实验报告

一、实验目的

用雅克比迭代法求解下列线性方程组的解：

$$\begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 \\ -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}=\begin{bmatrix} 0 \\ 5 \\ -2 \\ 5 \\ -2 \\ 6 \end{bmatrix}$$

1) 分析并证明收敛性；

2) 编程求出使$\left\|X^{(k+1)}-X^{(k)}\right\|_2 \le 0.001$的近似解以及相应的迭代次数；

3) 用 MATLAB 绘制$\left\|X^{(k+1)}-X^{(k)}\right\|_2$的时间变化曲线。

二、实验方法（要求用自己的语言描述算法）

1）解线性方程组$\boldsymbol{Ax}=\boldsymbol{b}$，可以使用雅可比迭代方程$\boldsymbol{x}^{(\boldsymbol{k}+\boldsymbol{1})}=\boldsymbol{B}\boldsymbol{x}^{(\boldsymbol{k})}+\boldsymbol{f}$。使用雅可比迭代法收敛的条件是$\boldsymbol{B}$的谱半径小于 1，即$\rho(\boldsymbol{B})<1$。其必要条件是$\boldsymbol{B}$的任意一个范数小于 1。

我们求B的 1 范数，即 $\|\boldsymbol{B}\|_1=\max\limits_{j=1\to n}\sum_{i=1}^{n} a_{ij}$

$$D=\begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix} \quad L=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \quad U=\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B=D^{-1}(L+U)=\frac{1}{4}(L+U)=\frac{1}{4}\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$B^T B = B^2 = \frac{1}{16}\begin{bmatrix} 2 & 0 & 2 & 0 & 2 & 0 \\ 0 & 3 & 0 & 3 & 0 & 2 \\ 2 & 0 & 3 & 0 & 3 & 0 \\ 0 & 3 & 0 & 3 & 0 & 2 \\ 2 & 0 & 3 & 0 & 3 & 0 \\ 0 & 2 & 0 & 2 & 0 & 2 \end{bmatrix} \qquad \|\boldsymbol{B}\|_1 = \max_{j=1\to n} \sum_{i=1}^{n} |a_{ij}| = \frac{1}{2} < 1。$$

因此可以得到雅可比迭代式是收敛的。

2) 伪代码

代码关键在于循环内的迭代运算和精度计算。伪代码如下。

```
def main():
    matrix A = {...}  # 初始化矩阵A
    matrix b = {...}  # 初始化向量B
    matrix D = get_dialog(A)  # 初始化对角矩阵D，值参照A
    matrix B = (D ^ -1) * (D - A)  # 计算矩阵B
    matrix f = (D ^ -1) * b  # 计算向量f
    matrix x0 = {...}  # 初始化向量x0 = (1, 1, 1, 1, 1, 1)^T
    matrix x1 = x0
    double eps = 0.001
    do:
        x0 = x1
        x1 = B * x0 + f
        ir = (x1 - x0).vector_norm2()  # 迭代计算
    while ir > eps
```

三、实验代码

main.cpp

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

using namespace std;
int main() {
    matrix A = {
            {4, -1, 0, -1, 0, 0},
            {-1, 4, -1, 0, -1, 0},
            {0, -1, 4, -1, 0, -1},
            {-1, 0, -1, 4, -1, 0},
            {0, -1, 0, -1, 4, -1},
            {0, 0, -1, 0, -1, 4}
    };

```

```
    matrix b = {
            {0},
            {5},
            {-2},
            {5},
            {-2},
            {6}
    };

    A.print_with_title("A");
    //A.print_with_title("A");
    matrix D = A.get_dialog();
    matrix B = (D ^ -1) * (D - A);
    B.print_with_title("B");
    matrix f = (D ^ -1) * b;
    matrix x0 = {{1},{1},{1},{1},{1},{1}};
    matrix x1 = x0;
    double eps = 0.001;
    double ir = 0;

    //int index = 0;
    vector<double> results;
    do{
        //++index;
        x0 = x1;
        x1 = B * x0 + f;
        ir = (x1 - x0).vector_norm2();

        results.emplace_back(ir);
        //cout << index << ":" << ir << endl;
        //x1.print_with_title("x?");

    } while(ir > eps);

    x1.print_with_title("x = ");

    int length = results.size();
    cout << "x = [";
    for (int i = 0; i < length; ++i) {
        cout << (i + 1) << " ";
    }
    cout << "]" << endl << "y = [";
    for (int i = 0; i < length; ++i) {
        cout << results[i] << " ";
```

