```cpp
#include <iostream>
#include <iomanip>
#include <chrono>
#include "Matrix.h"
#include "EigenUtils.h"
#include "LinearEqUtil.h"
#include "ZeroRangeGuard.h"

using namespace std;
using namespace chrono;

/**
 * @brief 自定义的输出浮点数函数
 * @param arg
 * @return std::string
 * 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
 */

string toScientific(double arg) {
    stringstream ss;
    ss << fixed << setprecision(12);
    auto exp = (arg == 0) ? 0 : 1 + (int) floor(log10(fabs(arg)));
    auto base = (long long) round(arg * pow(10, ss.precision() - exp));
    if (base < 0)
        ss << '-';
    ss << '.' << abs(base) << 'E' << (exp >= 0 ? '+' : '-') << setw(2) << setfill('0') << abs(exp);
    return ss.str();
}

int main() {
    ios::sync_with_stdio(false);
    //如果是调试模式，则输出，否则不输出
#ifdef NDEBUG
    cout.setstate(ios_base::failbit);
#endif
    auto start = system_clock::now();

    auto matA = Matrix::getDefaultMatrix();
    auto n = matA.size();

    //计算矩阵的特征值
    ZeroRangeGuard guard1(1e-13);
    auto eigenValues = EigenUtils::solveEigenValuesWithQrMethod(matA, true);

    //计算矩阵的特征向量
    ZeroRangeGuard guard2(1e-12);
    for (int i = 0; i < n; i++) {
        //如果特征值非实数，则不计算特征向量
        if (eigenValues[i].imag() != 0) {
            cout << "eigenValue: " << "(" << toScientific(eigenValues[i].real()) << ", "
                 << toScientific(eigenValues[i].imag()) << ")" << endl;
            cout << endl;
            continue;
        }
        cout << "eigenValue: " << toScientific(eigenValues[i].real()) << "\t";

        //求解齐次线性方程组
        auto eigenVector = LinearEqUtil::solveHomoLinearEq(matA - eigenValues[i].real(), 1.0);
        eigenVector.normalize();

        cout << "eigenVector: [";
        for (int j = 1; j <= n; j++)
            cout << toScientific(eigenVector.at(j)) << (j == n ? "]" : ", ");
        cout << endl;

        //计算||A*x-lambda*x||，衡量误差
        cout << "||A*x-lambda*x||="
             << toScientific((matA * eigenVector - eigenValues[i].real() * eigenVector).normInf()) << endl;
        cout << endl;
    }

    auto end = system_clock::now();
    auto duration = duration_cast<microseconds>(end - start);
#ifdef NDEBUG
    cout.clear();
#endif
    //输出程序运行时间
    cout << "time: " << duration.count() << " microseconds" << endl;
    return 0;
}

```

```
//
// Created by 40461 on 2021/11/9.
//

#ifndef NUMERICALANALYSIST2_MATRIX_H
#define NUMERICALANALYSIST2_MATRIX_H

#include <vector>
#include <cassert>
#include "Vector.h"

/**
 * @brief 矩阵类
 * 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
 * @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
 */

class Matrix {
public:
    explicit Matrix(int _n);

    inline double &at(int i, int j) {
        assert(i > 0 && i <= n && j > 0 && j <= n);
        return data[(i - 1) * n + j - 1];
    }

    inline const double &at(int i, int j) const {
        return const_cast<Matrix *>(this)->at(i, j);
    }

    inline int size() const {
        return n;
    }

    static Matrix getDefaultMatrix();

    Matrix transpose() const;

    Vector operator*(const Vector &v) const;

    Matrix operator+(const Matrix &m) const;

    Matrix operator-(const Matrix &m) const;

    Matrix operator*(const Matrix &m) const;

    Matrix operator*(double d) const;

    friend Matrix operator*(double d, const Matrix &m);

    Matrix operator+(double d) const;

    Matrix operator-(double d) const;

    Matrix getPrefixMat(int m) const;

    void setPrefixMat(const Matrix &mat);

    void print() const;
```

```cpp

private:
    int n;
    std::vector<double> data;
};


#endif //NUMERICALANALYSIST2_MATRIX_H

```

```
//
// Created by 40461 on 2021/11/9.
