**Задание №2. Решение систем линейных алгебраических уравнений**

***Цель задания***: *практическое освоение точных и итерационных методов решения систем линейных алгебраических уравнений*

1. Решить СЛАУ с помощью программной реализации ниже указанных методов:

Точные методы:

* Метод Гаусса (*LU -* разложение)
* Метод Отражений (*QR -* разложение (метод Хаусхолдера))

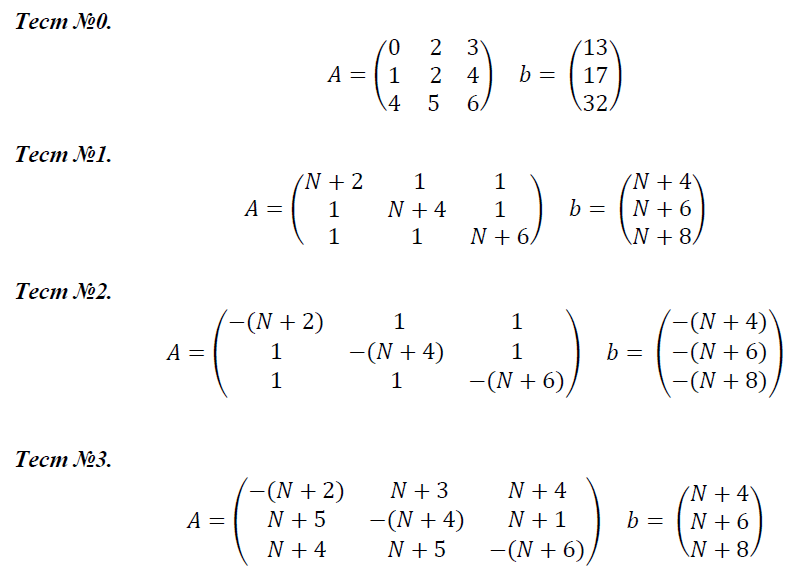
Итерационные методы:

* Метод простой итерации
* Метод Зейделя

*Примечание:* пользоваться встроенными функциями языка программирования можно только при вычислении абсолютной погрешности решения для заполнения таблиц «Результаты тестирования». Операции с матрицами и векторами необходимо запрограммировать самостоятельно. Для вычисления квадратного корня используйте итерационную формулу Герона.

1. Используя программную реализацию методов из п. 1, заполнить таблицы «*Результаты тестирования*» для тестов №0 - №5.

*Примечание:* Параметр *N*в тестах №0 - №5 должен совпадать с Вашим номером в списке группы.

****

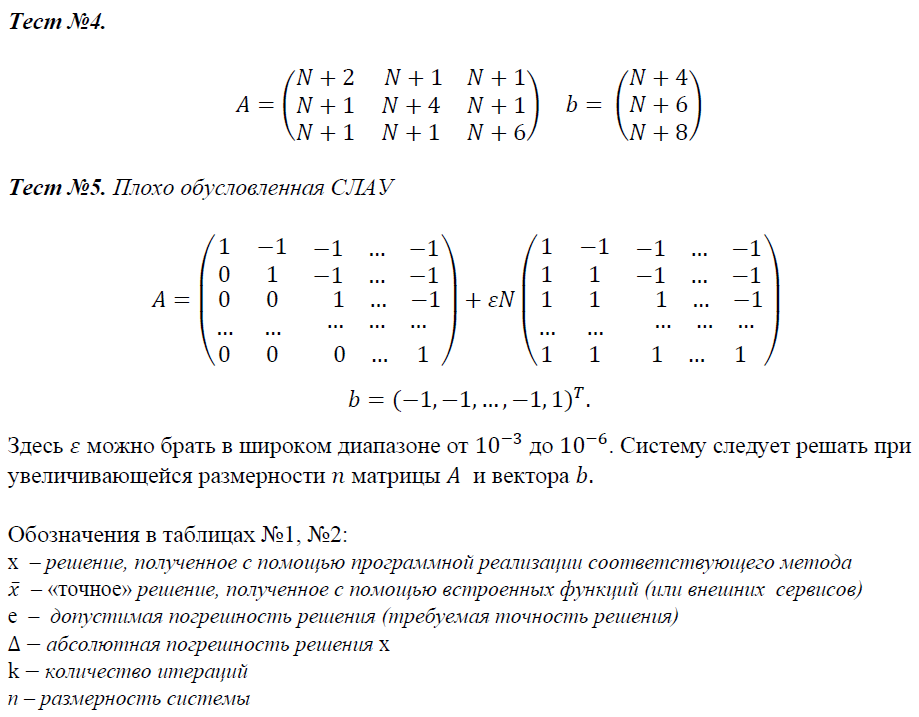
****

Таблица №1. Результаты тестирования №0 - №4.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № теста |  | *e* | МПИ | | | Метод Зейделя | | | М-д Гаусса (LU) | | М-д Хаусхолдера | |
| x |  | k | x |  | k | x |  | x |  |
| 0 | 1  2  3 | 10-2 | 1.07258  2.01162  2.95583 | 0.072 | 60 | 1.04848  2.05257  2.93839 | 0.061 | 65 | 1  2  3 | 0 | 1  2  3 | 3,55e-15 |
| 10-3 | 1.00697  2.00197  2.99513 | 0.007 | 147 | 0.99151  2.03699  2.97699 | 0.037 | 138 |
| 10-4 | 1.00043  2.00085  2.99916 | 0.00084 | 233 | 0.996748  2.00699  2.99644 | 0.007 | 716 |
| 1 | 1  1  1 | 10-2 | 0.997164  1.00054  1.00062 | 0.0028 | 4 | 0.99761  0.999978  1.00021 | 0.0024 | 4 | 1  1  1 | 0 | 1  1  1 | 3,0e-15 |
| 10-3 | 0.999639  1.0001  1.00006 | 0.00036 | 6 | 0.999726  1.00002  1.00002 | 0.00027 | 6 |
| 10-4 | 0.999954  1.00001  1.00001 | 4.65e-5 | 8 | 0.999968  1  1 | 3.2e-5 | 8 |
| 2 | 916/661  882/661  856/661 | 10-2 | 1.38187  1.3325  1.2938 | 0.0039 | 11 | 1.38111  1.33293  1.29423 | 0.0081 | 10 | 1.385779  1.334342  1.295008 | 2,2e-16 | 1.385779  1.334342  1.295008 | 3,3e-15 |
| 10-3 | 1.38528  1.33411  1.29485 | 0.0005 | 15 | 1.38528  1.33419  1.29492 | 0.0005 | 14 |
| 10-4 | 1.38574  1.33432  1.295 | 3.88e-5 | 20 | 1.38572  1.33433  1.295 | 5.43e-5 | 18 |
| 3 | 3404/2577  2902/2577  2800/2577 | 10-2 | 1.31443  1.1201  1.07959 | 0.0069 | 26 | 1.31655  1.12237  1.08255 | 0.0044 | 22 | 1.320916  1.126116  1.086535 | 7,1e-15 | 1.320916  1.126116  1.086535 | 1,8e-15 |
| 10-3 | 1.32027  1.12552  1.08585 | 0.00069 | 38 | 1.32046  1.12572  1.08611 | 0.00046 | 31 |
| 10-4 | 1.32085  1.12606  1.08647 | 6.83e-5 | 50 | 1.32087  1.12607  1.08649 | 4.87e-5 | 40 |
| 4 | -8/67  42/67  52/67 | 10-2 | -0.0786722  0.599911  0.765358 | 0.041 | 50 | -0.0851307  0.606942  0.765334 | 0.034 | 45 | -0.119403  0.626866  0.776119 | 2,4e-15 | -0.119403  0.626866  0.776119 | 8,9e-16 |
| 10-3 | -0.115285  0.62409  0.775079 | 0.0041 | 97 | -0.116011  0.624862  0.77508 | 0.0034 | 85 |
| 10-4 | -0.119005  0.626597  0.776019 | 0.0004 | 145 | -0.119066  0.626666  0.776016 | 0.00034 | 125 |

