ESAM445 HW1 Computing Report

Mingfu Liang, Student ID:3146919, NetID:MLQ4767

May 2019

1. Introduction

In this report, I am going to employ Point Jacobi, Weighted Jacobi, Gauss-Seidel, Red-black Gauss-Seidel, SOR and SSOR with $N = 16, 32, 64$ and $K = 0, -2, 2$ to solve the Helmholtz equation with $u = 0$ on all boundaries. I will illustrate first what happens for $K = 2$ using these schemes and then I will demonstrate what vector norm I will use in this report to monitor convergence for different N. Then I will indicate the smooth mode and high frequency mode of two kind of starting point we are going to use which is (a) $u_{i,j} = 1$ and (b) $u_{i,j} = (-1)^{i+j}$. Then I will go through all these schemes and tell you about how they perform with different K and N and even different ω, which is the weight parameter in several schemes, in solving the Helmholtz equation. I will also demonstrate the sensitivity analysis of the convergence results to ω for weighted Jacobi, SOR and SSOR. Finally I will give you a 'bottom line', conclusion, about which method is preferred for each K and as N increases.

2. How some of these schemes deal with the Helmholtz equation?

2.1 Theoretical and Computationally Analysis why $K = 2$ all the numerical methods are not convergent numerically

First we analysis theoretically what happens when $K = 2$ as (1) using the Fredholm alternative theorem.

$$u_{xx} + u_{yy} + 2\pi^2 u = 32xy\,(x-1)\,(1-y)\,, u\,(x,0) = u\,(x,1) = u\,(0,y) = u\,(1,y) = 0 \tag{1}$$

Using the separation of variable we can calculate that the corresponding homogeneous solution should be $u_H\,(x,y) = \sin(\pi x)\sin(\pi y)$, which is a nontrivial solution. Then we calculate the integral

$$\int_0^1\int_0^1 f\,(x,y) u_H\,(x,y)\,dxdy = \frac{512}{\pi^6} \neq 0 \tag{2}$$

where $f(x,y) = 32xy(x-1)(1-y)$, which implies that (1) does not have solutions.

Numerically, we will see that for all the methods deploying in this homework, the infinity norm of the residual vector keeps increasing as the iteration increasing. From Figure (2), (3), (4), (5), (6), (7) which are the loglog plot of residual for different method when the iteration are large enough (100000 iterations) as $K = 2$, we can see that with the starting point (a) $u_{i,j} = 1$ and for different N, all the methods are going to blow up when $K = 2$. For simplicity I only include the graphs of starting point (a) $u_{i,j} = 1$ but the phenomenon are the same when using starting point (b) $u_{i,j} = (-1)^{i+j}$. And we also see that even the N is going large from 16 to 64, we still can not see any convergence of the residual although the residual do not diverge.

To further illustrate this, if we plot one of the solution among those methods when $K = 2$ at 1000000 iteration (plot only one method for concise since all the methods have the same phenomenon), we will see something like Figure (1). The magnitude of the solution blow up and it makes no sense that it should be the solution.

Therefore, for $K = 2$ I give the analysis about how these relaxation schemes deal with the Helmholtz. They do not have convergence of the numerical scheme and also do not converge to a solution of the continuous problem as I refine the grid, which implies that they don't work for $K = 2$.

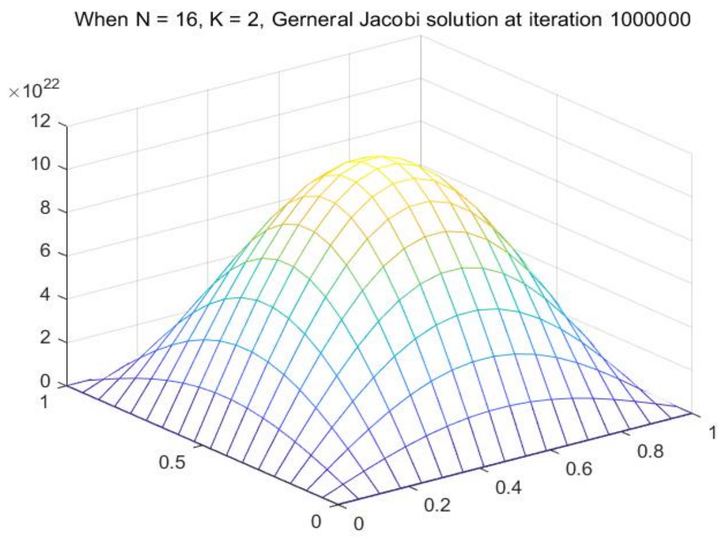

Figure 1: Result of Jacobi when $K = 2$, $N = 16$ and using starting point (a)

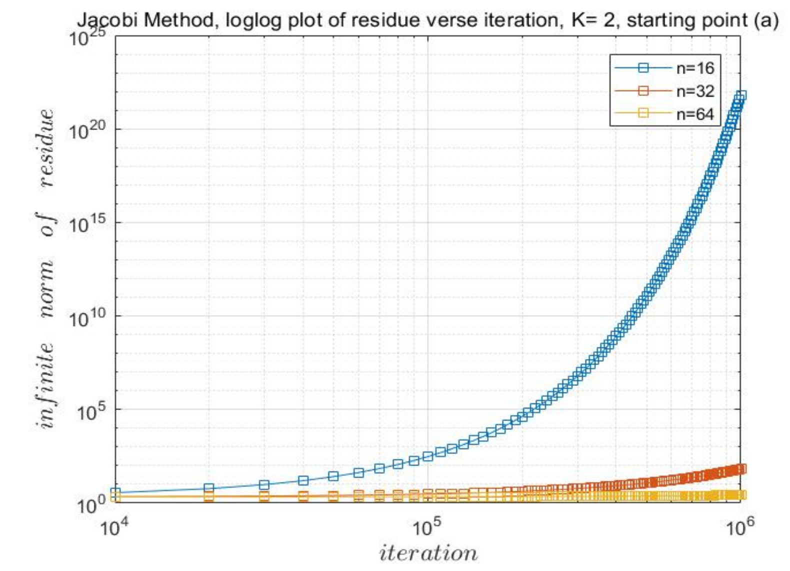

Figure 2: loglog plot of residue verse iteration when $K = 2$ using Jacobi Method

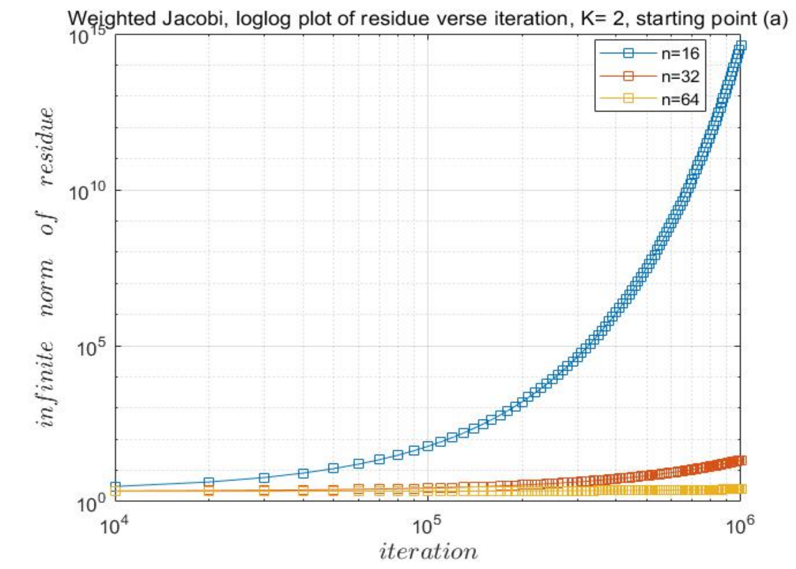

Figure 3: loglog plot of residue verse iteration when $K = 2$ using Weight Jacobi Method

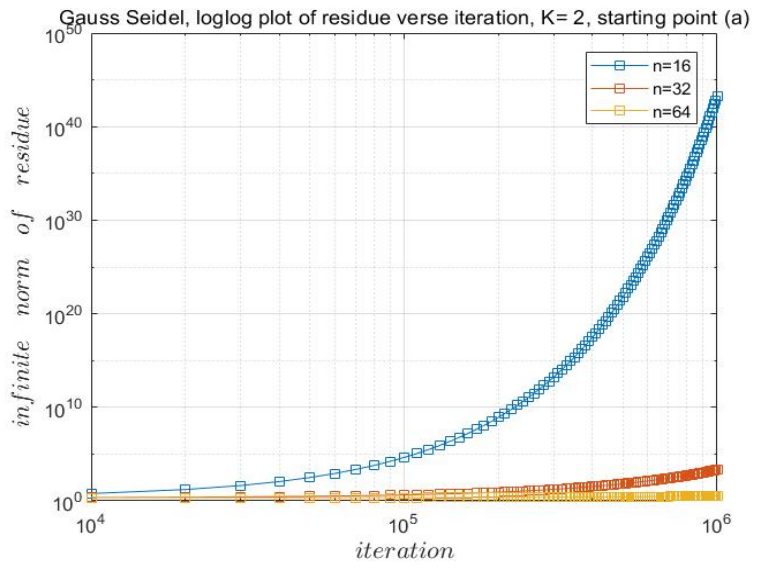

Figure 4: loglog plot of residue verse iteration when $K = 2$ using Gauss Seidel

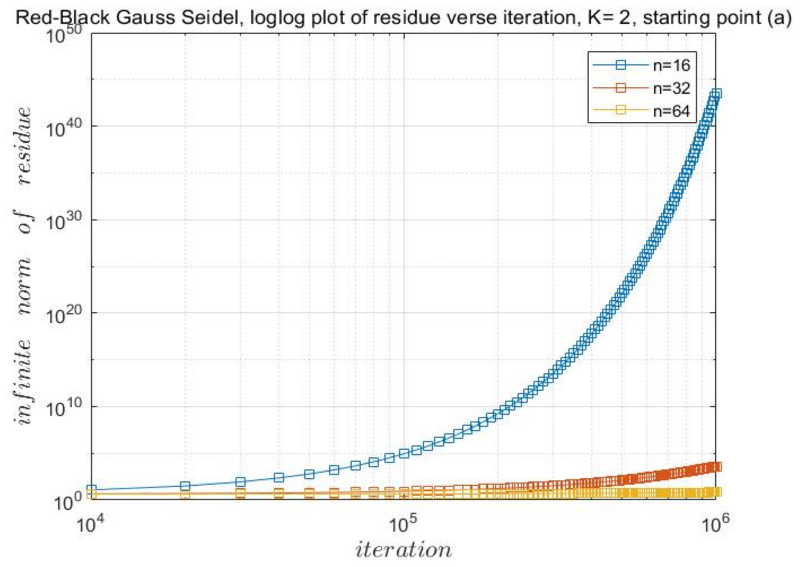

Figure 5: loglog plot of residue verse iteration when $K = 2$ using Red Black Gauss Seidel Method

