

```

1  #include <iostream>
2  #include <iomanip>
3  #include <chrono>
4  #include "Matrix.h"
5  #include "EigenUtils.h"
6  #include "LinearEqUtil.h"
7  #include "ZeroRangeGuard.h"
8
9  using namespace std;
10 using namespace chrono;
11
12 /**
13  * @brief 自定义的输出浮点数函数
14  * @param arg
15  * @return std::string
16  * 采用e型输出实型数来表示arg, 显示12位有效数字, 返回表示的字符串
17  */
18
19 string toScientific(double arg) {
20     stringstream ss;
21     ss << fixed << setprecision(12);
22     auto exp = (arg == 0) ? 0 : 1 + (int) floor(log10(fabs(arg)));
23     auto base = (long long) round(arg * pow(10, ss.precision() - exp));
24     if (base < 0)
25         ss << '-';
26     ss << '.' << abs(base) << 'E' << (exp >= 0 ? '+' : '-') << setw(2) << setfill('0') << abs(exp);
27     return ss.str();
28 }
29
30 int main() {
31     ios::sync_with_stdio(false);
32     //如果是调试模式, 则输出, 否则不输出
33     #ifdef NDEBUG
34         cout.setstate(ios_base::failbit);
35     #endif
36     auto start = system_clock::now();
37
38     auto matA = Matrix::getDefaultMatrix();
39     auto n = matA.size();
40
41     //计算矩阵的特征值
42     ZeroRangeGuard guard1(1e-13);
43     auto eigenValues = EigenUtils::solveEigenValuesWithQrMethod(matA, true);
44
45     //计算矩阵的特征向量
46     ZeroRangeGuard guard2(1e-12);
47     for (int i = 0; i < n; i++) {
48         //如果特征值非实数, 则不计算特征向量
49         if (eigenValues[i].imag() != 0) {
50             cout << "eigenValue: " << "(" << toScientific(eigenValues[i].real()) << ", "
51                 << toScientific(eigenValues[i].imag()) << ")" << endl;
52             cout << endl;
53             continue;
54         }
55         cout << "eigenValue: " << toScientific(eigenValues[i].real()) << "\t";
56
57         //求解齐次线性方程组
58         auto eigenVector = LinearEqUtil::solveHomoLinearEq(matA - eigenValues[i].real(), 1.0);
59         eigenVector.normalize();

```

```
60
61     cout << "eigenVector: [";
62     for (int j = 1; j <= n; j++)
63         cout << toScientific(eigenVector.at(j)) << (j == n ? "]" : ", ");
64     cout << endl;
65
66     //计算||A*x-lambda*x||, 衡量误差
67     cout << "||A*x-lambda*x||="
68         << toScientific((matA * eigenVector - eigenValues[i].real() * eigenVector).normInf()) <<
        endl;
69     cout << endl;
70 }
71
72 auto end = system_clock::now();
73 auto duration = duration_cast<microseconds>(end - start);
74 #ifdef NDEBUG
75     cout.clear();
76 #endif
77 //输出程序运行时间
78 cout << "time: " << duration.count() << " microseconds" << endl;
79 return 0;
80 }
81
```

```

1 //
2 // Created by 40461 on 2021/11/9.
