Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики

Лабораторная работа №1

по дисциплине

«Уравнения математической физики»

Группа	ПМ-71
Студент	Востриков В.
	Аникина П.
Преподаватель	Задорожный А.Г.
Вариант	4

Новосибирск

2020

1. Цель работы

Разработать программу решения эллиптической краевой задачи методом конечных разностей. Протестировать программу и численно оценить порядок аппроксимации.
Область имеет Т-образную форму.

2. Анализ

Начальное уравнение :

$$-\operatorname{div}(\lambda \operatorname{grad} u)+\gamma u = f$$

Для функции:

$$u = u(x, y)$$

Оператор Лапласа:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

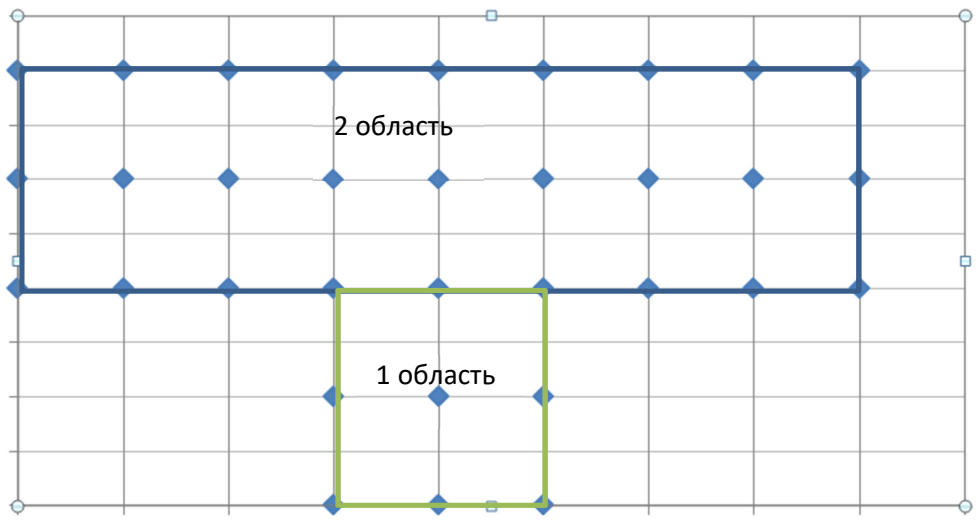
Дискретный аналог оператора Лапласа как пятиточечное разностоное выражение:

$$\Delta_h u_{i,j} = \frac{2u_{i-1,j}}{h^x_{i-1}\left(h^x_i + h^x_{i-1}\right)} + \frac{2u_{i,j-1}}{h^y_{j-1}\left(h^y_j + h^y_{j-1}\right)} + \frac{2u_{i+1,j}}{h^x_i\left(h^x_i + h^x_{i-1}\right)} + \frac{2u_{i,j+1}}{h^y_j\left(h^y_j + h^y_{j-1}\right)} - \left(\frac{2}{h^x_{i-1}h^x_i} + \frac{2}{h^y_{j-1}h^y_j}\right)u_{i,j}$$

Подставив его в наше начальное уравнение, получим:

$$-\frac{2\lambda u_{i-1,j}}{h^x_{i-1}\left(h^x_i + h^x_{i-1}\right)} - \frac{2\lambda u_{i,j-1}}{h^y_{j-1}\left(h^y_j + h^y_{j-1}\right)} - \frac{2\lambda u_{i+1,j}}{h^x_i\left(h^x_i + h^x_{i-1}\right)} - \frac{2\lambda u_{i,j+1}}{h^y_j\left(h^y_j + h^y_{j-1}\right)} + \left(\frac{2}{h^x_{i-1}h^x_i} + \frac{2}{h^y_{j-1}h^y_j} + \gamma\right)u_{i,j} = f_{i,j}$$

При построении сетки, узлы нумеруем снизу вверх и слева направо. Для решения поставленной задачи из линий сетки получаем координаты узлов. Для удобства разделим нашу область на две части.



Матрица A формируется одним проходом по всем узлам (отдельно обрабатывается первая область, граница между первой и второй областью и вторая область), для регулярных узлов заполняется согласно пятиточечному шаблону, для граничных узлов в соответствии с краевыми условиями. Матрица A хранится как 5-ти диагональная матрица.
В случае с равномерной сеткой мы представляем матрицу, как 7-ми диагональную. То есть записываем в матрицу только узлы, которые принадлежат области. И храним два коэффициента сдвига. Примерно таким образом выглядит матрица (рис. 1). Также можно заметить, что не смотря на 7-ми диагональную структуру, достаточно хранить только 5 диагоналей.

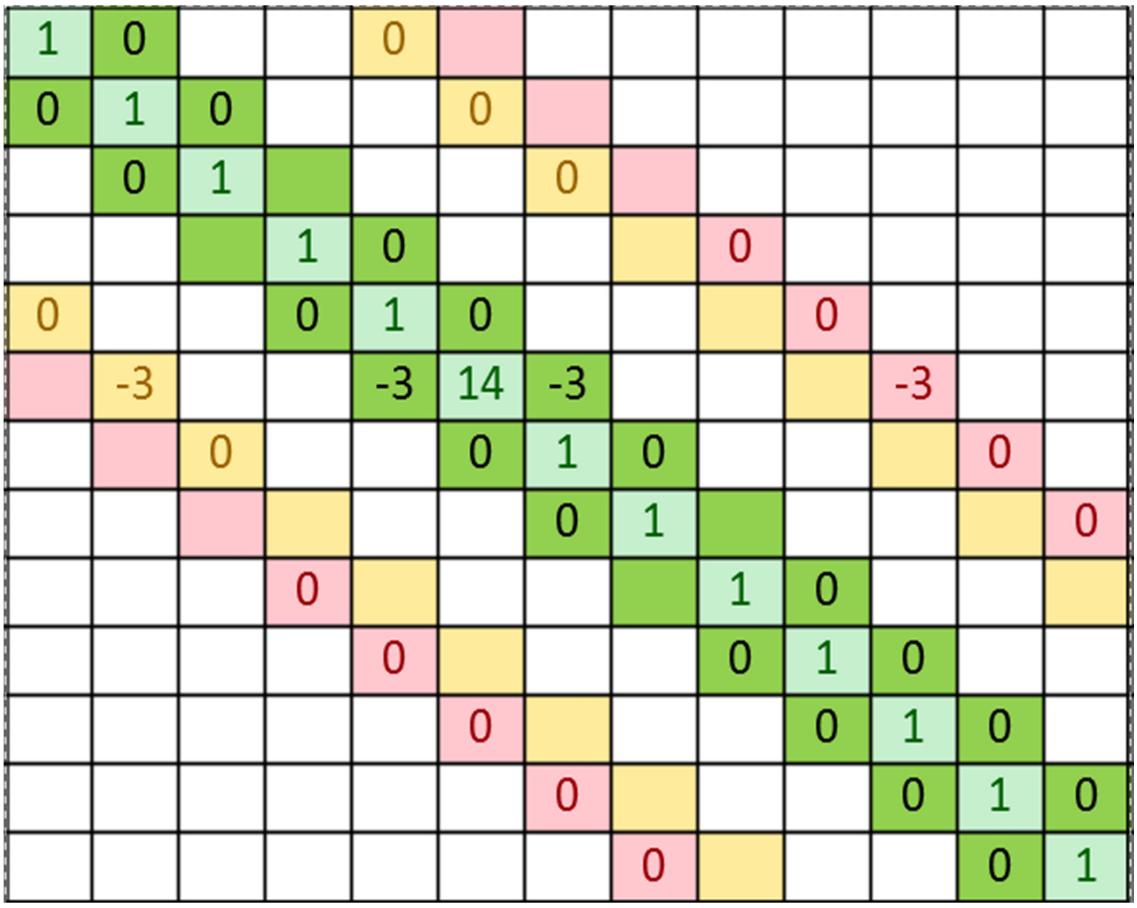


Рис. 1. Хранение матрицы в 7-ми диагональной структуре.