Таблица №2. Результаты тестирования №5.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № теста | n |  |  | *e* | МПИ | | | | Метод Зейделя | | | | М-д Гаусса (LU) | | | М-д Хаусхолдера | | |
| x |  | k | x | |  | k | x | |  | x | |  |
| 5 | 4 | 10-3 | 0;0;0;  0.(9900) | *10-2* | -0.126088;  -0.0635615  -0.031662  0.964842 | 0.126 | 9 | -0.117677  -0.0585741  -0.028436  0.966798 | | 0.118 | 8 | 0;0;0;  0.(9900) | | 5,5e-16 | 0;0;0;  0.(9900) | | 1,72e-16 |
| *10-3* | -0.0360032  -0.018143  -0.00982616  0.983421 | 0.036 | 191 | -0.0318765  -0.0159815  -0.00864037  0.984209 | | 0.032 | 176 |
| *10-4* | -0.00359947  -0.00181387  -0.000982384  0.989431 | 0.0036 | 526 | -0.00320634  -0.00160753  -0.000869105  0.989507 | | 0.0032 | 473 |
| 10-6 | 0;0;0;0;  0.(999990000) | *10-2* | -0.126582  -0.064358  -0.0322382  0.975077 | 0.127 | 9 | -0.11796  -0.0592495  -0.0289815  0.977019 | | 0.118 | 8 | 0;0;0;0;  0.(999990000) | | 4,4e-16 | 0;0;0;0;  0.(999990000) | | 2,18e-16 |
| *10-3* | -0.0445113  -0.022633  -0.0122634  0.99186 | 0.045 | 196 | -0.0395186  -0.0200112  -0.0108274  0.992794 | | 0.04 | 181 |
| *10-4* | -0.00444871  -0.00226207  -0.00122568  0.999177 | 0.0044 | 609 | -0.0039669  -0.00200874  -0.00108686  0.999268 | | 0.004 | 547 |
| 5 | 10-3 | 0;0;0;0;  0.(9900) | *10-2* | -0.0712858  -0.0359919  -0.0180529  -0.00446812  0.978633 | 0.07 | 13 | -0.0611844  -0.0309558  -0.0164041  -0.0031138  0.980866 | | 0.061 | 12 | 0;0;0;0;  0.(9900) | | 1,1e-16 | 0;0;0;0;  0.(9900) | | 1,73e-16 |
| *10-3* | -0.0704593  -0.0351533  -0.0179504  -0.00917893  0.982691 | 0.07 | 20 | -0.0604174  -0.0301173  -0.015524  -0.00793774  0.983922 | | 0.06 | 19 |
| *10-4* | -0.0164253  -0.00817082  -0.00415413  -0.00229199  0.98848 | 0.016 | 977 | -0.0151986  -0.00755238  -0.00383788  -0.00211813  0.988601 | | 0.015 | 859 |
| 10-6 | 0;0;0;0;  0.(999990000) | *10-2* | -0.068236  -0.0347352  -0.017481  -0.0042503  0.989346 | 0.068 | 13 | -0.0580644  -0.0296056  -0.0157599  -0.00289592  0.991508 | | 0.058 | 12 | 0;0;0;0;  0.(999990000) | | 0 | 0;0;0;0;  0.(999990000) | | 2,9e-16 |
| *10-3* | -0.0676749  -0.0340769  -0.0174773  -0.00883878  0.993258 | 0.067 | 20 | -0.0575439  -0.02896  -0.0149952  -0.00758041  0.994431 | | 0.057 | 19 |
| *10-4* | -0.025978  -0.0130453  -0.00666371  -0.0036286  0.997574 | 0.026 | 1016 | -0.0240279  -0.0120577  -0.0061574  -0.00335354  0.997755 | | 0.024 | 860 |
| 6 | 10-3 | 0;0;0;0;0;  0.(9900) | *10-2* | -0.0396742  -0.0204241  -0.0110512  -0.00443147  0.00754591  0.978821 | 0.040 | 17 | -0.0311944  -0.0161413  -0.00960945  -0.00522177  0.00599282  0.983232 | | 0.031 | 17 | 0;0;0;0;0;  0.(9900) | | 1,1e-16 | 0;0;0;0;0;  0.(9900) | | 4,5e-16 |
| *10-3* | -0.0396964  -0.0197416  -0.00998714  -0.00517435  -0.00190041  0.987261 | 0.040 | 29 | -0.031471  -0.0156468  -0.00801592  -0.0043705  -0.00139136  0.987987 | | 0.031 | 27 |
| *10-4* | -0.0395568  -0.0196118  -0.00980322  -0.00504835  -0.00273776  0.987889 | 0.040 | 40 | -0.0314015  -0.0155596  -0.00778267  -0.00404312  -0.00216723  0.988366 | | 0.031 | 37 |
| 10-6 | 0;0;0;0;0;  0.(999990000) | *10-2* | -0.034674  -0.0180319  -0.00986594  -0.00380632  0.00769693  0.989419 | 0.035 | 17 | -0.0263868  -0.0138155  -0.00845063  -0.00459723  0.00614523  0.993722 | | 0.026 | 17 | 0;0;0;0;0;  0.(999990000) | | 0 | 0;0;0;0;0;  0.(999990000) | | 1,18e-16 |
| *10-3* | -0.0347515  -0.0174517  -0.00890043  -0.00459713  -0.00153056  0.997705 | 0.035 | 29 | -0.0267124  -0.0134132  -0.00694327  -0.00379578  -0.001047  0.998354 | | 0.027 | 27 |
| *10-4* | -0.0347217  -0.0173826  -0.00875259  -0.00449452  -0.00233658  0.998301 | 0.035 | 40 | -0.0267279  -0.0133749  -0.0067407  -0.00349479  -0.00179109  0.998709 | | 0.027 | 37 |

**Выводы**

По результатам всех тестов, можно сделать вывод, что методы (LU и QR разложения) дают точный результат, присутствует только погрешность, связанная с ограниченным числом десятичных знаков в пердставленнии чисел с плавающей запятой в компьютере (везде использовался тип double). Поэтому погрешности точных методов составлют порядка .

По результатм решения тестов №0-4 делаем вывод, что метод Зейделя сходится быстрее метода простой итерации, поскольку использует новые приближения на той же итерации. При этом чем больше необходимая точность, тем большее число итераций требуется.

Для матриц с диагональнвм преобладанием получаем норму матрицы B (x = Bx+c) меньше 1, в результате реальная достигнутая точность согласно критерию, учитывающему норму матрицы B, не превышает заданное пользователем значение.

Для не положительно определённых матриц, даже после умножения системы на транспонированную матрицу, норма матрицы B будет больше 1. Скорость сходимости таких систем низкая (для тестов 0 и 4 потребовалось более 100 итераций для достижения точности ).

Плохо обусловленная СЛАУ высокого порядка практически не сходится к точному решению. Так, если для СЛАУ 4-го порядка наблюдаем, что при увеличении числа итераций точность решения увеличивается, то для СЛАУ 6-го аболютна погрешность решения не уменьшается с ростом числа итераций. А для СЛАУ 5-го порядка видим, что при увеличении числа итераций с 20 до 1016 абсолютная погрешность уменьшилась всего с 0,067 до 0,026. Таким образом, итерационные методы решения плохо подходят для плохо обусловленных систем.

**Приложения**

*Итерационные методы (для тестов 0-4)*

#include <iostream>

#include <fstream>

#include "conio.h"

#include <locale>

#include <cmath>

using namespace std;

int const N=3; //размерность матрицы

double A[N][N],BB[N][N],Xtru[N],Xdelta[N];

double B[N],c[N],X[N],Xold[N],Xeps[N];

double Eps0;

//вывод системы на экран

void Printsystem()

{

int i,j,n=N;

cout << "Решение СЛАУ A\*x=b" <<endl;

cout << "Матрица A=" <<endl;

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

{

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

cout<<A[i][j]<<'\t';

cout<<endl;

}

cout << "Вектор b:" << endl;

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

cout<<B[i]<<'\t';

cout<<'\n'<< endl;

}

//загрузка примера №k

void LoadSystem(int k)

{

int i,j,n;

n=N;

cout<<"-------------------Тeст №"<<k<<"-------------------"<<endl<<endl;

switch (k)

{

case 0:

{ifstream An("A0.txt");

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

{

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

An>>A[i][j];

}

An.close();

ifstream Bn("B0.txt");

for(i=0;i<n;i++) Bn>>B[i];

Bn.close();

//точное решение

Xtru[0]=1;Xtru[1]=2;Xtru[2]=3;}

break;

case 1:

{ifstream An("A1.txt");

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

{

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

An>>A[i][j];

}

An.close();

ifstream Bn("B1.txt");

for(i=0;i<n;i++) Bn>>B[i];

Bn.close();

Xtru[0]=1;Xtru[1]=1;Xtru[2]=1;}

break;

case 2:

{ifstream An("A2.txt");

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

{

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

An>>A[i][j];