```
    }
    cout << "]" << endl;

    //std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

matrix matrix.h matrix.cpp

matrix.h

```
//
// Created by t1542 on 2020/3/28.
//

#pragma once
#include <iostream>
#include <iomanip>
#include <string>
#include <initializer_list>
#include <sstream>
using namespace std;

class matrix {
public:
    matrix(initializer_list<initializer_list<double>> data){
        get_row_column(data, _row_count, _column_count);
        fill(data, fills, _row_count, _column_count);
    }
    matrix(const matrix& m){
        _is_error = m._is_error;
        _row_count = m._row_count;
        _column_count = m._column_count;
        fills = new double[_row_count * _column_count];
        for (int i = 0; i < _row_count * _column_count; ++i) {
            fills[i] = m.fills[i];
        }
    }
    int row_count() { return _row_count; }
    int column_count() { return _column_count; }
    void print(int w = 8);
    void print_with_title(const string& title, int w = 8);
    double vector_norm2();

    matrix t();
```

```
    matrix get_dialog();

    ~matrix(){
        if(fills != nullptr){
            delete[] fills;
        }
    }

    friend matrix operator + (const matrix& m1, const matrix& m2);
    friend matrix operator - (const matrix& m1, const matrix& m2);
    friend matrix operator * (const matrix& m1, const matrix& m2);
    friend matrix operator ^ (const matrix& m1, int p);

    matrix& operator = (const matrix& m);

    matrix& operator -();

    static matrix error_instance(){
        matrix m(0,0);
        m._is_error = true;
        return m;
    }
    static matrix e(int row);
    static matrix dialog(initializer_list<double> data);


private:
    matrix(int _row_count, int _column_count): _row_count(_row_count),
_column_count(_column_count){}
    static void
get_row_column(initializer_list<initializer_list<double>>& data, int&
row_count, int& column_count);
    static void fill(initializer_list<initializer_list<double>>& data,
double*& fills, int& row_count, int& column_count);
    static void fill(double*& data, int length);
    matrix inverse() const;
    bool is_dialog() const;

    //void resize(int _rows, int _columns);

    bool _is_error = false;
    // 行数
    int _row_count;
    // 列数
```

```
    int _column_count;
    //填充
    double* fills = nullptr;
};
```

