```cpp
#include <iostream>
#include <iomanip>
#include "SymBandMatrix.h"
#include "EigenValueUtils.h"
#include "LinearEqLuSolver.h"
#include "TimerUtil.h"
#include <functional>

using namespace std;
using EigenValueMethodType = function<pair<double, Vector>(const SymBandMatrix &)>;

/**
 * @brief 自定义的输出浮点数函数
 * @param arg
 * 采用e型输出实型数显示arg，显示12位有效数字
 */

void customPrint(double arg) {
    TimerUtil::finish("solve");
    cout << fixed << setprecision(12);
    auto exp = (arg == 0) ? 0 : 1 + (int) floor(log10(fabs(arg)));
    auto base = (long long) round(arg * pow(10, cout.precision() - exp));
    if (base < 0)
        cout << '-';
    cout << '.' << abs(base) << 'E' << (exp >= 0 ? '+' : '-') << setw(2) << setfill('0') << abs(exp) << endl;
    cout << defaultfloat;
    TimerUtil::start("solve");
}

/**
 * @brief 求解主体
 * @param 函数对象powerMethod
 * @param 函数对象invPowerMethod
 * 传入幂法和反幂法套件，求解，输出答案
 */

void solve(const EigenValueMethodType &powerMethod, const EigenValueMethodType &invPowerMethod) {
    auto matrix = SymBandMatrix::genDefaultMatrix();
    TimerUtil::start("solve");
    auto lambda1 = powerMethod(matrix).first;
    customPrint(lambda1);

    matrix.addConstOnDiagonal(-lambda1);
    auto lambda501 = powerMethod(matrix).first + lambda1;
    customPrint(lambda501);
    matrix.addConstOnDiagonal(lambda1);

    auto lambdaS = invPowerMethod(matrix).first;
    customPrint(lambdaS);

    for (int k = 1; k < 40; k++) {
        auto mu = lambda1 + k * (lambda501 - lambda1) / 40;
        matrix.addConstOnDiagonal(-mu);
        auto lambda = invPowerMethod(matrix).first + mu;
        customPrint(lambda);
        matrix.addConstOnDiagonal(mu);
    }
```

```
58
59      auto cond2 = abs(lambda1 / lambdaS);
60      customPrint(cond2);
61
62      LinearEqLuSolver solver(matrix);
63
64      double det = 1;
65      for (int i = 1; i <= matrix.size(); i++)
66          det *= solver.getU().at(i, i);
67      customPrint(det);
68      TimerUtil::finish("solve");
69  }
70
71  /**
72   * @brief 对EigenValueUtils中函数的包装器
73   * @tparam Func
74   * @param x
75   * @return 一个lambda表达式，相当于为函数设置默认eps=1E-12的绑定
76   */
77  template<typename Func>
78  inline auto wrapEps(Func x) {
79      return [=](auto &&param) { return x(forward<decltype(param)>(param), 1E-12); };
80  }
81
82  int main() {
83      ios::sync_with_stdio(false);
84      //2-范数套件
85      solve(wrapEps(EigenValueUtils::powerMethodWithNorm2),
86            wrapEps(EigenValueUtils::invPowerMethodWithNorm2));
87      TimerUtil::printAllTime();
88      TimerUtil::clear();
89      //无穷范数套件
90      solve(wrapEps(EigenValueUtils::powerMethodWithNormInf),
91            wrapEps(EigenValueUtils::invPowerMethodWithNormInf));
92      TimerUtil::printAllTime();
93      return 0;
94  }
```

```
1  //
2  // Created by 40461 on 2021/10/30.
3  //
4
5  #ifndef NUMERICALANALYSIST1_VECTOR_H
6  #define NUMERICALANALYSIST1_VECTOR_H
7
8  #include <vector>
9  #include <random>
10 #include <cassert>
11
12 /**
13  * @brief 向量类
14  * 实现向量的基本运算，可以使用随机化的方式初始化
15  */
16
17 class Vector {
18 public:
19     Vector() = default;
20
21     explicit Vector(int n, bool randomInit = false);
22
23     explicit Vector(std::vector<double> &&_data);
24
25     double norm2() const;
26
27     double normInf() const;
28
29     int maxNormElement() const;
30
31     inline double &at(int i) {
32         assert(i >= 1 && i <= data.size());
33         return data[i - 1];
34     }
35
36     inline const double &at(int i) const {
37         return const_cast<Vector *>(this)->at(i);
38     }
39
40     inline int length() const {
41         return (int) data.size();
42     }
43
44     Vector operator/(double t) const;
45
46     double dot(const Vector &v) const;
47
48 private:
49     std::vector<double> data;
50 };
51
52
53 #endif //NUMERICALANALYSIST1_VECTOR_H
54
```

```cpp
//
// Created by 40461 on 2021/10/30.