}

An.close();

ifstream Bn("B2.txt");

for(i=0;i<n;i++) Bn>>B[i];

Bn.close();

Xtru[0]=916.0/661.0;Xtru[1]=882.0/661.0;Xtru[2]=856.0/661.0;}

break;

case 3:

{ifstream An("A3.txt");

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

{

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

An>>A[i][j];

}

An.close();

ifstream Bn("B3.txt");

for(i=0;i<n;i++) Bn>>B[i];

Bn.close();

Xtru[0]=3404.0/2577.0;Xtru[1]=2902.0/2577.0;Xtru[2]=2800.0/2577.0;}

break;

case 4:

{ifstream An("A4.txt");

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

{

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

An>>A[i][j];

}

An.close();

ifstream Bn("B4.txt");

for(i=0;i<n;i++) Bn>>B[i];

Bn.close();

Xtru[0]=-8.0/67.0;Xtru[1]=42.0/67.0;Xtru[2]=52.0/67.0;}

break;

}

Printsystem();

}

//бесконечная норма вектора

double NormInfVect(double b[N])

{

double maxb=0;

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

{

if (maxb<abs(b[j])) maxb=abs(b[j]);

}

return maxb;

}

//норма матрицы

double NormMatrix(double a[N][N])

{

double S[N], maxa=0;

for (int i=0; i<N; i++)

{

S[i] = 0;

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

{

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

}

if (maxa<S[i]) maxa=S[i];

}

return maxa;

}

//умножение матрицы на её транспонированную

void MultA()

{

double S,An[N][N];

for (int i=0; i<N; i++)

{

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

{

S=0;

for (int k=0; k<N; k++)

{

S+=A[k][i]\*A[k][j];

}

An[i][j]=S;

}

}

for (int i=0; i<N; i++)

{

for (int j=0; j<N; j++) A[i][j]= An[i][j];

}

}

//умножение транспонированной матрицы на вектор

void MultAb()

{

double S, Bn[N];

for (int i=0; i<N; i++)

{

S = 0;

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

{

S+=A[j][i]\*B[j];

}

Bn[i]=S;

}

for (int i=0; i<N; i++) B[i]=Bn[i];

MultA();

Printsystem();

}

double OneIteration(int n) //одна итерация метода простой итерации

{

int i,j;

double sum,Xnew[N];

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

{

sum=c[i];

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

{

sum=sum+BB[i][j]\*X[j];

}

Xnew[i]=sum;

//разница между итерациями

Xeps[i]=X[i]-Xnew[i];

}

//вывод текущего приближения

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

{

X[i]=Xnew[i];

//cout<<X[i]<<'\t';

}

return NormInfVect(Xeps);

}

double OneIterationZ(int n) //одна итерация метода Зейделя

{

int i,j;

double sum,Xold[N];

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

{

Xold[i]=X[i];

sum=c[i];

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

{

sum=sum+BB[i][j]\*X[j];

}

X[i]=sum;

//разница между итерациями

Xeps[i]=X[i]-Xold[i];

}

//вывод текущего приближения

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

{

//cout<<X[i]<<'\t';

}

return NormInfVect(Xeps);

}

double OneIterationJ() //одна итерация метода Якоби

{

int i,j, n=N;

double sum,Xnew[N];;

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

{

Xold[i]=X[i];

sum=B[i];

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

{

if(i!=j)

{

sum=sum-A[i][j]\*X[j];

}

}

Xnew[i]=sum/A[i][i];

//разница между итерациями

Xeps[i]=Xnew[i]-X[i];

}

//вывод текущего приближения

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

{

X[i]=Xnew[i];

cout<<X[i]<<'\t';

}

cout<<'\n';

return NormInfVect(Xeps);

}

//метод Якоби для матриц с диагональным преобладанием

void Jakobi()

{

int i,j,n=N;

double t;

//Расчёт матрицы BB (x =BB\*x+c)

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

{

t=A[i][i];

for(j=0;j<n;j++) BB[i][j]=-A[i][j]/t;

BB[i][i]=0;

c[i]=B[i]/t;

}

cout << "Решение СЛАУ x=B\*x+c" <<endl;

cout << "Матрица B=" <<endl;

double bnorm=NormMatrix(BB);

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

{

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

cout<<BB[i][j]<<'\t';

cout<<endl;

}

cout << "Норма матрицы B: " <<bnorm<< endl;

cout << "Вектор c:" << endl;

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

cout<<c[i]<<'\t';

cout<<'\n'<< endl;

//начальное приближение - вектор c

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

{

X[i]=c[i];

}

cout << "-------Метод Якоби-------"<<endl;

//итерации метода

int k=1;

double eps;

do

{

cout << "Итерация " << k << " : ";

eps = OneIterationJ();

cout << "Оценка точности: " << bnorm\*eps/(1-bnorm) <<endl<<endl;

k++;

}

while(eps>(1-bnorm)\*Eps0/bnorm);

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

{

Xdelta[i]=fabs(X[i]-Xtru[i]);

}

cout << "Реально достигнутая точность: " << NormInfVect(Xdelta) <<endl << endl;

}

//методы простой итерации и Зейделя

void Simpleiteration()

{

int i,j,n=N;

//приводим к виду x =BB\*x+c

double mu = 1/NormMatrix(A);//норма исходной матрицы

cout << "Норма матрицы A: " <<NormMatrix(A)<< endl;

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

{

for(j=0;j<n;j++) BB[i][j]=-mu\*A[i][j];

BB[i][i]+=1;

c[i]=mu\*B[i];

}

cout << "Решение СЛАУ x=B\*x+c" <<endl;

cout << "Матрица B=" <<endl;

double bnorm=NormMatrix(BB);

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

{

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

cout<<BB[i][j]<<'\t';

cout<<endl;

}

cout << "Норма матрицы B: " <<bnorm<< endl;

cout << "Вектор c:" << endl;

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

cout<<c[i]<<'\t';

cout<<'\n'<< endl;

//начальное приближение - вектор c

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

{

X[i]=c[i];

}

//критерий близости двух итераций в зависимости от нормы матрицы

double Eps1=(1-bnorm)\*Eps0/bnorm;

if (bnorm>1) Eps1=(bnorm-1)\*Eps0/bnorm;

cout << "-------Метод простой итерации-------"<<endl;

//итерации метода

int k=1;

double eps;

do

{

eps = OneIteration(n);

k++;

}

while(eps>Eps1);

cout << "Итерация " << k-1 << " : ";

//вывод текущего приближения

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

{

cout<<X[i]<<'\t';

}

cout<<'\n';

if (bnorm<=1)

cout << "Оценка точности: " << bnorm\*eps/(1-bnorm) <<endl<<endl;

else

cout << "Оценка точности: " << bnorm\*eps/(bnorm-1) <<endl<<endl;

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

{

Xdelta[i]=fabs(X[i]-Xtru[i]);

}

cout << "Реально достигнутая точность: " << NormInfVect(Xdelta) <<endl << endl;

cout << "-----------Метод Зейделя----------"<<endl;

k=1;

//начальное приближение - вектор c

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

{

X[i]=c[i];

}

do

{

eps = OneIterationZ(n);

k++;

}

while(eps>Eps1);

cout << "Итерация " << k-1 << " : ";

//вывод текущего приближения

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

{

cout<<X[i]<<'\t';

}

cout<<'\n';

if (bnorm<=1)

cout << "Оценка точности: " << bnorm\*eps/(1-bnorm) <<endl<<endl;

else

cout << "Оценка точности: " << bnorm\*eps/(bnorm-1) <<endl<<endl;

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

{

Xdelta[i]=fabs(X[i]-Xtru[i]);

}

cout << "Реально достигнутая точность: " << NormInfVect(Xdelta) <<endl << endl;

}

int main()

{

setlocale(LC\_ALL, "russian");

cout << "Введите точность вычисления" << endl;

cin >> Eps0;

//Тест №0//матрица не положительно определённая

LoadSystem(0);//загружаем систему

//Метод простой итерации и Зейделя

MultAb(); //умножаем на транспонированную

Simpleiteration();

//Тест №1//матрица положительно определена

LoadSystem(1);

//Метод Якоби

Jakobi();

//Метод простой итерации и Зейделя

Simpleiteration();

//Тест №2//матрица не положительно определённая

LoadSystem(2);//загружаем систему

//Метод Якоби

Jakobi();

//Метод простой итерации и Зейделя

MultAb(); //умножаем на транспонированную

Simpleiteration();

