实验报告

一、实验目的

利用顺序高斯消元法和列主元高斯消元法，求解下面方程组的解。

$$\begin{bmatrix} 1.1348 & 3.8326 & 1.1651 & 3.4017 \\ 0.5301 & 1.7875 & 2.5330 & 1.5435 \\ 3.4129 & 4.9317 & 8.7643 & 1.3142 \\ 1.2371 & 4.9998 & 10.6721 & 0.0147 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 9.5342 \\ 6.3941 \\ 18.4231 \\ 16.9237 \end{bmatrix}$$

通过自己的编程实践，体会列主元高斯消元法可以提高精度这一点。

二、实验方法（要求用自己的语言描述算法）

基于列主元消元法是对于顺序高斯消元法的改进，所以写在一起。（蓝色为列主元高斯消元法添加的部分。）

其中解方程组需要用到公式

$$x_i = \frac{b_i - (\sum_{k=i+1}^{n} a_{i,k} x_k)}{a_{i,i}}$$

求解步骤：

操作 1：进行高斯消元

新建 martrix 对象 data，并初始化其规模为[rows,rows+1]，其中 rows 为矩阵的行数。

新建解向量对象 xs，并初始化为 xs(x1…xrows)，即 rows 维矩阵。

初始化 martrix 为方程组的增广矩阵。

循环：i = 0 -> rows - 1 //表示操作的列标。

    找到 data[i][i] .. data[rows-1][i] 最大的元素，并将最大行和第 i 行交换，同时，解的下标也同样交换。

    循环：j = i+1 -> rows //表示操作的行标。

        计算系数 p := - data[j][i] / data[i][i] //进行高斯消元法的系数。

        data[i] += data[j] * p //进行高斯消元，使 data[j][i]变成 0。

操作 2：解方程组的解

新建向量对象 x(rows)，用于存放解的数值。

循环: i = rows - 1 -> 1 //解的下标

    x[i] = data[i][rows - 1] // 相当于 bi

    循环: j = i+1 -> rows - 1

        x[i] -= data[i][j] * x[j] // 相当于减去 ai,j*xj

    x[i] /= data[i][i] //除去系数 ai,i

    最后打印解

三、实验代码

class::matrix matrix.h

```cpp
#pragma once
#include<iostream>
#include<string>
#include<iomanip>
#include<vector>
#include<initializer_list>
using namespace std;

template <typename T>
class matrix {
public:
    matrix(int rows, int columns):  rows(rows), columns(columns){

    }
    matrix(const matrix& o){
        this->rows = o.rows;
        this->columns = o.columns;
        this->data = o.data;
        this->xs = o.xs;
    }

    void set(initializer_list<initializer_list<T>> d){
        int i = 0, j = 0;
        for(initializer_list<T> d1: d){
            data.emplace_back(vector<T>());
            for(T d2: d1){
                this->data[i].emplace_back(d2);
                j++;
            }
            i++;
        }
    }
    void set_x(initializer_list<string> x){
        for(const string& p: x){
            xs.emplace_back(p);
        }
    }

    void print_matrix(){
        cout << "matrix<" << rows << "x" << columns << ">" << endl;
        for(int i = 0; i < rows; ++i){
            for(int j = 0; j < columns; ++j){
```

```
                cout << setw(16) << data[i][j];
            }
            cout << endl;
        }
    };

    void print_m_with(){
        cout << "matrix<" << rows << "x" << columns << ">with x" <<
endl;
        for(int i = 0; i < rows; ++i){
            cout << setw(4) << xs[i];
            for(int j = 0; j < columns; ++j){
                cout << setw(16) << data[i][j];
            }
            cout << endl;
        }
    }

    //顺序高斯消元法。
    void gs(){
        for(int i = 0; i < columns - 1; ++i){
            for(int j = i+1; j < rows; ++j){
                T a = data[i][i];
                T b = data[j][i];
                T p = -b/a;
                row_apply(j, i, p);
            }
        }
    }

    //主元高斯消元法
    void main_gs(){
        for(int i = 0; i < columns - 1; ++i){
            //讲绝对值最大的列设为主元。
            int max_index = i;
            for(int j = i+1; j < rows; ++ j){
                if(abs(data[j][i]) > abs(data[max_index][i])){
                    max_index = j;
                }
            }
            if(i != max_index){
                swap_rows(i, max_index);
```

```
            }
            //print_m_with();
            // 高斯消元法
            for(int j = i+1; j < rows; ++j){
                T a = data[i][i];
                T b = data[j][i];
                T p = -b/a;
                row_apply(j, i, p);
            }

            //print_m_with();
        }
    }

    void p_solve(){
        cout << "solves" << endl;
        auto* x = new T[rows];

        for(int i = rows - 1; i >= 0; --i){
            T r = data[i][rows];
            for(int j = i+1; j < rows; ++j){
                r-= data[i][j] * x[j];
            }
            x[i] = r / data[i][i];
        }

        for(int i = 0; i < rows; ++i){
            cout << xs[i] << " = " << x[i] << endl;
        }

        delete[] x;
    }

    void row_apply(int line1, int line2, T p){
        for(int i = 0; i < columns; ++i){
            data[line1][i] += data[line2][i] * p;
        }
    }

    void swap_rows(int line1, int line2){
        for(int i = 0; i < columns; ++i){
            T temp = data[line1][i];
```

```
            data[line1][i] = data[line2][i];
            data[line2][i] = temp;
        }
        string x = xs[line1];
        xs[line1] = xs[line2];
        xs[line2] = x;
    }


private:
    vector<vector<T>> data;
    vector<string> xs;
    int columns = 0;
    int rows = 0;
};
```