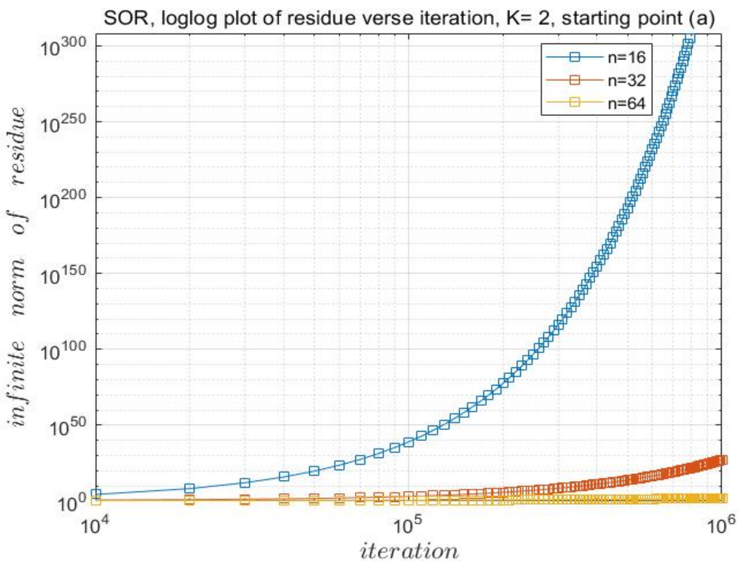

Figure 6: loglog plot of residue verse iteration when $K = 2$ using SOR

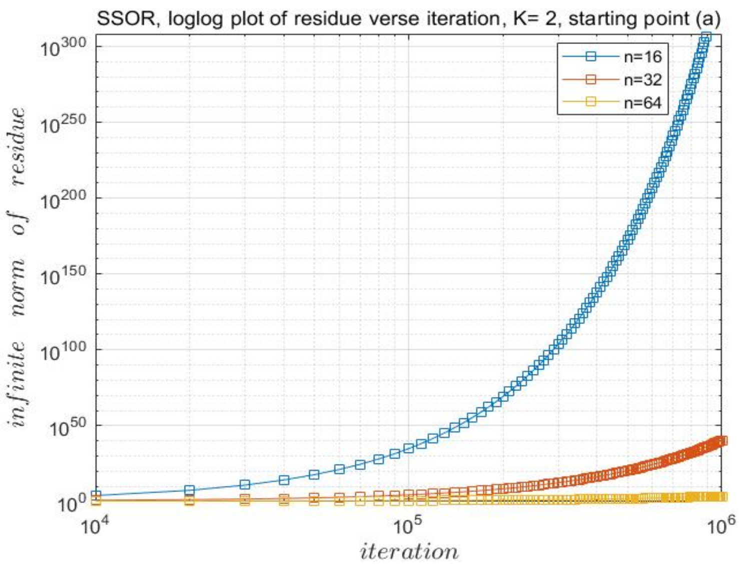

Figure 7: loglog plot of residue verse iteration when $K = 2$ using SSOR

2.2 What about $K = -2$ and $K = 0$?

For $K = -2$ and $K = 2$, all the methods using in this homework can have convergence of the numerical scheme and also converge to a solution of the continuous problem as I refine the grid. For simplicity I will just use SOR method and give the solution of $N = 32$ and $N = 64$ with starting data (a) to verify my statement since the conclusion are the same for other methods.

From Figure 8 and 9 we can see that for $K = 0$, when N increase from 32 to 64, which means using finer and finer grids, the solution are the same. And From Figure 10 and 11 we can also see that for $K = -2$, when N increase from 32 to 64, which menas using finer and finer grids, the solution are the same. And these conclusions are true for all the other methods when $K = 0$ and $K = -2$ and starting data (b). They are all converging to the solution of the corresponding continues problems when $K = 0$ and $K = -2$.

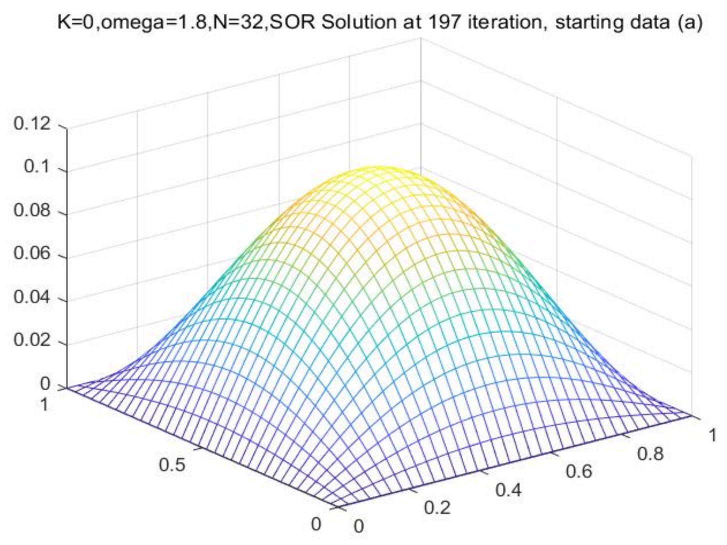

Figure 8: SOR method solution when $K = 0$ and $N = 32$

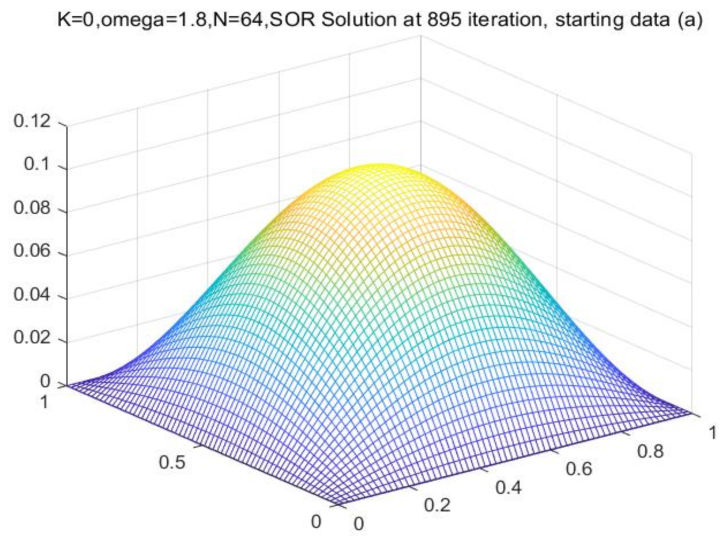

Figure 9: SOR method solution when $K = 0$ and $N = 64$

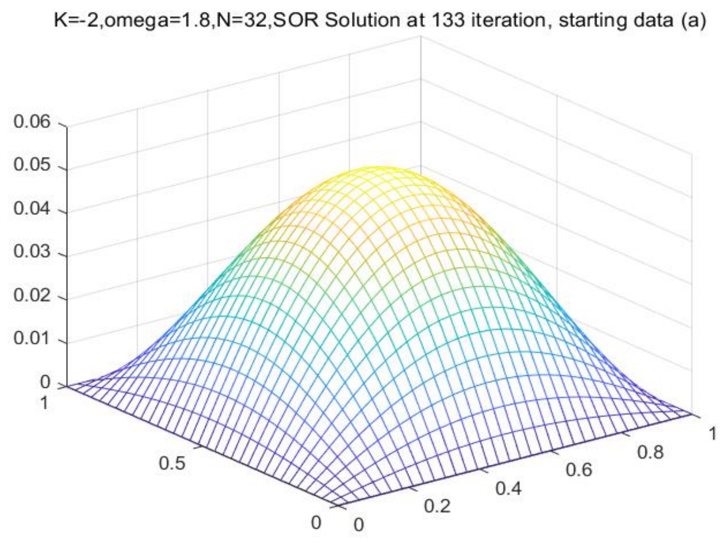

Figure 10: SOR method solution when $K = -2$ and $N = 32$

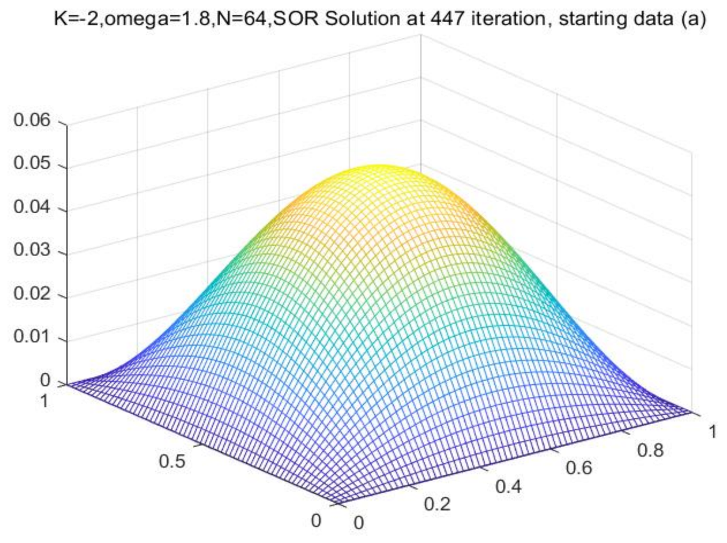

Figure 11: SOR method solution when $K = -2$ and $N = 64$

3. What would be an appropriate norm?

To monitor convergence for different N, the infinite-norm should be a good choice since no matter how the N changes, it finds the maximum absolute entry over all the N entries in the residual vector, which means that it does not depend on N but only depend on the propriety of the residual vector. However if you choose $2-norm$, for example, the $2-norm$ of residue vector will depends on N since when N become larger, the value of $2-norm$ of residual vector will also become larger if you do not average it to reduce the impact of N, and it will lead to more iteration to make the $2-norm$ of residual vector less than tol.

4. Smooth Mode and High Frequency Mode of starting point choice

In this homework we have two starting point which are $(a) u_{i,j} = 1$ and $(b) u_{i,j} = (-1)^{i+j}$. Starting point (a) represents smooth mode on the grid and starting point (b) represents high frequency mode on the grid.

5. Convergence

In the following sections, if I do not give any further specification, then when I talk about total iterations for convergence of the method in fix K, N and ω, I mean the total iteration it needs such that the infinite norm of the residual vector is smaller than the tol where $tol = 10^{-7}$ by default.

6. Point Jacobi

First let's look at Point Jacobi and see how it performs in solving Helmholtz equation. Since as indicate before we have two different starting point represent smooth mode and high frequency mode, I will demonstrate the result for different starting point as following.

6.1 starting data as $u_{i,j} = 1$

When $u_{i,j} = 1$ as the smooth mode, we can see that as N goes larger which means the grid be finer, the total iterations for convergence are increasing. When $K = -2$, it needs less iterations to get convergence, which is the same of our expectation since we know theoretically that in Point Jacobi method, if the matrix A is more diagonally dominant, we should expect more rapid convergence it can get. And since when $K = -2$ has more diagonally dominant matrix A when using Point Jacobi than $K = 0$, therefore it makes sense as what we see in Table 1.

Table 1: Total Iteration for convergence using Point Jacobi when $K = 0, -2$ and starting point as $u_{i,j} = 1$

	$N = 16$	$N = 32$	$N = 64$
$K = 0$	1136	4299	16701
$K = -2$	594	2234	8667

6.2 starting data as $u_{i,j} = (-1)^{i+j}$

For $u_{i,j} = (-1)^{i+j}$, which represent high frequency mode, from Table 2 we get the same conclusion as mentioned in $u_{i,j} = 1$. We can see that Point Jacobi is not good at dealing with high frequency mode since it needs larger iterations to converge than the smooth mode.

Table 2: Total Iteration for convergence using Point Jacobi when $K = 0, -2$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

	$N = 16$	**N=32**	$N = 64$
$K = 0$	1417	5656	23132
$K = -2$	714	2834	11572

7. Weighted Jacobi

For Weighted Jacobi scheme, we need to also evaluate how the weight ω influence the convergence and what is the optimal ω for different N and K and starting data. And we can also see that $K = -2$ needs less iteration to converge than $K = 0$ with same ω, N and starting data.

7.1 starting data as $u_{i,j} = 1$

For starting data as $u_{i,j} = 1$ (smooth mode), from Table 3 we can see that for $K = 0$ and $N = 16, 32, 64$, the optimal ω should be 1, 1 and 1. It needs 1136, 4299 and 16701 iterations for $K = 0$ to get convergence with respect to different N and need 594, 2234 and 8667 for $K = -2$ to get convergence with respect to different N. This is the same as our expectation from theoretical knowledge. When $K = -2$ as shown in Table 4, the conclusion is the same as when $K = 0$. And we can also get the same conclusion that $K = -2$ need less iteration to get convergence than $K = 0$.