3 //
4
5 #ifndef NUMERICALANALYSIST2_MATRIX_H
6 #define NUMERICALANALYSIST2_MATRIX_H
7
8 #include <vector>
9 #include <cassert>
10 #include "Vector.h"
11
12 /**
13  * @brief 矩阵类
14  * 实现了一些基本的矩阵和数字，矩阵和向量以及矩阵之间的运算
15  * @todo 更完善的运算符重载，以及右值重载，减少频繁内存申请的开销
16  */
17
18 class Matrix {
19 public:
20     explicit Matrix(int _n);
21
22     inline double &at(int i, int j) {
23         assert(i > 0 && i <= n && j > 0 && j <= n);
24         return data[(i - 1) * n + j - 1];
25     }
26
27     inline const double &at(int i, int j) const {
28         return const_cast<Matrix*>(this)->at(i, j);
29     }
30
31     inline int size() const {
32         return n;
33     }
34
35     static Matrix getDefaultMatrix();
36
37     Matrix transpose() const;
38
39     Vector operator*(const Vector &v) const;
40
41     Matrix operator+(const Matrix &m) const;
42
43     Matrix operator-(const Matrix &m) const;
44
45     Matrix operator*(const Matrix &m) const;
46
47     Matrix operator*(double d) const;
48
49     friend Matrix operator*(double d, const Matrix &m);
50
51     Matrix operator+(double d) const;
52
53     Matrix operator-(double d) const;
54
55     Matrix getPrefixMat(int m) const;
56
57     void setPrefixMat(const Matrix &mat);
58
59     void print() const;

```

```
60
61 private:
62     int n;
63     std::vector<double> data;
64 };
65
66
67 #endif //NUMERICALANALYSIST2_MATRIX_H
68
```

```

1 //
2 // Created by 40461 on 2021/11/9.
3 //
4
5 #ifndef NUMERICALANALYSIST2_VECTOR_H
6 #define NUMERICALANALYSIST2_VECTOR_H
7
8 #include <vector>
9 #include <cassert>
10
11 class Matrix;
12
13 /**
14  * @brief 向量类
15  * 实现了一些基本运算
16  * @todo 更完善的运算符重载, 以及右值重载, 减少频繁内存申请的开销
17  */
18
19 class Vector {
20 public:
21     explicit Vector(int n);
22
23     inline double &at(int i) {
24         assert(i >= 1 && i <= data.size());
25         return data[i - 1];
26     }
27
28     inline const double &at(int i) const {
29         return const_cast<Vector*>(this)->at(i);
30     }
31
32     inline int length() const {
33         return (int) data.size();
34     }
35
36     Vector operator/(double x) const;
37
38     Vector operator-(const Vector &v) const;
39
40     /**
41      * @brief 向量点乘
42      * @param v
43      * @return  $self^T * v$ 
44      */
45     double dot(const Vector &v) const;
46
47     /**
48      * @brief 向量外积
49      * @param v
50      * @return  $self * v^T$ 
51      */
52     Matrix outer(const Vector &v) const;
53
54     Vector operator*(double x) const;
55
56     friend Vector operator*(double x, const Vector &v);
57
58     void print() const;
59

```

```
60  double normInf() const;  
61  
62  double norm2() const;  
63  
64  void normalize();  
65  
66  private:  
67      std::vector<double> data;  
68  };  
69  
70  
71  #endif //NUMERICALANALYSIST2_VECTOR_H  
72
```

```

1  //
2  // Created by 40461 on 2021/11/9.