//

#ifndef NUMERICALANALYSIST2_VECTOR_H
#define NUMERICALANALYSIST2_VECTOR_H

#include <vector>
#include <cassert>

class Matrix;

/**
 * @brief 向量类
 * 实现了一些基本运算
 * @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
 */

class Vector {
public:
    explicit Vector(int n);

    inline double &at(int i) {
        assert(i >= 1 && i <= data.size());
        return data[i - 1];
    }

    inline const double &at(int i) const {
        return const_cast<Vector *>(this)->at(i);
    }

    inline int length() const {
        return (int) data.size();
    }

    Vector operator/(double x) const;

    Vector operator-(const Vector &v) const;

    /**
     * @brief 向量点乘
     * @param v
     * @return self^T * v
     */
    double dot(const Vector &v) const;

    /**
     * @brief 向量外积
     * @param v
     * @return self * v^T
     */
    Matrix outer(const Vector &v) const;

    Vector operator*(double x) const;

    friend Vector operator*(double x, const Vector &v);

    void print() const;

    double normInf() const;

    double norm2() const;

    void normalize();

private:
    std::vector<double> data;
};


#endif //NUMERICALANALYSIST2_VECTOR_H

```

```
//
// Created by 40461 on 2021/11/9.
//

#include "Matrix.h"
#include <cmath>
#include <iostream>
#include <iomanip>

using namespace std;

Matrix::Matrix(int _n) : n(_n), data(n * n) {}

Matrix Matrix::getDefaultMatrix() {
    Matrix mat(10);
    for (int i = 1; i <= 10; i++)
        for (int j = 1; j <= 10; j++)
            mat.at(i, j) = i == j ? 1.52 * cos(i + 1.2 * j) : sin(0.5 * i + 0.2 * j);
    return mat;
}

Matrix Matrix::transpose() const {
    Matrix mat(n);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            mat.at(j, i) = at(i, j);
    return mat;
}

Vector Matrix::operator*(const Vector &v) const {
    assert(n == v.length());
    Vector vec(n);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            vec.at(i) += at(i, j) * v.at(j);
    return vec;
}

Matrix Matrix::operator-(const Matrix &m) const {
    assert(n == m.n);
    Matrix mat(n);
    for (int i = 0; i < data.size(); i++)
        mat.data[i] = data[i] - m.data[i];
    return mat;
}

Matrix Matrix::operator*(const Matrix &m) const {
    assert(n == m.n);
    Matrix mat(n);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            for (int k = 1; k <= n; k++)
                mat.at(i, j) += at(i, k) * m.at(k, j);
    return mat;
}

Matrix Matrix::operator*(double d) const {
    Matrix mat(n);
    for (int i = 0; i < data.size(); i++)
```

```cpp
        mat.data[i] = data[i] * d;
    return mat;
}

Matrix operator*(double d, const Matrix &m) {
    return m * d;
}

void Matrix::print() const {
    cout << fixed << setprecision(3);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++)
            cout << at(i, j) << "\t";
        cout << endl;
    }
    cout << endl;
}

Matrix Matrix::getPrefixMat(int m) const {
    assert(m <= n);
    Matrix mat(m);
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= m; j++)
            mat.at(i, j) = at(i, j);
    return mat;
}

void Matrix::setPrefixMat(const Matrix &mat) {
    assert(mat.n <= n);
    for (int i = 1; i <= mat.n; i++)
        for (int j = 1; j <= mat.n; j++)
            at(i, j) = mat.at(i, j);
}

Matrix Matrix::operator+(const Matrix &m) const {
    assert(n == m.n);
    Matrix mat(n);
    for (int i = 0; i < data.size(); i++)
        mat.data[i] = data[i] + m.data[i];
    return mat;
}

Matrix Matrix::operator+(double d) const {
    Matrix mat(*this);
    for (int i = 1; i <= n; i++)
        mat.at(i, i) += d;
    return mat;
}

Matrix Matrix::operator-(double d) const {
    return *this + (-d);
}

```

```cpp
//
// Created by 40461 on 2021/11/9.
