上机题第四题实验报告

王伟任　　计 33　　2013011333

一、题目要求及分析

第四章上机题 1：考虑 10 阶 Hilbert 矩阵作为系数阵的方程组$\boldsymbol{Ax}=\boldsymbol{b}$，其中$\boldsymbol{b}$=$[1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\frac{1}{5},\frac{1}{6},\frac{1}{7},\frac{1}{8},\frac{1}{9},\frac{1}{10}]^T$。取初始解$\boldsymbol{x}^{(0)}=\boldsymbol{0}$，编写程序用 Jacobi 与 SOR 迭代法求解该方程组，将$||\boldsymbol{x}^{(k+1)}-\boldsymbol{x}^{(k)}||_\infty<10^{-4}$作为终止迭代的判据。

（1）分别用 Jacobi 与 SOR（$\omega=1.25$）迭代法求解，观察收敛情况；

（2）改变ω的值，试验 SOR 迭代法的效果，考察解的准确度。

迭代法求解是否收敛的判断依据就是迭代矩阵的谱半径是否小于一，当谱半径小于 1 时迭代法收敛。但是由于谱半径求解算法实现起来较为复杂，因此不另作计算。

二、实验结果及分析

（1） 利用 Jacobi 迭代法求解：

利用 Jacobi 迭代法求解时，迭代法不收敛，得到的结果很快达到数据类型上限。迭代过程记录储存在 test1.txt 中，记录了每一次迭代的近似解。部分截图如下：

1.000000	1.500000	1.666667	1.750000	1.800000	1.833333	1.857143	1.875000	1.888889	1.900000
-2.308169	-6.391991	-9.384380	-11.663481	-13.466459	-14.934078	-16.155150	-17.188848	-18.076355	-18.847342
23.772000	54.483063	75.814960	92.025143	104.910442	115.459057	124.282671	131.787833	138.258356	143.899769
-178.724898	-419.430157	-587.585510	-715.174253	-816.383023	-899.083765	-968.151618	-1026.821516	-1077.346977	-1121.356475
1396.995308	3267.253438	4573.073625	5564.257480	6350.831367	6993.789729	7530.913653	7987.283731	8380.379477	8722.837483
-10861.539156	-25414.696857	-35576.295377	-43288.914506	-49408.997203	-54411.369740	-58590.133051	-62140.507585	-65198.538508	-67862.571061
84507.824863	197725.373932	276778.043673	336779.441579	384391.979102	423309.372263	455819.481185	483440.977821	507232.206659	527958.239045
-657448.002062	-1538263.297644	-2153281.244744	-2620082.344186	-2990499.565988	-3293269.718795	-3546192.137928	-3761081.924365	-3946172.867166	-4107417.117422
5114831.330489	11967407.489261	16752130.549147	20383755.525523	23265532.629880	25621028.510147	27588718.485623	29260521.738587	30700495.559208	31954946.544655

由此可知，此时的迭代矩阵的谱半径大于等于一，不满足收敛条件。

（2） 利用 SOR（$\omega=1.25$）迭代法求解：

利用 SOR（$\omega=1.25$）迭代法求解时，迭代法收敛。

最终得到的近似解$\hat{\boldsymbol{x}}$为

$[1.002023,-0.021610,0.051496,-0.028416,-0.006260,-0.006193,-0.000537,0.001835,0.003637,0.004567$