В случае с неравномерной сеткой мы представляем матрицу, как 5-ти диагональную. То есть записываем все узлы прямоугольника, в котором лежит изучаемая T — область. Храним один коэффициент сдвига (рис. 2).

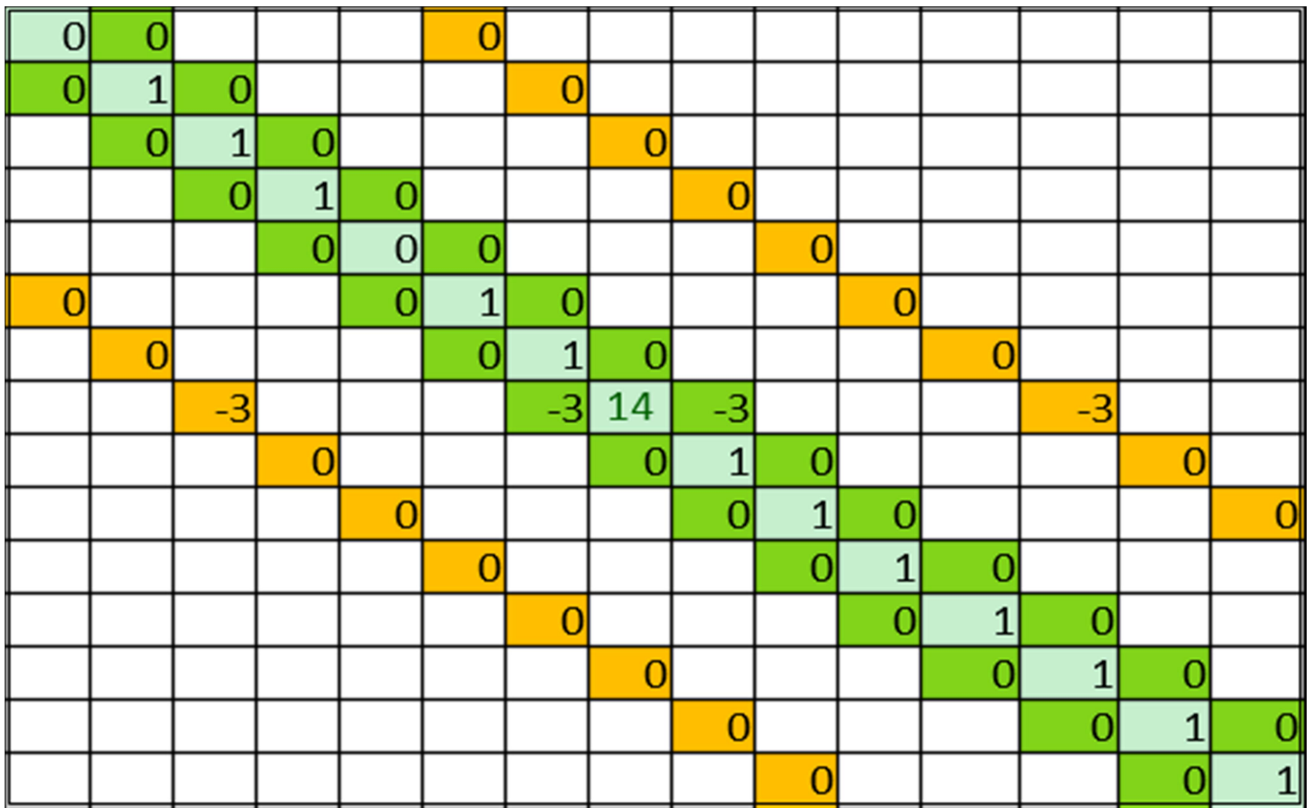


Рис. 2. Хранение матрицы в 5-ти диагональной структуре

Общий вывод после реализации каждой матрицы: *намного легче работать с 5-ти диагональной матрицей, чем с 7-ми, как это было сделано с неравномерной сеткой, так как с 7-ми диагональной матрицей было необходимо рассмотреть больше случаев по сравнению с 5-ти.*

После построения матрицы необходимо решить Au=f.
Для решения СЛАУ будем использовать метод Якоби.

3. Текст программы

4. Тестирование

Равномерная сетка:

1) $u=x^3+y^3$, $\lambda=1$, $\gamma=2$, $f=2*(x^3+y^3)-6*(x+y)$

X =	Y =	U =	U* =	|U - U*|
1	0	1	1	0
2	0	8	8	0
3	0	27	27	0
0	1	1	1	0
1	1	2	2	0
2	1	9	9	0

3	1	28	28	0
4	1	65	65	0
0	2	8	8	0
1	2	9	9	0
2	2	16	16	0
3	2	35	35	0
4	2	72	72	0

2) $u=e^x+e^y$, $\lambda=1$, $\gamma=2$, $f= e^x+e^y$

X =	Y =	U =	U* =	|U - U*|
1	0	3,71828	3,71828	0
1,5	0	5,48169	5,48169	0
2	0	8,38906	8,38906	0
2,5	0	13,1825	13,1825	0
3	0	21,0855	21,0855	0
1	0,5	4,367	4,367	0
1,5	0,5	6,1515	6,13041	0,0210854
2	0,5	9,07057	9,03778	0,0327903
2,5	0,5	13,8649	13,8312	0,0336961
3	0,5	21,7343	21,7343	0
0	1	3,71828	3,71828	0
0,5	1	4,367	4,367	0
1	1	5,43656	5,43656	0
1,5	1	7,22987	7,19997	0,0298977
2	1	10,1526	10,1073	0,0453089
2,5	1	14,947	14,9008	0,0462014
3	1	22,8038	22,8038	0
3,5	1	35,8337	35,8337	0
4	1	57,3164	57,3164	0

0	1,5	5,48169	5,48169	0
0,5	1,5	6,14148	6,13041	0,0110665
1	1,5	7,21757	7,19997	0,0176027
1,5	1,5	8,99371	8,96338	0,0303318
2	1,5	11,9127	11,8707	0,0419174
2,5	1,5	16,7148	16,6642	0,0506436
3	1,5	24,6195	24,5672	0,0522581
3,5	1,5	37,6526	37,5971	0,0554924
4	1,5	59,0798	59,0798	0
0	2	8,38906	8,38906	0
0,5	2	9,03778	9,03778	0
1	2	10,1073	10,1073	0
1,5	2	11,8707	11,8707	0
2	2	14,7781	14,7781	0
2,5	2	19,5716	19,5716	0
3	2	27,4746	27,4746	0
3,5	2	40,5045	40,5045	0
4	2	61,9872	61,9872	0