//

#include "Vector.h"
#include "Matrix.h"
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

Vector::Vector(int n) : data(n) {}

Vector Vector::operator/(double x) const {
//    Vector v(length());
//    for (int i = 0; i < data.size(); ++i)
//        v.data[i] = data[i] / x;
//    return v;
    return *this * (1 / x);
}

double Vector::dot(const Vector &v) const {
    assert(length() == v.length());
    double sum = 0;
    for (int i = 0; i < data.size(); ++i)
        sum += data[i] * v.data[i];
    return sum;
}

Vector Vector::operator-(const Vector &v) const {
    Vector w(length());
    for (int i = 0; i < data.size(); ++i)
        w.data[i] = data[i] - v.data[i];
    return w;
}

Matrix Vector::outer(const Vector &v) const {
    assert(length() == v.length());
    Matrix m(length());
    for (int i = 1; i <= length(); ++i)
        for (int j = 1; j <= v.length(); ++j)
            m.at(i, j) = at(i) * v.at(j);
    return m;
}

Vector Vector::operator*(double x) const {
    Vector v(length());
    for (int i = 0; i < data.size(); ++i)
        v.data[i] = data[i] * x;
    return v;
}

Vector operator*(double x, const Vector &v) {
    return v * x;
}

void Vector::print() const {
    cout << fixed << setprecision(3);
    for (double i: data)
        cout << i << "\t";
    cout << endl;
}

double Vector::normInf() const {
    double v = 0;
    for (auto i: data)
        v = max(v, abs(i));
    return v;
}

void Vector::normalize() {
    double n = norm2();
    for (double &i: data)
        i /= n;
}

double Vector::norm2() const {
    double v = 0;
    for (auto i: data)
        v += i * i;
    return sqrt(v);
}

```

```
//
// Created by 40461 on 2021/11/9.
//

#ifndef NUMERICALANALYSIST2_EIGENUTILS_H
#define NUMERICALANALYSIST2_EIGENUTILS_H

#include "Matrix.h"
#include <complex>

//采用std::complex作为复数的默认实现

using Complex = std::complex<double>;

/**
 * @brief 计算矩阵的特征值
 * 采用了双步位移优化的QR方法
 */

class EigenUtils {
public:
    /**
     *
     * @param mat
     * @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
     * @return std::vector<Complex> 特征值列表
     */
    static std::vector<Complex>
    solveEigenValuesWithQrMethod(const Matrix &mat, bool printInfo = false);

private:

    /**
     * @brief 计算拟上三角(Hessenberg)矩阵
     * @param mat
     * @return Hessenberg矩阵
     */
    static Matrix getHessenbergMatrix(const Matrix &mat);

    /**
     * @brief 通过M矩阵的QR分解来更新矩阵A
     * @param matA，可以修改，A=Q^T*A*Q
     * @param matM
     * @return void
     */
    static void updateWithDoubleStep(Matrix &matA, const Matrix &matM);
};


#endif //NUMERICALANALYSIST2_EIGENUTILS_H

```

```
cmake_minimum_required(VERSION 3.16)
project(NumericalAnalysisT2)

set(CMAKE_CXX_STANDARD 17)

add_compile_options("$<$<C_COMPILER_ID:MSVC>:/utf-8>")
add_compile_options("$<$<CXX_COMPILER_ID:MSVC>:/utf-8>")

add_executable(NumericalAnalysisT2 main.cpp Matrix.cpp Matrix.h EigenUtils.cpp EigenUtils.h Vector.cpp Vector.h LinearEqUtil.cpp LinearEqUtil.h ZeroRangeGuard.h)
```

```cpp
//
// Created by 40461 on 2021/11/9.