//Тест №3//матрица не положительно определённая

LoadSystem(3);//загружаем систему

//Метод простой итерации и Зейделя

MultAb(); //умножаем на транспонированную

Simpleiteration();

//Тест №4//матрица положительно определена

LoadSystem(4);

//Метод простой итерации и Зейделя

Simpleiteration();

cout << "Press any key" << endl;

getch();

return 0;

}

*Итерационные методы (для теста 5)*

#include <iostream>

#include <fstream>

#include "conio.h"

#include <locale>

#include <cmath>

using namespace std;

int const N=6; //размерность матрицы

double A[N][N],BB[N][N],Xtru[N];

double B[N],c[N],X[N],Xold[N],Xeps[N],Xdelta[N];

double Eps0;

//вывод системы на экран

void Printsystem()

{

int i,j,n=N;

cout << "Решение СЛАУ A\*x=b (после умножения системы на транспонированную матрицу)" <<endl;

cout << "Матрица A=" <<endl;

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

{

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

cout<<A[i][j]<<'\t';

cout<<endl;

}

cout << "Вектор b:" << endl;

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

cout<<B[i]<<'\t';

cout<<'\n'<< endl;

}

//загрузка примера №5

void LoadSystem(double e)

{

int i,j,n;

n=N;

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

{

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

{

if (i>=j) A[i][j]=10\*e;

else A[i][j]=-(1+10\*e);

}

A[i][i]+=1;

B[i]=-1;

Xtru[i]=0;

}

B[n-1]=1;

//точное решение

if (e==0.001) Xtru[n-1]=0.99009900990099;

else Xtru[n-1]=0.99999000099999;

}

//бесконечная норма вектора

double NormInfVect(double b[N])

{

double maxb=0;

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

{

if (maxb<abs(b[j])) maxb=abs(b[j]);

}

return maxb;

}

//норма матрицы

double NormMatrix(double a[N][N])

{

double S[N], maxa=0;

for (int i=0; i<N; i++)

{

S[i] = 0;

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

{

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

}

if (maxa<S[i]) maxa=S[i];

}

return maxa;

}

//умножение матрицы на её транспонированную

void MultA()

{

double S,An[N][N];

for (int i=0; i<N; i++)

{

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

{

S=0;

for (int k=0; k<N; k++)

{

S+=A[k][i]\*A[k][j];

}

An[i][j]=S;

}

}

for (int i=0; i<N; i++)

{

for (int j=0; j<N; j++) A[i][j]= An[i][j];

}

}

//умножение транспонированной матрицы на вектор

void MultAb()

{

double S, Bn[N];

for (int i=0; i<N; i++)

{

S = 0;

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

{

S+=A[j][i]\*B[j];

}

Bn[i]=S;

}

for (int i=0; i<N; i++) B[i]=Bn[i];

MultA();

Printsystem();

}

double OneIteration(int n) //одна итерация метода простой итерации

{

int i,j;

double sum,Xnew[N];

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

{

sum=c[i];

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

{

sum=sum+BB[i][j]\*X[j];

}

Xnew[i]=sum;

//разница между итерациями

Xeps[i]=X[i]-Xnew[i];

}

//вывод текущего приближения

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

{

X[i]=Xnew[i];

//cout<<X[i]<<'\t';

}

return NormInfVect(Xeps);

}

double OneIterationZ(int n) //одна итерация метода Зейделя

{

int i,j;

double sum,Xold[N];

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

{

Xold[i]=X[i];

sum=c[i];

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

{

sum=sum+BB[i][j]\*X[j];

}

X[i]=sum;

//разница между итерациями

Xeps[i]=X[i]-Xold[i];

}

//вывод текущего приближения

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

{

//cout<<X[i]<<'\t';

}

return NormInfVect(Xeps);

}

//методы простой итерации и Зейделя

void Simpleiteration()

{

int i,j,n=N;

//приводим к виду x =BB\*x+c

double mu = 1/NormMatrix(A);//норма исходной матрицы

cout << "Норма матрицы A: " <<NormMatrix(A)<< endl;

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

{

for(j=0;j<n;j++) BB[i][j]=-mu\*A[i][j];

BB[i][i]+=1;

c[i]=mu\*B[i];

}

cout << "Решение СЛАУ x=B\*x+c" <<endl;

cout << "Матрица B=" <<endl;

double bnorm=NormMatrix(BB);

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

{

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

cout<<BB[i][j]<<'\t';

cout<<endl;

}

cout << "Норма матрицы B: " <<bnorm<< endl;

cout << "Вектор c:" << endl;

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

cout<<c[i]<<'\t';

cout<<'\n'<< endl;

//начальное приближение - вектор c

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

{

X[i]=c[i];

}

double Eps1=(1-bnorm)\*Eps0/bnorm;

if (bnorm>1) Eps1=(bnorm-1)\*Eps0/bnorm;

cout << "-------Метод простой итерации-------"<<endl;

//итерации метода

int k=1;

double eps;

do//делаем итерации

{

eps = OneIteration(n);

k++;

}

while(eps>Eps1);

cout << "Итерация " << k-1 << " : ";

//вывод текущего приближения

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

{

cout<<X[i]<<'\t';

//сравнение с точным решением

Xdelta[i]=fabs(X[i]);

}

Xdelta[n-1]=fabs(X[n-1]-Xtru[n-1]);

cout<<'\n';

if (bnorm<=1)

cout << "Оценка точности: " << bnorm\*eps/(1-bnorm) <<endl<<endl;

else

cout << "Оценка точности: " << bnorm\*eps/(bnorm-1) <<endl;

cout << "Реально достигнутая точность: " << NormInfVect(Xdelta) <<endl<<endl;

cout << "-----------Метод Зейделя----------"<<endl;

k=1;

//начальное приближение - вектор c

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

{

X[i]=c[i];

}

do

{

eps = OneIterationZ(n);

k++;

}

while(eps>Eps1);

cout << "Итерация " << k-1 << " : ";

//вывод текущего приближения

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

{

cout<<X[i]<<'\t';

Xdelta[i]=fabs(X[i]);

}

Xdelta[n-1]=fabs(X[n-1]-Xtru[n-1]);

cout<<'\n';

if (bnorm<=1)

cout << "Оценка точности: " << bnorm\*eps/(1-bnorm) <<endl<<endl;

else

cout << "Оценка точности: " << bnorm\*eps/(bnorm-1) <<endl<<endl;

cout << "Реально достигнутая точность: " << NormInfVect(Xdelta) <<endl<<endl;

}

int main()

{

setlocale(LC\_ALL, "russian");

cout << "Введите точность вычисления" << endl;

cin >> Eps0;

double e =0.001;//параметр в 5-ом тесте

//Тест №5

LoadSystem(e);//загружаем систему

MultAb(); //умножаем на транспонированную

Simpleiteration();

e =0.000001;

LoadSystem(e);//загружаем систему

MultAb(); //умножаем на транспонированную

Simpleiteration();

cout << "Press any key" << endl;

getch();

return 0;

}

*Метод Гаусса LU и классический (для тестов 1-4)*

#include <iostream>

#include <fstream>

#include "conio.h"

#include <locale>

#include <cmath>

using namespace std;

int const N=3; //размерность матрицы

double A[N][N],newA[N][N],L[N][N],U[N][N],Xtru[N];

double B[N],Y[N],X[N];

//вывод системы на экран

void Printsystem()

{

int i,j,n=N;

cout << "Решение СЛАУ A\*x=b" <<endl;

cout << "Матрица A=" <<endl;

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

{

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

cout<<A[i][j]<<'\t';

cout<<endl;

}

cout << "Вектор b:" << endl;

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

cout<<B[i]<<'\t';

cout<<'\n'<< endl;

}

void PrintLU()

{

int i,j,n=N;

double er[N];

cout << "Matrix L:" << endl;

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

{

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

cout<<L[i][j]<<'\t';

cout<<endl;

}

cout << "Matrix U:" << endl;

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

{

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

cout<<U[i][j]<<'\t';

cout<<endl;

}

cout << "Vector X:" << endl;

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

cout<<X[i]<<'\t';

cout<<'\n';

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

{

er[i]=A[i][0]\*X[0]-B[i];

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

{

er[i]=er[i]+A[i][j]\*X[j];

}

}

cout << "Невязка Ax-b: " << endl;

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

{

cout << "Ax-b[" << i << "]=";

printf(" %e\n",er[i]);

}

}

//загрузка примера №k

void LoadSystem(int k)

{

int i,j,n;

n=N;

cout<<"-------------------Тeст №"<<k<<"-------------------"<<endl<<endl;

switch (k)