Table 3: Total Iteration for convergence using Weighted Jacobi when $K = 0$ and starting point as $u_{i,j} = 1$

ω	$N = 16$	$N = 32$	$N = 64$
0.03333	34343	129253	501297
0.06667	17167	64622	250644
0.1	11442	43078	167093
0.13333	8579	32306	125317
0.16667	6861	25843	100252
0.2	5716	21534	83542
0.23333	4898	18457	71606
0.26667	4285	16148	62654
0.3	3808	14353	55691
0.33333	3426	12917	50121
0.36667	3114	11742	45564
0.4	2853	10763	41766
0.43333	2633	9934	38553
0.46667	2444	9224	35798
0.5	2281	8608	33411
0.53333	2138	8070	31322
0.56667	2011	7594	29479
0.6	1899	7172	27841
0.63333	1799	6794	26375
0.66667	1708	6454	25056
0.7	1627	6146	23862
0.73333	1552	5866	22777
0.76667	1484	5611	21787
0.8	1422	5377	20879
0.83333	1365	5161	20043
0.86667	1312	4962	19272
0.9	1263	4778	18558
0.93333	1218	4607	17895
0.96667	1175	4448	17277
1	1136	4299	16701

Table 4: Total Iteration for convergence using Weighted Jacobi when $K = -2$ and starting point as $u_{i,j} = 1$

ω	$N = 16$	$N = 32$	$N = 64$
0.03333	18088	67313	260277
0.06667	9039	33652	130134

Continued on next page

Table 4 – Total Iteration for convergence using Weighted Jacobi when $K = -2$ and starting point as $u_{i,j} = 1$

ω	$N = 16$	$N = 32$	$N = 64$
0.1	6023	22431	86753
0.13333	4515	16821	65062
0.16667	3610	13455	52048
0.2	3007	11211	43371
0.23333	2576	9608	37174
0.26667	2253	8406	32526
0.3	2001	7471	28911
0.33333	1800	6723	26019
0.36667	1636	6111	23653
0.4	1499	5601	21681
0.43333	1383	5169	20012
0.46667	1283	4799	18582
0.5	1197	4479	17343
0.53333	1121	4198	16258
0.56667	1055	3951	15301
0.6	996	3731	14451
0.63333	943	3534	13690
0.66667	895	3357	13005
0.7	852	3196	12385
0.73333	813	3051	11822
0.76667	777	2917	11307
0.8	744	2795	10836
0.83333	714	2683	10402
0.86667	686	2580	10001
0.9	661	2484	9631
0.93333	637	2395	9286
0.96667	614	2312	8966
1	594	2234	8667

7.2 starting data as $u_{i,j} = (-1)^{i+j}$

For $u_{i,j} = (-1)^{i+j}$ (high frequency mode), from Table 5 and 6 we can see that the optimal ω for $N = 16, 32, 64$ and $K = 0, -2$ are 0.9667. It needs 1417, 5656 and 23132 iterations for $K = 0$ to get convergence with respect to different N and need 714, 2834 and 11572 for $K = -2$ to get convergence with respect to different N. But optimal ω can not be 1. This is the same as what we expect from the theoretical knowledge since when ω is exactly 1, the high frequency mode will be strong. We can also see that when $u_{i,j} = (-1)^{i+j}$, which is the high frequency mode, the iteration we need to get convergence are less than when $u_{i,j} = 1$. It implies that Weighted Jacobi method perform well with high frequency mode.

Table 5: Total Iteration for convergence using Weighted Jacobi when $K = 0$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

ω	$N = 16$	$N = 32$	$N = 64$
0.03333	29708	111775	433477
0.06667	14850	55884	216735
0.1	9897	37253	144487
0.13333	7421	27938	108363
0.16667	5935	22349	86689
0.2	4945	18623	72240
0.23333	4237	15961	61919
0.26667	3707	13965	54178
0.3	3294	12412	48157
0.33333	2964	11170	43341
0.36667	2694	10154	39400
0.4	2468	9307	36116
0.43333	2278	8591	33337
0.46667	2115	7977	30955
0.5	1973	7444	28891
0.53333	1849	6978	27085
0.56667	1740	6568	25491
0.6	1643	6202	24075

Continued on next page

Table 5 –Total Iteration for convergence using Weighted Jacobi when $K = 0$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

ω	$N = 16$	$N = 32$	$N = 64$
0.63333	1556	5875	22807
0.66667	1478	5581	21666
0.7	1407	5315	20634
0.73333	1343	5073	19696
0.76667	1284	4852	18839
0.8	1230	4650	18054
0.83333	1181	4463	17331
0.86667	1135	4291	16665
0.9	1093	4132	16047
0.93333	1053	3984	15474
0.96667	1017	3847	14940
1	1417	5656	23132

Table 6: Total Iteration for convergence using Weighted Jacobi when $K = -2$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

ω	$N = 16$	$N = 32$	$N = 64$
0.03333	15082	56116	216967
0.06667	7537	28054	108479
0.1	5022	18700	72317
0.13333	3765	14023	54236
0.16667	3010	11217	43387
0.2	2507	9346	36155
0.23333	2148	8010	30988
0.26667	1878	7008	27114
0.3	1669	6228	24100
0.33333	1501	5604	21690
0.36667	1364	5094	19717
0.4	1250	4669	18073
0.43333	1153	4309	16682
0.46667	1070	4001	15490
0.5	998	3734	14457
0.53333	935	3500	13553
0.56667	880	3293	12755
0.6	830	3110	12046
0.63333	786	2946	11412
0.66667	747	2798	10841
0.7	711	2665	10324
0.73333	678	2543	9855
0.76667	648	2432	9426
0.8	621	2331	9033
0.83333	596	2237	8671
0.86667	572	2151	8337
0.9	551	2071	8028
0.93333	531	1996	7741
0.96667	512	1927	7474
1	714	2834	11572

8. Gauss-Seidel

8.1 starting data as $u_{i,j} = 1$

When $u_{i,j} = 1$ (smooth mode), from table 7 we can get the same conclusion as we have before for point Jacobi and Weighted Jacobi that $K = -2$ use less iteration to converge and it needs larger iteration for large N to converge.

Table 7: Total Iteration for convergence using Gauss-Siedel when $K = 0, -2$ and starting point as $u_{i,j} = 1$

	$N = 16$	$N = 32$	$N = 64$
$K = 0$	569	2151	8352

Continued on next page

Table 7 – Total Iteration for convergence using Gauss-Siedel when $K = 0, -2$ and starting point as $u_{i,j} = u_{i,j} = (-1)^{(i+j)}$

	$N = 16$	$N = 32$	$N = 64$
$K = -2$	299	1119	4335

8.2 starting data as $u_{i,j} = (-1)^{i+j}$

For $u_{i,j} = (-1)^{i+j}$ (high frequency mode), from Table 8 we can see that for $K = 0, -2$ and $N = 16, 32, 64$, it needs less iterations to converge than $u_{i,j} = 1$. And $K = -2$ needs less iterations to get convergence than $K = 0$. It implies that Gauss-Seidel method perform well with high frequency mode.

Table 8: Total Iteration for convergence using Gauss-Siedel when $K = 0, -2$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

	$N = 16$	$N = 32$	$N = 64$
$K = 0$	493	1860	7222
$K = -2$	249	933	3614

9. Red-black Gauss-Seidel

Since Red-black Gauss-Seidel is just an alternative implementation of the Gauss-Seidel, therefore we expect that we should get the same conclusion as Gauss-Seidel.

9.1 starting data as $u_{i,j} = 1$

For $u_{i,j} = 1$ we do get the same conclusion from Table 9 as mentioned in Gauss-Seidel when $u_{i,j} = 1$.

Table 9: Total Iteration for convergence using Red-black Gauss-Siedel when $K = 0, -2$ and starting point as $u_{i,j} = 1$

	$N = 16$	$N = 32$	$N = 64$
$K = 0$	589	2227	8648
$K = -2$	307	1156	4482

9.2 starting data as $u_{i,j} = (-1)^{i+j}$

For $u_{i,j} = (-1)^{i+j}$ we also do get the same conclusion from Table 9 as mentioned in Gauss-Seidel when $u_{i,j} = (-1)^{i+j}$.

Table 10: Total Iteration for convergence using Red-black Gauss-Siedel when $K = 0, -2$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

	$N = 16$	$N = 32$	$N = 64$
$K = 0$	593	2241	8705
$K = -2$	308	1160	4496

10. SOR

Now it turns to SOR and we need to find out the optimal ω for different K, N and starting data. We will see the iterations that needs to get convergence will first decrease to optimal and then increase again. And we can also see that $K = -2$ needs less iteration to converge than $K = 0$ with same ω, N and starting data.

10.1 starting data as $u_{i,j} = 1$

For starting data $u_{i,j} = 1$, from Table 11 we know that for $K = 0$ and $N = 16, 32, 64$, the corresponding optimal ω should be 1.69, 1.83 and 1.91 and need 69, 138 and 276 iterations to get convergence. From Table 12 we know that for $K = -2$ and $N = 16, 32, 64$, the corresponding optimal ω should be 1.6, 1.76 and 1.87 and need 53, 110 and 227 iterations to get convergence.

Table 11: Total Iteration for convergence using SOR when $K = 0$ and starting point as $u_{i,j} = 1$

ω	$N = 16$	$N = 32$	$N = 64$
1.5	180	708	2775
1.51	174	688	2701
1.52	169	669	2628
1.53	163	651	2556
1.54	158	632	2484
1.55	153	614	2414
1.56	148	595	2345
1.57	142	577	2276
1.58	137	559	2208
1.59	132	542	2141
1.6	126	524	2075
1.61	121	507	2010
1.62	116	490	1945
1.63	110	473	1881
1.64	105	456	1818
1.65	99	440	1756
1.66	93	423	1694
1.67	87	407	1633
1.68	79	391	1573
1.69	69	375	1514
1.7	72	359	1455
1.71	73	343	1396
1.72	76	327	1339
1.73	81	311	1281
1.74	84	295	1225
1.75	86	280	1169
1.76	93	264	1113
1.77	101	248	1058
1.78	102	231	1003
1.79	103	214	949
1.8	103	197	895
1.81	113	178	842
1.82	122	156	789
1.83	135	138	736
1.84	137	144	682
1.85	142	162	629
1.86	160	179	576
1.87	171	198	522
1.88	183	199	466
1.89	205	209	407
1.9	222	249	342
1.91	244	265	276
1.92	275	303	337
1.93	321	333	390
1.94	376	397	417
1.95	444	474	521
1.96	564	597	650
1.97	753	795	895

Continued on next page

Table 11 –Total Iteration for convergence using SOR when $K = 0$ and starting point as $u_{i,j} = 1$

ω	$N = 16$	$N = 32$	$N = 64$
1.98	1131	1200	1300
1.99	2278	2413	2601

Table 12: Total Iteration for convergence using SOR when $K = -2$ and starting point as $u_{i,j} = 1$

ω	$N = 16$	$N = 32$	$N = 64$
1.5	88	364	1436
1.51	85	353	1397
1.52	82	343	1359
1.53	78	333	1322
1.54	75	323	1285
1.55	71	314	1248
1.56	68	304	1212
1.57	64	294	1176
1.58	60	285	1140
1.59	55	275	1105
1.6	53	266	1071
1.61	55	257	1037
1.62	58	247	1003
1.63	61	238	970
1.64	65	229	937
1.65	67	220	904
1.66	68	211	872
1.67	69	202	840
1.68	69	193	808
1.69	70	184	777
1.7	72	174	746
1.71	73	165	715
1.72	75	156	684
1.73	77	146	654
1.74	84	135	624
1.75	88	124	594
1.76	92	110	565
1.77	103	112	535
1.78	104	122	506
1.79	104	132	477
1.8	105	133	447
1.81	110	135	418
1.82	122	137	389
1.83	138	139	359
1.84	139	144	328
1.85	141	157	296
1.86	158	179	261
1.87	174	199	227
1.88	181	199	260
1.89	208	209	262
1.9	219	250	267
1.91	244	267	275
1.92	278	302	339
1.93	318	334	391
1.94	382	399	419
1.95	452	474	521
1.96	567	598	651
1.97	752	799	878
1.98	1142	1205	1302
1.99	2286	2397	2604

10.2 starting data as $u_{i,j} = (-1)^{i+j}$

For starting data $u_{i,j} = (-1)^{i+j}$, from Table 13 we know that for $K = 0$ and $N = 16, 32, 64$, the corresponding optimal ω should be 1.68, 1.82 and 1.90 and needs 72, 145 and 296 iterations to get convergence. From Table 14 we know that for $K = -2$ and $N = 16, 32, 64$, the corresponding optimal ω should be 1.57, 1.74 and 1.86 and needs 56, 114 and 232 iterations to get convergence. And we can also see that when $u_{i,j} = (-1)^{i+j}$, which is high frequency mode, it needs less iteration to get convergence than $u_{i,j} = 1$ for corresponding K, N and ω. It implies that SOR method perform well with high frequency mode.