Неравномерная сетка:

3) $u=x^4+y^4$, $\lambda=1$, $\gamma=2$, $f=2*(x^4+y^4)-12*(x^2+y^2)$

X =	Y =	U =	U* =	|U - U*|
1	0	1	1	0
1.54353	0	5.67623	5.67623	0
2	0	16	16	0
2.78886	0	60.4932	60.4932	0
3	0	81	81	0
1	0.3333	1.01234	1.01234	0
1.54353	0.3333	5.84203	5.68857	0.153453

2	0.3333	16.5062	16.0123	0.493825
2.78886	0.3333	60.015	60.5056	0.490575
3	0.3333	81.0123	81.0123	0
1	0.67788	1.21116	1.21116	0
1.54353	0.67788	6.16257	5.88739	0.275179
2	0.67788	16.9434	16.2112	0.732221
2.78886	0.67788	60.116	60.7044	0.588403
3	0.67788	81.2112	81.2112	0
0	1	1	1	0
1	1	2	2	0
1.54353	1	7.06766	6.67623	0.391428
2	1	17.8907	17	0.8907
2.78886	1	60.9948	61.4932	0.498487
3	1	82	82	0
4	1	257	257	0
0	1.66684	7.71926	7.71926	0
1	1.66684	8.1855	8.71926	0.533765
1.54353	1.66684	13.0958	13.3955	0.29967
2	1.66684	23.8439	23.7193	0.12462
2.78886	1.66684	67.6439	68.2125	0.568581
3	1.66684	89.294	88.7193	0.574765
4	1.66684	263.719	263.719	0
0	1.7777	9.98697	9.98697	0
1	1.7777	10.6437	10.987	0.343226
1.54353	1.7777	15.4907	15.6632	0.172535
2	1.7777	26.1369	25.987	0.149935
2.78886	1.7777	70.1074	70.4802	0.372811
3	1.7777	91.5318	90.987	0.544833
4	1.7777	265.987	265.987	0

0	2	16	16	0
1	2	17	17	0
1.54353	2	21.6762	21.6762	0
2	2	32	32	0
2.78886	2	76.4932	76.4932	0
3	2	97	97	0
4	2	272	272	0

4) $u=x^5+y^5$, $\lambda=1$, $\gamma=2$, $f=2*(x^5+y^5)-20*(x^3+y^3)$

X =	Y =	U =	U* =	|U - U*|
1	0	1	1	0
1.54353	0	8.76144	8.76144	0
2	0	32	32	0
2.78886	0	168.707	168.707	0
3	0	243	243	0
1	0.3333	1.00411	1.00411	0
1.54353	0.3333	9.68147	8.76555	0.915918
2	0.3333	34.6815	32.0041	2.67737
2.78886	0.3333	165.401	168.711	3.30981
3	0.3333	243.004	243.004	0
1	0.67788	1.14314	1.14314	0
1.54353	0.67788	10.4591	8.90458	1.55456
2	0.67788	36.0437	32.1431	3.90051
2.78886	0.67788	164.819	168.85	4.03138
3	0.67788	243.143	243.143	0
0	1	1	1	0
1	1	2	2	0
1.54353	1	11.7305	9.76144	1.96911
2	1	37.5187	33	4.51869

2.78886	1	166.052	169.707	3.65515
3	1	244	244	0
4	1	1025	1025	0
0	1.66684	12.8668	12.8668	0
1	1.66684	12.3341	13.8668	1.53271
1.54353	1.66684	21.0571	21.6282	0.571128
2	1.66684	46.5384	44.8668	1.6716
2.78886	1.66684	179.825	181.574	1.74885
3	1.66684	262.755	255.867	6.88825
4	1.66684	1036.87	1036.87	0
0	1.7777	17.7538	17.7538	0
1	1.7777	17.8486	18.7538	0.905226
1.54353	1.7777	26.2925	26.5153	0.222801
2	1.7777	51.2392	49.7538	1.48538
2.78886	1.7777	185.362	186.461	1.0992
3	1.7777	266.592	260.754	5.83809
4	1.7777	1041.75	1041.75	0
0	2	32	32	0
1	2	33	33	0
1.54353	2	40.7614	40.7614	0
2	2	64	64	0
2.78886	2	200.707	200.707	0
3	2	275	275	0
4	2	1056	1056	0

5) $u=x^2+y^2$, $\lambda=1$, $\gamma=2$, $f=2*(x^2+y^2)-4$

X =	Y =	U =	U* =	|U - U*|
1	0	1	1	0
1.54353	0	2.38248	2.38248	0

2	0	4	4	0
2.78886	0	7.77774	7.77774	0
3	0	9	9	0
1	0.3333	1.11109	1.11109	0
1.54353	0.3333	2.49357	2.49357	8.88178e-16
2	0.3333	4.11109	4.11109	8.88178e-16
2.78886	0.3333	7.88883	7.88883	2.66454e-15
3	0.3333	9.11109	9.11109	0
1	0.67788	1.45952	1.45952	0
1.54353	0.67788	2.84201	2.84201	8.88178e-16
2	0.67788	4.45952	4.45952	1.77636e-15
2.78886	0.67788	8.23726	8.23726	3.55271e-15
3	0.67788	9.45952	9.45952	0
0	1	1	1	0
1	1	2	2	0
1.54353	1	3.38248	3.38248	0
2	1	5	5	8.88178e-16
2.78886	1	8.77774	8.77774	1.77636e-15
3	1	10	10	0
4	1	17	17	0
0	1.66684	2.77836	2.77836	0
1	1.66684	3.77836	3.77836	4.44089e-16
1.54353	1.66684	5.16084	5.16084	8.88178e-16
2	1.66684	6.77836	6.77836	1.77636e-15
2.78886	1.66684	10.5561	10.5561	1.77636e-15
3	1.66684	11.7784	11.7784	1.77636e-15
4	1.66684	18.7784	18.7784	0
0	1.7777	3.16022	3.16022	0
1	1.7777	4.16022	4.16022	0

1.54353	1.7777	5.5427	5.5427	8.88178e-16
2	1.7777	7.16022	7.16022	8.88178e-16
2.78886	1.7777	10.938	10.938	0
3	1.7777	12.1602	12.1602	1.77636e-15
4	1.7777	19.1602	19.1602	0
0	2	4	4	0
1	2	5	5	0
1.54353	2	6.38248	6.38248	0
2	2	8	8	0
2.78886	2	11.7777	11.7777	0
3	2	13	13	0
4	2	20	20	0

6) u=x+y , λ=1, γ=2, f=2*(x+y)

X =	Y =	U =	U* =	|U - U*|
1	0	1	1	0
1.54353	0	1.54353	1.54353	0
2	0	2	2	0
2.78886	0	2.78886	2.78886	0
3	0	3	3	0
1	0.3333	1.3333	1.3333	0
1.54353	0.3333	1.87683	1.87683	4.44089e-16
2	0.3333	2.3333	2.3333	4.44089e-16
2.78886	0.3333	3.12216	3.12216	4.44089e-16
3	0.3333	3.3333	3.3333	0
1	0.67788	1.67788	1.67788	0
1.54353	0.67788	2.22141	2.22141	8.88178e-16
2	0.67788	2.67788	2.67788	4.44089e-16
2.78886	0.67788	3.46674	3.46674	8.88178e-16