3  //
4
5  #include "Matrix.h"
6  #include <cmath>
7  #include <iostream>
8  #include <iomanip>
9
10 using namespace std;
11
12 Matrix::Matrix(int _n) : n(_n), data(n * n) {}
13
14 Matrix Matrix::getDefaultMatrix() {
15     Matrix mat(10);
16     for (int i = 1; i <= 10; i++)
17         for (int j = 1; j <= 10; j++)
18             mat.at(i, j) = i == j ? 1.52 * cos(i + 1.2 * j) : sin(0.5 * i + 0.2 * j);
19     return mat;
20 }
21
22 Matrix Matrix::transpose() const {
23     Matrix mat(n);
24     for (int i = 1; i <= n; i++)
25         for (int j = 1; j <= n; j++)
26             mat.at(j, i) = at(i, j);
27     return mat;
28 }
29
30 Vector Matrix::operator*(const Vector &v) const {
31     assert(n == v.length());
32     Vector vec(n);
33     for (int i = 1; i <= n; i++)
34         for (int j = 1; j <= n; j++)
35             vec.at(i) += at(i, j) * v.at(j);
36     return vec;
37 }
38
39 Matrix Matrix::operator-(const Matrix &m) const {
40     assert(n == m.n);
41     Matrix mat(n);
42     for (int i = 0; i < data.size(); i++)
43         mat.data[i] = data[i] - m.data[i];
44     return mat;
45 }
46
47 Matrix Matrix::operator*(const Matrix &m) const {
48     assert(n == m.n);
49     Matrix mat(n);
50     for (int i = 1; i <= n; i++)
51         for (int j = 1; j <= n; j++)
52             for (int k = 1; k <= n; k++)
53                 mat.at(i, j) += at(i, k) * m.at(k, j);
54     return mat;
55 }
56
57 Matrix Matrix::operator*(double d) const {
58     Matrix mat(n);
59     for (int i = 0; i < data.size(); i++)

```

```

60     mat.data[i] = data[i] * d;
61     return mat;
62 }
63
64 Matrix operator*(double d, const Matrix &m) {
65     return m * d;
66 }
67
68 void Matrix::print() const {
69     cout << fixed << setprecision(3);
70     for (int i = 1; i <= n; i++) {
71         for (int j = 1; j <= n; j++)
72             cout << at(i, j) << "\t";
73         cout << endl;
74     }
75     cout << endl;
76 }
77
78 Matrix Matrix::getPrefixMat(int m) const {
79     assert(m <= n);
80     Matrix mat(m);
81     for (int i = 1; i <= m; i++)
82         for (int j = 1; j <= m; j++)
83             mat.at(i, j) = at(i, j);
84     return mat;
85 }
86
87 void Matrix::setPrefixMat(const Matrix &mat) {
88     assert(mat.n <= n);
89     for (int i = 1; i <= mat.n; i++)
90         for (int j = 1; j <= mat.n; j++)
91             at(i, j) = mat.at(i, j);
92 }
93
94 Matrix Matrix::operator+(const Matrix &m) const {
95     assert(n == m.n);
96     Matrix mat(n);
97     for (int i = 0; i < data.size(); i++)
98         mat.data[i] = data[i] + m.data[i];
99     return mat;
100 }
101
102 Matrix Matrix::operator+(double d) const {
103     Matrix mat(*this);
104     for (int i = 1; i <= n; i++)
105         mat.at(i, i) += d;
106     return mat;
107 }
108
109 Matrix Matrix::operator-(double d) const {
110     return *this + (-d);
111 }
112

```



```

1 //
2 // Created by 40461 on 2021/11/9.
3 //
4
5 #include "Vector.h"
6 #include "Matrix.h"
7 #include <iostream>
8 #include <iomanip>
9 #include <cmath>
10
11 using namespace std;
12
13 Vector::Vector(int n) : data(n) {}
14
15 Vector Vector::operator/(double x) const {
16     // Vector v(length());
17     // for (int i = 0; i < data.size(); ++i)
18     //     v.data[i] = data[i] / x;
19     // return v;
20     return *this * (1 / x);
21 }
22
23 double Vector::dot(const Vector &v) const {
24     assert(length() == v.length());
25     double sum = 0;
26     for (int i = 0; i < data.size(); ++i)
27         sum += data[i] * v.data[i];
28     return sum;
29 }
30
31 Vector Vector::operator-(const Vector &v) const {
32     Vector w(length());
33     for (int i = 0; i < data.size(); ++i)
34         w.data[i] = data[i] - v.data[i];
35     return w;
36 }
37
38 Matrix Vector::outer(const Vector &v) const {
39     assert(length() == v.length());
40     Matrix m(length());
41     for (int i = 1; i <= length(); ++i)
42         for (int j = 1; j <= v.length(); ++j)
43             m.at(i, j) = at(i) * v.at(j);
44     return m;
45 }
46
47 Vector Vector::operator*(double x) const {
48     Vector v(length());
49     for (int i = 0; i < data.size(); ++i)
50         v.data[i] = data[i] * x;
51     return v;
52 }
53
54 Vector operator*(double x, const Vector &v) {
55     return v * x;
56 }
57
58 void Vector::print() const {
59     cout << fixed << setprecision(3);

```

```
60     for (double i: data)
61         cout << i << "\t";
62     cout << endl;
63 }
64
65 double Vector::normInf() const {
66     double v = 0;
67     for (auto i: data)
68         v = max(v, abs(i));
69     return v;
70 }
71
72 void Vector::normalize() {
73     double n = norm2();
74     for (double &i: data)
75         i /= n;
76 }
77
78 double Vector::norm2() const {
79     double v = 0;
80     for (auto i: data)
81         v += i * i;
82     return sqrt(v);
83 }
84
```

```

1 //
2 // Created by 40461 on 2021/11/9.