//

#include "Vector.h"
#include <random>
#include <chrono>

using namespace std;

Vector::Vector(int n, bool randomInit) : data(n) {
    if (randomInit) {
        static default_random_engine engineForVecGen(chrono::system_clock::now().
time_since_epoch().count());
        normal_distribution<double> distribution(0, 3.2);
        for (int i = 1; i <= n; i++)
            at(i) = distribution(engineForVecGen);
    }
}

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

Vector Vector::operator/(double t) const {
    Vector r(length());
    for (int i = 1; i <= length(); i++)
        r.at(i) = at(i) / t;
    return r;
}

double Vector::dot(const Vector &v) const {
    double ans = 0;
    for (int i = 1; i <= length(); i++)
        ans += at(i) * v.at(i);
    return ans;
}

Vector::Vector(vector<double> &&_data) : data(move(_data)) {}

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

int Vector::maxNormElement() const {
    int r = 1;
    for (int i = 2; i <= length(); i++)
        if (abs(at(r)) < abs(at(i)))
            r = i;
    return r;
}

```

```
1  //
2  // Created by 40461 on 2021/10/31.
3  //
4
5  #ifndef NUMERICALANALYSIST1_TIMERUTIL_H
6  #define NUMERICALANALYSIST1_TIMERUTIL_H
7
8
9  #include <chrono>
10 #include <map>
11 #include <string>
12
13 /**
14  * @brief 一个简单的计时器
15  * @attention 调用时建议采用字符串字面量作为参数
16  */
17
18 class TimerUtil {
19 public:
20     static void start(std::string_view name);
21
22     static void finish(std::string_view name);
23
24     static void printAllTime();
25
26     static void clear();
27
28 private:
29     inline static std::map<std::string_view, std::pair<decltype(std::chrono::system_clock::now()),
    double>> timer;
30 };
31
32 #endif //NUMERICALANALYSIST1_TIMERUTIL_H
33
```

```cpp
//
// Created by 40461 on 2021/10/31.
//

#include "TimerUtil.h"
#include <iostream>

using namespace std;
using namespace chrono;

void TimerUtil::start(string_view name) {
    timer[name].first = system_clock::now();
}

void TimerUtil::finish(string_view name) {
    auto &item = timer[name];
    item.second += (double) duration_cast<microseconds>(system_clock::now() - item.first).count();
}

void TimerUtil::printAllTime() {
    for (auto &[name, item]:timer)
        cout << name << ": " << item.second * microseconds::period::num / microseconds::period::den << endl;
}

void TimerUtil::clear() {
    timer.clear();
}

```

```
cmake_minimum_required(VERSION 3.16)
project(NumericalAnalysisT1)

set(CMAKE_CXX_STANDARD 17)

add_executable(NumericalAnalysisT1 main.cpp SymBandMatrix.cpp SymBandMatrix.h
EigenValueUtils.cpp EigenValueUtils.h Vector.cpp Vector.h LinearEqLuSolver.cpp
LinearEqLuSolver.h TimerUtil.cpp TimerUtil.h)
```

```
1  //
2  // Created by 40461 on 2021/10/30.
3  //
4
5  #ifndef NUMERICALANALYSIST1_SYMBANDMATRIX_H
6  #define NUMERICALANALYSIST1_SYMBANDMATRIX_H
7
8  #include <vector>
9  #include <cassert>
10 #include "Vector.h"
11
12 /**
13  * @brief 实对称带状矩阵
14  * 内部采用压缩存储方式 A(i,j)
15  */
16
17 class SymBandMatrix {
18 public:
19     SymBandMatrix(int _n, int _width);
20
21     /**
22      * @brief 下标索引方法
23      * 我们只存储上三角部分，并将上三角部分进行压缩映射
24      * @param i 行参数
25      * @param j 列参数
26      * @return 返回未压缩矩阵(i,j)位置元素的引用，注意(i,j)和(j,i)返回的是同一个引用
27      */
28     inline double &at(int i, int j) {
29         if (i > j)
30             std::swap(i, j);
31         assert(i >= 1 && j <= n && j - i <= w);
32         return data[(i - j + w) * n + (j - 1)];
33     }
34
35     inline const double &at(int i, int j) const {
36         return const_cast<SymBandMatrix *>(this)->at(i, j);
37     }
38
39     inline int size() const {
40         return n;
41     }
42
43     inline int width() const {
44         return w;
45     }
46
47     /**
48      * @brief 和向量的乘法
49      * @param v
50      * @return product A*v
51      */
52     Vector operator*(const Vector &v) const;
53
54     /**
55      * @brief 带原点平移
56      * @param cv
57      * A=A+cv*I
58      */
59     void addConstOnDiagonal(double cv);
```

```
60
61      /**
62       * @brief 生成题目中需要求解的默认矩阵
63       * @return 待求解矩阵
64       */
65      static SymBandMatrix genDefaultMatrix();
66
67  private:
68      int n, w;
69      //std::vector<std::vector<double>> data;
70      std::vector<double> data;
71  };
72
73  #endif //NUMERICALANALYSIST1_SYMBANDMATRIX_H
```

```
1  //
2  // Created by 40461 on 2021/10/30.