Table 13: Total Iteration for convergence using SOR when $K = 0$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

ω	$N = 16$	$N = 32$	$N = 64$
1.5	155	612	2399
1.51	151	595	2335
1.52	146	579	2272
1.53	141	562	2210
1.54	137	546	2148
1.55	132	530	2087
1.56	127	515	2027
1.57	123	499	1968
1.58	118	484	1909
1.59	114	468	1851
1.6	109	453	1794
1.61	105	438	1738
1.62	100	424	1682
1.63	95	409	1627
1.64	90	394	1572
1.65	85	380	1518
1.66	80	366	1465
1.67	75	352	1412
1.68	72	338	1360
1.69	74	324	1308
1.7	76	310	1257
1.71	79	296	1207
1.72	83	282	1157
1.73	89	269	1107
1.74	93	255	1058
1.75	98	241	1010
1.76	100	227	962
1.77	102	214	914
1.78	104	199	867
1.79	113	185	820
1.8	119	170	773
1.81	129	153	727
1.82	133	145	681
1.83	138	151	635
1.84	152	168	589
1.85	164	185	543
1.86	169	196	497
1.87	188	201	450
1.88	200	217	401
1.89	224	247	351
1.9	237	263	296
1.91	265	290	322
1.92	300	328	383
1.93	337	380	403
1.94	403	443	490
1.95	475	522	551
1.96	607	654	694
1.97	810	853	912
1.98	1216	1298	1399
1.99	2443	2571	2723

Table 14: Total Iteration for convergence using SOR when $K = -2$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

ω	$N = 16$	$N = 32$	$N = 64$
1.5	74	303	1197
1.51	71	295	1165
1.52	68	286	1133
1.53	65	278	1102
1.54	63	270	1071
1.55	60	261	1040
1.56	57	253	1010
1.57	56	245	980

Continued on next page

Table 14 –Total Iteration for convergence using SOR when $K = -2$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

ω	$N = 16$	$N = 32$	$N = 64$
1.58	57	237	951
1.59	59	229	921
1.6	61	222	893
1.61	63	214	864
1.62	65	206	836
1.63	66	199	808
1.64	67	191	781
1.65	68	183	753
1.66	69	176	726
1.67	70	168	700
1.68	72	161	673
1.69	74	153	647
1.7	76	145	621
1.71	79	138	596
1.72	82	130	570
1.73	89	122	545
1.74	93	114	520
1.75	98	114	495
1.76	101	122	471
1.77	103	127	446
1.78	105	130	422
1.79	108	133	397
1.8	119	136	373
1.81	129	140	348
1.82	134	145	324
1.83	138	151	299
1.84	146	165	273
1.85	161	185	246
1.86	171	196	232
1.87	183	201	254
1.88	201	216	262
1.89	217	247	271
1.9	239	264	285
1.91	268	290	322
1.92	304	329	383
1.93	342	381	403
1.94	408	440	491
1.95	482	524	552
1.96	608	655	694
1.97	821	856	912
1.98	1210	1300	1398
1.99	2450	2579	2726

11. SSOR

Now it turns to SSOR and we need to find out the optimal ω for different K,N and starting data.We will see the iterations that needs to get convergence will first decrease to optimal and then increase again. And we can also see that $K = -2$ needs less iteration to converge than $K = 0$ with same ω, N and starting data

11.1 starting data as $u_{i,j} = 1$

For $u_{i,j} = 1$, from Table 15 we know that for$K = 0$ and $N = 16, 32, 64$, the corresponding optimal ω should be 1.72, 1.84 and 1.92 and need 78, 157 and 321 iterations to get convergence. From Table 16 we know that for $K = -2$ and $N = 16, 32, 64$, the corresponding optimal ω should be 1.63, 1.78 and 1.89 and needs 47, 93 and 189 iterations to converge.

Table 15: Total Iteration for convergence using SSOR when $K = 0$ and starting point as $u_{i,j} = 1$

ω	$N = 16$	$N = 32$	$N = 64$
1.5	109	373	1407
1.51	107	364	1370
1.52	105	355	1334
1.53	103	346	1299
1.54	101	337	1264
1.55	99	329	1229
1.56	97	320	1195
1.57	95	312	1162
1.58	94	304	1128
1.59	92	296	1096
1.6	90	288	1063
1.61	89	280	1032
1.62	87	273	1000
1.63	86	265	969
1.64	85	258	939
1.65	83	251	908
1.66	82	244	879
1.67	81	237	849
1.68	80	230	821
1.69	80	223	792
1.7	79	217	764
1.71	79	211	736
1.72	**78**	205	709
1.73	78	199	682
1.74	79	193	656
1.75	79	188	630
1.76	80	182	604
1.77	81	177	579
1.78	83	173	555
1.79	85	170	531
1.8	88	166	507
1.81	91	163	484
1.82	96	161	463
1.83	101	159	444
1.84	107	**157**	425
1.85	114	157	407
1.86	122	158	389
1.87	132	160	373
1.88	143	166	358
1.89	157	175	344
1.9	174	189	333
1.91	194	209	325
1.92	219	234	**321**
1.93	252	269	324
1.94	295	315	348
1.95	356	380	407
1.96	446	477	508
1.97	598	639	681
1.98	901	963	1027
1.99	1811	1935	2063

Table 16: Total Iteration for convergence using SSOR when $K = -2$ and starting point as $u_{i,j} = 1$

ω	$N = 16$	$N = 32$	$N = 64$
1.5	58	195	731
1.51	57	190	712
1.52	56	185	693
1.53	55	181	675
1.54	54	176	656
1.55	53	172	639
1.56	52	167	621
1.57	51	163	603
1.58	50	159	586
1.59	50	154	569
1.6	49	150	552
1.61	48	146	536
1.62	48	142	520
1.63	**47**	139	503
1.64	47	135	488

Continued on next page

Table 16 –Total Iteration for convergence using SSOR when $K = -2$ and starting point as $u_{i,j} = 1$

ω	$N = 16$	$N = 32$	$N = 64$
1.65	47	131	472
1.66	48	127	457
1.67	48	124	441
1.68	49	120	426
1.69	51	117	412
1.7	52	114	397
1.71	54	111	383
1.72	56	108	369
1.73	58	105	355
1.74	61	102	341
1.75	63	99	328
1.76	66	97	314
1.77	70	95	301
1.78	73	**93**	289
1.79	77	93	276
1.8	81	94	264
1.81	86	96	252
1.82	91	100	241
1.83	97	106	231
1.84	103	112	222
1.85	111	120	213
1.86	119	129	204
1.87	129	140	197
1.88	141	152	190
1.89	154	167	**189**
1.9	170	184	200
1.91	190	206	221
1.92	215	233	249
1.93	247	267	286
1.94	289	313	335
1.95	349	377	404
1.96	438	474	508
1.97	587	635	680
1.98	885	957	1025
1.99	1779	1923	2059

11.2 starting data as $u_{i,j} = (-1)^{i+j}$

For starting data $u_{i,j} = (-1)^{i+j}$, from Table 17 we know that for$K = 0$ and $N = 16, 32, 64$, the corresponding optimal ω should be 1.67, 1.82 and 1.91 and needs 71, 141 and 289 iterations to get convergence. From Table 18 we know that for$K = -2$ and $N = 16, 32, 64$, the corresponding optimal ω should be 1.57, 1.74 and 1.86 and needs 45, 88 and 179 iterations. And we can also see that when $u_{i,j} = (-1)^{i+j}$, which is high frequency mode, it needs less iteration to get convergence than $u_{i,j} = 1$ for corresponding K, N and ω. It implies that SSOR method perform well with high frequency mode.

Table 17: Total Iteration for convergence using SSOR when $K = 0$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

ω	$N = 16$	$N = 32$	$N = 64$
1.5	95	323	1217
1.51	93	315	1185
1.52	91	307	1154
1.53	90	300	1124
1.54	88	292	1093
1.55	86	285	1063
1.56	85	278	1034
1.57	83	270	1005
1.58	82	263	976
1.59	80	256	948
1.6	79	250	920
1.61	78	243	893
1.62	76	236	865
1.63	75	230	839
1.64	74	224	812

Continued on next page

Table 17 –Total Iteration for convergence using SSOR when $K = 0$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

ω	$N = 16$	$N = 32$	$N = 64$
1.65	73	217	786
1.66	72	211	761
1.67	71	205	735
1.68	71	200	710
1.69	71	194	686
1.7	71	188	661
1.71	72	183	638
1.72	75	178	614
1.73	77	173	591
1.74	80	168	568
1.75	84	163	546
1.76	88	159	524
1.77	92	155	502
1.78	97	151	481
1.79	102	148	460
1.8	108	146	440
1.81	114	143	420
1.82	121	141	402
1.83	129	141	386
1.84	138	147	370
1.85	148	156	355
1.86	159	168	340
1.87	172	182	326
1.88	187	198	314
1.89	205	217	303
1.9	227	240	294
1.91	254	268	289
1.92	287	303	319
1.93	329	348	366
1.94	386	408	430
1.95	466	492	518
1.96	585	618	651
1.97	784	828	872
1.98	1182	1248	1315
1.99	2375	2507	2642

Table 18: Total Iteration for convergence using SSOR when $K = -2$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

ω	$N = 16$	$N = 32$	$N = 64$
1.5	49	162	609
1.51	48	159	594
1.52	47	155	578
1.53	46	151	563
1.54	45	147	547
1.55	45	143	533
1.56	45	140	518
1.57	45	136	503
1.58	45	133	489
1.59	46	129	475
1.6	48	126	461
1.61	49	122	447
1.62	51	119	433
1.63	52	116	420
1.64	54	113	407
1.65	56	110	394
1.66	58	107	381
1.67	60	104	368
1.68	62	101	356
1.69	65	98	344
1.7	67	95	331
1.71	70	93	319
1.72	73	90	308
1.73	76	88	296
1.74	80	88	285
1.75	83	90	274
1.76	87	93	263
1.77	91	97	252
1.78	96	102	241
1.79	101	108	231
1.8	107	114	221
1.81	113	120	211

Continued on next page

Table 18 –Total Iteration for convergence using SSOR when $K = -2$ and starting point as $u_{i,j} = (-1)^{(i+j)}$

ω	$N = 16$	$N = 32$	$N = 64$
1.82	120	127	202
1.83	128	136	194
1.84	137	145	186
1.85	146	155	179
1.86	158	167	179
1.87	171	181	192
1.88	186	197	208
1.89	204	216	228
1.9	225	239	253
1.91	252	267	282
1.92	285	302	319
1.93	327	347	366
1.94	383	407	429
1.95	462	491	518
1.96	581	616	651
1.97	778	826	872
1.98	1172	1245	1314
1.99	2356	2502	2641

