将希尔伯特矩阵条件数结果输出至文件，然后利用excel 画图，横坐标为矩阵的维数，纵坐标是条件数的对数，近似成一条直线，说明希尔伯特矩阵的条件数随维数呈指数级增加。

（n >= 14 时，double型数据越界，所以后两个数据我是用matlab算出来的）

上图为解方程的运行结果：对于n=6，9，15分别输出LU法、Jacobi法、GS法的解

condr 为余项r=b-Hx的∞范数 condx 为解的误差的∞范数

可见，雅克比法求解希尔伯特矩阵是发散的（调用该函数进行运算时，迭代次数过多，返回地址为0，所以输出 -1.#IND00）

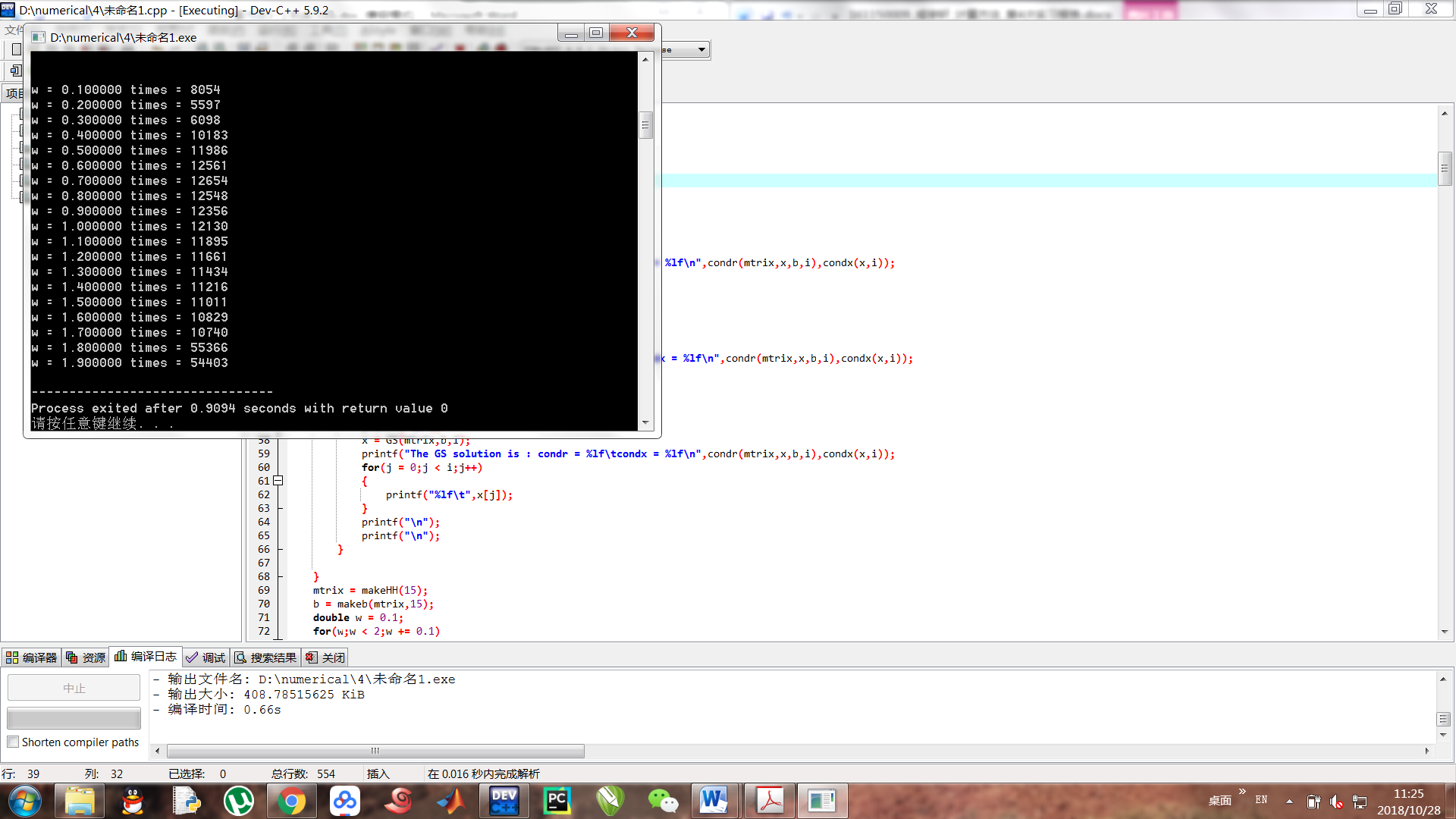
LU法的解精度随着矩阵维数增大而减小，n=6 condr = condx = 0