{

case 0:

{ifstream An("A0.txt");

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

{

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

An>>A[i][j];

}

An.close();

ifstream Bn("B0.txt");

for(i=0;i<n;i++) Bn>>B[i];

Bn.close();

//точное решение

Xtru[0]=1;Xtru[1]=2;Xtru[2]=3;}

break;

case 1:

{ifstream An("A1.txt");

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

{

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

An>>A[i][j];

}

An.close();

ifstream Bn("B1.txt");

for(i=0;i<n;i++) Bn>>B[i];

Bn.close();

Xtru[0]=1;Xtru[1]=1;Xtru[2]=1;}

break;

case 2:

{ifstream An("A2.txt");

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

{

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

An>>A[i][j];

}

An.close();

ifstream Bn("B2.txt");

for(i=0;i<n;i++) Bn>>B[i];

Bn.close();

Xtru[0]=916.0/661.0;Xtru[1]=882.0/661.0;Xtru[2]=856.0/661.0;}

break;

case 3:

{ifstream An("A3.txt");

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

{

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

An>>A[i][j];

}

An.close();

ifstream Bn("B3.txt");

for(i=0;i<n;i++) Bn>>B[i];

Bn.close();

Xtru[0]=3404.0/2577.0;Xtru[1]=2902.0/2577.0;Xtru[2]=2800.0/2577.0;}

break;

case 4:

{ifstream An("A4.txt");

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

{

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

An>>A[i][j];

}

An.close();

ifstream Bn("B4.txt");

for(i=0;i<n;i++) Bn>>B[i];

Bn.close();

Xtru[0]=-8.0/67.0;Xtru[1]=42.0/67.0;Xtru[2]=52.0/67.0;}

break;

}

Printsystem();

}

void crout() //разложение на L и U

{

int i,j,n=N,s;

double sum;

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

{

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

{

L[i][j]=0;

U[i][j]=0;

}

Y[i]=0;

X[i]=0;

U[i][i]=1;

}

for(i=0;i<n;i++) L[i][0]=A[i][0];

for(j=0;j<n;j++) U[0][j]=A[0][j]/A[0][0];

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

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

{

sum=0;

if(i>=j)

{

for(s=0;s<=j-1;s++)

sum+=(L[i][s])\*(U[s][j]);

L[i][j]=A[i][j]-sum;

}

else

{

for(s=0;s<=i-1;s++)

sum+=L[i][s]\*U[s][j];

U[i][j]=(A[i][j]-sum)/L[i][i];

}

}

}

void solve() //решение системы

{

int i,j,n=N;

double sum;

crout(); //сначала делаем LU разложение

// прямой ход:

Y[0]=B[0]/L[0][0];

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

{

sum=0;

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

sum+=L[i][j]\*Y[j];

Y[i]=(B[i]-sum)/L[i][i];

}

// обратный ход:

X[n-1]=Y[n-1];

for(i=n-2;i>=0;i--)

{

sum=0;

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

sum+=X[j]\*U[i][j];

X[i]=Y[i]-sum;

}

PrintLU();//вывод ответа

}

void Gauss()

{

cout<<"---------Решение классическим методом Гаусса------"<<endl;

int n=N, i, j, k;

double d, s,er[N];

for (k = 0; k < n; k++) // прямой ход

{

cout<<"Шаг "<<k<<endl;

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

{

for(j=0;j<n;j++) cout <<fixed <<A[i][j]<<'\t';

cout<<B[i];

cout<<endl;

}

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

{

d = A[j][k] / A[k][k];

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

{

A[j][i] = A[j][i] - d \* A[k][i];

}

B[j] = B[j] - d \* B[k];

}

}

for (k = n-1; k >= 0; k--) // обратный ход

{

d = 0;

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

{

s = A[k][j] \* X[j];

d = d + s;

}

X[k] = (B[k] - d) / A[k][k];

}

cout << "Решение системы: " << endl;

for( i = 0; i < n; i++) cout << "x[" << i << "]=" << X[i] << " " << endl;

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

{

er[i]=A[i][0]\*X[0]-B[i];

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

{

er[i]=er[i]+A[i][j]\*X[j];

}

}

cout << "Невязка Ax-b: " << endl;

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

{

cout << "Ax-b[" << i << "]=";

printf(" %e\n",er[i]);

}

}

int main()

{

setlocale(LC\_ALL, "russian");

int n=N;

//Тест №0

LoadSystem(0);//загружаем систему

//LU разложение

solve(); // решеаем систему LU разложением

//Классический метод Гаусса

Gauss();

//Тест №1

LoadSystem(1);

//LU разложение

solve(); // решеаем систему LU разложением

//Классический метод Гаусса

Gauss();

//Тест №2

LoadSystem(2);//загружаем систему

//LU разложение

solve(); // решеаем систему LU разложением

//Классический метод Гаусса

Gauss();

//Тест №3

LoadSystem(3);

//LU разложение

solve(); // решеаем систему LU разложением

//Классический метод Гаусса

Gauss();

//Тест №4

LoadSystem(4);

//LU разложение

solve(); // решеаем систему LU разложением

//Классический метод Гаусса

Gauss();

cout << "Press any key" << endl;

getch();

return 0;

}

*Метод Гаусса LU (для теста 5)*

#include <iostream>

#include <fstream>

#include "conio.h"

#include <locale>

#include <cmath>

using namespace std;

int const N=6; //размерность матрицы

double A[N][N],newA[N][N],L[N][N],U[N][N],Xtru[N];

double B[N],Y[N],X[N];

//вывод системы на экран

void Printsystem()

{

int i,j,n=N;

cout << "Решение СЛАУ A\*x=b" <<endl;

cout << "Матрица A=" <<endl;

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

{

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

cout<<A[i][j]<<'\t';

cout<<endl;

}

cout << "Вектор b:" << endl;

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

cout<<B[i]<<'\t';

cout<<'\n'<< endl;

}

void PrintLU()

{

double er[N];

int i,j,n=N;

cout << "Matrix L:" << endl;

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

{

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

cout<<L[i][j]<<'\t';

cout<<endl;

}

cout << "Matrix U:" << endl;

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

{

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

cout<<U[i][j]<<'\t';

cout<<endl;

}

cout << "Vector X:" << endl;

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

{

cout<<X[i]<<'\t';

er[i]=A[i][0]\*X[0]-B[i];

}

cout<<'\n'<< endl;

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

{

er[i]=er[i]+A[i][j]\*X[j];

}

cout << "Невязка Ax-b: " << endl;

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

{

cout << "Ax-b[" << i << "]=";

printf(" %e\n",er[i]);

}

}

//загрузка примера №5

void LoadSystem(double e)

{

int i,j,n;

n=N;

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

{

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

{

if (i>=j) A[i][j]=10\*e;

else A[i][j]=-(1+10\*e);

}

A[i][i]+=1;

B[i]=-1;

}

B[n-1]=1;

}

void crout() //разложение на L и U

{

int i,j,n=N,s;

double sum;

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

{

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

{

L[i][j]=0;

U[i][j]=0;

}

Y[i]=0;

X[i]=0;

U[i][i]=1;

}

for(i=0;i<n;i++) L[i][0]=A[i][0];

for(j=0;j<n;j++) U[0][j]=A[0][j]/A[0][0];

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

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

{

sum=0;

if(i>=j)

{

for(s=0;s<=j-1;s++)

sum+=(L[i][s])\*(U[s][j]);

L[i][j]=A[i][j]-sum;

}

else

{

for(s=0;s<=i-1;s++)

sum+=L[i][s]\*U[s][j];

U[i][j]=(A[i][j]-sum)/L[i][i];

}

}

}

void solve() //решение системы

{

int i,j,n=N;

double sum;

crout(); //сначала делаем LU разложение

// прямой ход:

Y[0]=B[0]/L[0][0];

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

{

sum=0;

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

sum+=L[i][j]\*Y[j];

Y[i]=(B[i]-sum)/L[i][i];

}

// обратный ход:

X[n-1]=Y[n-1];

for(i=n-2;i>=0;i--)

{

sum=0;

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

sum+=X[j]\*U[i][j];

X[i]=Y[i]-sum;

}

PrintLU();//вывод ответа

}

int main()

{

setlocale(LC\_ALL, "russian");

double e;

e=0.001;

LoadSystem(e);//загружаем систему 5 c e=0.001

solve(); // решаем систему LU разложением

e=0.000001;

LoadSystem(e);//загружаем систему 5 c e=0.000001

solve(); // решаем систему LU разложением

cout << "Press any key" << endl;

getch();

return 0;