3 //
4
5 #ifndef NUMERICALANALYSIST2_EIGENUTILS_H
6 #define NUMERICALANALYSIST2_EIGENUTILS_H
7
8 #include "Matrix.h"
9 #include <complex>
10
11 //采用std::complex作为复数的默认实现
12
13 using Complex = std::complex<double>;
14
15 /**
16  * @brief 计算矩阵的特征值
17  * 采用了双步位移优化的QR方法
18  */
19
20 class EigenUtils {
21 public:
22     /**
23      *
24      * @param mat
25      * @param printInfo 如果为true, 则打印一些过程信息, 包括Hessenberg矩阵, 迭代次数等
26      * @return std::vector<Complex> 特征值列表
27      */
28     static std::vector<Complex>
29     solveEigenValuesWithQrMethod(const Matrix &mat, bool printInfo = false);
30
31 private:
32
33     /**
34      * @brief 计算拟上三角(Hessenberg)矩阵
35      * @param mat
36      * @return Hessenberg矩阵
37      */
38     static Matrix getHessenbergMatrix(const Matrix &mat);
39
40     /**
41      * @brief 通过M矩阵的QR分解来更新矩阵A
42      * @param matA, 可以修改,  $A=Q^T A Q$ 
43      * @param matM
44      * @return void
45      */
46     static void updateWithDoubleStep(Matrix &matA, const Matrix &matM);
47 };
48
49
50 #endif //NUMERICALANALYSIST2_EIGENUTILS_H
51

```

```
1 cmake_minimum_required(VERSION 3.16)
2 project(NumericalAnalysisT2)
3
4 set(CMAKE_CXX_STANDARD 17)
5
6 add_compile_options("$<$<C_COMPILER_ID:MSVC>:/utf-8>")
7 add_compile_options("$<$<CXX_COMPILER_ID:MSVC>:/utf-8>")
8
9 add_executable(NumericalAnalysisT2 main.cpp Matrix.cpp Matrix.h EigenUtils.cpp EigenUtils.
  h Vector.cpp Vector.h LinearEqUtil.cpp LinearEqUtil.h ZeroRangeGuard.h)
```

```

1  //
2  // Created by 40461 on 2021/11/9.