3	0.67788	3.67788	3.67788	0
0	1	1	1	0
1	1	2	2	0
1.54353	1	2.54353	2.54353	0
2	1	3	3	4.44089e-16
2.78886	1	3.78886	3.78886	8.88178e-16
3	1	4	4	0
4	1	5	5	0
0	1.66684	1.66684	1.66684	0
1	1.66684	2.66684	2.66684	1.77636e-15
1.54353	1.66684	3.21037	3.21037	1.77636e-15
2	1.66684	3.66684	3.66684	1.33227e-15
2.78886	1.66684	4.4557	4.4557	8.88178e-16
3	1.66684	4.66684	4.66684	1.77636e-15
4	1.66684	5.66684	5.66684	0
0	1.7777	1.7777	1.7777	0
1	1.7777	2.7777	2.7777	8.88178e-16
1.54353	1.7777	3.32123	3.32123	8.88178e-16
2	1.7777	3.7777	3.7777	8.88178e-16
2.78886	1.7777	4.56656	4.56656	0
3	1.7777	4.7777	4.7777	8.88178e-16
4	1.7777	5.7777	5.7777	0
0	2	2	2	0
1	2	3	3	0
1.54353	2	3.54353	3.54353	0
2	2	4	4	0
2.78886	2	4.78886	4.78886	0
3	2	5	5	0
4	2	6	6	0

7) $u=x^5+y^5$, $\lambda=1$, $\gamma=2$, $f=2*(x^5+y^5)-20*(x^3+y^3)$ (с большим количеством дроблений)

X =	Y =	U =	U* =	|U - U*|
1	0	1	1	0
1.3333	0	4.21347	4.21347	0
1.54353	0	8.76144	8.76144	0
1.77777	0	17.7573	17.7573	0
2	0	32	32	0
2.3333	0	69.1597	69.1597	0
2.55555	0	108.999	108.999	0
2.78886	0	168.707	168.707	0
3	0	243	243	0
1	0.16667	1.00013	1.00013	0
1.3333	0.16667	4.19143	4.21359	0.0221608
1.54353	0.16667	8.8398	8.76157	0.078233
1.77777	0.16667	17.8897	17.7575	0.132215
2	0.16667	32.2475	32.0001	0.247371
2.3333	0.16667	69.0275	69.1598	0.132341
2.55555	0.16667	108.991	108.999	0.00791777
2.78886	0.16667	168.676	168.707	0.0316342
3	0.16667	243	243	0
1	0.3333	1.00411	1.00411	0
1.3333	0.3333	4.20617	4.21758	0.0114042
1.54353	0.3333	8.90268	8.76555	0.137126
1.77777	0.3333	17.9923	17.7615	0.23083
2	0.3333	32.3812	32.0041	0.377114
2.3333	0.3333	69.0014	69.1638	0.162364
2.55555	0.3333	108.989	109.003	0.0139789

2.78886	0.3333	168.671	168.711	0.0403697
3	0.3333	243.004	243.004	0
1	0.43333	1.01528	1.01528	0
1.3333	0.43333	4.23196	4.22874	0.00321376
1.54353	0.43333	8.94677	8.77672	0.170055
1.77777	0.43333	18.0526	17.7726	0.279981
2	0.43333	32.4487	32.0153	0.433456
2.3333	0.43333	69.0164	69.1749	0.158532
2.55555	0.43333	109.006	109.014	0.00818499
2.78886	0.43333	168.686	168.722	0.036304
3	0.43333	243.015	243.015	0
1	0.67788	1.14314	1.14314	0
1.3333	0.67788	4.38039	4.35661	0.0237789
1.54353	0.67788	9.11815	8.90458	0.213575
1.77777	0.67788	18.2444	17.9005	0.343874
2	0.67788	32.6443	32.1431	0.501208
2.3333	0.67788	69.1623	69.3028	0.140517
2.55555	0.67788	109.146	109.142	0.00378905
2.78886	0.67788	168.82	168.85	0.0298446
3	0.67788	243.143	243.143	0
1	0.89999	1.59046	1.59046	0
1.3333	0.89999	4.85183	4.80392	0.0479107
1.54353	0.89999	9.60024	9.35189	0.248345
1.77777	0.89999	18.734	18.3478	0.386159
2	0.89999	33.1363	32.5905	0.545797
2.3333	0.89999	69.6467	69.7501	0.103381
2.55555	0.89999	109.633	109.589	0.0434448
2.78886	0.89999	169.303	169.298	0.00507322
3	0.89999	243.59	243.59	0

0	1	1	1	0
0.3333	1	1.00411	1.00411	0
0.55555	1	1.05292	1.05292	0
0.7777	1	1.28449	1.28449	0
1	1	2	2	0
1.3333	1	5.3053	5.21347	0.0918387
1.54353	1	10.0557	9.76144	0.294281
1.77777	1	19.1898	18.7573	0.432497
2	1	33.5932	33	0.59324
2.3333	1	70.108	70.1597	0.0516627
2.55555	1	110.102	109.999	0.103059
2.78886	1	169.778	169.707	0.0707626
3	1	244	244	0
3.3333	1	412.502	412.502	0
3.55555	1	569.243	569.243	0
3.77777	1	770.445	770.445	0
4	1	1025	1025	0
0	1.33333	4.21394	4.21394	0
0.3333	1.33333	4.11544	4.21805	0.102616
0.55555	1.33333	4.15954	4.26686	0.107314
0.7777	1.33333	4.41234	4.49842	0.086086
1	1.33333	5.17857	5.21394	0.0353699
1.3333	1.33333	8.32846	8.4274	0.0989426
1.54353	1.33333	13.0435	12.9754	0.0680751
1.77777	1.33333	22.162	21.9713	0.190736
2	1.33333	36.5686	36.2139	0.354612
2.3333	1.33333	73.1285	73.3736	0.24507
2.55555	1.33333	113.189	113.213	0.0236666
2.78886	1.33333	173.03	172.921	0.108541

3	1.33333	247.658	247.214	0.443712
3.3333	1.33333	415.092	415.716	0.624093
3.55555	1.33333	572.155	572.457	0.30174
3.77777	1.33333	773.531	773.659	0.127721
4	1.33333	1028.21	1028.21	0
0	1.55555	9.10793	9.10793	0
0.3333	1.55555	9.03922	9.11205	0.0728274
0.55555	1.55555	9.09437	9.16085	0.0664828
0.7777	1.55555	9.3526	9.39242	0.039819
1	1.55555	10.1201	10.1079	0.0121182
1.3333	1.55555	13.2365	13.3214	0.0848621
1.54353	1.55555	17.9268	17.8694	0.0574427
1.77777	1.55555	27.0252	26.8653	0.159963
2	1.55555	41.4285	41.1079	0.320595
2.3333	1.55555	78.0412	78.2676	0.226389
2.55555	1.55555	118.116	118.107	0.00895961
2.78886	1.55555	177.981	177.815	0.166017
3	1.55555	252.646	252.108	0.538047
3.3333	1.55555	419.981	420.61	0.628843
3.55555	1.55555	577.063	577.351	0.287792
3.77777	1.55555	778.442	778.553	0.111158
4	1.55555	1033.11	1033.11	0
0	1.66684	12.8668	12.8668	0
0.3333	1.66684	12.8829	12.8709	0.0119687
0.55555	1.66684	12.9477	12.9197	0.0279704
0.7777	1.66684	13.2057	13.1513	0.0544155
1	1.66684	13.9695	13.8668	0.10269
1.3333	1.66684	17.0832	17.0802	0.00296414
1.54353	1.66684	21.756	21.6282	0.127826