}

*Метод QR* *(для тестов 1-4)*

#include <iostream>

#include <stdio.h>

#include <conio.h>

#include <math.h>

#include <cmath>

#define \_N 3 // число уравнений и неизвестных

using namespace std;

//умножение двух матриц

double\*\* MultAB(int N, double \*\*A, double \*\*B)

{

double \*\*MultAB = new double\* [N];

for (int i=0; i<N; i++)

{

MultAB[i] = new double [N];

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

{

double skalar = 0;

for(int k = 0; k < N; k++)

skalar += A[i][k] \* B[k][j];

MultAB[i][j]=skalar;

}

}

return MultAB;

}

//умножение матрицы на вектор

double\* MultA\_vect(int N, double \*\*A, double \*b)

{

double \*MultA\_vect;

MultA\_vect= new double [N];

for (int i=0; i<N; i++)

{

double skalar = 0;

for(int k = 0; k < N; k++) skalar += A[i][k] \* b[k];

MultA\_vect[i]=skalar;

}

return MultA\_vect;

}

//разность векторов

double\* Razn\_vect(int N, double \*a, double \*b)

{

double \*Razn\_vect;

Razn\_vect= new double [N];

for (int i=0; i<N; i++)

{

Razn\_vect[i]=a[i]-b[i];

}

return Razn\_vect;

}

//приравнять одну матрицу к другой

double\*\* Ravno(int N, double \*\*A )

{

double \*\*Ravno = new double\* [N];

for (int i=0; i<N; i++)

{

Ravno[i] = new double [N];

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

{

Ravno[i][j]=A[i][j];

}

}

return Ravno;

}

//бесконечная норма матрицы(максимум из сумм модулей элементов разных строк)

double NormInf(int N, double \*\*A)

{

double maxA=0;

for (int i=0; i<N; i++)

{

double s =0;

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

{

s+=abs(A[i][j]);

}

if (maxA<s) maxA=s;

}

return maxA;

}

//норма 1 матрицы (максимум из сумм модулей элементов разных столбцов)

double Norm1(int N, double \*\*A)

{

double maxA=0;

for (int i=0; i<N; i++)

{

double s =0;

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

{

s+=abs(A[j][i]);

}

if (maxA<s) maxA=s;

}

return maxA;

}

//бесконечная норма вектора

double NormInfVect(int N, double \*b)

{

double maxb=0;

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

{

if (maxb<abs(b[j])) maxb=abs(b[j]);

}

return maxb;

}

//бесконечная норма вектора

double Norm1Vect(int N, double \*b)

{

double s=0;

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

{

s+=abs(b[j]);

}

return s;

}

//класс - СЛАУ

class matrix

{

public:

matrix(int n);//конструктор

~matrix();//деструктор

void seta(int i, int j, double value); //ввод значение в ячейку (i,j) матрицы системы

void setb(int i, double value);//ввод значение в ячейку (i) столбца свободных членов

void Q\_R(); //QR разложение + вычисление с его помощью обратной матрицы

void print\_slau(); //вывод на печать исходной системы, а также Q и R матриц

void Reshenie(); //решение системы с помощью найденной обратной матрицы

void print\_x(); //вывод на печать решения

void Setcond(); //вычисление числа обусловленности

void ApproxCond(); //оценка числа обусловленности через возмущённое решение

void PrintA\_Aobr();//вывод на печать произведения обратной и исходной матриц

private:

double \*\*a; //матрица системы

double \*\*q; //отртогональная матрица Q

double \*\*r; //верхняя треугольная матрица

double \*\*Aobr; //обратная матрица

double \*b; //столбец свободных членов

double \*x; //решение СЛАУ

double cond; //число обусловленности

int n; //размерность системы

int Indic; //индикатор (1 - используем норму 1, иначе бесконечную)

};

// ------------------------------------------------------

// Конструктор

// ------------------------------------------------------

matrix::matrix(int N)

{

n = N;

a = new double\* [n];

Aobr = new double\* [n];

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

{

a[i] = new double [n];

Aobr[i] = new double [n];

for (int j=0; j<n; j++) {a[i][j]=0;Aobr[i][j]=0;}

}

b = new double [n];

x = new double [n];

for (int i=0; i<n; i++) { b[i]=0; x[i]=0; }

Indic = 0; //по умолчанию используем бесконечную норму

}

// ------------------------------------------------------

// Деструктор

// ------------------------------------------------------

matrix::~matrix() //освобождаем память, занятую массивами

{

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

{

delete a[i]; delete Aobr[i]; delete r[i]; delete q[i];

}

delete[] a;delete[] Aobr; delete[] r; delete[] q;

delete b; delete x;

}

// ------------------------------------------------------

// Вывод СЛАУ на экран

// ------------------------------------------------------

void matrix::print\_slau()

{

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

{

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

{

printf("%.6f \t",a[i][j]);

}

printf("| %.6f ",b[i]);

printf("\n");

}

printf("\n");

printf("= Ortogonal matrix Q ===============================\n");

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

{

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

{

printf("%.6f \t",q[i][j]);

}

printf("\n");

}

printf("\n");

printf("= Matrix R =========================================\n");

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

{

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

{

printf("%.6f \t",r[i][j]);

}

printf("\n");

}

printf("\n");

printf("= Obratnaja A^(-1) =================================\n");

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

{

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

{

printf("%.6f \t",Aobr[i][j]);

}

printf("\n");

}

}

// ------------------------------------------------------

// Печать решения и невязки решения (максимальная по модулю ошибка)

// ------------------------------------------------------

void matrix::print\_x()

{

double max=0,h;

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

h=0;

for (int j=0; j<n; j++) h=h+x[j]\*a[i][j];

if (max<fabs(b[i]-h)) max=fabs(b[i]-h);

printf("x[%i]=%.6f ",i,x[i]);

}

printf("\nMaximal error: %e\n",max);

printf("\n");

}

// ------------------------------------------------------

// Установить значение матрицы a[][]

// ------------------------------------------------------

void matrix::seta(int i, int j, double value)

{

a[i][j] = value;

}

// ------------------------------------------------------

// Установить значение вектора b[]

// ------------------------------------------------------

void matrix::setb(int i, double value)

{

b[i] = value;

}

//находим число обусловленности

void matrix::Setcond()

{

cond = NormInf(n,a)\*NormInf(n,Aobr);

printf("Cond A = %.3f\n",cond);

}

// ------------------------------------------------------

// QR-алгоритм реализуем с помощью метода вращений

// ------------------------------------------------------

void matrix::Q\_R()

{

double \*\*T = new double\* [n]; // T - матрица вращения

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

{

T[i] = new double [n];

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

{

if (i==j) T[i][j]=1;

else T[i][j]=0;

}

}

r = new double\* [n];

q = new double\* [n];

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

{

r[i] = new double [n];

q[i] = new double [n];

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

{

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

q[i][j]=T[i][j];

}

}

//собственно сам алгоритм метода вращений

for (int k=0;k<n-1; k++)

{

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

{

if (r[i][k]!=0)

{

T[k][k]=r[k][k]/sqrt(r[k][k]\*r[k][k]+r[i][k]\*r[i][k]);

T[i][i]=T[k][k];

T[k][i]=r[i][k]/sqrt(r[k][k]\*r[k][k]+r[i][k]\*r[i][k]);

T[i][k]=(-1)\*T[k][i];

}

r=MultAB(n, T, r);

q=MultAB(n, T, q);

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

{

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

{

if (i==j) T[i][j]=1; else T[i][j]=0;

}

}

}

}

//получили верхнюю треугольную матрицу R и обратную q (или что тоже самое транспанированную)

// к ортогональной матрице Q

//находим обратную матрицу A^(-1) = R^(-1)\*Q^(T), решая систему R\*A^(-1) =Q^(T)

// Обратный ход

double h;

for (int P=0;P<n;P++)

{

Aobr[n-1][P]=q[n-1][P]/r[n-1][n-1];

for (int l=(n-1);l>=1;l--)

{

h=q[l-1][P];

for (int k=(l+1);k<=n;k++) h=h-Aobr[k-1][P]\*r[l-1][k-1];

Aobr[l-1][P]=h/r[l-1][l-1];

}

}

//теперь транспонируем q и получаем искомую Q

T=Ravno(n, q);

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

{

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

{

q[i][j]=T[j][i];

}

}

//освобождаем память, занятую массивом T

for (int i=0; i<n; i++) delete T[i]; delete[] T;