一共迭代 187 步。

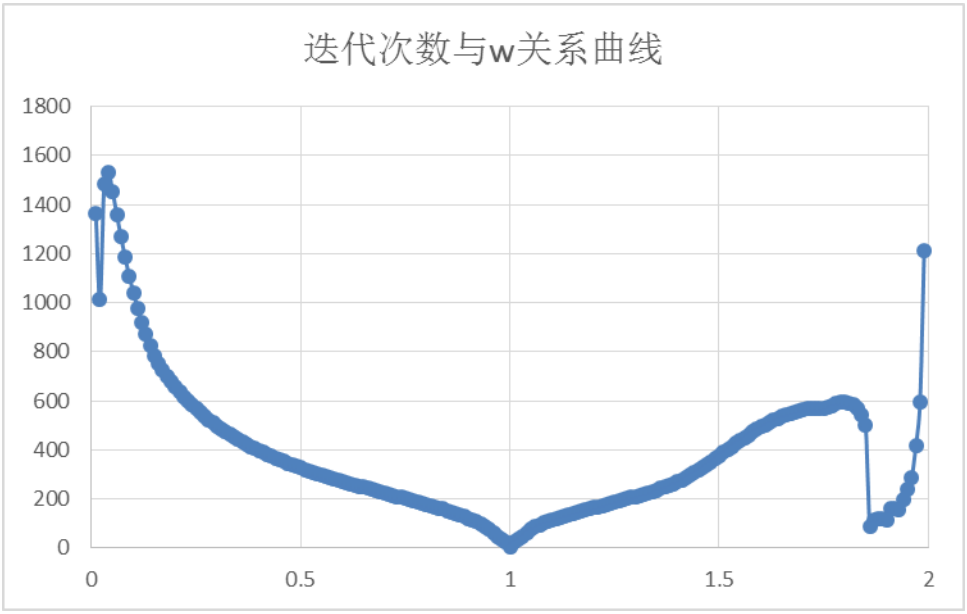
迭代过程记录储存在 test2.txt 中，记录了每一次迭代的近似解。部分截图如下：