n=9 condr = 0 condx = 0.000023 n = 15 condr = 0 condx = 8.695876

GS 法的解精度随着矩阵维数增大而减小，但GS法是三者精度最高的方法。

n=6 condr = condx =0 n= 9 condr = 0 condx = 0.025836

n= 15 condr = 0 condx = 0.02400.



上图为n =15时，通过超松弛迭代来求方程组的解。超松弛因子在0到2之间变化时，收敛速度的变化。W为超松弛因子，times 为超松弛迭代次数，可见在w=0.2附近迭代速度最快，w=1 时 即为GS迭代，所以通过超松弛迭代因子的设定可以减小迭代次数，增加效率。

总结：

病态矩阵可通过条件数来甄别，当条件数特别大时，矩阵方程两边有微小的扰动都会导致解变化很大。所以不适合用直接法解，本次实验用其他三种方法：LU、Jacobi,GS。

不同迭代方法的迭代矩阵不同，对于相同的矩阵，其迭代效果也会不同，所以要选择合适的迭代方法。

在用超松弛迭代法时，选择合适的松弛因子可以增加收敛速度，因此确立合适的松弛因子十分重要。

以下为程序流程图及代码



#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define N 15

#define epsilon 1e-6

double \*\*makeHH(int n);//make hillbert

double condA(double \*\*a,int n);//return fanshu

double \*\*LUni(double \*\*U,double \*\*L,int n);// return U-1\*L-1

double \*\*ALU(double \*\*a,double n);//L U fenjie,return A-1

double \*makeb(double \*\*a,double n);//make vector b[n] and return

double \*LUsolve(double \*\*a,double \*b,int n);//sovle equation im LU method, return x

double \*Jacobi(double \*\*a,double \*b,int n);//sovle equation in Jacobi method, return x

double \*GS(double \*\*a,double \*b,int n);//sovle equation in GS method, return x

double condr(double \*\*a,double \*x,double \*b,int n);//return condition(r)

double condx(double \*x,int n);//return condition(delta x)

void multi(double \*\*a,double \*\*b,int n,double \*\*ans);//matrix multiply

int relax(double \*\*a,double \*b,double n,double w);//solve equation in over-relaxtion ineration

int main()

{

int i,j,k;

double \*\*mtrix;

double \*\*nimtrix;

double cond[20] = {0};

double \*b,\*x;

FILE \*fp;

fp = fopen("C:\\condition.txt","w+");

for(i = 1;i <= N;i++)

{

mtrix = makeHH(i);

cond[i] = condA(mtrix,i);

nimtrix = ALU(mtrix,i);

cond[i] \*= condA(nimtrix,i);

fprintf(fp,"%e\n",cond[i]);

free(mtrix);

}

fclose(fp);

for(i = 0;i < 16;i++)

{

if(i == 6 || i == 9 || i == 15)

{

printf("x = %d:\n",i);

mtrix = makeHH(i);

b = makeb(mtrix,i);

x = LUsolve(mtrix,b,i);

printf("The LU solution is : condr = %lf\tcondx = %lf\n",condr(mtrix,x,b,i),condx(x,i));

for(j = 0;j < i;j++)

{

printf("%lf\t",x[j]);

}

x = Jacobi(mtrix,b,i);

printf("\n");

printf("The Jacobi solution is :condr = %lf\tcondx = %lf\n",condr(mtrix,x,b,i),condx(x,i));

for(j = 0;j < i;j++)

{

printf("%lf\t",x[j]);

}

printf("\n");

x = GS(mtrix,b,i);

printf("The GS solution is : condr = %lf\tcondx = %lf\n",condr(mtrix,x,b,i),condx(x,i));

for(j = 0;j < i;j++)

