1

基本上是对hw1中代码的复制，一共定义了三个函数：

高斯消元法解方程组gem函数，

cholesky递归分解矩阵cholesky\_rec函数

cholesky解方程组cholesky函数

#include<iostream>

#include<cmath>

#include<vector>

//高斯消元法解方程组

std::vector<double> gem(std::vector<std::vector<double>> matrix){

    int n = matrix.size();//矩阵阶数

    std::vector<double> b = {0.23, 0.32,0.33, 0.31};

    for(int i =0;i<n;i++){

        matrix[i].push\_back(b[i]);//加上b，变成增广矩阵

    }

    for(int i=0;i<n;i++){   //高斯消元，变成上三角矩阵

        for(int j =i+1;j<n;j++){

            double multip = matrix[i][j] / matrix[i][i] ;

            for(int k = i;k<n+1;k++){

                matrix[j][k] -= matrix[i][k] \* multip;

            }

        }

    }

    std::vector<double> solution(n,0);  //初始化解向量

    for(int i=n-1;i>-1;i--){    //反代解方程

        for(int j = i+1;j<n;j++){

            matrix[i][n] -= matrix[i][j] \* solution[j];

        }

        solution[i] = matrix[i][n] / matrix[i][i];

    }

    return solution;

}

//cholesky的递归函数

std::vector<std::vector<double>>  cholesky\_rec(int depth,std::vector<std::vector<double>>  matrix){

            if(depth == 0) return {{sqrt(0.05)}};  //一阶分解返回1

            auto matrix\_H = cholesky\_rec(depth-1, matrix);  //递归调用

            std::vector<double> solution(depth, 0); //h向量

            double sum=0;   //为了方便beta的计算

            for(int i =0;i<depth;i++){  //反代解h

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

                    matrix[i][depth] -= solution[j] \* matrix\_H[i][j];

                }

                solution[i] = matrix[i][depth] / matrix\_H[i][i];

                sum += solution[i] \* solution[i];

            }

            double beta =  std::sqrt(matrix[depth][depth] - sum);   //beta

            //创建新的H

            solution.push\_back(beta);

            matrix\_H.push\_back(solution);

            return matrix\_H;

}

//cholesky解方程组

std::vector<double> cholesky(std::vector<std::vector<double>> matrix)

{

    int n = matrix.size();

    //递归地cholesky分解hilbert矩阵，返回下三角矩阵L；

    std::vector<std::vector<double>> L = cholesky\_rec(n-1, matrix);

    //向量y初始化

    std::vector<double> solution\_y(n,0);

    std::vector<double> b = {0.23, 0.32,0.33, 0.31};

    //反代法解向量y

    for(int i=0;i<n;i++){

        for(int j=0;j<i;j++){

            b[i] -= solution\_y[j] \* L[i][j];

        }

        solution\_y[i] = b[i] / L[i][i];

    }

    //x向量

    std::vector<double> solution\_x(n, 0);

    //转置下三角L，获得上三角U

    std::vector<std::vector<double>> U(n, std::vector<double>(n,0));

    for(int i =0;i<n;i++){

        for(int j =0;j<n;j++){

            U[i][j] = L[j][i];

        }

    }

    //反代法解x

    for(int i=n-1;i>-1;i--){

        for(int j = i+1;j<n;j++){

            solution\_y[i] -= U[i][j] \* solution\_x[j];

        }

        solution\_x[i] = solution\_y[i] / U[i][i];

    }

    return solution\_x;

}

int main(){

    //创建矩阵

    std::vector<std::vector<double>> mtx ={{0.05,0.07,0.06,0.05},{0.07,0.10,

    0.08,0.07},{0.06,0.08,0.10,0.09},{0.05,0.07,0.09,0.10}};

    int n =4;

    //两种方法获得解

    auto solution\_gem = gem(mtx);

    auto solution\_cho = cholesky(mtx);

    //输出解

    for(int i=0;i<n;i++){

        std::cout<< solution\_gem[i] << " ";

    }

    std::cout<< std::endl;

    for(int i=0;i<n;i++){

        std::cout<< solution\_cho[i] << " ";

    }

    return 0;

}

## 3

因为之后的程序都是ipynb文件，所以把源代码说明的部分和源代码合并在ipynb中一起提交了。