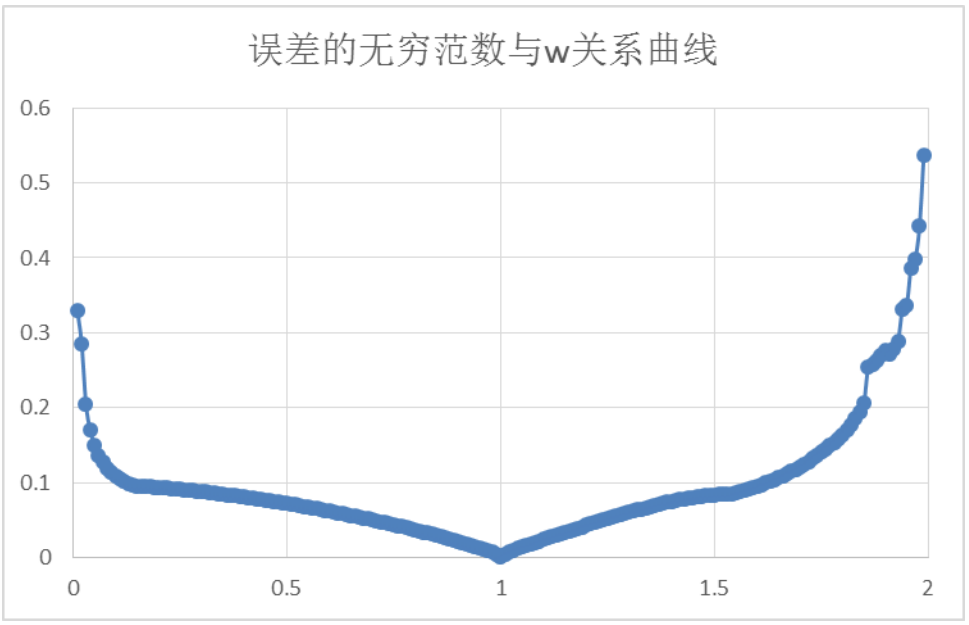
1.002652	-0.025753	0.056758	-0.028606	-0.006331	-0.006805	-0.001028	0.001471	0.003449	0.004536
1.002631	-0.025619	0.056599	-0.028608	-0.006335	-0.006790	-0.001014	0.001484	0.003459	0.004543
1.002611	-0.025487	0.056442	-0.028610	-0.006338	-0.006775	-0.000999	0.001497	0.003468	0.004549
1.002590	-0.025357	0.056287	-0.028611	-0.006342	-0.006760	-0.000985	0.001509	0.003478	0.004555
1.002570	-0.025230	0.056134	-0.028611	-0.006344	-0.006745	-0.000971	0.001522	0.003486	0.004560
1.002551	-0.025104	0.055983	-0.028612	-0.006347	-0.006730	-0.000957	0.001534	0.003495	0.004565
1.002531	-0.024981	0.055834	-0.028611	-0.006349	-0.006715	-0.000943	0.001546	0.003503	0.004570
1.002512	-0.024860	0.055687	-0.028611	-0.006351	-0.006700	-0.000930	0.001557	0.003511	0.004574
1.002494	-0.024740	0.055542	-0.028610	-0.006352	-0.006685	-0.000916	0.001569	0.003518	0.004578
1.002476	-0.024623	0.055398	-0.028609	-0.006353	-0.006670	-0.000903	0.001580	0.003526	0.004582
1.002458	-0.024507	0.055257	-0.028608	-0.006354	-0.006655	-0.000890	0.001591	0.003533	0.004585
1.002440	-0.024393	0.055117	-0.028606	-0.006355	-0.006640	-0.000877	0.001601	0.003540	0.004589
1.002423	-0.024281	0.054979	-0.028604	-0.006355	-0.006625	-0.000864	0.001612	0.003546	0.004591
1.002406	-0.024171	0.054843	-0.028601	-0.006355	-0.006610	-0.000852	0.001622	0.003552	0.004594
1.002389	-0.024063	0.054708	-0.028598	-0.006355	-0.006595	-0.000839	0.001632	0.003558	0.004596
1.002373	-0.023956	0.054575	-0.028595	-0.006355	-0.006580	-0.000827	0.001642	0.003564	0.004598
1.002357	-0.023851	0.054443	-0.028592	-0.006354	-0.006566	-0.000815	0.001651	0.003569	0.004600
1.002341	-0.023748	0.054313	-0.028588	-0.006353	-0.006551	-0.000803	0.001660	0.003574	0.004601
1.002325	-0.023646	0.054185	-0.028584	-0.006352	-0.006536	-0.000791	0.001669	0.003579	0.004603
1.002310	-0.023545	0.054058	-0.028580	-0.006350	-0.006522	-0.000779	0.001678	0.003584	0.004604
1.002295	-0.023447	0.053933	-0.028575	-0.006348	-0.006507	-0.000767	0.001687	0.003588	0.004604
1.002280	-0.023350	0.053809	-0.028570	-0.006347	-0.006492	-0.000755	0.001695	0.003593	0.004605
1.002266	-0.023254	0.053686	-0.028565	-0.006344	-0.006478	-0.000744	0.001704	0.003597	0.004605
1.002252	-0.023160	0.053565	-0.028560	-0.006342	-0.006463	-0.000733	0.001712	0.003600	0.004605
1.002238	-0.023067	0.053445	-0.028554	-0.006339	-0.006449	-0.000721	0.001720	0.003604	0.004605
1.002224	-0.022975	0.053327	-0.028549	-0.006336	-0.006434	-0.000710	0.001727	0.003607	0.004604
1.002210	-0.022885	0.053210	-0.028543	-0.006333	-0.006420	-0.000699	0.001735	0.003610	0.004604
1.002197	-0.022797	0.053094	-0.028536	-0.006330	-0.006405	-0.000688	0.001742	0.003613	0.004603
1.002184	-0.022709	0.052979	-0.028530	-0.006327	-0.006391	-0.000678	0.001750	0.003616	0.004602
1.002171	-0.022623	0.052866	-0.028523	-0.006323	-0.006377	-0.000667	0.001757	0.003619	0.004600
1.002159	-0.022538	0.052754	-0.028516	-0.006319	-0.006362	-0.000656	0.001764	0.003621	0.004599
1.002146	-0.022455	0.052643	-0.028509	-0.006315	-0.006348	-0.000646	0.001770	0.003623	0.004597
1.002134	-0.022372	0.052533	-0.028501	-0.006311	-0.006334	-0.000636	0.001777	0.003625	0.004595
1.002122	-0.022291	0.052425	-0.028494	-0.006307	-0.006320	-0.000625	0.001783	0.003627	0.004593
1.002110	-0.022211	0.052317	-0.028486	-0.006302	-0.006305	-0.000615	0.001790	0.003629	0.004591
1.002099	-0.022132	0.052211	-0.028478	-0.006297	-0.006291	-0.000605	0.001796	0.003630	0.004589
1.002087	-0.022054	0.052106	-0.028470	-0.006292	-0.006277	-0.000595	0.001802	0.003632	0.004586
1.002076	-0.021978	0.052001	-0.028461	-0.006287	-0.006263	-0.000585	0.001808	0.003633	0.004583
1.002065	-0.021902	0.051898	-0.028453	-0.006282	-0.006249	-0.000576	0.001813	0.003634	0.004581
1.002054	-0.021827	0.051796	-0.028444	-0.006277	-0.006235	-0.000566	0.001819	0.003635	0.004577
1.002043	-0.021754	0.051695	-0.028435	-0.006271	-0.006221	-0.000556	0.001824	0.003636	0.004574
1.002033	-0.021681	0.051595	-0.028426	-0.006265	-0.006207	-0.000547	0.001830	0.003636	0.004571
1.002023	-0.021610	0.051496	-0.028416	-0.006260	-0.006193	-0.000537	0.001835	0.003637	0.004567
Answer is:									
1.002023	-0.021610	0.051496	-0.028416	-0.006260	-0.006193	-0.000537	0.001835	0.003637	0.004567
counter = 187									