{

printf("%lf\t",x[j]);

}

printf("\n");

printf("\n");

}

}

mtrix = makeHH(15);

b = makeb(mtrix,15);

double w = 0.1;

for(w;w < 2;w += 0.1)

{

printf("w = %lf times = %d\n",w,relax(mtrix,b,15,w));

}

return 0;

}

int relax(double \*\*a,double \*b,double n,double w)

{

double \*\*l = (double\*\*)malloc(n\*sizeof(double\*));

double \*\*u = (double\*\*)malloc(n\*sizeof(double\*));

double \*d = (double\*)malloc(n\*sizeof(double));

double \*newx = (double\*)malloc(n\*sizeof(double));

double \*oldx = (double\*)malloc(n\*sizeof(double));

double \*eps = (double\*)malloc(n\*sizeof(double));

double suml,sumu;

int i,j,k;

int count = 0;

for(i = 0;i < n;i++)

{

u[i] = (double\*)malloc(n\*sizeof(double));

l[i] = (double\*)malloc(n\*sizeof(double));

d[i] = a[i][i];

newx[i] = 0;

oldx[i] = 0;

eps[i] = 0;

}

for(i = 0;i < n;i++)

{

for(j = 0;j < n;j++)

{

u[i][j] = 0;

l[i][j] = 0;

}

}

for(i = 0;i < n;i++)

{

for(j = i+1;j < n;j++)

{

u[i][j] = a[i][j];

l[j][i] = a[j][i];

}

}

while(1)

{

count++;

for(i = 0;i < n;i++)

{

suml = 0;

sumu = 0;

for(j = 0;j < n;j++)

{

suml += l[i][j] \* newx[j];

sumu += u[i][j] \* oldx[j];

}

newx[i] = w \* (b[i] - sumu - suml) / d[i] + (1 - w) \* oldx[i];

}

for(i = 0;i < n;i++)

{

eps[i] = fabs(newx[i] - oldx[i]);

if(eps[i] > epsilon)

{

break;

}

}

if(i == n)

{

return count;

}

for(i = 0;i < n;i++)

{

oldx[i] = newx[i];

}

}

}

double condx(double \*x,int n)

{

double \*r = (double\*)malloc(n\*sizeof(double));

int i;

for(i = 0;i < n;i++)

{

r[i] = x[i] - 1;

}

double max = 0;

max = fabs(r[0]);

for(i = 1;i < n;i++)

{

if(max < fabs(r[i]))

{

max = fabs(r[i]);

}

}

return max;

}

double condr(double \*\*a,double \*x,double \*b,int n)

{

double \*r = (double\*)malloc(n\*sizeof(double));

int i,j;

for(i = 0;i < n;i++)

{

r[i] = 0;

}

for(i = 0;i < n;i++)

{

for(j = 0;j < n;j++)

{

r[i] += a[i][j] \* x[j];

}

r[i] = b[i] - r[i];

}

double max = 0;

max = fabs(r[0]);

for(i = 1;i < n;i++)

{

if(max < fabs(r[i]))

{

max = fabs(r[i]);

}

}

return max;

}

double \*GS(double \*\*a,double \*b,int n)

{

double \*\*l = (double\*\*)malloc(n\*sizeof(double\*));

double \*\*u = (double\*\*)malloc(n\*sizeof(double\*));

double \*d = (double\*)malloc(n\*sizeof(double));

double \*newx = (double\*)malloc(n\*sizeof(double));

double \*oldx = (double\*)malloc(n\*sizeof(double));

double \*eps = (double\*)malloc(n\*sizeof(double));

double suml,sumu;

int i,j,k;

for(i = 0;i < n;i++)

{

u[i] = (double\*)malloc(n\*sizeof(double));

l[i] = (double\*)malloc(n\*sizeof(double));

d[i] = a[i][i];

newx[i] = 0;

oldx[i] = 0;

eps[i] = 0;

}

for(i = 0;i < n;i++)

{

for(j = 0;j < n;j++)

{

u[i][j] = 0;

l[i][j] = 0;