1.77777	1.66684	30.8375	30.6241	0.213408
2	1.66684	45.2317	44.8668	0.364958
2.3333	1.66684	81.8974	82.0264	0.12899
2.55555	1.66684	121.96	121.866	0.0947502
2.78886	1.66684	181.812	181.574	0.237558
3	1.66684	256.464	255.867	0.596985
3.3333	1.66684	423.872	424.369	0.496622
3.55555	1.66684	580.937	581.11	0.172978
3.77777	1.66684	782.284	782.312	0.0278656
4	1.66684	1036.87	1036.87	0
0	1.7777	17.7538	17.7538	0
0.3333	1.7777	17.8508	17.758	0.0928249
0.55555	1.7777	17.9213	17.8068	0.114533
0.7777	1.7777	18.1755	18.0383	0.1372
1	1.7777	18.9315	18.7538	0.177675
1.3333	1.7777	22.0555	21.9673	0.0881778
1.54353	1.7777	26.7032	26.5153	0.187898
1.77777	1.7777	35.7617	35.5112	0.250534
2	1.7777	50.1345	49.7538	0.380696
2.3333	1.7777	86.8903	86.9135	0.0231988
2.55555	1.7777	126.92	126.753	0.167465
2.78886	1.7777	186.738	186.461	0.277458
3	1.7777	261.345	260.754	0.590842
3.3333	1.7777	428.935	429.256	0.320546
3.55555	1.7777	585.955	585.997	0.0421302
3.77777	1.7777	787.257	787.199	0.058409
4	1.7777	1041.75	1041.75	0
0	2	32	32	0
0.3333	2	32.0041	32.0041	0

0.55555	2	32.0529	32.0529	0
0.7777	2	32.2845	32.2845	0
1	2	33	33	0
1.3333	2	36.2135	36.2135	0
1.54353	2	40.7614	40.7614	0
1.77777	2	49.7573	49.7573	0
2	2	64	64	0
2.3333	2	101.16	101.16	0
2.55555	2	140.999	140.999	0
2.78886	2	200.707	200.707	0
3	2	275	275	0
3.3333	2	443.502	443.502	0
3.55555	2	600.243	600.243	0
3.77777	2	801.445	801.445	0
4	2	1056	1056	0

7.1) в (2, 1) с шагом в 1 точное значение составляет 33, численное – 38, абс. погрешность = 5.

7.2) в (2, 1) с шагом в 0.5 точное значение = 33, численное = 34.504, абс. погрешность = 1.50398.

7.3) в (2, 1) с шагом в 0.25 точное значение = 33, численное = 33.4, абс. погрешность = 0.4.

7.4) в (2.1) с шагом в 0.125 точное значение = 33, численное = 33,0966, абс. погрешность = 0,0966

Порядок k = 1.733 исходя из 7.1 и 7.2

Порядок k = 1.91 исходя из 7.2 и 7.3

Порядок k = 2.05 исходя из 7.3 и 7.4

5. Код программы

Файл Lab_1.cpp:

```
#include <iostream>

#include <fstream>

#define MAX_ITER 70000
```

```
double epsilon = 1e-15;
using namespace std;
typedef double type;
//
ofstream fout("output.txt");
ifstream fout_x;
ifstream fout_y;
type x_0, x_n, y_0, y_m; // Параметры сетки
type x_0_border, x_n_border; // Границы области T по x
type y_del; // Граница области T по y
int N_x, M_y; // Количество биений по x и y ( +1 - количество узлов)
int k_1_x, k_2_x; // ширина (по узлам) в каждой области x
int k_1_y, k_2_y; // ширина (по узлам) в каждой области y
type step_x, step_y;  // шаг по x и y
int sum_nodes; // Количество узлов
type* res_u;
int choice;

type resh_U(type x, type y) // считает истинное решение U
{
   return pow(x, 1) + pow(y, 1);
}

type m_function(type x, type y) // правая функция
{
   return 2*(pow(x, 1) + pow(y, 1));
}

type lambda(type x, type y)
{
   return 1;
}

type gamma(type x, type y)
{
```

```
    return 2;
}

type first_boundary_condition(type x, type y) // первое краевое условие
{
    return resh_U(x, y);
}

#include "even.h"
#include "uneven.h"

int read_grid()
{

    ifstream info("info.txt");

    choice = 1;
    info >> choice; // Если 1, то равномерная сетка, иначе неравномерная

    if (choice == 1)
    {
        fout_x.open("gridx.txt");
        fout_y.open("gridy.txt");

        if (fout_x) // начало инт, конец инт, начало инт T, конец инт T, количествой биений
            fout_x >> x_0 >> x_n >> x_0_border >> x_n_border >> N_x;
        else return 0;

        if (fout_y) // начало инт, конец инт, уровень деления, количествой биений
            fout_y >> y_0 >> y_m >> y_del >> M_y;
        else return 0;

        if (!calc_grid()) return 0;
    }
    else
```

```
    {
        info >> x_0_border >> x_n_border >> y_del >> N_x >> M_y;
        fout_x.open("gridx_1.txt");
        fout_y.open("gridy_1.txt");
        if (!read_massive_ox_oy()) return 0;
        ///////////!!!!!!!!!!!!!!!
    }

    info.close();
    fout_x.close();
    fout_y.close();
    return 1;
}

int main()
{
    // Начало этапа "сетка"
    type* a,* f, * u_1;
    setlocale(LC_ALL, "rus");
    if (!read_grid()) return -1;



    f = new type[sum_nodes]{ 0 }; // Значение правой части в узлах
    u_1 = new type[sum_nodes]{ 0 }; // значения U(x, y) в узлах
    a = new type[sum_nodes * 5]{ 0 }; // матрица A
    res_u = new type[sum_nodes]{ 0 }; // Истинное решение u

    if (choice == 1)
    {
        if (choice == 1 && !matrix_filling(a, f)) return -2;
    }
    else
        matrix_filling_uneven(a, f, res_u); /////!!!!!!!!!!

    ////////////////////////// Для проверки
```

```
    cout << "Для матрицы " << "\n";
    for (int i = 0; i < sum_nodes; i++)
    {
        cout << "Узел номер " << i + 1 << "\t \t" <<
            *(a + i) << '\t' << *(a + i + sum_nodes) << '\t' << *(a + i + 2 * sum_nodes) <<
'\t'
            << *(a + i + 3 * sum_nodes) << '\t' << *(a + i + 4 * sum_nodes) << '\n';
    }
    cout << "\nДля правой части " << "\n";
    for (int i = 0; i < sum_nodes; i++)
    {
        cout << "Узел номер " << i + 1 << "\t \t" << *(f + i) << endl;
    }
    ////////////////////////
    if (choice == 1)
    {
        if (!calc_SLAE_1(u_1, f, a)) return -2;
    }
    else
        if(!calc_SLAE_2(u_1, f, a)) return -2;
    delete(f);
    delete(a);
    if (choice == 1)
    {
        fout << "\tU =\tU* =\t|U - U*|\n";
        for (int i = 0; i < sum_nodes; i++)
            fout  << *(u_1 + i) << '\t' << *(res_u + i)<< '\t' << abs(*(res_u + i) - *(u_1 +
i)) << endl;
    }
    else
        out_f(u_1);
    delete(res_u);
    fout.close();
    return 1;
    system("exit");
}
```