3  //
4
5  #ifndef NUMERICALANALYSIST1_EIGENVALUEUTILS_H
6  #define NUMERICALANALYSIST1_EIGENVALUEUTILS_H
7
8  #include "SymBandMatrix.h"
9  #include <algorithm>
10
11 /**
12  * @brief 特征值工具类
13  * 包含幂法，和反幂法，分别对2-范数和无穷范数进行了实现
14  */
15
16 class EigenValueUtils {
17 public:
18     static std::pair<double, Vector> powerMethodWithNorm2(const SymBandMatrix &matrix,
    double eps);
19
20     static std::pair<double, Vector> invPowerMethodWithNorm2(const SymBandMatrix &matrix,
    double eps);
21
22     static std::pair<double, Vector> powerMethodWithNormInf(const SymBandMatrix &matrix,
    double eps);
23
24     static std::pair<double, Vector> invPowerMethodWithNormInf(const SymBandMatrix &matrix
    , double eps);
25 };
26
27
28 #endif //NUMERICALANALYSIST1_EIGENVALUEUTILS_H
29
```

```cpp
//
// Created by 40461 on 2021/10/30.
//

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

using namespace std;

SymBandMatrix::SymBandMatrix(int _n, int _width) : n(_n), w(_width), data((_width + 1) * n) {}

SymBandMatrix SymBandMatrix::genDefaultMatrix() {
    constexpr int N = 501;
    constexpr int WIDTH = 2;
    SymBandMatrix matrixA(N, WIDTH);

    for (int i = 1; i <= N; i++) {
        matrixA.at(i, i) = (1.64 - 0.024 * i) * sin(0.2 * i) - 0.64 * exp(0.1 / i);
        if (i < N)
            matrixA.at(i, i + 1) = 0.16;
        if (i + 1 < N)
            matrixA.at(i, i + 2) = -0.064;
    }
    return matrixA;
}

Vector SymBandMatrix::operator*(const Vector &v) const {
    Vector r(v.length());
    for (int i = 1; i <= n; i++) {
        for (int j = max(1, i - w); j <= min(n, i + w); j++)
            r.at(i) += at(i, j) * v.at(j);
    }
    return r;
}

void SymBandMatrix::addConstOnDiagonal(double cv) {
    for (int i = 1; i <= n; i++)
        at(i, i) += cv;
}

```

```
1  //
2  // Created by 40461 on 2021/10/30.
3  //
4
5  #ifndef NUMERICALANALYSIST1_LINEAREQLUSOLVER_H
6  #define NUMERICALANALYSIST1_LINEAREQLUSOLVER_H
7
8  #include "Vector.h"
9  #include "SymBandMatrix.h"
10
11 using LeftMatrix = SymBandMatrix;
12 using UpMatrix = SymBandMatrix;
13
14 /**
15  * @brief 线性方程组求解类
16  */
17
18 class LinearEqLuSolver {
19 public:
20     /**
21      * @brief 构造函数
22      * 通过参数matrix分解出LU矩阵并存储
23      * @param matrix
24      */
25     explicit LinearEqLuSolver(const SymBandMatrix &matrix);
26
27     /**
28      * @brief 求解线性方程组
29      * 求解方式Ly=b,Ux=y
30      * @param 常数向量
31      * @return 返回Ax=b的解
32      */
33     Vector solveEq(const Vector &b);
34
35 private:
36     LeftMatrix l;
37     UpMatrix u;
38 public:
39     /**
40      * @brief 得到L
41      * @return 下三角矩阵L的常引用
42      */
43     const LeftMatrix &getL() const;
44     /**
45      * @brief 得到R
46      * @return 上三角矩阵U的常引用
47      */
48     const UpMatrix &getU() const;
49
50 private:
51     int n,s;
52 };
53
54
55 #endif //NUMERICALANALYSIST1_LINEAREQLUSOLVER_H
56
```

```cpp
//
// Created by 40461 on 2021/10/30.