}

}

for(i = 0;i < n;i++)

{

for(j = i+1;j < n;j++)

{

u[i][j] = a[i][j];

l[j][i] = a[j][i];

}

}

while(1)

{

for(i = 0;i < n;i++)

{

suml = 0;

sumu = 0;

for(j = 0;j < n;j++)

{

suml += l[i][j] \* newx[j];

sumu += u[i][j] \* oldx[j];

}

newx[i] = (b[i] - sumu - suml) / d[i];

}

for(i = 0;i < n;i++)

{

eps[i] = fabs(newx[i] - oldx[i]);

if(eps[i] > epsilon)

{

break;

}

}

if(i == n)

{

return newx;

}

for(i = 0;i < n;i++)

{

oldx[i] = newx[i];

}

}

}

double \*Jacobi(double \*\*a,double \*b,int n)

{

double \*\*l = (double\*\*)malloc(n\*sizeof(double\*));

double \*\*u = (double\*\*)malloc(n\*sizeof(double\*));

double \*d = (double\*)malloc(n\*sizeof(double));

double \*newx = (double\*)malloc(n\*sizeof(double));

double \*oldx = (double\*)malloc(n\*sizeof(double));

double \*eps = (double\*)malloc(n\*sizeof(double));

double suml,sumu;

int i,j,k;

for(i = 0;i < n;i++)

{

u[i] = (double\*)malloc(n\*sizeof(double));

l[i] = (double\*)malloc(n\*sizeof(double));

d[i] = a[i][i];

newx[i] = 0;

oldx[i] = 0;

eps[i] = 0;

}

for(i = 0;i < n;i++)

{

for(j = 0;j < n;j++)

{

u[i][j] = 0;

l[i][j] = 0;

}

}

for(i = 0;i < n;i++)

{

for(j = i+1;j < n;j++)

{

u[i][j] = a[i][j];

l[j][i] = a[j][i];

}

}

while(1)

{

for(i = 0;i < n;i++)

{

suml = 0;

sumu = 0;

for(j = 0;j < n;j++)

{

suml += l[i][j] \* oldx[j];

sumu += u[i][j] \* oldx[j];

}

newx[i] = (b[i] - sumu - suml) / d[i];

}

for(i = 0;i < n;i++)

{

eps[i] = fabs(newx[i] - oldx[i]);

if(eps[i] > epsilon)

{

break;

}

}

if(i == n)

{

return newx;

}

for(i = 0;i < n;i++)

{

oldx[i] = newx[i];

}

}

}

double \*makeb(double \*\*a,double n)

{

double \*b = (double\*)malloc(n\*sizeof(double));

int i,j;

double sum;

for(i = 0;i < n;i++)

{

sum = 0;

for(j = 0;j < n;j++)

{

sum += a[i][j];

}

b[i] = sum;

}

return b;

}

double \*LUsolve(double \*\*a,double \*b,int n)

{

double \*\*l = (double\*\*)malloc(n\*sizeof(double\*));

double \*\*u = (double\*\*)malloc(n\*sizeof(double\*));

int i, j,r, k;

for(i = 0;i < n;i++)

{

l[i] = (double\*)malloc(n\*sizeof(double));

u[i] = (double\*)malloc(n\*sizeof(double));

}

//make u row 1

for (i = 0; i<n; i++)

{

u[0][i] = a[0][i];

}

//make l column 1

for (i = 1; i<n; i++)

{

l[i][0] = a[i][0] / u[0][0];

}

//make u l

for (r = 1; r<n; r++)

{

for (i = r; i <n; i++)

{

double sum1 = 0;

for (k = 0; k < r; k++)

{

sum1 += l[r][k] \* u[k][i];

}

u[r][i] = a[r][i] - sum1;

}

if(r!=n)

for(i=r+1;i<n;i++)

{

double sum2 = 0;

for (k = 0; k<r; k++)

{

sum2 += l[i][k] \* u[k][r];

}

l[i][r] = (a[i][r] - sum2) / u[r][r];

}

}

double \*x = (double\*)malloc(n\*sizeof(double));

double \*y = (double\*)malloc(n\*sizeof(double));

double \*bb = (double\*)malloc(n\*sizeof(double));

y[0] = b[0];

for(i = 0;i < n;i++)