Содержимое «uneven.h»:

```
#pragma once
type* x_mas, * y_mas;

int read_massive_ox_oy()
{
   x_mas = new type[N_x];
   y_mas = new type[M_y];

   if (fout_x)
      for (int i = 0; i < N_x; i++)
         fout_x >> x_mas[i];

   if (fout_y)
      for (int i = 0; i < M_y; i++)
         fout_y >> y_mas[i];
   sum_nodes = N_x * M_y;
   return 1;
}

type step(type x_, type x_1)
{
   return x_1 - x_;
}

int matrix_filling_uneven(type* A, type* F, type* res_u) // проблема в степах
{
   int i, z = 0;
   for(i = 0; y_mas[i] < y_del; i++)
      for (int j = 0; j < N_x; j++)
      {
         if (x_mas[j] >= x_0_border && x_mas[j] <= x_n_border)
         {
            if (i == 0 || x_mas[j] == x_0_border || x_mas[j] == x_n_border)
```

```
            {
               *(A + 2 * sum_nodes + j + i * N_x) = 1;
               F[j + i * N_x] = resh_U(x_mas[j], y_mas[i]);
            }
            else
            {
               *(A + j + i * N_x) = - 2 * lambda(x_mas[j], y_mas[i]) / (step(y_mas[i -
1], y_mas[i])*(step(y_mas[i - 1], y_mas[i]) + step(y_mas[i], y_mas[i +
1])));//pow(step_y, 2); ///
               *(A + sum_nodes + j + i * N_x) = - 2 * lambda(x_mas[j], y_mas[i]) /
(step(x_mas[j - 1], x_mas[j]) * (step(x_mas[j - 1], x_mas[j]) + step(x_mas[j], x_mas[j +
1])));///
               *(A + 2 * sum_nodes + j + i * N_x) = +2 * lambda(x_mas[j], y_mas[i]) * (1.
/ (step(x_mas[j - 1], x_mas[j]) * step(x_mas[j], x_mas[j + 1])) + 1. / (step(y_mas[i -
1], y_mas[i]) * step(y_mas[i], y_mas[i + 1]))) + gamma(x_mas[j], y_mas[i]); // диагональ
               *(A + 3 * sum_nodes + j + i * N_x) = -2 * lambda(x_mas[j], y_mas[i]) /
(step(x_mas[j], x_mas[j + 1]) * (step(x_mas[j - 1], x_mas[j]) + step(x_mas[j], x_mas[j +
1])));///
               *(A + 4 * sum_nodes + j + i * N_x) = -2 * lambda(x_mas[j], y_mas[i]) /
(step(y_mas[i], y_mas[i + 1]) * (step(y_mas[i - 1], y_mas[i]) + step(y_mas[i], y_mas[i +
1])));//pow(step_y, 2); ///;////
               F[j + i * N_x] = m_function(x_mas[j], y_mas[i]);
            }
            res_u[z] = resh_U(x_mas[j], y_mas[i]);
            z++;
         }
         else
         {
            F[j + i * N_x] = 0;
         }
      }
   for(i; i < M_y; i++)
      for (int j = 0; j < N_x; j++)
      {
         if ((y_mas[i] == y_del && (x_mas[j] <= x_0_border || x_mas[j] >= x_n_border)) ||
i == M_y - 1 || j == 0 || j == N_x - 1)
         {
            *(A + 2 * sum_nodes + j + i * N_x) = 1;
```

```
            F[j + i * N_x] = resh_U(x_mas[j], y_mas[i]);

         }

         else

         {

            *(A + j + i * N_x) = -2 * lambda(x_mas[j], y_mas[i]) / (step(y_mas[i - 1],
y_mas[i]) * (step(y_mas[i - 1], y_mas[i]) + step(y_mas[i], y_mas[i + 1])));//pow(step_y,
2); ///

            *(A + sum_nodes + j + i * N_x) = -2 * lambda(x_mas[j], y_mas[i]) /
(step(x_mas[j - 1], x_mas[j]) * (step(x_mas[j - 1], x_mas[j]) + step(x_mas[j], x_mas[j +
1])));///

            *(A + 2 * sum_nodes + j + i * N_x) = +2 * lambda(x_mas[j], y_mas[i]) * (1. /
(step(x_mas[j - 1], x_mas[j]) * step(x_mas[j], x_mas[j + 1])) + 1. / (step(y_mas[i - 1],
y_mas[i]) * step(y_mas[i], y_mas[i + 1]))) + gamma(x_mas[j], y_mas[i]); // диагональ

            *(A + 3 * sum_nodes + j + i * N_x) = -2 * lambda(x_mas[j], y_mas[i]) /
(step(x_mas[j], x_mas[j + 1]) * (step(x_mas[j - 1], x_mas[j]) + step(x_mas[j], x_mas[j +
1])));///

            *(A + 4 * sum_nodes + j + i * N_x) = -2 * lambda(x_mas[j], y_mas[i]) /
(step(y_mas[i], y_mas[i + 1]) * (step(y_mas[i - 1], y_mas[i]) + step(y_mas[i], y_mas[i +
1])));//pow(step_y, 2); ///;////

            F[j + i * N_x] = m_function(x_mas[j], y_mas[i]);

         }

         res_u[z] = resh_U(x_mas[j], y_mas[i]);

         z++;

      }

   return 1;

}



/**/

type* multiply_matr_vect_1(type* A, type* vec) // 20%

{

   sum_nodes = M_y * N_x;

   type* v = new type[sum_nodes]{ 0 };

   int i, sum_nodes_1, k;



   for (i = 0; y_mas[i] < y_del; i++)

   {

      for (int j = 0; j < N_x; j++)
```

```
      {
         if (i * N_x + j - N_x >= 0)
            *(v + i * N_x + j) += *(A + i * N_x + j) * *(vec + i * N_x + j - N_x); // Это
под вопросом
         if (i - 1 >= 0)
            *(v + i * N_x + j) += *(A + 1 * sum_nodes + i * N_x + j) * *(vec + i * N_x +
j - 1);


         *(v + i * N_x + j) += *(A + 2 * sum_nodes + i * N_x + j) * *(vec + i * N_x + j);
         *(v + i * N_x + j) += *(A + 3 * sum_nodes + i * N_x + j) * *(vec + i * N_x + j +
1);
         *(v + i * N_x + j) += *(A + 4 * sum_nodes + i * N_x + j) * *(vec + i * N_x + j +
N_x);
      }
   }
   i = i * N_x;
   k = i;
   // граница между 1 и 2
   for (i; i < k + N_x; i++)
   {
      if (i - N_x >= 0)
         *(v + i) += *(A + i) * *(vec + i - N_x); // Это под вопросом
      if (i - 1 >= 0)
         *(v + i) += *(A + 1 * sum_nodes + i) * *(vec + i - 1);


      *(v + i) += *(A + 2 * sum_nodes + i) * *(vec + i);
      *(v + i) += *(A + 3 * sum_nodes + i) * *(vec + i + 1);
      *(v + i) += *(A + 4 * sum_nodes + i) * *(vec + i + N_x);
   }


   // Область 2
   for (; i < sum_nodes; i++)
   {
      *(v + i) += *(A + i) * *(vec + i - N_x);
      *(v + i) += *(A + 1 * sum_nodes + i) * *(vec + i - 1);
```

```
        *(v + i) += *(A + 2 * sum_nodes + i) * *(vec + i);
        if (i + 1 <= sum_nodes)
            *(v + i) += *(A + 3 * sum_nodes + i) * *(vec + i + 1);
        if (i + N_x <= sum_nodes)
            *(v + i) += *(A + 4 * sum_nodes + i) * *(vec + i + N_x);
    }
    return v;
}

int calc_SLAE_2(type* X_0, type* F, type* A) //
{
    // Реализация Якоби
    type* residual = new type[sum_nodes]{ 0 };
    type* sup;

    int z = 0;
    while (z < MAX_ITER)
    {
        sup = multiply_matr_vect_1(A, X_0);
        for (int i = 0; i < M_y; i++) {
            for (int j = 0; j < N_x; j++) {
                if (!(y_mas[i] < y_del && (x_mas[j] < x_0_border || x_mas[j] > x_n_border)))
{
                    X_0[i * N_x + j] += F[i * N_x + j] / A[2 * sum_nodes + i * N_x + j];
                    X_0[i * N_x + j] -= sup[i * N_x + j] / A[2 * sum_nodes + i * N_x + j];
                }
            }
        }

        // Выход по невязке
        sup = multiply_matr_vect_1(A, X_0);
        for (int i = 0; i < sum_nodes; i++)
            residual[i] = F[i] - sup[i];
```

```
        if (vector_norm(residual) / vector_norm(F) < epsilon)
            break; // По относительной невязке

        z++;
    }
    return 1;
}

void out_f(type* U_)
{
    fout << "X =\t\t Y = \t\tU = \t\tU* =\t|U - U*|\n";
    int k = 0;
    int i;
    for (i = 0; y_mas[i] != y_del; i++)
        for (int j = 0; j < N_x; j++)
            if (x_mas[j] >= x_0_border && x_mas[j] <= x_n_border)
            {
                k++;
                fout << x_mas[j] << '\t' << y_mas[i] << '\t'  << *(U_ + i * N_x + j) << '\t'
<< *(res_u + k - 1) << '\t' << abs(*(U_ + i * N_x + j) - *(res_u + k - 1)) <<endl;
            }
    for (; i < M_y; i++)
        for (int j = 0; j < N_x; j++)
        {
            k++;
            fout << x_mas[j] << '\t' << y_mas[i] << '\t' << *(U_ + i * N_x + j) << '\t' <<
*(res_u + k - 1) << '\t' << abs(*(U_ + i * N_x + j) - *(res_u + k - 1)) << endl;
        }
}
```