matrix.cpp

```
//
// Created by t1542 on 2020/3/28.
//

#include <cmath>
#include "matrix.h"

void matrix::get_row_column(initializer_list<initializer_list<double>>&
data, int &row_count, int &column_count) {
    row_count = data.size();
    int max_column = data.begin()->size();
    for(auto p: data){
        if(p.size() > max_column){
            max_column = p.size();
        }
    }
    column_count = max_column;
}

void matrix::fill(initializer_list<initializer_list<double>> &data,
double*& fills, int &row_count, int &column_count) {
    fills = new double[row_count * column_count];
    int i = 0;
    for(auto p1: data){
        int j = 0;
        for(auto p2: p1){
            fills[i * column_count + j++] = p2;
        }
        while(j < column_count){
            fills[i * column_count + j++] = 0;
        }
        ++i;
    }
}

void matrix::print(int w) {
    int fill1 = 2;
    if(_is_error){
```

```
        cout << "error!matrix" << endl;
    } else {
        for (int i = 0; i < _row_count; ++i) {
            for (int j = 0; j < _column_count; ++j) {
                cout << setw(w) << fills[i * _column_count + j];
            }
            cout << endl;
        }
    }
}

void matrix::print_with_title(const string& title, int w) {
    int title_fill = title.size();
    int n_fill = 4;
    if(_is_error){
        cout << title << " = error!matrix" << endl;
    } else {
        for (int i = 0; i < _row_count; ++i) {
            cout << setw(title_fill);
            if (i == 0) {
                cout << title;
            } else {
                cout << " ";
            }
            cout << setw(n_fill);
            if (i == 0) {
                cout << " = [";
            } else {
                cout << " ";
            }
            for (int j = 0; j < _column_count; ++j) {
                cout << setw(w) << fills[i * _column_count + j];

            }
            if (i == _row_count - 1){
                cout << " ]";
            }
            cout << endl;
        }
    }
}

matrix matrix::t() {
    matrix m(_column_count, _row_count);
```

```
    m.fills = new double[_row_count * _column_count];
    for (int i = 0; i < _row_count; ++i) {
        for (int j = 0; j < _column_count; ++j) {
            m.fills[j * _row_count + i] = fills[i * _column_count + j];
        }
    }
    return m;
}

matrix operator+(const matrix &m1, const matrix &m2) {
    if(m1._row_count == m2._row_count && m1._column_count ==
m2._column_count){
        int row = m1._row_count;
        int column = m1._column_count;

        matrix m(row, column);
        m.fills = new double[row * column];
        for (int i = 0; i < row; ++i) {
            for (int j = 0; j < column; ++j) {
                m.fills[i * column + j] = m1.fills[i * column + j] +
m2.fills[i * column +j];
            }
        }
        return m;
    } else {
        return matrix::error_instance();
    }
}

matrix &matrix::operator-() {
    for (int i = 0; i < _row_count * _column_count; ++i) {
        fills[i] = - fills[i];
    }
    return *this;
}

matrix operator-(const matrix &m1, const matrix &m2) {
    matrix right = m2;
    return m1 + (-right);
}

matrix operator*(const matrix &m1, const matrix &m2) {
    if (m1._column_count == m2._row_count) {
        int row = m1._row_count;
```

```
        int column = m2._column_count;
        matrix m(row, column);
        m.fills = new double[row * column];
        for (int i = 0; i < row; ++i) {
            for (int j = 0; j < column; ++j) {
                double v = 0;
                for (int k = 0; k < m1._row_count; ++k) {
                    v += m1.fills[i * m1._column_count + k] * m2.fills[k *
m2._column_count + j];
                }
                m.fills[i * column + j] = v;
            }
        }
        return m;
    } else {
        return matrix::error_instance();
    }
}

matrix matrix::e(int row) {
    matrix m(row, row);
    m.fills = new double[row * row];
    fill(m.fills, row*row);

    for (int i = 0; i < row; ++i) {
        m.fills[i] = 1;
    }

    return m;
}

matrix matrix::dialog(initializer_list<double> data) {
    int row = data.size();
    matrix m(row, row);
    m.fills = new double[row * row];
    fill(m.fills, row*row);

    int i = 0;
    for(auto p: data){
        m.fills[i * row + i] = p;
        ++i;
    }

    return m;
```

```
}

void matrix::fill(double*& data, int length) {
    for (int i = 0; i < length; ++i) {
        data[i] = 0;
    }
}

matrix matrix::get_dialog() {
    if(_row_count == _column_count){
        int row = _row_count;
        matrix m(row, row);
        m.fills = new double [row * row];
        fill(m.fills, row * row);

        for (int i = 0; i < row; ++i) {
            m.fills[i * row + i] = fills[i * row + i];
        }
        return m;
    } else {
        return matrix::error_instance();
    }
}

matrix matrix::inverse() const {
    if(_row_count != _column_count){
        return matrix::error_instance();
    }

    int row = _row_count;
    if(is_dialog()){
        matrix m(row, row);
        m.fills = new double[row * row];
        fill(m.fills, row * row);
        for (int i = 0; i < row; ++i) {
            m.fills[i * row + i] = 1.0 / fills[i * row + i];
        }
        return m;
    } else {
        //need to update
        return *this;
    }
}
```

```
bool matrix::is_dialog() const {
    if(_row_count != _column_count){
        return false;
    }
    for (int i = 0; i < _row_count; ++i) {
        for (int j = 0; j < _column_count; ++j) {
            if(i != j && fills[i * _column_count + j] != 0){
                return false;
            }
        }
    }
    return true;
}

matrix operator^(const matrix& m, int p) {
    if(m._row_count != m._column_count){
        return matrix::error_instance();
    }

    if(p == 0){
        return matrix::e(m._row_count);
    } else if(p > 0){
        matrix a = m;
        for (int i = 2; i <= p; ++i) {
            a = a * m;
        }
        return a;
    } else {
        matrix l = m.inverse();
        matrix a = l;
        for (int i = 2; i <= -p; ++i){
            a = a * l;
        }
        return a;
    }
}

matrix &matrix::operator=(const matrix &m) {
    _is_error = m._is_error;
    _row_count = m._row_count;
    _column_count = m._column_count;
    if(fills != nullptr){
        delete[] fills;
        fills = nullptr;
```

```
    }
    fills = new double[_row_count * _column_count];
    for (int i = 0; i < _row_count * _column_count; ++i) {
        fills[i] = m.fills[i];
    }
    return *this;
}

double matrix::vector_norm2() {
    if(_row_count == 1 || _column_count == 1){
        int length = _row_count * _column_count;
        double v = 0;
        for (int i = 0; i < length; ++i) {
            v += fills[i] * fills[i];
        }
        return sqrt(v);
    }
    return -1;
}
```