由以上可知，此时的迭代矩阵的谱半径小于一，满足收敛条件。

（3） 改变ω的值，试验 SOR 迭代法的效果

令ω从 0.01 变化到 1.99，每次增加 0.01，观察在每一个ω下迭代法的收敛情况。计算结果显示，在 0.01-1.99 范围内，无论ω取何值，SOR 迭代法均收敛，但是收敛速度不同，得到近似解的误差大小也不同。当ω=1.00 时，迭代速度最快，只需要迭代两步即可得到解，而且误差也最小（为零）。





全部计算结果储存在 test0.txt 中。部分截图如下：

```
w = 0.980000
Answer is:
0.998921        0.006518        -0.006982       -0.001337       0.000347        0.000724        0.000718        0.000608        0.000483        0.000371
wu cha fan shu: 0.006982
counter = 33

w = 0.990000
Answer is:
0.999265        0.003955        -0.003804       -0.000885       0.000041        0.000313        0.000369        0.000352        0.000313        0.000269
wu cha fan shu: 0.003955
counter = 22

w = 1.000000
Answer is:
1.000000        -0.000000       0.000000        0.000000        0.000000        -0.000000       0.000000        0.000000        -0.000000       0.000000
wu cha fan shu: 0.000000
counter = 2

w = 1.010000
Answer is:
1.000724        -0.003938       0.003889        0.000779        -0.000065       -0.000313       -0.000361       -0.000341       -0.000301       -0.000258
wu cha fan shu: 0.003938
counter = 22

w = 1.020000
Answer is:
1.001049        -0.006453       0.007234        0.000942        -0.000418       -0.000705       -0.000674       -0.000560       -0.000437       -0.000330
wu cha fan shu: 0.007234
counter = 33
```