12. Sensitivity Analysis of SSOR, SOR and Weighted Jacobi with respect to ω

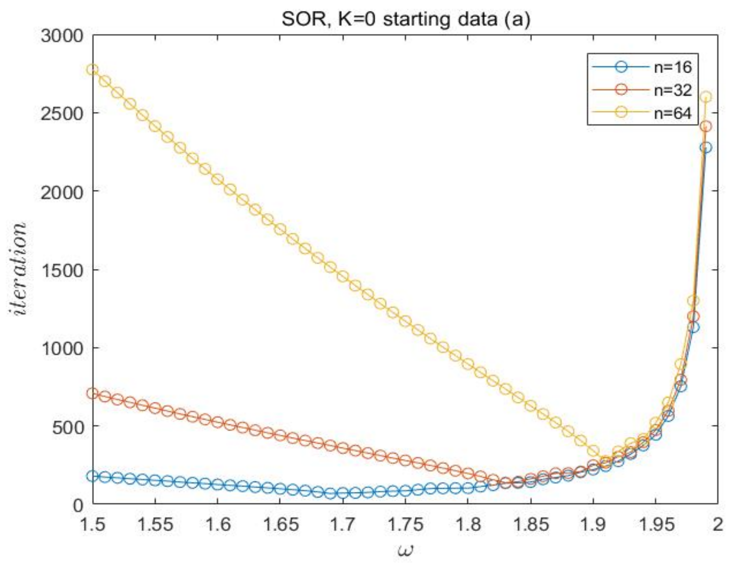

Figure 12: The Sensitivity analysis for SOR with respect to ω using starting data $u_{i,j} = 1$

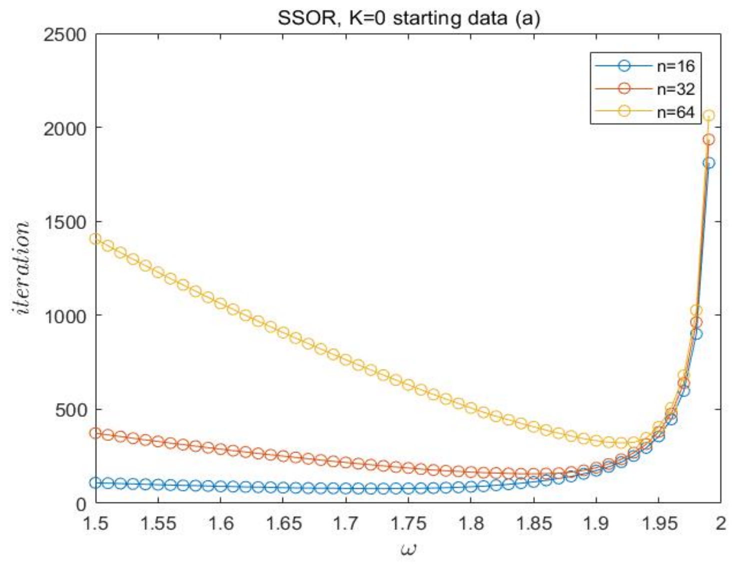

Figure 13: The Sensitivity analysis for SSOR with respect to ω using starting data $u_{i,j} = 1$

From Figure 12 and 13 we can see that as N increase, SOR and SSOR are both more sensitive to the ω when the ω is smaller than the optimal ω. For simplicity here I just show you the figure for starting data (a), but the conclusions are the same with starting data (b).

From Figure 14 and 15 we can see that for both starting data, SOR is more sensitive to the ω than SSOR when the ω is smaller than the optimal ω. For concise here I only include the graphs of $N = 64$ and $K = 0$, but the conclusions are the same for the other alternative N and K.

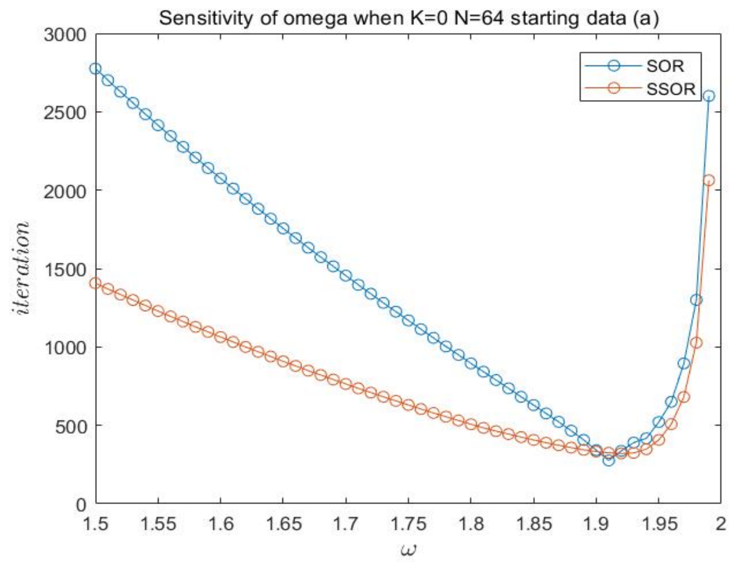

Figure 14: The Sensitivity analysis between SSOR and SOR with respect to ω using starting data $u_{i,j} = 1$

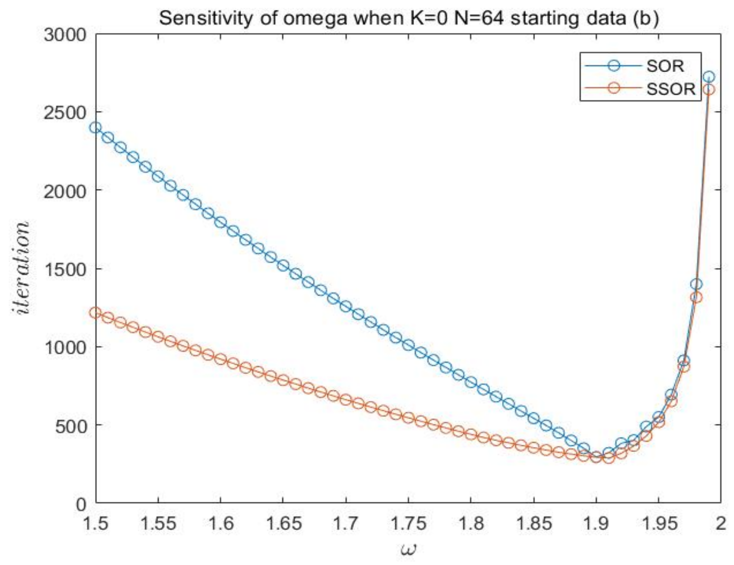

Figure 15: The Sensitivity analysis between SSOR and SOR with respect to ω using starting data $u_{i,j} = (-1)^{i+j}$

From Figure 16 and 17 we can see that for different K with the same starting data $u_{i,j} = 1$, when N is smaller, the Weighted Jacobi will be always more sensitive to the ω when the ω is smaller than the optimal ω. For simplicity I just show you the figure for starting data (a), but the conclusions are the same with starting data (b).

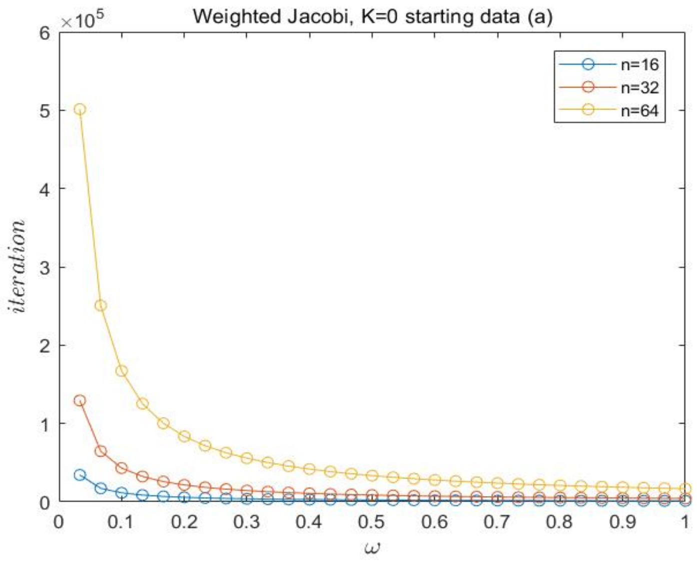

Figure 16: The Sensitivity analysis for Weighted Jacobi with respect to ω using starting data $u_{i,j} = 1$

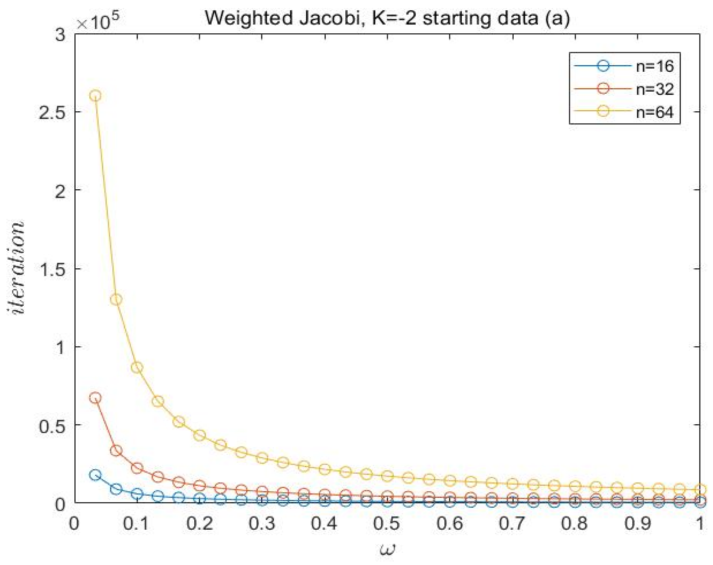

Figure 17: The Sensitivity analysis for Weighted Jacobi with respect to ω using starting data $u_{i,j} = 1$

13. set $\omega = 1$ for SSOR

If we set $\omega = 1$ for SSOR, which implies each SOR step become a Gauss-Seidel Step and as shown in Figure 18 and 19, we can see that SSOR is simply a symmetrized version of Gauss-Seidel with respect to the iterations they need to get convergence separately. SSOR needs 4183 iterations to get convergence, which is approximately the half of Gauss Seidel that needs 8352 iterations to get convergence. For simplicity here I only give $N = 64$ and $K = 0$ but the conclusions are the same with alternative N and K.

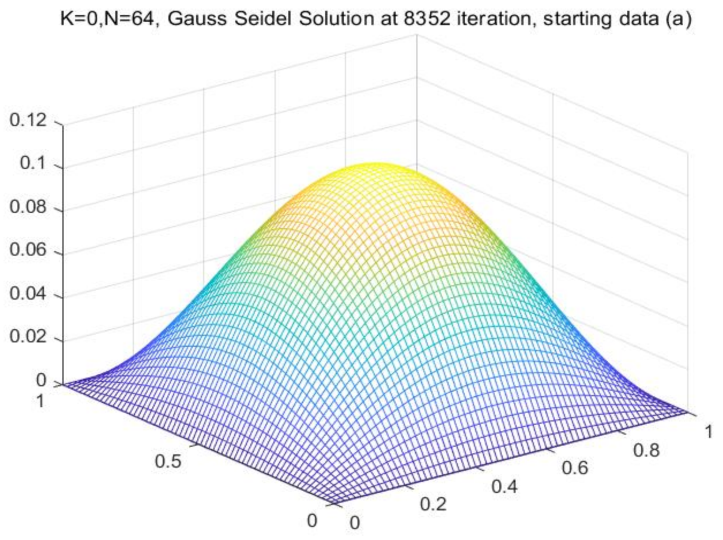

Figure 18: Gauss Seidel Solution when $K = 0$, $N = 64$ using starting data (a)

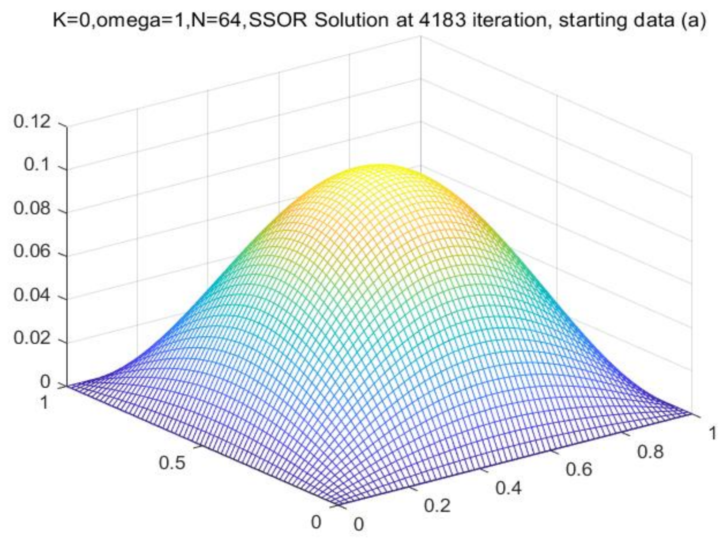

Figure 19: SSOR Solution when $K = 0$,$N = 64$ using starting data (a)

14. Take away points for six methods in solving Helmholtz Equation

From the discussion above, we can see that Point Jacobi performs better at smooth mode when $u_{i,j} = 1$ and Weighted Jacobi, Gauss-Seidel, Red-Black Gauss-Seidel, SOR and SSOR all performs better in high frequency mode when $u_{i,j} = (-1)^{i+j}$. All these methods can solve the Helmholtz Equation when $K = 0, -2$ and the most efficient way is using SOR since when the ω is optimal and considering the less computational cost, it is the best one with less enough iterations and low computational cost.