//

#include "EigenUtils.h"
#include "ZeroRangeGuard.h"
#include <iostream>
#include <algorithm>

using namespace std;

Matrix EigenUtils::getHessenbergMatrix(const Matrix &mat) {
    auto matA(mat);
    int n = matA.size();
    for (int r = 1; r <= n - 2; r++) {
        /*  bool allZero = true;
            for (int i = r + 2; i <= n && allZero; i++)
                if (!isZero(matA.at(i, r)))
                    allZero = false;
            if (allZero)
                continue;
        */
        double c = 0;
        for (int i = r + 1; i <= n; i++)
            c += matA.at(i, r) * matA.at(i, r);
        c = sqrt(c);
        if (matA.at(r + 1, r) > 0)
            c = -c;
        auto h = c * c - c * matA.at(r + 1, r);
        if (ZeroRangeGuard::isZero(h))
            continue;
        Vector u(n);
        for (int i = r + 1; i <= n; i++)
            u.at(i) = matA.at(i, r);
        u.at(r + 1) -= c;
        auto p = matA.transpose() * u / h;
        auto q = matA * u / h;
        auto t = p.dot(u) / h;
        auto omega = q - t * u;
        matA = matA - omega.outer(u) - u.outer(p);
    }
    return matA;
}

vector<Complex> EigenUtils::solveEigenValuesWithQrMethod(const Matrix &mat, bool printInfo) {
    auto matA = getHessenbergMatrix(mat);
    if (printInfo) {
        cout << "Hessenberg Matrix:" << endl;
        matA.print();
    }
    int n = matA.size(), m = n, k = 1;

    vector<Complex> eigenValues;
    eigenValues.reserve(n);
    while (m > 0) {
        if (m == 1 || ZeroRangeGuard::isZero(matA.at(m, m - 1))) {
            eigenValues.emplace_back(matA.at(m, m));
            --m;
        } else {
            auto t = matA.at(m - 1, m - 1) * matA.at(m, m) - matA.at(m, m - 1) * matA.at(m - 1, m);
            auto s = matA.at(m - 1, m - 1) + matA.at(m, m);
            if ((m == 2 || ZeroRangeGuard::isZero(matA.at(m - 1, m - 2)))) {
                auto delta = sqrt(Complex(s * s - 4 * t));
                eigenValues.emplace_back((s + delta) / 2.0);
                eigenValues.emplace_back((s - delta) / 2.0);
                m -= 2;
            } else {
                auto pMatA = matA.getPrefixMat(m);
                //auto matM = pMatA *pMatA- s * pMatA +t;
                auto matM = (pMatA - s) * pMatA + t;
                updateWithDoubleStep(pMatA, matM);
                matA.setPrefixMat(pMatA);
                ++k;
            }
        }
    }
    if (printInfo)
        cout << "k = " << k << endl;
    reverse(eigenValues.begin(), eigenValues.end());
    if (printInfo) {
        cout << "Matrix after QR Method:" << endl;
        matA.print();
    }
    return eigenValues;
}

void EigenUtils::updateWithDoubleStep(Matrix &matA, const Matrix &matM) {
    auto matB = matM;
    auto &matC = matA;
    int m = matA.size();
    for (int r = 1; r <= m - 1; r++) {
        double c = 0;
        for (int i = r; i <= m; i++)
            c += matB.at(i, r) * matB.at(i, r);
        c = sqrt(c);
        if (matB.at(r, r) > 0)
            c = -c;
        auto h = c * c - c * matB.at(r, r);
        if (ZeroRangeGuard::isZero(h))
            continue;
        Vector u(m);
        for (int i = r; i <= m; i++)
            u.at(i) = matB.at(i, r);
        u.at(r) -= c;

        auto v = matB.transpose() * u / h;
        matB = matB - u.outer(v);
        auto p = matC.transpose() * u / h;
        auto q = matC * u / h;
        auto t = p.dot(u) / h;
        auto omega = q - t * u;
        matC = matC - omega.outer(u) - u.outer(p);
    }
}

```

```cpp
//
// Created by 40461 on 2021/11/15.