{

bb[i] = b[i];

}

for(i = 1;i < n;i++)//soulv ly = b

{

for(j = 0;j < i;j++)

{

bb[i] -= y[j] \* l[i][j];

}

y[i] = bb[i];

}

x[n-1] = y[n-1] / u[n-1][n-1];

for(i = n - 2;i >= 0;i--)//solve ux = y

{

for(j = n - 1;j > i;j--)

{

y[i] -= u[i][j] \* x[j];

}

x[i] = y[i] / u[i][i];

}

return x;

}

void multi(double \*\*a,double \*\*b,int n,double \*\*ans)

{

int i,j,k;

double temp;

for(i = 0;i < n;i++)

{

for(j = 0;j < n;j++)

{

temp = 0;

for(k = 0;k < n;k++)

{

temp += a[i][k] \* b[k][j];

}

ans[i][j] = temp;

}

}

}

double \*\*LUni(double \*\*U,double \*\*L,int n)

{

int i,j,k,s;

double \*\*u = (double\*\*)malloc(n\*sizeof(double\*));

double \*\*r = (double\*\*)malloc(n\*sizeof(double\*));

double \*\*ans = (double\*\*)malloc(n\*sizeof(double\*));

for(i = 0;i < n;i++)

{

u[i] = (double\*)malloc(n\*sizeof(double));

r[i] = (double\*)malloc(n\*sizeof(double));

ans[i] = (double\*)malloc(n\*sizeof(double));

}

for (i=0;i<n;i++) // U-1

{

u[i][i]=1/U[i][i];

for (k=i-1;k>=0;k--)

{

s=0;

for (j=k+1;j<=i;j++)

s=s+U[k][j]\*u[j][i];

u[k][i]=-s/U[k][k];

}

}

for (i=0;i<n;i++) // l-1

{

r[i][i]=1;

for (k=i+1;k<n;k++)

{

for (j=i;j<=k-1;j++)

r[k][i]=r[k][i]-L[k][j]\*r[j][i];

}

}

multi(u,r,n,ans);

return ans;

}

double \*\*ALU(double \*\*a,double n)

{

double \*\*l = (double\*\*)malloc(n\*sizeof(double\*));

double \*\*u = (double\*\*)malloc(n\*sizeof(double\*));

int i, r, k;

for(i = 0;i < n;i++)

{

l[i] = (double\*)malloc(n\*sizeof(double));

u[i] = (double\*)malloc(n\*sizeof(double));

}

for (i = 0; i<n; i++)

{

u[0][i] = a[0][i];

}

for (i = 1; i<n; i++)

{

l[i][0] = a[i][0] / u[0][0];

}

for (r = 1; r<n; r++)

{

for (i = r; i <n; i++)

{

double sum1 = 0;

for (k = 0; k < r; k++)

{

sum1 += l[r][k] \* u[k][i];

}

u[r][i] = a[r][i] - sum1;

}

if(r!=n)

for(i=r+1;i<n;i++)

{

double sum2 = 0;

for (k = 0; k<r; k++)

{

sum2 += l[i][k] \* u[k][r];

}

l[i][r] = (a[i][r] - sum2) / u[r][r];

}

}

double \*\*ni;

ni = LUni(u,l,n);

return ni;

}

double condA(double \*\*a,int n)

{

double count1 = 0,count2 = 0;

int i,j,k;

for(i = 0;i < n;i++)

{

count2 = 0;

for(j = 0;j < n;j++)

{

count2 += fabs(a[i][j]);

}

if (count2 > count1)

{

count1 = count2;

}

}

return count1;

}

double \*\*makeHH(int n)

{

double \*\*a = (double\*\*)malloc(n\*sizeof(double\*));

int i,j;

double para = 0;

for(i = 0;i < n;i++)

{

a[i] = (double\*)malloc(n\*sizeof(double));

}

for(i = 0;i < n;i++)

{

para = 2 \* i + 1;

for(j = i;j < n;j++)

{

a[i][j] = 1 / para;

a[j][i] = a[i][j];

para++;

}

}

return a;

}