When $K = 0$, the Helmholtz Equation become Laplace equation and using forward Euler for discretization, we know that the matrix A is not diagonally dominant since we know for each row we have $|a_{ii}| = \sum\limits_{i \neq j}^{N} |a_{ij}| = 4$. But when $K = -2$, the matrix A is more diagonally dominant and lead the numerical method to converge much faster, which is the same as what we discuss numerically above about the difference of iterations between $K = 0$ and $K = -2$.

15. Conclusion about the optimal choice for each K and as N increases

Therefore, based on the analysis provided above, when we choose the optimal ω for different N and K, SOR should be the preferred choice since for $K = 0$ and starting data (a) $u_{i,j} = 1$, and ω is optimal, it needs 69,138 and 276 iteration to converge when the infinity norm of residual vector are less than the default tolerance and for $K = -2$ it needs 72, 110 and 227 correspondingly; for $K = 0$ and starting data (b) $u_{i,j} = (-1)^{i+j}$, and ω is optimal, it needs 56,114 and 232 iteration to converge when the infinity norm of residual vector are less than the default tolerance and for $K = -2$ it needs 56, 114 and 232 correspondingly. For $K = 0, -2$ and as N increases, it needs smallest iterations to get the same solution if we assume that the computational cost of one sweep for Point Jacobi, Weighted Jacobi, Gauss-Siedel, red-black Gauss-Siedel. SSOR use twice computational cost to get less iterations than SOR since SSOR sweep twice in one iteration and therefore SOR is better at this viewpoint although SSOR has less iterations than SOR.

16. MATLAB code

16.1 Point Jacobi

```matlab
function ESAM445_PointJacobi_MingfuLiang(K,N,starting_method)
% Author: Mingfu Liang
% Date: 2019/05/07
%
% Implementation of Point Jacobi Method for solving Helmholtz equation.
%
% Input:
%           K:
%                   Input -2, 0 or 2 to choose different Helmholtz equation.
%
%                   the parameter K in the Helmholtz equation. In this homework
%                   we only have three choice of K as -2, 0, 2.
%
%           N:
%                    Input 16, 32 or 64 to choose the grid size.
%
%                    the size of the grid we are going to create. In this
%                    homework we have three alternative choice which is N=16
%                    N=32 and N=64.
%
%           starting_method:
%                    Input 1 or 2 to choose the starting method.
%
%                    the choice of the starting data we are going to use. In
%                    this homework we have two type of starting method, which
%                    are:
%                             (a) u_{i,j} = 1
%                             (b) u_{i,j} = (-1)^{i+j}
%                     To use the starting method (a), please input 1.
%                     To use the starting method (b), please input 2.
%
% Example:
%
%            ESAM445_PointJacobi_MingfuLiang(-2,16,1)
%
```

```
%           means that you are going to set the grid size N=16 to do the
    Point
%           Jacobi method for Helmholtz equation when K=-2 and using the
%           starting data (a) u_{i,j}=1.
%

tic
%%% initialize parameter

nonhom = @(X,Y) 32.*X.*Y.*(X-1).*(1-Y); % define the nonhomogenous part of
    the equation, which is the right handside
h      = 1/(N+1);        % grid spacing. Here I use N+1 since I want to be
    consistent with the notebook
tol    = 1e-7;         % tolerance

u    = zeros(N+2,N+2);     % storage for solution, here I use N+2 since I want
     to be consistent with the notebook
                                              % we denote that 0,1,2,...,N,N+1,
                                                 which means
                                              % that if we have N=16, we actually
                                                 have N+2
                                              % points although the at 0 and N+1
                                                 it should
                                              % be zero as the boundary condition
                                                 in this
                                              % Homework since we are setting at
                                                 all the
                                              % boundary u=0.

res      = u;% storage for residual, here residual is a matrix
u_update  = u;          % intialize the solution update in each iteration
loop_count     = 0;         % while loop counter
[X,Y] = meshgrid(0:h:1,0:h:1); % coordinates for final solution generation.
f      = nonhom(X,Y); % generate the nonhomogenous part matrix

%%%

%Now start the iteration solver, stop when
%infinite norm of the residual vector < tolerance

```

```
figure;

%%%% vectorized the iteration, generate the index set such that all the calculation can done simultaneous %%%%%%%%%

%%%% initial vectorize index for generating intial guess of starting points %%%%

l =2:N+1;
m=2:N+1;

%%%% initial guess of starting points %%%%

if starting_method ==1
    u(l,m)=1;
    starting_name = '(a)';
end

if starting_method ==2
    for i =2:N+1
        for j =2:N+1
            u(i,j)=(-1)^(i+j);
        end
    end
    starting_name = '(b)';
end

%%%% generate the index to do vectorize calculation later

i = 2:N+1; % start from 2 and end at N+1 since I don't want to update the boundary value since they should be zero all the time
j = 2:N+1; % start from 2 and end at N+1 since I don't want to update the boundary value since they should be zero all the time

%%%% initial the boundary value %%%%
%%%% In this homework we set u=0 at all the boundary %%%%

    u(1,:) =0;
    u(:,1) =0;
    u(N+2,:) =0;
    u(:,N+2) =0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while 1

    %%% Point Jacobi Implementation. Here I parallel all the calculation at
        the
    %%% same time since in Point Jacobi the update of each individual
    %%% element in solution matrix is independent with each other.
    %%% Here I use the Forward Euler to get the general update formula
    %%% for the u(i,j). i and j are the vectorize index which I defined
    %%% before. Also for the residue vector, I can also use the similar
    %%% formula to calculate each element in residue matrix correspondingly.

    u_update(i,j) = (1/(4 - (h^2)*pi*pi*K))*( u(i-1,j) + u(i+1,j) + ...
                        u(i,j-1) + u(i,j+1) - h^2 * f(i,j) );
    res(i,j) = f(i,j) - (1/h^2)*( u(i-1,j) + ...
            u(i+1,j) + u(i,j-1) + u(i,j+1) - (4-(h^2)*pi*pi*K)*u(i,j) ); %
                get the corresponding residual matrix for each individual
                solution

    %%% convergence check using infinite vector norm of residue matrix
    %%% be careful here you should first use res(:) to change the matrix to
    %%% a vector so that you can use vector norm correspondingly.

    if norm(res(:),inf) < tol
        break
    end

    %%% update the solution %%%%

    u = u_update;

    %%%% initial the boundary value %%%%

    u(1,:) =0;
    u(:,1) =0;
    u(N+2,:) =0;
    u(:,N+2) =0;

    loop_count = loop_count+1; % update the iteration count of loop
end

%%% plot the solution at convergence iteration

mesh(X,Y,u);
title1 = ['K=', num2str(K),',N=',num2str(N), ', Point Jacobi Solution at ',
    num2str(loop_count),' iteration, starting data ', starting_name];
title(title1)
toc
end
```

16.2 Weight Jacobi

```
function ESAM445_WeightJacobi_MingfuLiang(K,N,omega,starting_method)
% Author: Mingfu Liang
% Date: 2019/05/07
%
% Implementation of Weight Jacobi Method for solving Helmholtz equation.
%
% Input:
%           K:
%                 Input -2, 0 or 2 to choose different Helmholtz equation.
%
%                 the parameter K in the Helmholtz equation. In this homework
%                 we only have three choice of K as -2, 0, 2.
%
%           N:
%                  Input 16, 32 or 64 to choose the grid size.
%
%                  the size of the grid we are going to create. In this
%                  homework we have three alternative choice which is N=16
%                  N=32 and N=64.
%           omega:
%                  Input value between 0 to 1 for weight
%
%                 The weight parameter for weight Jacobi method.
%
%           starting_method:
%                  Input 1 or 2 to choose the starting method.
```

```
%
%                   the choice of the starting data we are going to use. In
%                   this homework we have two type of starting method, which
%                   are:
%                          (a) u_{i,j} = 1
%                          (b) u_{i,j} = (-1)^{i+j}
%                    To use the starting method (a), please input 1.
%                    To use the starting method (b), please input 2.
%
% Example:
%
%             ESAM445_WeightJacobi_MingfuLiang(-2,16,2/3,1)
%
%             means that you are going to set the grid size N=16 and omega
%             =2/3 to do the Weight Jacobi method for Helmholtz equation
%             when K=-2 and using the starting data (a) u_{i,j}=1.
%

tic
%%%  initialize parameter
w = omega;
nonhom = @(X,Y) 32.*X.*Y.*(X-1).*(1-Y); % define the nonhomogenous part of
    the equation, which is the right handside
h      = 1/(N+1);          % grid spacing. Here I use N+1 since I want to be
    consistent with the notebook
tol    = 1e-7;          % tolerance

u   = zeros(N+2,N+2);     % storage for solution, here I use N+2 since I want
     to be consistent with the notebook
                                                % we denote that 0,1,2,...,N,N+1,
                                                    which means
                                                % that if we have N=16, we actually
                                                    have N+2
                                                % points although the at 0 and N+1
                                                    it should
                                                % be zero as the boundary condition
                                                    in this
                                                % Homework since we are setting at
                                                    all the
                                                % boundary u=0.
```

```

res      = u;% storage for residual, here residual is a matrix
u_update  = u;          % intialize the solution update in each iteration
loop_count      = 0;          % while loop counter
[X,Y] = meshgrid(0:h:1,0:h:1); % coordinates for final solution generation.
f      = nonhom(X,Y); % generate the nonhomogenous part matrix

%%

%Now start the iteration solver, stop when
%infinite norm of the residual vector < tolerance

figure;

%%% vectorized the iteration, generate the index set such that all the
    calculation can done simultaneous %%%%%%%%%%

%%% initial vectorize index for generating intial guess of starting points
    %%%

l  =2:N+1;
m=2:N+1;

%%% initial guess of starting points %%%

if starting_method ==1
    u(l,m)=1;
    starting_name = '(a)';
end

if starting_method ==2
    for i  =2:N+1
        for j  =2:N+1
            u(i,j)=(-1)^(i+j);
        end
    end
    starting_name = '(b)';
end

%%% generate the index to do vectorize calculation later
```

```

i = 2:N+1; % start from 2 and end at N+1 since I don't want to update the
    boundary value since they should be zero all the time
j = 2:N+1; % start from 2 and end at N+1 since I don't want to update the
    boundary value since they should be zero all the time

%%%% initial the boundary value %%%%
%%%% In this homework we set u=0 at all the boundary %%%%

     u(1,:) =0;
     u(:,1) =0;
     u(N+2,:) =0;
     u(:,N+2) =0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while 1

     %%% Weight Jacobi Implementation. Here I parallel all the calculation at
          the
     %%% same time since in Weight Jacobi the update of each individual
     %%% element in solution matrix is independent with each other.
     %%% Here I use the Forward Euler to get the general update formula
     %%% for the u(i,j). i and j are the vectorize index which I defined
     %%% before. Also for the residue vector, I can also use the similar
     %%% formula to calculate each element in residue matrix correspondingly.

     u_update(i,j) = (1-w)*(u(i,j))+ ...
                         w*(1/(4 - (h^2)*pi*pi*K))*( u(i-1,j) + u(i+1,j) + ...
                        u(i,j-1) + u(i,j+1) - h^2 * f(i,j));
     res(i,j) = f(i,j) - (1/h^2)*( u(i-1,j) + ...
               u(i+1,j) + u(i,j-1) + u(i,j+1) - (4-(h^2)*pi*pi*K)*u(i,j) ); %
                   get the corresponding residual matrix for each individual
                   solution

     %%% convergence check using infinite vector norm of residue matrix
     %%% be careful here you should first use res(:) to change the matrix to
     %%% a vector so that you can use vector norm correspondingly.

     if norm(res(:),inf) < tol
```

```
            break
        end

        %%% update the solution %%%%

        u = u_update;

        %%%% initial the boundary value %%%%

        u(1,:) =0;
        u(:,1) =0;
        u(N+2,:) =0;
        u(:,N+2) =0;

        loop_count = loop_count+1; % update the iteration count of loop
end

%%% plot the solution at convergence iteration

mesh(X,Y,u);
title1 = ['K=', num2str(K), ',omega=',num2str(w),',N=',num2str(N), ',Weight
    Jacobi Solution at ',num2str(loop_count),' iteration, starting data ',
    starting_name];
title(title1)
toc
end
```