在这些情况下迭代矩阵的谱半径均小于 1，满足收敛条件。

三、实验代码

采用 C++语言实现。

仅附 Jacobi 迭代法和试验不同 ω 条件下 SOR 迭代法效果的代码：

Jacobi 迭代法代码：

```
#include <cstdio>
#include <cmath>

int n = 10;

double** createHilbert(){
    double** temp = new double*[n];
    for (int i = 0; i < n; i++){
        temp[i] = new double[n];
        for (int j = 0; j < n; j++)
            temp[i][j] = 1.0 / (i + j + 1);
```

```
        }
        return temp;
}

double* multiply(double** A, double* bl){
        double* ans = new double[n];
        for (int i = 0; i < n; i++){
                ans[i] = 0;
                for (int j = 0; j < n; j++)
                        ans[i] += A[i][j] * bl[j];
        }
        return ans;
}

double norm(double* A, double* B){
        double mmax = 0.0;
        for (int i = 0; i < n; i++)
                if (fabs(A[i] - B[i]) > mmax) mmax = fabs(A[i] - B[i]);
        return mmax;
}

int main(){
        double** Hilbert = createHilbert();
        double* b = new double[n];
        for (int i = 0; i < n; i++) b[i] = 1.0 / (i + 1);
        double* x = new double[n];
        double* y = new double[n];
        for (int i = 0; i < n; i++) x[i] = 0;

        FILE* fp = fopen("test1.txt", "w");

        do{                                             //Jacobi
                for (int i = 0; i < n; i++) y[i] = x[i];
                for (int i = 0; i < n; i++){
                        double s = 0.0;
                        for (int j = 0; j < n; j++)
                                if (j != i) s += Hilbert[i][j] * y[j];
                        x[i] = (b[i] - s) / Hilbert[i][i];
                }

                for (int i = 0; i < n; i++)
                        fprintf(fp, "%f\t", x[i]);
                fprintf(fp, "\n");
```

```
    } while(norm(x, y) >= 0.0001);


    fprintf(fp, "Answer is:\n");
    for (int i = 0; i < n; i++)
        fprintf(fp, "%f\t", x[i]);
    fprintf(fp, "\n");
    fclose(fp);


    for (int i = 0; i < n; i++)
        delete[] Hilbert[i];
    delete[] Hilbert;
    delete[] b;
    delete[] x;
    delete[] y;
    return 0;
}
```

SOR 迭代法代码：

```
#include <cstdio>
#include <cmath>

int n = 10;
double w = 0.01;

double** createHilbert(){
    double** temp = new double*[n];
    for (int i = 0; i < n; i++){
        temp[i] = new double[n];
        for (int j = 0; j < n; j++)
            temp[i][j] = 1.0 / (i + j + 1);
    }
    return temp;
}

double norm(double* A, double* B){
    double mmax = 0.0;
    for (int i = 0; i < n; i++)
        if (fabs(A[i] - B[i]) > mmax) mmax = fabs(A[i] - B[i]);
    return mmax;
}

int main(){
```

```
double** Hilbert = createHilbert();
double* b = new double[n];
for (int i = 0; i < n; i++) b[i] = 1.0 / (i + 1);
double* x = new double[n];
double* y = new double[n];
double* x0 = new double[n];
x0[0] = 1.0;
for (int i = 1; i < n; i++) x0[i] = 0;

FILE* fp = fopen("test0.txt", "w");

for (;w < 2; w += 0.01){
	for (int i = 0; i < n; i++) x[i] = 0;
	int counter = 0;
	do{ 				//SOR
		for (int i = 0; i < n; i++) y[i] = x[i];
		for (int i = 0; i < n; i++){
			double s = 0.0;
			for (int j = 0; j < n; j++)
				if (j != i) s += Hilbert[i][j] * x[j];
			x[i] = (1 - w) * x[i] + w * (b[i] - s) / Hilbert[i][i];
		}
		counter++;
	} while(norm(x, y) >= 0.0001);

	fprintf(fp, "w = %f\n", w);
	fprintf(fp, "Answer is:\n");
	for (int i = 0; i < n; i++)
		fprintf(fp, "%f\t", x[i]);
	fprintf(fp, "\n");
	double nor = norm(x, x0);
	fprintf(fp, "wu cha fan shu: %f\n", nor);
	fprintf(fp, "counter = %d\n\n", counter);
}
fclose(fp);


for (int i = 0; i < n; i++)
	delete[] Hilbert[i];
delete[] Hilbert;
delete[] b;
delete[] x;
delete[] y;
return 0;
```

```
}
```