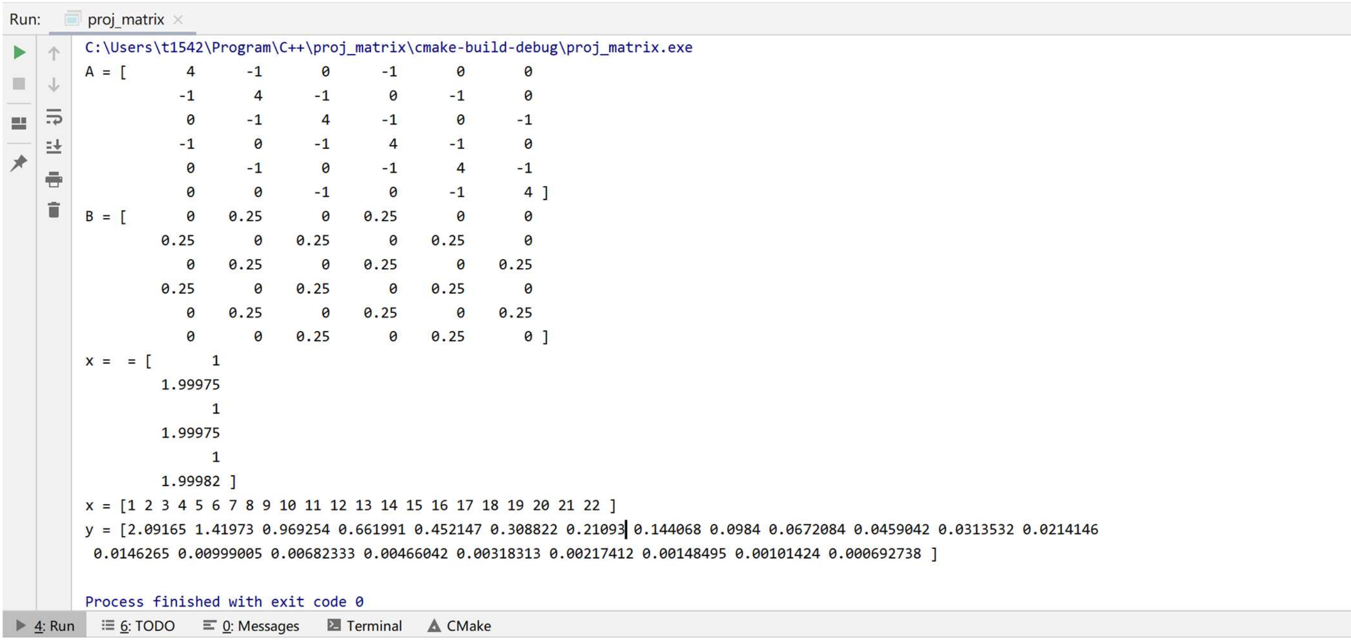
matlab 脚本

```
clear
clc



figure()
x = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 ];
y = [2.09165 1.41973 0.969254 0.661991 0.452147 0.308822 0.21093
0.144068 0.0984 0.0672084 0.0459042 0.0313532 0.0214146 0.0146265
0.00999005 0.00682333 0.00466042 0.00318313 0.00217412 0.00148495
0.00101424 0.000692738 ];
plot(x,y,'-.r*')
```

四、实验结果及其讨论



```
Run:   proj_matrix
C:\Users\t1542\Program\C++\proj_matrix\cmake-build-debug\proj_matrix.exe
A = [       4      -1       0      -1       0       0
           -1       4      -1       0      -1       0
            0      -1       4      -1       0      -1
           -1       0      -1       4      -1       0
            0      -1       0      -1       4      -1
            0       0      -1       0      -1       4 ]
B = [       0    0.25       0    0.25       0       0
         0.25       0    0.25       0    0.25       0
            0    0.25       0    0.25       0    0.25
         0.25       0    0.25       0    0.25       0
            0    0.25       0    0.25       0    0.25
            0       0    0.25       0    0.25       0 ]
x =  = [       1
         1.99975
               1
         1.99975
               1
         1.99982 ]
x = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 ]
y = [2.09165 1.41973 0.969254 0.661991 0.452147 0.308822 0.21093 0.144068 0.0984 0.0672084 0.0459042 0.0313532 0.0214146
 0.0146265 0.00999005 0.00682333 0.00466042 0.00318313 0.00217412 0.00148495 0.00101424 0.000692738 ]

Process finished with exit code 0
4: Run   6: TODO   0: Messages   Terminal   CMake
```