16.3 Gauss Seidel

```
function ESAM445_GaussSeidel_MingfuLiang(K,N,starting_method)
% Author: Mingfu Liang
% Date: 2019/05/07
%
% Implementation of Gauss Seidel Method for solving Helmholtz equation.
%
% Input:
%          K:
%                 Input -2, 0 or 2 to choose different Helmholtz equation.
%
%                 the parameter K in the Helmholtz equation. In this homework
%                 we only have three choice of K as -2, 0, 2.
```

```
%
%           N:
%                   Input 16, 32 or 64 to choose the grid size.
%
%                   the size of the grid we are going to create. In this
%                   homework we have three alternative choice which is N=16
%                   N=32 and N=64.
%
%           starting_method:
%                   Input 1 or 2 to choose the starting method.
%
%                   the choice of the starting data we are going to use. In
%                   this homework we have two type of starting method, which
%                   are:
%                         (a) u_{i,j} = 1
%                         (b) u_{i,j} = (-1)^{i+j}
%
%                    To use the starting method (a), please input 1.
%                    To use the starting method (b), please input 2.
%
% Example:
%
%            ESAM445_GaussSeidel_MingfuLiang(-2,16,1)
%
%            means that you are going to set the grid size N=16 to do the
%            Gauss Seidel method for Helmholtz equation when K=-2 and
%            using the starting data (a) u_{i,j}=1.
%


tic
%%  initialize parameter

nonhom = @(X,Y)  32.*X.*Y.*(X-1).*(1-Y); % define the nonhomogenous part of
    the equation, which is the right handside
h      = 1/(N+1);          % grid spacing. Here I use N+1 since I want to be
    consistent with the notebook
tol    = 1e-7;          % tolerance

u    = zeros(N+2,N+2);     % storage for solution, here I use N+2 since I want
     to be consistent with the notebook
```

```
                                                % we denote that 0,1,2,...,N,N+1,
                                                   which means
                                                % that if we have N=16, we actually
                                                   have N+2
                                                % points although the at 0 and N+1
                                                   it should
                                                % be zero as the boundary condition
                                                   in this
                                                % Homework since we are setting at
                                                   all the
                                                % boundary u=0.

res      = u;% storage for residual, here residual is a matrix
loop_count     = 0;        % while loop counter
[X,Y] = meshgrid(0:h:1,0:h:1); % coordinates for final solution generation.
f     = nonhom(X,Y); % generate the nonhomogenous part matrix

%%

%Now start the iteration solver, stop when
%infinite norm of the residual vector < tolerance

figure;

%%%% vectorized the iteration, generate the index set such that all the
   calculation can done simultaneous %%%%%%%%%

%%%% initial vectorize index for generating intial guess of starting points
   %%%%

l  =2:N+1;
m=2:N+1;

%%%% initial guess of starting points %%%%

if starting_method ==1
    u(l,m)=1;
    starting_name = '(a)';
end

```

```
if starting_method ==2
    for i =2:N+1
        for j =2:N+1
            u(i,j)=(-1)^(i+j);
        end
    end
    starting_name = '(b)';
end

%%%% initial the boundary value %%%%
%%%% In this homework we set u=0 at all the boundary %%%%

    u(1,:) =0;
    u(:,1) =0;
    u(N+2,:) =0;
    u(:,N+2) =0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while 1

    %%% Gauss Seidel Implementation. In Gauss Seidel the update of
    %%% each individual is dependent to the previous update and hard
    %%% to parallel.
    %%% Here I use the Forward Euler to get the general update formula
    %%% for the u(i,j). The outer loop is in i and inner loop is in j,
    %%% which mean the outer loop is for row loop and inner loop is for
    %%% column loop. For example for i=1, which means for the first row,
    %%% we loop j=1:N+1, which means loop for all the columns in first row.
    %%% For example for i=1, j=1, we first update u(1,1), then i is still 1
    %%% and j become 2, and since I just use the same u matrix to storage
    %%% the solution, therefore when calculate u(1,2) by the fomular below,
    %%% it automatically use the updated u(1,1) by the previous iteration.
        So
    %%% I save the memory by just using one matrix to storage and update
    %%% the solution and also satisfies the requirement of Gauss Seidel.

    for i=2:N+1
        for j=2:N+1
            u(i,j) =  (1/(4 - (h^2)*pi*pi*K))*( u(i,j+1) + u(i,j-1) +u(i-1,j
            ) + u(i+1,j) -h^2*f(i,j));
        end
    end

    res(l,m) = f(l,m) - (1/h^2)*( u(l-1,m) + u(l+1,m) + u(l,m-1) + u(l,m+1)
        - (4-(h^2)*pi*pi*K)*u(l,m) ); % get the corresponding residual matrix
        for each individual solution

    %%% convergence check using infinite vector norm of residue matrix
    %%% be careful here you should first use res(:) to change the matrix to
    %%% a vector so that you can use vector norm correspondingly.

    loop_count = loop_count+1; % update the iteration count of loop

    if norm(res(:),inf) < tol
        break
    end

    %%%% initial the boundary value %%%%

    u(1,:) =0;
    u(:,1) =0;
    u(N+2,:) =0;
    u(:,N+2) =0;

end

%%% plot the solution at convergence iteration

mesh(X,Y,u);
title1 = ['K=', num2str(K),',N=',num2str(N), ', Gauss Seidel Solution at ',
    num2str(loop_count),' iteration, starting data ', starting_name];
title(title1)
toc
end
```

16.4 Red Black Gauss Seidel

```
function ESAM445_RedBlackGaussSeidel_MingfuLiang(K,N,starting_method)
% Author: Mingfu Liang
% Date: 2019/05/07
```

```
%
% Implementation of Red Black Gauss Seidel Method for solving Helmholtz
  equation.
%
% Input:
%          K:
%                Input -2, 0 or 2 to choose different Helmholtz equation.
%
%                the parameter K in the Helmholtz equation. In this homework
%                we only have three choice of K as -2, 0, 2.
%
%          N:
%                 Input 16, 32 or 64 to choose the grid size.
%
%                 the size of the grid we are going to create. In this
%                 homework we have three alternative choice which is N=16
%                 N=32 and N=64.
%
%          starting_method:
%                 Input 1 or 2 to choose the starting method.
%
%                 the choice of the starting data we are going to use. In
%                 this homework we have two type of starting method, which
%                 are:
%                        (a) u_{i,j} = 1
%                        (b) u_{i,j} = (-1)^{i+j}
%
%                  To use the starting method (a), please input 1.
%                  To use the starting method (b), please input 2.
%
% Example:
%
%           ESAM445_RedBlackGaussSeidel_MingfuLiang(-2,16,1)
%
%           means that you are going to set the grid size N=16 to do the
%           Red Black Gauss Seidel method for Helmholtz equation when K=-2
  and
%           using the starting data (a) u_{i,j}=1.
%

```

```
tic
%%  initialize parameter

nonhom = @(X,Y)  32.*X.*Y.*(X-1).*(1-Y); % define the nonhomogenous part of the equation, which is the right handside
h      = 1/(N+1);         % grid spacing. Here I use N+1 since I want to be consistent with the notebook
tol    = 1e-7;          % tolerance

u    = zeros(N+2,N+2);     % storage for solution, here I use N+2 since I want to be consistent with the notebook
                                    % we denote that 0,1,2,...,N,N+1, which means
                                    % that if we have N=16, we actually have N+2
                                    % points although the at 0 and N+1 it should
                                    % be zero as the boundary condition in this
                                    % Homework since we are setting at all the
                                    % boundary u=0.

res       = u;% storage for residual, here residual is a matrix
loop_count      = 0;          % while loop counter
[X,Y] = meshgrid(0:h:1,0:h:1); % coordinates for final solution generation.
f      = nonhom(X,Y); % generate the nonhomogenous part matrix

%%

%Now start the iteration solver, stop when
%infinite norm of the residual vector < tolerance

figure;

%%% vectorized the iteration, generate the index set such that all the calculation can done simultaneous %%%%%%%%

%%% initial vectorize index for generating intial guess of starting points %%%%
```

```

l  =2:N+1;
m=2:N+1;

% generate red & black point index

% red_i = 2:2:N;
% red_j = 2:2:N;
%
% black_i  =2:2:N;
% black_j  =3:2:N+1;
%
% red_red_i=3:2:N+1;
% red_red_j=3:2:N+1;
%
% black_black_i  =3:2:N+1;
% black_black_j  =2:2:N;

%%%% initial guess of starting points %%%%

if starting_method ==1
    u(l,m)=1;
    starting_name = '(a)';
end

if starting_method ==2
    for i  =2:N+1
        for j  =2:N+1
            u(i,j)=(-1)^(i+j);
        end
    end
    starting_name = '(b)';
end

%%%% initial the boundary value %%%%
%%%% In this homework we set u=0 at all the boundary %%%%

    u(1,:)  =0;
    u(:,1)  =0;
    u(N+2,:)  =0;
    u(:,N+2) =0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while 1

    %%% Red Black Gauss Seidel Implementation. Here I
    %%% first update all the red point, which means the index
    %%% i and j of u(i,j) should be mod(i+j,2)==0, which implies the
    %%% even points. Then I update the black point, which is defined
    %%% similar before.

    %%%%%%%%%%%%%% Update Red points %%%%%%%%%%%%%%%%%

    for i = 2:N+1
        for j= 2:N+1
            if mod(i+j,2)==0
                u(i,j) = (1/(4 - (h^2)*pi*pi*K))*( u(i,j+1) + u(i,j-1) +u(i
                    -1,j) + u(i+1,j) -h^2*f(i,j));
            end
        end
    end

    %%%%%%%%%%%%%% Update black points %%%%%%%%%%%%%%%%%

     for i = 2:N+1
        for j= 2:N+1
            if mod(i+j,2)==1
                u(i,j) = (1/(4 - (h^2)*pi*pi*K))*( u(i,j+1) + u(i,j-1) +u(i
                    -1,j) + u(i+1,j) -h^2*f(i,j));
            end
        end
     end

    res(l,m) = f(l,m) - (1/h^2)*( u(l-1,m) + u(l+1,m) + u(l,m-1) + u(l,m+1)
       - (4-(h^2)*pi*pi*K)*u(l,m) ); % get the corresponding residual matrix
        for each individual solution

    %%% convergence check using infinite vector norm of residue matrix
    %%% be careful here you should first use res(:) to change the matrix to
    %%% a vector so that you can use vector norm correspondingly.

    loop_count = loop_count+1; % update the iteration count of loop

    if norm(res(:),inf) < tol
        break
    end

    %%%% initial the boundary value %%%%

    u(1,:) =0;
    u(:,1) =0;
    u(N+2,:) =0;
    u(:,N+2) =0;

end

%%% plot the solution at convergence iteration

mesh(X,Y,u);
title1 = ['K=', num2str(K),',N=',num2str(N), ', Red Black Gauss Seidel
    Solution at ',num2str(loop_count),' iteration, starting data ',
    starting_name];
title(title1)
toc
end
```