}

//решаем систему с помощью обратной матрицы

void matrix::Reshenie()

{

x = MultA\_vect(n,Aobr,b);

}

//печатаем произведение исходной матрицы на обратную

void matrix::PrintA\_Aobr()

{

double \*\*T = new double\* [n];

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

{

T[i] = new double [n];

}

T = MultAB(n,Aobr,a);

printf("\n");

printf(" A^(-1)\*A ===================================\n");

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

{

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

{

printf("%.2e\t ",T[i][j]);//выводим в експоненциальном формате

}

printf("\n");

}

//освобождаем память, занятую массивом T

for (int i=0; i<n; i++) delete T[i]; delete[] T;

}

//оценка числа обусловленности

void matrix::ApproxCond()

{

double deltaX, deltab;

double AppCond;

double \*newb = new double [n]; // вектор свободных членов с возмущением

double \*newX = new double [n]; // вектор решения возмущённой СЛАУ

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

newb[k]=b[k]+0.01\*cos(M\_PI\*k);

newX = MultA\_vect(n,Aobr,newb);//также для получения возмущ реш исп обратную матрицу

printf("\n");

printf("Reshenie s vozmusheniem================================\n");

for (int j=0; j<n; j++) printf("x[%i]=%.5f ",j,newX[j]);

printf("\n");

//получаем оценку числа обусловленности в зависимости от выбора нормы

if (Indic == 1) //норма 1

{

deltab = n\*0.01/Norm1Vect(n,b);

deltaX = Norm1Vect(n,Razn\_vect(n,newX,x))/Norm1Vect(n,x);

AppCond = deltaX/deltab;

printf("Cond A >=%.3f,\n", AppCond);

}

else //бесконечная норма

{

deltab = 0.01/NormInfVect(n,b);

deltaX = NormInfVect(n,Razn\_vect(n,newX,x))/NormInfVect(n,x);

AppCond = deltaX/deltab;

printf("Cond A >=%.3f,\n", AppCond);

}

//освобождаем память, занятую массивами newb, newX

delete newb; delete newX;

}

// ------------------------------------------------------

// Main

// ------------------------------------------------------

int main()

{

//система 0

matrix A(\_N);

A.seta(0,0,1);A.seta(0,1,2);A.seta(0,2,4);

A.seta(1,0,0);A.seta(1,1,2);A.seta(1,2,3);

A.seta(2,0,4);A.seta(2,1,5);A.seta(2,2,6);

A.setb(0,17);

A.setb(1,13);

A.setb(2,32);

printf("\*\*\*QR-razlojenie (test 0)\*\*\*\n");

printf("Ax=b ==============================================\n");

A.Q\_R();

A.print\_slau();

printf("\n=Otvet==============================================\n");

A.Reshenie();

A.print\_x();

printf("\n\n");

//система 1

matrix B(\_N);

B.seta(0,0,12);B.seta(0,1,1);B.seta(0,2,1);

B.seta(1,0,1);B.seta(1,1,14);B.seta(1,2,1);

B.seta(2,0,1);B.seta(2,1,1);B.seta(2,2,16);

B.setb(0,14);

B.setb(1,16);

B.setb(2,18);

printf("\*\*\*QR-razlojenie (test 1)\*\*\*\n");

printf("Ax=b ==============================================\n");

B.Q\_R();

B.print\_slau();

printf("\n=Otvet==============================================\n");

B.Reshenie();

B.print\_x();

printf("\n\n");

//система 2

matrix C(\_N);

C.seta(0,0,-12);C.seta(0,1,1);C.seta(0,2,1);

C.seta(1,0,1);C.seta(1,1,-14);C.seta(1,2,1);

C.seta(2,0,1);C.seta(2,1,1);C.seta(2,2,-16);

C.setb(0,-14);

C.setb(1,-16);

C.setb(2,-18);

printf("\*\*\*QR-razlojenie (test 2)\*\*\*\n");

printf("Ax=b ==============================================\n");

C.Q\_R();

C.print\_slau();

printf("\n=Otvet==============================================\n");

C.Reshenie();

C.print\_x();

printf("\n\n");

//система 3

matrix D(\_N);

D.seta(0,0,-12);D.seta(0,1,13);D.seta(0,2,14);

D.seta(1,0,15);D.seta(1,1,-14);D.seta(1,2,11);

D.seta(2,0,14);D.seta(2,1,15);D.seta(2,2,-16);

D.setb(0,14);

D.setb(1,16);

D.setb(2,18);

printf("\*\*\*QR-razlojenie (test 3)\*\*\*\n");

printf("Ax=b ==============================================\n");

D.Q\_R();

D.print\_slau();

printf("\n=Otvet==============================================\n");

D.Reshenie();

D.print\_x();

printf("\n\n");

//система 4

matrix E(\_N);

E.seta(0,0,12);E.seta(0,1,11);E.seta(0,2,11);

E.seta(1,0,11);E.seta(1,1,14);E.seta(1,2,11);

E.seta(2,0,11);E.seta(2,1,11);E.seta(2,2,16);

E.setb(0,14);

E.setb(1,16);

E.setb(2,18);

printf("\*\*\*QR-razlojenie (test 4)\*\*\*\n");

printf("Ex=b ==============================================\n");

E.Q\_R();

E.print\_slau();

printf("\n=Otvet==============================================\n");

E.Reshenie();

E.print\_x();

printf("\n\n");

getch();

return 0;

}

*Метод QR* *(для теста 5)*

#include <iostream>

#include <stdio.h>

#include <conio.h>

#include <math.h>

#include <cmath>

#define \_N 6 // число уравнений и неизвестных

using namespace std;

//умножение двух матриц

double\*\* MultAB(int N, double \*\*A, double \*\*B)

{

double \*\*MultAB = new double\* [N];

for (int i=0; i<N; i++)

{

MultAB[i] = new double [N];

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

{

double skalar = 0;

for(int k = 0; k < N; k++)

skalar += A[i][k] \* B[k][j];

MultAB[i][j]=skalar;

}

}

return MultAB;

}

//умножение матрицы на вектор

double\* MultA\_vect(int N, double \*\*A, double \*b)

{

double \*MultA\_vect;

MultA\_vect= new double [N];

for (int i=0; i<N; i++)

{

double skalar = 0;

for(int k = 0; k < N; k++) skalar += A[i][k] \* b[k];

MultA\_vect[i]=skalar;

}

return MultA\_vect;

}

//разность векторов

double\* Razn\_vect(int N, double \*a, double \*b)

{

double \*Razn\_vect;

Razn\_vect= new double [N];

for (int i=0; i<N; i++)

{

Razn\_vect[i]=a[i]-b[i];

}

return Razn\_vect;

}

//приравнять одну матрицу к другой

double\*\* Ravno(int N, double \*\*A )

{

double \*\*Ravno = new double\* [N];

for (int i=0; i<N; i++)

{

Ravno[i] = new double [N];

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

{

Ravno[i][j]=A[i][j];

}

}

return Ravno;

}

//бесконечная норма матрицы(максимум из сумм модулей элементов разных строк)

double NormInf(int N, double \*\*A)

{

double maxA=0;

for (int i=0; i<N; i++)

{

double s =0;

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

{

s+=abs(A[i][j]);

}

if (maxA<s) maxA=s;

}

return maxA;

}

//норма 1 матрицы (максимум из сумм модулей элементов разных столбцов)

double Norm1(int N, double \*\*A)

{

double maxA=0;

for (int i=0; i<N; i++)

{

double s =0;

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

{

s+=abs(A[j][i]);

}

if (maxA<s) maxA=s;

}

return maxA;

}

//бесконечная норма вектора

double NormInfVect(int N, double \*b)

{

double maxb=0;

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

{

if (maxb<abs(b[j])) maxb=abs(b[j]);

}

return maxb;

}

//бесконечная норма вектора

double Norm1Vect(int N, double \*b)

{

double s=0;

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

{

s+=abs(b[j]);

}

return s;

}

//класс - СЛАУ

class matrix

{

public:

matrix(int n);//конструктор

~matrix();//деструктор

void seta(int i, int j, double value); //ввод значение в ячейку (i,j) матрицы системы

void setb(int i, double value);//ввод значение в ячейку (i) столбца свободных членов

void Q\_R(); //QR разложение + вычисление с его помощью обратной матрицы

void print\_slau(); //вывод на печать исходной системы, а также Q и R матриц

void Reshenie(); //решение системы с помощью найденной обратной матрицы

void print\_x(); //вывод на печать решения

void Setcond(); //вычисление числа обусловленности

void ApproxCond(); //оценка числа обусловленности через возмущённое решение

void PrintA\_Aobr();//вывод на печать произведения обратной и исходной матриц

private:

double \*\*a; //матрица системы

double \*\*q; //отртогональная матрица Q

double \*\*r; //верхняя треугольная матрица

double \*\*Aobr; //обратная матрица

double \*b; //столбец свободных членов

double \*x; //решение СЛАУ

double cond; //число обусловленности

int n; //размерность системы

int Indic; //индикатор (1 - используем норму 1, иначе бесконечную)

