys_mat, int n, int m, double* sys_right_vec, double* sys_sol, double relax_param, double epsilon, int max_iter) {

    int sys_mat_main_diag = 4;

    if (method == JACOBI_RELAX_PARAM) {
        double* mult = mult_mat_diag_vec(sys_mat, sys_sol, n, m);

        for (int i = 0; i < n; i++) {
            sys_sol[i] = sys_sol[i] + (relax_param / sys_mat[i][sys_mat_main_diag] * (sys_right_vec[i] - mult[i]));
        }
    }
    else {
        double mult;

        for (int i = 0; i < n; i++) {
            mult = scal_mult_diag(sys_mat[i], i, sys_sol, n, m);

            sys_sol[i] = sys_sol[i] + (relax_param / sys_mat[i][sys_mat_main_diag] * (sys_right_vec[i] - mult));
        }
    }
}

double* solve_sys(METHOD method, double** sys_mat, int n, int m, double* sys_right_vec, double* sys_sol_begin_aprox, int& total_iter, double relax_param, double epsilon, int max_iter) {
    double* sys_sol = sys_sol_begin_aprox;

    total_iter = 0;
    double rel;
    do {
        iter_step(method, sys_mat, n, m, sys_right_vec, sys_sol, relax_param, epsilon, max_iter);
    }
}

```

```

        rel = relative_discrepancy(sys_mat, n, m, sys_right_vec, sys_sol);
        /*std::cout << "iter: " << total_iter << "; rel: " << rel << std::endl;*/

        total_iter++;
    } while (rel > epsilon and total_iter < max_iter);

    return sys_sol;
}

double* mult_mat_LDU_bandform_vec(double*** mat_elems, double* vec_elems, int n) {
    double* mult_elems = new double[n] {};

    int half_width = 1;
    int L = 0;
    int D = 1;
    int U = 2;

    for (int i = 0; i < n; i++) {
        for (int j = i - half_width; j <= i; j++) {
            if (j >= 0) {
                if (j == i)
                    mult_elems[i] += mat_elems[D][0][i] * vec_elems[i];
                else {
                    mult_elems[i] += mat_elems[L][i][j - (i - half_width)] * vec_elems[j];
                    mult_elems[j] += mat_elems[U][i][j - (i - half_width)] * vec_elems[i];
                }
            }
        }
    }

    return mult_elems;
}

void make_LDU(double** sys_mat, int dim, int half_width) {
    double sum = 0;
    double mult = 0;

    for (int i = 0; i < dim; i++) {
        for (int k = 0; k < i; k++) {
            sys_mat[i][i] -
= sys_mat[i][k] * sys_mat[k][k] * sys_mat[k][i];
        }

        if (i + 1 <= dim - 1) {
            for (int j = 0; j < i + 1; j++) {
                for (int k = 0; k < j; k++) {

                    mult = sys_mat[i + 1][k] * sys_mat[k][k] * sys_mat[k][i + 1];

```



```

        sys_mat[j][i+1] -= mult;
        sys_mat[i + 1][j] -= mult;
    }
    sys_mat[j][i + 1] /= sys_mat[j][j];

    sys_mat[i + 1][j] /= sys_mat[j][j];
}
}
sum = 0;
mult = 0;
}
}

void solve_L(double** sys_mat_elems, double* sys_right_side_vec_elems, int n) {
    int half_width = 1;

    int L = 1;
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < i; k++) {
            sys_right_side_vec_elems[i] -= sys_mat_elems[i][k] * sys_right_side_vec_elems[k];
        }
    }
}

void solve_D(double** sys_mat_elems, double* sys_right_side_vec_elems, int n) {
    int half_width = 1;

    int D = 0;
    for (int i = 0; i < n; i++) {
        sys_right_side_vec_elems[i] /= sys_mat_elems[i][i];
    }
}

void solve_U(double** sys_mat_elems, double* sys_right_side_vec_elems, int n) {
    int half_width = 1;

    for (int i = n - 1; i >= 0; i--) {
        for (int k = i + 1; k < n; k++) {
            sys_right_side_vec_elems[i] -= sys_mat_elems[i][k] * sys_right_side_vec_elems[k];
        }
    }
}

double* solve_sys_LDU(double** sys_mat_elems, double* sys_right_side_vec_elems, int n) {
    solve_L(sys_mat_elems, sys_right_side_vec_elems, n);
    solve_D(sys_mat_elems, sys_right_side_vec_elems, n);
    solve_U(sys_mat_elems, sys_right_side_vec_elems, n);

    return sys_right_side_vec_elems;
}

```

```

double** solve_block_relax(double** sys_mat, int n, int block_dim, int diag_block_half_width, double** sys_right_vec, double** sys_sol_begin_aprox, double relax_param, int& total_iter, double epsilon, int max_iter) {
    double** sys_sol = sys_sol_begin_aprox;

    int sys_sol_blocks_amount = n / block_dim;

    int block_sys_mat_dim = n / block_dim;
    int half_width = 1;

    double** mult_diag_blocks = new double*[block_sys_mat_dim];

    for (int block_i = 0, diag_block = 0; block_i < block_sys_mat_dim; block_i++) {

        make_LDU(sys_mat[block_i][block_i], block_dim, diag_block_half_width);

        double* mult;

        total_iter = 0;
        double rel;
        do {
            for (int block_i=0; block_i < block_sys_mat_dim; block_i++) {

                mult = new double[block_dim] {};

                mult = vec_sum(mult, mult_mat_LDU_vec(sys_mat[block_i][block_i], sys_sol[block_i], block_dim), block_dim);

                for (int block_j=0; block_j < block_sys_mat_dim; block_j++) {
                    if (block_i!=block_j)

                        mult = vec_sum(mult, mult_mat_vec(sys_mat[block_i][block_j], sys_sol[block_j], block_dim), block_dim);
                }

                sys_sol[block_i] = vec_sum(solve_sys_LDU(sys_mat[block_i][block_i], mult_num_vec(relax_param, vec_substr(sys_right_vec[block_i], mult, block_dim), block_dim), block_dim), sys_sol[block_i], block_dim);

                }

                rel = block_relative_discrepancy(sys_mat, n, block_dim, sys_right_vec, sys_sol);
                /*std::cout << "iter: " << total_iter << "; rel: " << rel << std::endl;*/

                total_iter++;
            }
        } while (rel > epsilon);
    }
}

```

```
    } while (rel>epsilon and total_iter<=max_iter);  
  
    return sys_sol;  
}
```