main.cpp

```
#include <iostream>
#include "matrix.h"

using namespace std;
int main() {
    matrix<float> m(4,5);
    m.set({{1.1348, 3.8326, 1.1651, 3.4017, 9.5342},
          {0.5301, 1.7875, 2.5330, 1.5435, 6.3941},
          {3.4129, 4.9317, 8.7643, 1.3142, 18.4231},
          {1.2371, 4.9998, 10.6721, 0.0147, 16.9237}});
    m.set_x({"x1","x2","x3","x4"});
    matrix<float> m2 = m;

    m.print_m_with();
    cout << ">>>>gs<<<<" << endl;
    m.gs();
    m.print_m_with();
    m.p_solve();
    cout << ">>>>main_gs<<<<" << endl;
    m2.main_gs();
    m2.print_m_with();
    m2.p_solve();

    //cout << "hello world!" << endl;
    return 0;
```

```
}
```

四、实验结果及其讨论

```
C:\Users\t1542\Program\C++\ProjCalMartrix\cmake-build-debug\ProjCalMartrix.exe
matrix<4x5>with x
  x1        1.1348        3.8326        1.1651        3.4017        9.5342
  x2        0.5301        1.7875         2.533        1.5435        6.3941
  x3        3.4129        4.9317        8.7643        1.3142       18.4231
  x4        1.2371        4.9998       10.6721        0.0147       16.9237
>>>>gs<<<<
matrix<4x5>with x
  x1        1.1348        3.8326        1.1651        3.4017        9.5342
  x2             0    -0.0028255       1.98875    -0.0455388       1.94038
  x3             0             0      -4636.54       97.3727      -4539.17
  x4             0             0             0       -4.5934      -4.59338
solves
x1 = 0.99994
x2 = 1.00002
x3 = 1
x4 = 0.999996
>>>>main_gs<<<<
matrix<4x5>with x
  x3        3.4129        4.9317        8.7643        1.3142       18.4231
  x4             0       3.21217       7.49524     -0.461668       10.2457
  x1             0             0       -6.8657       3.27988      -3.58581
  x2             0             0             0      0.907269      0.907269
solves
x3 = 0.999999
x4 = 1
x1 = 1
x2 = 1

Process finished with exit code 0
```

上图中第 2 个矩阵为使用顺序高斯消元法，方程组的解为 $[\mathbf{0.99994\ 1.00002\ 1\ 0.999996}]$，第 3 个矩阵为使用列主元消元法，方程组的解为 $[\mathbf{1\ 1\ 0.999999\ 1}]$。不难看出，方程的精确解为 $[\mathbf{1\ 1\ 1\ 1}]$。

五、总结

观察上述程序的截图，发现使用顺序高斯消元法，$\boldsymbol{a_{3,3}}$ 的元素的绝对值很大，其为使用了过大的倍数乘积所致，其原因为主对角线上的元素的绝对值过下，乘积时导致绝对误差放大的原因，所以使用列主元消元法可以避免过大的倍数乘积。

所以在进行高斯消元法，选择适当的主元是有必要的。