//

#include "EigenValueUtils.h"
#include "LinearEqLuSolver.h"
#include <iostream>
#include "TimerUtil.h"

using namespace std;

pair<double, Vector> EigenValueUtils::powerMethodWithNorm2(const SymBandMatrix &matrix, const double eps) {
    TimerUtil::start(__func__);
    Vector u(matrix.size(), true), y;
    double beta = 0, oldBeta;
    int epochs = 0;
    do {
        oldBeta = beta;
        auto eta = u.norm2();
        y = u / eta;
        u = matrix * y;
        beta = y.dot(u);
        ++epochs;
#ifndef NDEBUG
        //cout << ++epochs << ":" << beta << endl;
#endif
    } while (eps <= abs((beta - oldBeta) / beta));
#ifndef NDEBUG
    cout << epochs << endl;
#endif
    TimerUtil::finish(__func__);
    return {beta, move(y)};
}

pair<double, Vector> EigenValueUtils::invPowerMethodWithNorm2(const SymBandMatrix & matrix, const double eps) {
    TimerUtil::start(__func__);
    Vector u(matrix.size(), true), y;
    LinearEqLuSolver solver(matrix);
    double beta = 0, oldBeta;
    int epochs = 0;
    do {
        oldBeta = beta;
        auto eta = u.norm2();
        y = u / eta;
        u = solver.solveEq(y);
        beta = y.dot(u);
        ++epochs;
#ifndef NDEBUG
        //cout << ++epochs << ":" << beta << endl;
#endif
    } while (eps <= abs((beta - oldBeta) / beta));
#ifndef NDEBUG
    cout << epochs << endl;
#endif
    TimerUtil::finish(__func__);
    return {1 / beta, move(y)};
}
```

```
 58
 59  pair<double, Vector> EigenValueUtils::powerMethodWithNormInf(const SymBandMatrix &
     matrix, const double eps) {
 60      TimerUtil::start(__func__);
 61      Vector u(matrix.size(), true), y;
 62      double beta = 0, oldBeta;
 63      bool prevHSign;
 64      int r = u.maxNormElement();
 65      int epochs = 0;
 66      do {
 67          oldBeta = beta;
 68          prevHSign = u.at(r) < 0;
 69          y = u / abs(u.at(r));
 70          u = matrix * y;
 71          r = u.maxNormElement();
 72          beta = prevHSign ? -u.at(r) : u.at(r);
 73          ++epochs;
 74      } while (eps <= abs((beta - oldBeta) / beta));
 75  #ifndef NDEBUG
 76      cout << epochs << endl;
 77  #endif
 78      TimerUtil::finish(__func__);
 79      return {beta, move(y)};
 80  }
 81
 82  pair<double, Vector> EigenValueUtils::invPowerMethodWithNormInf(const SymBandMatrix &
     matrix, const double eps) {
 83      TimerUtil::start(__func__);
 84      Vector u(matrix.size(), true), y;
 85      double beta = 0, oldBeta;
 86      bool prevHSign;
 87      int r = u.maxNormElement();
 88      int epochs = 0;
 89      LinearEqLuSolver solver(matrix);
 90      do {
 91          oldBeta = beta;
 92          prevHSign = u.at(r) < 0;
 93          y = u / abs(u.at(r));
 94          u = solver.solveEq(y);
 95          r = u.maxNormElement();
 96          beta = prevHSign ? -u.at(r) : u.at(r);
 97          ++epochs;
 98      } while (eps <= abs((beta - oldBeta) / beta));
 99  #ifndef NDEBUG
100      cout << epochs << endl;
101  #endif
102      TimerUtil::finish(__func__);
103      return {1 / beta, move(y)};
104  }
105
```

```cpp
//
// Created by 40461 on 2021/10/30.
//

#include "LinearEqLuSolver.h"
#include <algorithm>

using namespace std;

LinearEqLuSolver::LinearEqLuSolver(const SymBandMatrix &matrix) : l(matrix), u(matrix), n(matrix.size()),
                                                                  s(matrix.width()) {
    for (int i = 1; i <= n; i++)
        l.at(i, i) = 1;
    for (int k = 1; k <= n; k++) {
        for (int j = k; j <= min(k + s, n); j++)
            for (int t = max({1, k - s, j - s}); t <= k - 1; t++)
                u.at(k, j) -= l.at(k, t) * u.at(t, j);
        for (int i = k + 1; i <= min(k + s, n); i++) {
            for (int t = max({1, i - s, k - s}); t <= k - 1; t++)
                l.at(i, k) -= l.at(i, t) * u.at(t, k);
            l.at(i, k) /= u.at(k, k);
        }
    }
}

Vector LinearEqLuSolver::solveEq(const Vector &b) {
    Vector y(b);
    for (int i = 1; i <= n; i++)
        for (int t = max(1, i - s); t <= i - 1; t++)
            y.at(i) -= l.at(i, t) * y.at(t);
    Vector x(y);
    for (int i = n; i > 0; i--) {
        for (int t = i + 1; t <= min(i + s, n); t++)
            x.at(i) -= u.at(i, t) * x.at(t);
        x.at(i) /= u.at(i, i);
    }
    return x;
}

const LeftMatrix &LinearEqLuSolver::getL() const {
    return l;
}

const UpMatrix &LinearEqLuSolver::getU() const {
    return u;
}

```