3  //
4
5  #include "EigenUtils.h"
6  #include "ZeroRangeGuard.h"
7  #include <iostream>
8  #include <algorithm>
9
10 using namespace std;
11
12 Matrix EigenUtils::getHessenbergMatrix(const Matrix &mat) {
13     auto matA(mat);
14     int n = matA.size();
15     for (int r = 1; r <= n - 2; r++) {
16         /* bool allZero = true;
17            for (int i = r + 2; i <= n && allZero; i++)
18                if (!isZero(matA.at(i, r)))
19                    allZero = false;
20            if (allZero)
21                continue;
22         */
23         double c = 0;
24         for (int i = r + 1; i <= n; i++)
25             c += matA.at(i, r) * matA.at(i, r);
26         c = sqrt(c);
27         if (matA.at(r + 1, r) > 0)
28             c = -c;
29         auto h = c * c - c * matA.at(r + 1, r);
30         if (ZeroRangeGuard::isZero(h))
31             continue;
32         Vector u(n);
33         for (int i = r + 1; i <= n; i++)
34             u.at(i) = matA.at(i, r);
35         u.at(r + 1) -= c;
36         auto p = matA.transpose() * u / h;
37         auto q = matA * u / h;
38         auto t = p.dot(u) / h;
39         auto omega = q - t * u;
40         matA = matA - omega.outer(u) - u.outer(p);
41     }
42     return matA;
43 }
44
45 vector<Complex> EigenUtils::solveEigenValuesWithQrMethod(const Matrix &mat, bool printInfo) {
46     auto matA = getHessenbergMatrix(mat);
47     if (printInfo) {
48         cout << "Hessenberg Matrix:" << endl;
49         matA.print();
50     }
51     int n = matA.size(), m = n, k = 1;
52
53     vector<Complex> eigenValues;
54     eigenValues.reserve(n);
55     while (m > 0) {
56         if (m == 1 || ZeroRangeGuard::isZero(matA.at(m, m - 1))) {
57             eigenValues.emplace_back(matA.at(m, m));
58             --m;

```

```

59     } else {
60         auto t = matA.at(m - 1, m - 1) * matA.at(m, m) - matA.at(m, m - 1) * matA.at(m - 1, m);
61         auto s = matA.at(m - 1, m - 1) + matA.at(m, m);
62         if ((m == 2 || ZeroRangeGuard::isZero(matA.at(m - 1, m - 2)))) {
63             auto delta = sqrt(Complex(s * s - 4 * t));
64             eigenValues.emplace_back((s + delta) / 2.0);
65             eigenValues.emplace_back((s - delta) / 2.0);
66             m -= 2;
67         } else {
68             auto pMatA = matA.getPrefixMat(m);
69             //auto matM = pMatA * pMatA - s * pMatA + t;
70             auto matM = (pMatA - s) * pMatA + t;
71             updateWithDoubleStep(pMatA, matM);
72             matA.setPrefixMat(pMatA);
73             ++k;
74         }
75     }
76 }
77 if (printlnInfo)
78     cout << "k = " << k << endl;
79 reverse(eigenValues.begin(), eigenValues.end());
80 if (printlnInfo) {
81     cout << "Matrix after QR Method:" << endl;
82     matA.print();
83 }
84 return eigenValues;
85 }
86
87 void EigenUtils::updateWithDoubleStep(Matrix &matA, const Matrix &matM) {
88     auto matB = matM;
89     auto &matC = matA;
90     int m = matA.size();
91     for (int r = 1; r <= m - 1; r++) {
92         double c = 0;
93         for (int i = r; i <= m; i++)
94             c += matB.at(i, r) * matB.at(i, r);
95         c = sqrt(c);
96         if (matB.at(r, r) > 0)
97             c = -c;
98         auto h = c * c - c * matB.at(r, r);
99         if (ZeroRangeGuard::isZero(h))
100             continue;
101         Vector u(m);
102         for (int i = r; i <= m; i++)
103             u.at(i) = matB.at(i, r);
104         u.at(r) -= c;
105
106         auto v = matB.transpose() * u / h;
107         matB = matB - u.outer(v);
108         auto p = matC.transpose() * u / h;
109         auto q = matC * u / h;
110         auto t = p.dot(u) / h;
111         auto omega = q - t * u;
112         matC = matC - omega.outer(u) - u.outer(p);
113     }
114 }
115

```

```
1 //
2 // Created by 40461 on 2021/11/15.