//

#ifndef NUMERICALANALYSIST2_LINEAREQUTIL_H
#define NUMERICALANALYSIST2_LINEAREQUTIL_H


#include "Matrix.h"

/**
 * @brief 线性方程组的求解
 * 用列主元高斯消去法，求齐次线性方程组的一个解
 */

class LinearEqUtil {
public:
    /**
     * @brief 求解齐次线性方程组（解空间维度为1）
     * @param matA
     * @param freeVariable 为唯一的自由变量赋值
     * @return Vector 一个解向量
     */
    static Vector solveHomoLinearEq(const Matrix &matA, double freeVariable);

    static int gaussElimination(Matrix &mat);
};


#endif //NUMERICALANALYSIST2_LINEAREQUTIL_H

```

```cpp
//
// Created by 40461 on 2021/11/15.
//

#include "LinearEqUtil.h"
#include "ZeroRangeGuard.h"
#include <algorithm>

using namespace std;

int LinearEqUtil::gaussElimination(Matrix &matA) {
    auto n = matA.size();
    int s = 1, k = 1;
    for (; s <= n; ++s) {
        int pos = k;
        for (int i = k + 1; i <= n; i++)
            if (abs(matA.at(i, s)) > abs(matA.at(pos, s)))
                pos = i;
        if (ZeroRangeGuard::isZero(matA.at(pos, s)))
            continue;
        if (pos != k)
            for (int j = s; j <= n; j++)
                swap(matA.at(k, j), matA.at(pos, j));
        for (int i = k + 1; i <= n; i++) {
            auto m = matA.at(i, s) / matA.at(k, s);
            matA.at(i, s) = 0;
            for (int j = s + 1; j <= n; j++)
                matA.at(i, j) -= m * matA.at(k, j);
        }
        ++k;
    }
    return k - 1;
}

Vector LinearEqUtil::solveHomoLinearEq(const Matrix &matA, double freeVariable) {
    auto n = matA.size();
    Matrix matEliminated = matA;
    auto rank = gaussElimination(matEliminated);
    assert(("The dimension of solution space is greater than one", rank == n - 1));
    Vector res(n);
    for (int k = n - 1, p = n; k > 0; k--) {
        if (!ZeroRangeGuard::isZero(matEliminated.at(k, p)) && p > k && !ZeroRangeGuard::isZero(matEliminated.at(k, p - 1))) {
            res.at(p) = freeVariable;
            --p;
        }
        double t = 0;
        for (int i = p + 1; i <= n; i++)
            t -= matEliminated.at(k, i) * res.at(i);
        res.at(p) = t / matEliminated.at(k, p);
        --p;
    }
    return res;
}
```

```
//
// Created by 40461 on 2021/11/16.
//

#ifndef NUMERICALANALYSIST2_ZERORANGEGUARD_H
#define NUMERICALANALYSIST2_ZERORANGEGUARD_H

#include <vector>

/**
 * @brief      一个控制epsilon范围的守护类
 * 模仿了Python中的with ...:语句
 * 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
 */

class ZeroRangeGuard {
public:
    explicit ZeroRangeGuard(double range) {
        rangeList.emplace_back(range);
    }

    ~ZeroRangeGuard() {
        rangeList.pop_back();
    }

    inline static bool isZero(double value) {
        double &z = rangeList.back();
        return value <= z && value >= -z;
    }

    inline static const std::vector<double> &getRangeList() {
        return rangeList;
    }

private:
    inline static std::vector<double> rangeList{1E-15};
};

#endif //NUMERICALANALYSIST2_ZERORANGEGUARD_H
```