};

// ------------------------------------------------------

// Конструктор

// ------------------------------------------------------

matrix::matrix(int N)

{

n = N;

a = new double\* [n];

Aobr = new double\* [n];

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

{

a[i] = new double [n];

Aobr[i] = new double [n];

for (int j=0; j<n; j++) {a[i][j]=0;Aobr[i][j]=0;}

}

b = new double [n];

x = new double [n];

for (int i=0; i<n; i++) { b[i]=0; x[i]=0; }

Indic = 0; //по умолчанию используем бесконечную норму

}

// ------------------------------------------------------

// Деструктор

// ------------------------------------------------------

matrix::~matrix() //освобождаем память, занятую массивами

{

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

{

delete a[i]; delete Aobr[i]; delete r[i]; delete q[i];

}

delete[] a;delete[] Aobr; delete[] r; delete[] q;

delete b; delete x;

}

// ------------------------------------------------------

// Вывод СЛАУ на экран

// ------------------------------------------------------

void matrix::print\_slau()

{

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

{

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

{

printf("%.6f \t",a[i][j]);

}

printf("| %.6f ",b[i]);

printf("\n");

}

printf("\n");

printf("= Ortogonal matrix Q ===============================\n");

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

{

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

{

printf("%.6f \t",q[i][j]);

}

printf("\n");

}

printf("\n");

printf("= Matrix R =========================================\n");

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

{

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

{

printf("%.6f \t",r[i][j]);

}

printf("\n");

}

printf("\n");

printf("= Obratnaja A^(-1) =================================\n");

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

{

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

{

printf("%.6f \t",Aobr[i][j]);

}

printf("\n");

}

}

// ------------------------------------------------------

// Печать решения и невязки решения (максимальная по модулю ошибка)

// ------------------------------------------------------

void matrix::print\_x()

{

double max=0,h;

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

h=0;

for (int j=0; j<n; j++) h=h+x[j]\*a[i][j];

if (max<fabs(b[i]-h)) max=fabs(b[i]-h);

printf("x[%i]=%.10f ",i,x[i]);

}

printf("\nMaximal error: %e\n",max);

printf("\n");

}

// ------------------------------------------------------

// Установить значение матрицы a[][]

// ------------------------------------------------------

void matrix::seta(int i, int j, double value)

{

a[i][j] = value;

}

// ------------------------------------------------------

// Установить значение вектора b[]

// ------------------------------------------------------

void matrix::setb(int i, double value)

{

b[i] = value;

}

//находим число обусловленности

void matrix::Setcond()

{

cond = NormInf(n,a)\*NormInf(n,Aobr);

printf("Cond A = %.6f\n",cond);

}

// ------------------------------------------------------

// QR-алгоритм реализуем с помощью метода вращений

// ------------------------------------------------------

void matrix::Q\_R()

{

double \*\*T = new double\* [n]; // T - матрица вращения

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

{

T[i] = new double [n];

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

{

if (i==j) T[i][j]=1;

else T[i][j]=0;

}

}

r = new double\* [n];

q = new double\* [n];

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

{

r[i] = new double [n];

q[i] = new double [n];

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

{

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

q[i][j]=T[i][j];

}

}

//собственно сам алгоритм метода вращений

for (int k=0;k<n-1; k++)

{

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

{

if (r[i][k]!=0)

{

T[k][k]=r[k][k]/sqrt(r[k][k]\*r[k][k]+r[i][k]\*r[i][k]);

T[i][i]=T[k][k];

T[k][i]=r[i][k]/sqrt(r[k][k]\*r[k][k]+r[i][k]\*r[i][k]);

T[i][k]=(-1)\*T[k][i];

}

r=MultAB(n, T, r);

q=MultAB(n, T, q);

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

{

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

{

if (i==j) T[i][j]=1; else T[i][j]=0;

}

}

}

}

//получили верхнюю треугольную матрицу R и обратную q (или что тоже самое транспанированную)

// к ортогональной матрице Q

//находим обратную матрицу A^(-1) = R^(-1)\*Q^(T), решая систему R\*A^(-1) =Q^(T)

// Обратный ход

double h;

for (int P=0;P<n;P++)

{

Aobr[n-1][P]=q[n-1][P]/r[n-1][n-1];

for (int l=(n-1);l>=1;l--)

{

h=q[l-1][P];

for (int k=(l+1);k<=n;k++) h=h-Aobr[k-1][P]\*r[l-1][k-1];

Aobr[l-1][P]=h/r[l-1][l-1];

}

}

//теперь транспонируем q и получаем искомую Q

T=Ravno(n, q);

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

{

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

{

q[i][j]=T[j][i];

}

}

//освобождаем память, занятую массивом T

for (int i=0; i<n; i++) delete T[i]; delete[] T;

}

//решаем систему с помощью обратной матрицы

void matrix::Reshenie()

{

x = MultA\_vect(n,Aobr,b);

}

//печатаем произведение исходной матрицы на обратную

void matrix::PrintA\_Aobr()

{

double \*\*T = new double\* [n];

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

{

T[i] = new double [n];

}

T = MultAB(n,Aobr,a);

printf("\n");

printf(" A^(-1)\*A ===================================\n");

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

{

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

{

printf("%.2e\t ",T[i][j]);//выводим в експоненциальном формате

}

printf("\n");

}

//освобождаем память, занятую массивом T

for (int i=0; i<n; i++) delete T[i]; delete[] T;

}

//оценка числа обусловленности

void matrix::ApproxCond()

{

double deltaX, deltab;

double AppCond;

double \*newb = new double [n]; // вектор свободных членов с возмущением

double \*newX = new double [n]; // вектор решения возмущённой СЛАУ

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

newb[k]=b[k]+0.01\*cos(M\_PI\*k);

newX = MultA\_vect(n,Aobr,newb);//также для получения возмущ реш исп обратную матрицу

printf("\n");

printf("Reshenie s vozmusheniem================================\n");

for (int j=0; j<n; j++) printf("x[%i]=%.5f ",j,newX[j]);

printf("\n");

//получаем оценку числа обусловленности в зависимости от выбора нормы

if (Indic == 1) //норма 1

{

deltab = n\*0.01/Norm1Vect(n,b);

deltaX = Norm1Vect(n,Razn\_vect(n,newX,x))/Norm1Vect(n,x);

AppCond = deltaX/deltab;

printf("Cond A >=%.3f,\n", AppCond);

}

else //бесконечная норма

{

deltab = 0.01/NormInfVect(n,b);

deltaX = NormInfVect(n,Razn\_vect(n,newX,x))/NormInfVect(n,x);

AppCond = deltaX/deltab;

printf("Cond A >=%.3f,\n", AppCond);

}

//освобождаем память, занятую массивами newb, newX

delete newb; delete newX;

}

// ------------------------------------------------------

// Main

// ------------------------------------------------------

int main()

{

double e=0.001;

//тест 5

matrix A(\_N);

int i,j,n;

n=\_N;

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

{

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

{

if (i>=j) A.seta(i,j,10\*e);

else A.seta(i,j,-(1+10\*e));

}

A.seta(i,i,10\*e+1);

A.setb(i,-1);

}

A.setb(n-1,1);

printf("\*\*\*QR-razlojenie\*\*\*\n");

printf("Ax=b ==============================================\n");

A.Q\_R();

A.print\_slau();

printf("\n=Otvet==============================================\n");

A.Reshenie();

A.print\_x();

printf("\n\n");

e=0.000001;

matrix B(\_N);

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

{

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

{

if (i>=j) B.seta(i,j,10\*e);

else B.seta(i,j,-(1+10\*e));

}

B.seta(i,i,10\*e+1);

B.setb(i,-1);

}

B.setb(n-1,1);

printf("\*\*\*QR-razlojenie\*\*\*\n");

printf("Ax=b ==============================================\n");

B.Q\_R();

B.print\_slau();

printf("\n=Otvet==============================================\n");

B.Reshenie();

B.print\_x();

printf("\n\n");

getch();

return 0;

}