3 //
4
5 #ifndef NUMERICALANALYSIST2_LINEAREQUTIL_H
6 #define NUMERICALANALYSIST2_LINEAREQUTIL_H
7
8
9 #include "Matrix.h"
10
11 /**
12  * @brief 线性方程组的求解
13  * 用列主元高斯消去法，求齐次线性方程组的一个解
14  */
15
16 class LinearEqUtil {
17 public:
18     /**
19      * @brief 求解齐次线性方程组（解空间维度为1）
20      * @param matA
21      * @param freeVariable 为唯一的自由变量赋值
22      * @return Vector 一个解向量
23      */
24     static Vector solveHomoLinearEq(const Matrix &matA, double freeVariable);
25
26     static int gaussElimination(Matrix &mat);
27 };
28
29
30 #endif //NUMERICALANALYSIST2_LINEAREQUTIL_H
31
```

```

1 //
2 // Created by 40461 on 2021/11/15.
3 //
4
5 #include "LinearEqUtil.h"
6 #include "ZeroRangeGuard.h"
7 #include <algorithm>
8
9 using namespace std;
10
11 int LinearEqUtil::gaussElimination(Matrix &matA) {
12     auto n = matA.size();
13     int s = 1, k = 1;
14     for (; s <= n; ++s) {
15         int pos = k;
16         for (int i = k + 1; i <= n; i++)
17             if (abs(matA.at(i, s)) > abs(matA.at(pos, s)))
18                 pos = i;
19         if (ZeroRangeGuard::isZero(matA.at(pos, s)))
20             continue;
21         if (pos != k)
22             for (int j = s; j <= n; j++)
23                 swap(matA.at(k, j), matA.at(pos, j));
24         for (int i = k + 1; i <= n; i++) {
25             auto m = matA.at(i, s) / matA.at(k, s);
26             matA.at(i, s) = 0;
27             for (int j = s + 1; j <= n; j++)
28                 matA.at(i, j) -= m * matA.at(k, j);
29         }
30         ++k;
31     }
32     return k - 1;
33 }
34
35 Vector LinearEqUtil::solveHomoLinearEq(const Matrix &matA, double freeVariable) {
36     auto n = matA.size();
37     Matrix matEliminated = matA;
38     auto rank = gaussElimination(matEliminated);
39     assert(("The dimension of solution space is greater than one", rank == n - 1));
40     Vector res(n);
41     for (int k = n - 1, p = n; k > 0; k--) {
42         if (!ZeroRangeGuard::isZero(matEliminated.at(k, p)) && p > k && !ZeroRangeGuard::isZero(
matEliminated.at(k, p - 1))) {
43             res.at(p) = freeVariable;
44             --p;
45         }
46         double t = 0;
47         for (int i = p + 1; i <= n; i++)
48             t -= matEliminated.at(k, i) * res.at(i);
49         res.at(p) = t / matEliminated.at(k, p);
50         --p;
51     }
52     return res;
53 }

```



```
1 //
2 // Created by 40461 on 2021/11/16.
3 //
4
5 #ifndef NUMERICALANALYSIST2_ZERORANGEGUARD_H
6 #define NUMERICALANALYSIST2_ZERORANGEGUARD_H
7
8 #include <vector>
9
10 /**
11  * @brief 一个控制epsilon范围的守护类
12  * 模仿了Python中的with ...语句
13  * 一个作用域中定义ZeroRangeGuard类的变量, 新的epsilon将会被设置, 直到这个作用域结束才会失效
14  */
15
16 class ZeroRangeGuard {
17 public:
18     explicit ZeroRangeGuard(double range) {
19         rangeList.emplace_back(range);
20     }
21
22     ~ZeroRangeGuard() {
23         rangeList.pop_back();
24     }
25
26     inline static bool isZero(double value) {
27         double &z = rangeList.back();
28         return value <= z && value >= -z;
29     }
30
31     inline static const std::vector<double> &getRangeList() {
32         return rangeList;
33     }
34
35 private:
36     inline static std::vector<double> rangeList{1E-15};
37 };
38
39 #endif //NUMERICALANALYSIST2_ZERORANGEGUARD_H
```