图 1 程序执行结果

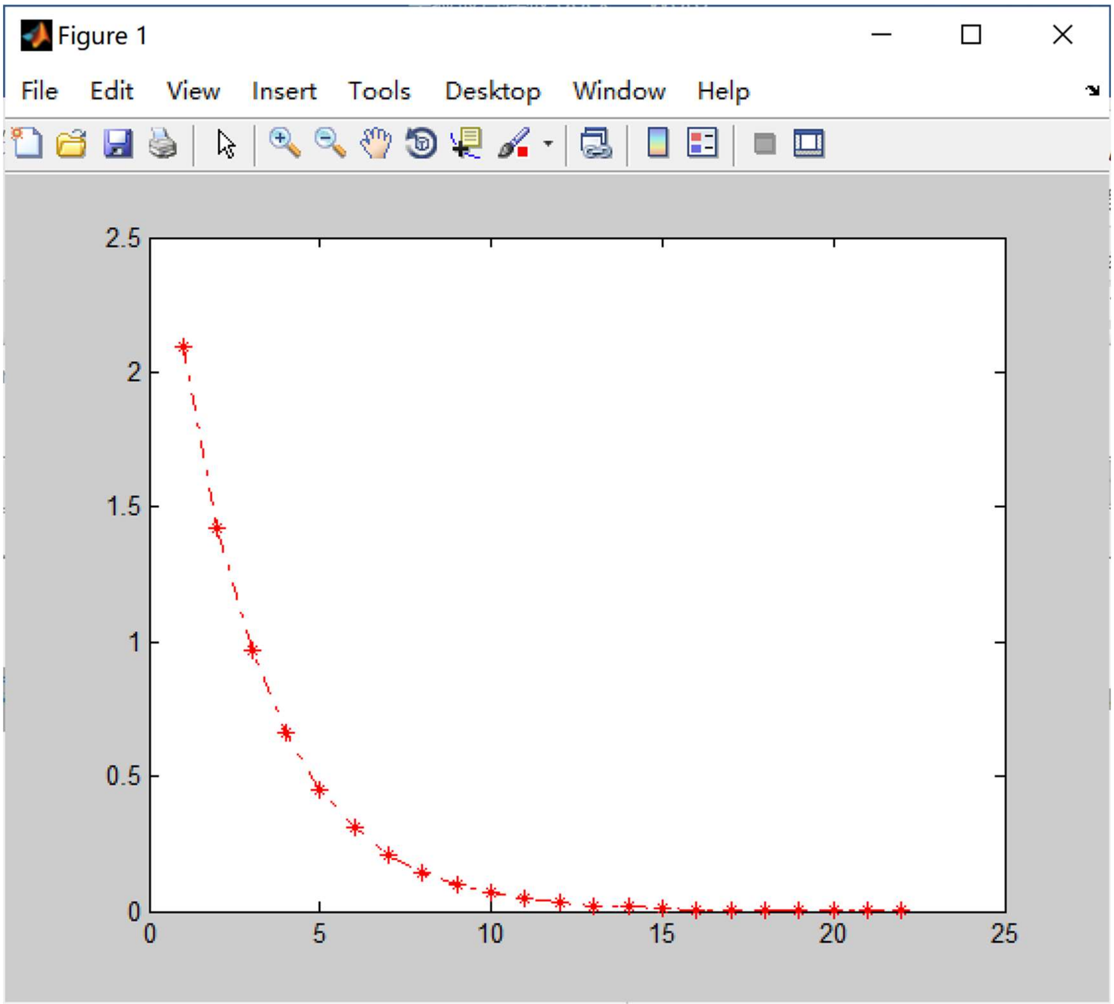


图 2 次数-误差曲线

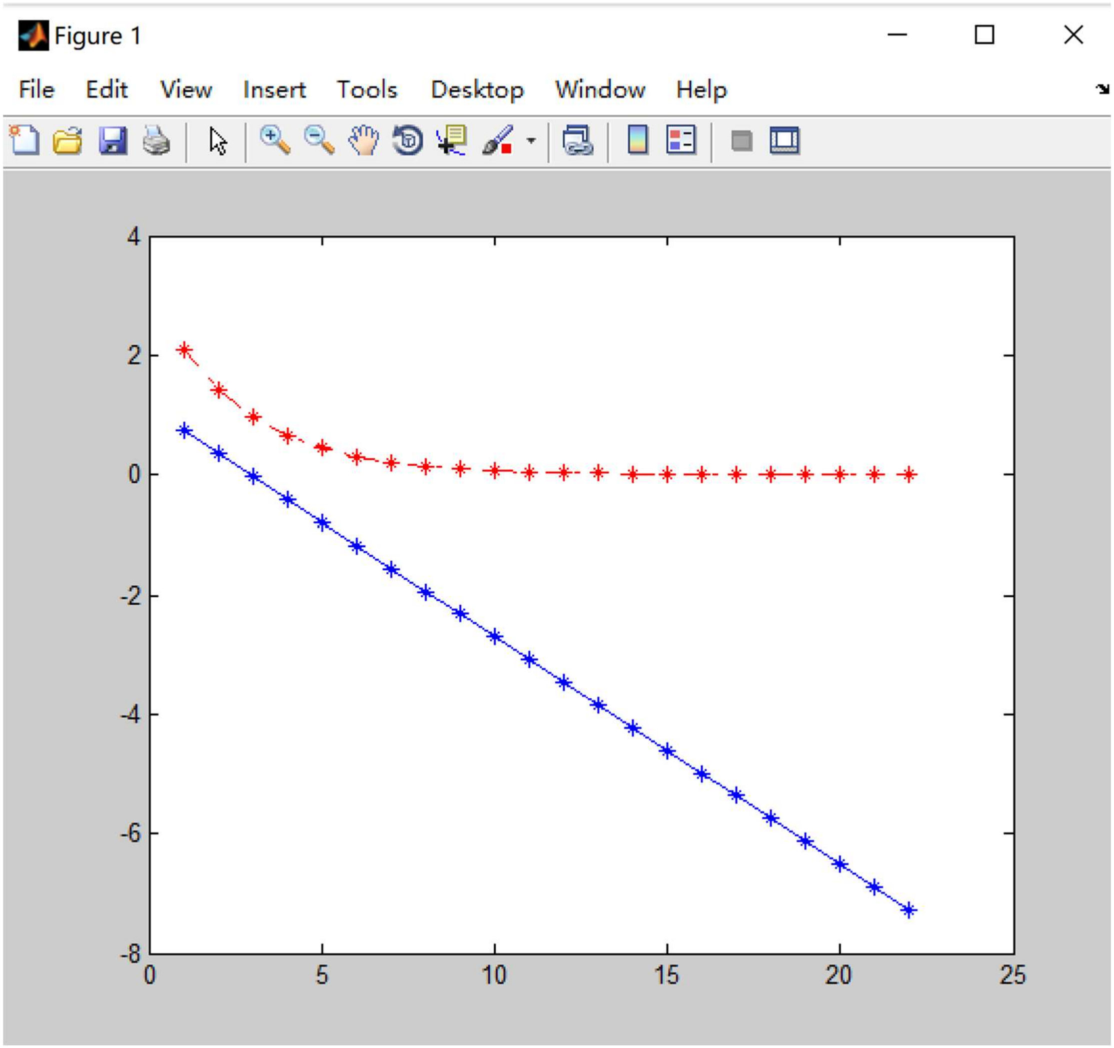


图 3 次数-误差-ln(误差)曲线

本实验从向量$x_0 = [1\ 1\ 1\ 1\ 1\ 1]$开始迭代。

从图 1 的结果可以看出，最后到达精度的解为 $x = [1\ 1.99975\ 1\ 1.99975\ 1\ 1.99982]$，符合精度的要求，猜测精确解为$x = [1\ 2\ 1\ 2\ 1\ 2]$。因此在这个方程组中，雅可比迭代法是有效的。

可以发现，雅可比迭代法的精度呈现指数下降的规律（图 3 的图像验证了这一点），越到后面精度下降越慢。因此在计算简单的方程组的解时，使用主元高斯消元法更加快一些。

五、总结

通过实现，从实践的角度对雅可比迭代法的计算步骤有了更加深刻的理解。在此方法中，涉及到的矩阵操作有：加减法，乘法，对角矩阵求逆，向量求 2 范数，矩阵求 1 范数。所以涉及到的操作并不复杂，没有用到普通矩阵求逆的运算。可以发现，雅可比迭代适合大规模的矩阵求解（而且大规模矩阵一般只有最小二乘解，并没有精确的解）。