Содержимое «even.h»:

```
#pragma once
```

```
int calc_grid()
{
   step_x = (x_n - x_0) / N_x;
   k_2_x = N_x + 1; // ширина в узлах
   k_1_x = type(x_n_border - x_0_border) / step_x + 1; // ширина в узлах

   step_y = (y_m - y_0) / M_y;

   k_1_y = (y_del - y_0) / step_y; // здесь не учитываем границу
   k_2_y = M_y - k_1_y + 1;

   sum_nodes = k_1_y * k_1_x + k_2_x * k_2_y;
   return 1;
}

int calc_ox_oy(type* X, type* Y)
{
   for (int i = 0; i < N_x + 1; i++)
      *(X + i) = x_0 + step_x * i;
   for (int i = 0; i < M_y + 1; i++)
      *(Y + i) = y_0 + step_y * i;
   return 1;
}

int calc_func(const type* X, const type* Y, type* F) //
{
   int i, k,
      supp_x = (x_0_border - x_0) / step_x;

   for (i = 0; i < k_1_y; i++)
      for (int j = 0; j < k_1_x; j++)
```

```
        {
            if (j == 0 || i == 0 || j == k_1_x - 1)
                *(F + i * k_1_x + j) = first_boundary_condition(*(X + j + supp_x), *(Y + i));
            else
                *(F + i * k_1_x + j) = m_function(*(X + j + supp_x), *(Y + i));
            *(res_u + i * k_1_x + j) = resh_U(*(X + j + supp_x), *(Y + i));
        }


    k = i;
    for (int j = 0; j < k_2_x; j++)
    {
        if (j <= supp_x || j >= k_2_x - 1 - supp_x)
            *(F + k * k_1_x + j) = first_boundary_condition(*(X + j), *(Y + i));
        else
            *(F + k * k_1_x + j) = m_function(*(X + j), *(Y + i));
        *(res_u + k * k_1_x + j) = resh_U(*(X + j), *(Y + i));
    }


    i++;
    for (i; i < M_y + 1; i++)
        for (int j = 0; j < k_2_x; j++)
        {
            if (j == 0 || j == k_2_x - 1 || i == M_y)
                *(F + k * k_1_x + (i - k) * k_2_x + j) = first_boundary_condition(*(X + j),
*(Y + i));
            else
                *(F + k * k_1_x + (i - k) * k_2_x + j) = m_function(*(X + j), *(Y + i));
            *(res_u + k * k_1_x + (i - k) * k_2_x + j) = resh_U(*(X + j), *(Y + i));
        }


    return 1;
}
```

```
int matrix_filling(type* A, type* F)  // Матрица заполняется правильно, что на счет
правой части??
{
   type* x = new type[N_x + 1]{ 0 };
   type* y = new type[M_y + 1]{ 0 };

   if (!calc_ox_oy(x, y)) return -1;
   if (!calc_func(x, y, F)) return -1;

   int i, k, supp_x = (x_0_border - x_0) / step_x;

   // первая область
   for (i = 0; i < k_1_y; i++)
      for (int j = 0; j < k_1_x; j++)
         if (j == 0 || i == 0 || j == k_1_x - 1)
            *(A + 2 * sum_nodes + j + i * k_1_x) = 1;
         else
         {
            *(A + j + i * k_1_x) = -lambda(x[j + supp_x], y[i]) / pow(step_y, 2);
            *(A + sum_nodes + j + i * k_1_x) = -lambda(x[j + supp_x], y[i]) / pow(step_x,
2);
            *(A + 2 * sum_nodes + j + i * k_1_x) = +2 * lambda(x[j + supp_x], y[i]) * (1.
/ pow(step_x, 2) + 1. / pow(step_y, 2)) + gamma(x[j + supp_x], y[i]); // диагональ
            *(A + 3 * sum_nodes + j + i * k_1_x) = *(A + sum_nodes + j + i * k_1_x);
            *(A + 4 * sum_nodes + j + i * k_1_x) = *(A + j + i * k_1_x);
         }

   k = i;
   // граница первой и второй области
   for (int j = 0; j < k_2_x; j++)
      if (j <= supp_x || j >= k_2_x - 1 - supp_x) // (Надо проверить)
         *(A + 2 * sum_nodes + k * k_1_x + j) = 1;
      else
      {
```

```
            *(A + j + i * k_1_x) = -lambda(x[j + supp_x], y[i]) / pow(step_y, 2);

            *(A + sum_nodes + k * k_1_x + j) = -lambda(x[j + supp_x], y[i]) / pow(step_x,
2);

            *(A + 2 * sum_nodes + k * k_1_x + j) = +2 * lambda(x[j + supp_x], y[i]) /
pow(step_x, 2) + 2 * lambda(x[j + supp_x], y[i]) / pow(step_y, 2) + gamma(x[j + supp_x],
y[i]); // диагональ

            *(A + 3 * sum_nodes + k * k_1_x + j) = *(A + sum_nodes + k * k_1_x + j);

            *(A + 4 * sum_nodes + k * k_1_x + j) = *(A + k * k_1_x + j);

         }



      // граница второй области

      i++;

      for (i; i < M_y + 1; i++)

         for (int j = 0; j < k_2_x; j++)

            if (j == 0 || j == k_2_x - 1 || i == M_y)

               *(A + 2 * sum_nodes + k * k_1_x + (i - k) * k_2_x + j) = 1;

            else

            {

               *(A + k * k_1_x + (i - k) * k_2_x + j) = -lambda(x[j + supp_x], y[i]) /
pow(step_y, 2);

               *(A + sum_nodes + k * k_1_x + (i - k) * k_2_x + j) = -lambda(x[j + supp_x],
y[i]) / pow(step_x, 2);

               *(A + 2 * sum_nodes + k * k_1_x + (i - k) * k_2_x + j) = 2 * lambda(x[j +
supp_x], y[i]) / pow(step_x, 2) + 2 * lambda(x[j + supp_x], y[i]) / pow(step_y, 2) +
gamma(x[j + supp_x], y[i]); // диагональ

               *(A + 3 * sum_nodes + k * k_1_x + (i - k) * k_2_x + j) = *(A + sum_nodes + k
* k_1_x + (i - k) * k_2_x + j);

               *(A + 4 * sum_nodes + k * k_1_x + (i - k) * k_2_x + j) = *(A + k * k_1_x + (i
- k) * k_2_x + j);

            }

      // Освобождение памяти

      delete(x);

      delete(y);

      return 1;

}



type vector_norm(type* vec) // 100%
```

```
{
   type val = 0;
   for (int i = 0; i < sum_nodes; i++)
      val += pow(*(vec + i), 2);
   return sqrt(val);
}

type* multiply_matr_vect(type* A, type* vec) // 100%
{
   int sum_area_1 = k_1_x * k_1_y;
   type* v = new type[sum_nodes]{ 0 };
   // область 1
   for (int i = 0; i < sum_area_1; i++)
   {
      if (i - k_1_x >= 0)
         *(v + i) += *(A + i) * *(vec + i - k_1_x);
      if (i - 1 >= 0)
         *(v + i) += *(A + 1 * sum_nodes + i) * *(vec + i - 1);

      *(v + i) += *(A + 2 * sum_nodes + i) * *(vec + i);
      *(v + i) += *(A + 3 * sum_nodes + i) * *(vec + i + 1);

      if (i + k_1_x < sum_area_1)
         *(v + i) += *(A + 4 * sum_nodes + i) * *(vec + i + k_1_x);
      else
         *(v + i) += *(A + 4 * sum_nodes + i) * *(vec + i + (k_1_x + k_2_x) / 2);
   }

   // граница между 1 и 2
   for (int i = sum_area_1; i < sum_area_1 + k_2_x; i++)
   {
      if (i - k_1_x >= 0)
         *(v + i) += *(A + i) * *(vec + i - (k_1_x + k_2_x) / 2); // Это под вопросом
```

```
        if (i - 1 >= 0)
            *(v + i) += *(A + 1 * sum_nodes + i) * *(vec + i - 1);


        *(v + i) += *(A + 2 * sum_nodes + i) * *(vec + i);
        *(v + i) += *(A + 3 * sum_nodes + i) * *(vec + i + 1);
        *(v + i) += *(A + 4 * sum_nodes + i) * *(vec + i + k_2_x);
    }


    // Область 2
    for (int i = sum_area_1 + k_2_x; i < sum_nodes; i++)
    {
        *(v + i) += *(A + i) * *(vec + i - k_2_x);
        *(v + i) += *(A + 1 * sum_nodes + i) * *(vec + i - 1);
        *(v + i) += *(A + 2 * sum_nodes + i) * *(vec + i);
        if (i + 1 <= sum_nodes)
            *(v + i) += *(A + 3 * sum_nodes + i) * *(vec + i + 1);
        if (i + k_2_x <= sum_nodes)
            *(v + i) += *(A + 4 * sum_nodes + i) * *(vec + i + k_2_x);
    }


    return v;
}


type multiply_vect_vect(type* a, type* b) // 100%
{
    type scalar = 0;
    for (int i = 0; i < sum_nodes; i++)
        scalar += a[i] * b[i];
    return scalar;
}


int calc_SLAE_1(type* X_0, type* F, type* A) //
{
```

```
    // Реализация Якоби
    type* residual = new type[sum_nodes]{ 0 };
    type* sup;

    int z = 0;
    while (z < MAX_ITER)
    {
        sup = multiply_matr_vect(A, X_0);
        for (int i = 0; i < sum_nodes; i++) {
            X_0[i] += F[i] / A[2 * sum_nodes + i];
            X_0[i] -= sup[i] / A[2 * sum_nodes + i]; // Это надо исправить!!!!
        }

        // Выход по невязке
        sup = multiply_matr_vect(A, X_0);
        for (int i = 0; i < sum_nodes; i++)
            residual[i] = F[i] - sup[i];

        if (vector_norm(residual) / vector_norm(F) < epsilon)
            break; // По относительной невязке

        z++;
    }
    return 1;
}
```