16.5 SOR

```
function ESAM445_SOR_MingfuLiang(K,N,omega,starting_method)
% Author: Mingfu Liang
% Date: 2019/05/07
%
% Implementation of SOR Method for solving Helmholtz equation.
%
% Input:
%           K:
%                   Input -2, 0 or 2 to choose different Helmholtz equation.
%
%                   the parameter K in the Helmholtz equation. In this homework
%                   we only have three choice of K as -2, 0, 2.
```

```
%
%           N:
%                 Input 16, 32 or 64 to choose the grid size.
%
%                 the size of the grid we are going to create. In this
%                 homework we have three alternative choice which is N=16
%                 N=32 and N=64.
%           omega:
%                 Input value between 1 to 2 for weight since
%                 overrelaxation.
%
%                The weight parameter for SOR method.
%
%           starting_method:
%                 Input 1 or 2 to choose the starting method.
%
%                 the choice of the starting data we are going to use. In
%                 this homework we have two type of starting method, which
%                 are:
%                        (a) u_{i,j} = 1
%                        (b) u_{i,j} = (-1)^{i+j}
%                  To use the starting method (a), please input 1.
%                  To use the starting method (b), please input 2.
%
% Example:
%
%            ESAM445_SOR_MingfuLiang(-2,16,1.8,1)
%
%            means that you are going to set the grid size N=16 and omega
%            =1.8 to do the SOR method for Helmholtz equation
%            when K=-2 and using the starting data (a) u_{i,j}=1.
%


tic
%%% initialize parameter
w = omega;
nonhom = @(X,Y) 32.*X.*Y.*(X-1).*(1-Y); % define the nonhomogenous part of
    the equation, which is the right handside
h     = 1/(N+1);        % grid spacing. Here I use N+1 since I want to be
    consistent with the notebook
```

```
tol    = 1e-7;           % tolerance

u    = zeros(N+2,N+2);     % storage for solution, here I use N+2 since I want
     to be consistent with the notebook
                                                 % we denote that 0,1,2,...,N,N+1,
                                                     which means
                                                 % that if we have N=16, we actually
                                                     have N+2
                                                 % points although the at 0 and N+1
                                                     it should
                                                 % be zero as the boundary condition
                                                     in this
                                                 % Homework since we are setting at
                                                     all the
                                                 % boundary u=0.

res      = u;% storage for residual, here residual is a matrix
loop_count      = 0;          % while loop counter
[X,Y] = meshgrid(0:h:1,0:h:1); % coordinates for final solution generation.
f      = nonhom(X,Y); % generate the nonhomogenous part matrix

%%

%Now start the iteration solver, stop when
%infinite norm of the residual vector < tolerance

figure;

%%% vectorized the iteration, generate the index set such that all the
    calculation can done simultaneous %%%%%%%%

%%% initial vectorize index for generating intial guess of starting points
    %%%

l  =2:N+1;
m=2:N+1;

%%% initial guess of starting points %%%

if starting_method ==1
```

```
    u(l,m)=1;
    starting_name = '(a)';
end

if starting_method ==2
    for i =2:N+1
        for j =2:N+1
            u(i,j)=(-1)^(i+j);
        end
    end
    starting_name = '(b)';
end

%%% generate the index to do vectorize calculation later

i = 2:N+1; % start from 2 and end at N+1 since I don't want to update the
    boundary value since they should be zero all the time
j = 2:N+1; % start from 2 and end at N+1 since I don't want to update the
    boundary value since they should be zero all the time

%%% initial the boundary value %%%
%%% In this homework we set u=0 at all the boundary %%%

    u(1,:) =0;
    u(:,1) =0;
    u(N+2,:) =0;
    u(:,N+2) =0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while 1

    %% SOR Implementation. Here in SOR, the update of each individual
    %% element in solution matrix is dependent with the previous update.
    %% Here I use the Forward Euler to get the general update formula
    %% for the u(i,j). The outer loop is in i and inner loop is in j,
    %% which mean the outer loop is for row loop and inner loop is for
    %% column loop. For example for i=1, which means for the first row,
    %% we loop j=1:N+1, which means loop for all the columns in first row.
    %% For example for i=1, j=1, we first update u(1,1), then i is still 1
    %%% and j become 2, and since I just use the same u matrix to storage
    %%% the solution, therefore when calculate u(1,2) by the fomular below,
    %%% it automatically use the updated u(1,1) by the previous iteration.
        So
    %%% I save the memory by just using one matrix to storage and update
    %%% the solution and also satisfies the requirement of SOR.

     for i=2:N+1
         for j=2:N+1
             u(i,j) = (1-w)*u(i,j) +w*(1/(4 - (h^2)*pi*pi*K))*( u(i,j+1) + u(
                 i,j-1) +u(i-1,j) + u(i+1,j) -h^2*f(i,j));
         end
     end


     res(l,m) = f(l,m) - (1/h^2)*( u(l-1,m) + u(l+1,m) + u(l,m-1) + u(l,m+1)
        - (4-(h^2)*pi*pi*K)*u(l,m) ); % get the corresponding residual matrix
          for each individual solution
     loop_count = loop_count+1; % update the iteration count of loop

    %%% convergence check using infinite vector norm of residue matrix
    %%% be careful here you should first use res(:) to change the matrix to
    %%% a vector so that you can use vector norm correspondingly.

     if norm(res(:),inf) < tol
          break
     end

end

%%% plot the solution at convergence iteration

mesh(X,Y,u);
title1 = ['K=', num2str(K), ',omega=',num2str(w),',N=',num2str(N), ',SOR
    Solution at ',num2str(loop_count),' iteration, starting data ',
    starting_name];
title(title1)
toc
end
```

16.6 SSOR

```
function ESAM445_SSOR_MingfuLiang(K,N,omega,starting_method)
% Author: Mingfu Liang
% Date: 2019/05/07
%
% Implementation of SSOR Method for solving Helmholtz equation.
%
% Input:
%           K:
%                  Input -2, 0 or 2 to choose different Helmholtz equation.
%
%                  the parameter K in the Helmholtz equation. In this homework
%                  we only have three choice of K as -2, 0, 2.
%
%           N:
%                   Input 16, 32 or 64 to choose the grid size.
%
%                   the size of the grid we are going to create. In this
%                   homework we have three alternative choice which is N=16
%                   N=32 and N=64.
%           omega:
%                   Input value between 1 to 2 for weight since
%                   overrelaxation.
%
%                  The weight parameter for SSOR method.
%
%           starting_method:
%                   Input 1 or 2 to choose the starting method.
%
%                   the choice of the starting data we are going to use. In
%                   this homework we have two type of starting method, which
%                   are:
%                          (a) u_{i,j} = 1
%                          (b) u_{i,j} = (-1)^{i+j}
%                    To use the starting method (a), please input 1.
%                    To use the starting method (b), please input 2.
%
% Example:
%
```

```
%              ESAM445_SSOR_MingfuLiang(-2,16,1.8,1)
%
%              means that you are going to set the grid size N=16 and omega
%              =1.8 to do the SSOR method for Helmholtz equation
%              when K=-2 and using the starting data (a) u_{i,j}=1.
%


tic
%%% initialize parameter
w = omega;
nonhom = @(X,Y) 32.*X.*Y.*(X-1).*(1-Y); % define the nonhomogenous part of
    the equation, which is the right handside
h      = 1/(N+1);         % grid spacing. Here I use N+1 since I want to be
    consistent with the notebook
tol    = 1e-7;          % tolerance

u    = zeros(N+2,N+2);     % storage for solution, here I use N+2 since I want
     to be consistent with the notebook
                                              % we denote that 0,1,2,...,N,N+1,
                                                 which means
                                              % that if we have N=16, we actually
                                                 have N+2
                                              % points although the at 0 and N+1
                                                 it should
                                              % be zero as the boundary condition
                                                 in this
                                              % Homework since we are setting at
                                                 all the
                                              % boundary u=0.

res      = u;% storage for residual, here residual is a matrix
loop_count      = 0;          % while loop counter
[X,Y] = meshgrid(0:h:1,0:h:1); % coordinates for final solution generation.
f      = nonhom(X,Y); % generate the nonhomogenous part matrix

%%%

%Now start the iteration solver, stop when
%infinite norm of the residual vector < tolerance

```

```
figure;

%%%% vectorized the iteration, generate the index set such that all the
    calculation can done simultaneous %%%%%%%%%%

%%%% initial vectorize index for generating intial guess of starting points
    %%%%

l =2:N+1;
m=2:N+1;

%%%% initial guess of starting points %%%%

if starting_method ==1
    u(l,m)=1;
    starting_name = '(a)';
end

if starting_method ==2
    for i =2:N+1
        for j =2:N+1
            u(i,j)=(-1)^(i+j);
        end
    end
    starting_name = '(b)';
end

%%%% generate the index to do vectorize calculation later

i = 2:N+1; % start from 2 and end at N+1 since I don't want to update the
    boundary value since they should be zero all the time
j = 2:N+1; % start from 2 and end at N+1 since I don't want to update the
    boundary value since they should be zero all the time

%%%% initial the boundary value %%%%
%%%% In this homework we set u=0 at all the boundary %%%%

    u(1,:) =0;
    u(:,1) =0;
    u(N+2,:) =0;
    u(:,N+2) =0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while 1

    %%% SSOR Implementation. Here in SSOR, the update of each individual
    %%% element in solution matrix is dependent with the previous update.
    %%% Here I use the Forward Euler to get the general update formula
    %%% for the u(i,j). The outer loop is in i and inner loop is in j,
    %%% which mean the outer loop is for row loop and inner loop is for
    %%% column loop. For example for i=1, which means for the first row,
    %%% we loop j=1:N+1, which means loop for all the columns in first row.
    %%% For example for i=1, j=1, we first update u(1,1), then i is still 1
    %%% and j become 2, and since I just use the same u matrix to storage
    %%% the solution, therefore when calculate u(1,2) by the fomular below,
    %%% it automatically use the updated u(1,1) by the previous iteration.
        So
    %%% I save the memory by just using one matrix to storage and update
    %%% the solution and also satisfies the requirement of SSOR. To
    %%% satisfy the requirement of the SSOR, I sweep the grid backward such
    %%% that it remove the asymmetry from SOR.

    %%%%%%%%% forward %%%%%%%%%%%%

    for i=2:N+1
        for j=2:N+1
            u(i,j) = (1-w)*u(i,j) +w*(1/(4 - (h^2)*pi*pi*K))*( u(i,j+1) + u(
                i,j-1) +u(i-1,j) + u(i+1,j) -h^2*f(i,j));
        end
    end

    %%%%%%%%% backward %%%%%%%%%%%%

    for i=N+1:-1:2
        for j=N+1:-1:2
            u(i,j) = (1-w)*u(i,j) +w*(1/(4 - (h^2)*pi*pi*K))*( u(i,j+1) + u(
                i,j-1) +u(i-1,j) + u(i+1,j) -h^2*f(i,j));
        end
    end
```

```matlab


    res(l,m) = f(l,m) - (1/h^2)*( u(l-1,m) + u(l+1,m) + u(l,m-1) + u(l,m+1) - (4-(h^2)*pi*pi*K)*u(l,m) ); % get the corresponding residual matrix for each individual solution
    loop_count = loop_count+1; % update the iteration count of loop

    %%% convergence check using infinite vector norm of residue matrix
    %%% be careful here you should first use res(:) to change the matrix to
    %%% a vector so that you can use vector norm correspondingly.

    if norm(res(:),inf) < tol
        break
    end

end

%%% plot the solution at convergence iteration

mesh(X,Y,u);
title1 = ['K=', num2str(K), ',omega=',num2str(w),',N=',num2str(N), ',SSOR Solution at ',num2str(loop_count),' iteration, starting data ', starting_name];
title(title1)
toc
end
```