6. Вывод

Если посмотреть на проделанные тесты, то можно сделать вывод, что метод конечных разностей с учетом первого краевого условия даёт достаточно точные значение для полиномонов степени 3 и меньше. Для полиномов степени выше 3 возникает *большая погрешность, которую можно уменьшить с помощью большего биения сетки* (для тест 7 максимальная разница истинного и численного значения составляет

0,628843. Для теста 4 — 6,88825). Следовательно, *порядок точности метода 3*. Также большая погрешность появляется для неполиномиальных функций.

Использование фиктивных узлов сохраняет 5-диагональную структуру матрицы СЛАУ. Для экономии вычислительных ресурсов можно отказаться от учета фиктивных узлов, но в этом случае матрица становится 7-диагональной. Физически такую матрицу можно также хранить в 5-диагональной структуре, но в этом случае придется хранить еще несколько переменных, которые отвечают за сдвиг (разрыв) диагоналей. Данный подход и был реализован в работе с неравномерной сеткой. В случае равномерной сетки фиктивные узлы не учитывались.

Исходя из тестов 7.1, 7.2, 7.3, 7.4 порядок сходимости примерно равен 2, что соответствует с теоретическим порядком сходимости метода конечных разностей.

Также проводилось исследование порядка аппроксимации, которое показало, что порядок аппроксимации на равномерной сетке равен 3, а на неравномерной - 2. Поиск порядка осуществлялся, перебором полиномов с 0 степени, пока погрешность решения была равна нулю.