```cpp
#include <iostream>
#include <cmath>
#include <iomanip>
#include <sstream>
#include <chrono>
#include "NonLinearEqUtil.h"
#include "InterpolationUtil.h"
#include "NonSqMatrix.h"
#include "Matrix.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 待输出的浮点数
 * @param precision 精度
 * @return std::string
 * 采用e型输出实型数来表示arg，显示precision（>0）位有效数字，返回表示的字符串
 */

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

    //关于x,y,t,u,v,w的非线性方程组
    vector<NonLinFormula> funcs = {
            {},
            {{0.5, COS},    {1,    LINEAR}, {1, LINEAR}, {1, LINEAR}, {-1, LINEAR},    {0,  CONSTANT}, {-2.67, CONSTANT}},
            {{1,    LINEAR}, {0.5, SIN},    {1, LINEAR}, {1, LINEAR}, {0,  CONSTANT}, {-1, LINEAR},    {-1.07, CONSTANT}},
            {{0.5, LINEAR}, {1,    LINEAR}, {1, COS},    {1, LINEAR}, {-1, LINEAR},    {0,  CONSTANT}, {-3.74, CONSTANT}},
            {{1,    LINEAR}, {0.5, LINEAR}, {1, LINEAR}, {1, SIN},    {0,  CONSTANT}, {-1, LINEAR},    {-0.79, CONSTANT}}
```

```
56      };
57
58      //t,u确定z，所产生的二维数表
59      Matrix zTable(6, {
60              {-0.5,  -0.34, 0.14,  0.94,  2.06,  3.5},
61              {-0.42, -0.5,  -0.26, 0.3,   1.18,  2.38},
62              {-0.18, -0.5,  -0.5,  -0.18, 0.46,  1.42},
63              {0.22,  -0.34, -0.58, -0.5,  -0.1,  0.62},
64              {0.78,  -0.02, -0.5,  -0.66, -0.5,  -0.02},
65              {1.5,   0.46,  -0.26, -0.66, -0.74, -0.5}
66      });
67
68      NonLinItemMatrix matA(4);
69      vector<NonLinFormula> fc{{}};
70      fc.reserve(5);
71      //计算关于t,u,v,w的非线性方程，包括系数非线性矩阵和常数项表达式
72      for (int i = 1; i <= 4; i++) {
73          for (int j = 1; j <= 4; j++)
74              matA.at(i, j) = funcs[i].at(j);
75          fc.emplace_back(initializer_list<NonLinItem>{funcs[i].at(5), funcs[i].at(6), funcs[i].at(7)});
76      }
77      int m = 10, n = 20;
78      NonSqMatrix matU(m + 1, n + 1);
79
80      /**
81       * @brief 计算二元函数z=f(x,y)，其中f(x,y)由 ①方程组 ②对zTable插值得到的关于t,u的二元函数共同确定
82       * @param x 参数x
83       * @param y 参数y
84       * @param z f(x,y)
85       */
86      auto f = [&](double x, double y) -> auto {
87          static Vector b(4), initialVec({1, 1, 1, 1});
88          for (int k = 1; k <= 4; k++)
89              b.at(k) = fc[k]({x, y, 0});
90          //使用牛顿迭代法来求出四元非线性方程组的解，从x,y得到确定的t,u
91          auto res = NonLinearEqUtil::solveByNewtonMethod(matA, -b, initialVec);
92
93          auto t = res.at(1), u = res.at(2);
94          auto pT = t > 0.8 ? 0.8 : (t < 0.2 ? 0.2 : round(t / 0.2) * 0.2);
95          auto pU = u > 1.6 ? 1.6 : (u < 0.4 ? 0.4 : round(u / 0.4) * 0.4);
96          auto iT = (int) round(pT / 0.2) + 1;
97          auto iU = (int) round(pU / 0.4) + 1;
98          //代入关于t,u的插值函数，得到f(x,y)
99          return InterpolationUtil::twoDimQuadLagrangeInterpolation(t, u, {pT - 0.2, pT, pT + 0.2},
100                                                                    {pU - 0.4, pU, pU + 0.4},
101                                                                    zTable.subMatrix(iT - 1, iU - 1, 3));
102     };
103
104     //求出数表(x_i,y_j,f(x_i,y_j)),并输出
105     for (int i = 0; i <= m; i++)
106         for (int j = 0; j <= n; j++) {
107             auto x = 0.08 * i, y = 0.5 + 0.05 * j;
108             matU.at(i + 1, j + 1) = f(x, y);
109             cout << toScientific(x, 2) << "\t" << toScientific(y, 3) << "\t"
110                  << toScientific(matU.at(i + 1, j + 1), 12) << endl;
111         }
112
113     int k = 0;
```

```
114     double delta;
115     ZeroRangeGuard guard(1E-7);
116     Matrix matC(0);
117     /**
118      * @brief 通过固定的系数矩阵C来求出函数值 z' = p(x,y) = \sum_{r=0}^k \sum_{s=0}^k c_{rs} x^r
        y^s
119      * @param x 参数x
120      * @param y 参数y
121      * @return z' p(x,y)
122      */
123     auto p = [&](double x, double y) -> auto {
124         double xi = 1, res = 0;
125         for (int i = 1; i <= matC.size(); i++) {
126             double yj = 1;
127             for (int j = 1; j <= matC.size(); j++) {
128                 res += matC.at(i, j) * xi * yj;
129                 yj *= y;
130             }
131             xi *= x;
132         }
133         return res;
134     };
135
136     //求最小的k值使得delta满足精度
137     do {
138         ZeroRangeGuard guard1(1E-12);
139         ++k;
140         NonSqMatrix matB(m + 1, k + 1), matG(n + 1, k + 1);
141         for (int i = 0; i <= m; i++) {
142             auto t = 0.08 * i;
143             matB.at(i + 1, 1) = 1;
144             for (int j = 2; j <= k + 1; j++)
145                 matB.at(i + 1, j) = matB.at(i + 1, j - 1) * t;
146         }
147         for (int j = 0; j <= n; j++) {
148             auto u = 0.5 + 0.05 * j;
149             matG.at(j + 1, 1) = 1;
150             for (int i = 2; i <= k + 1; i++)
151                 matG.at(j + 1, i) = matG.at(j + 1, i - 1) * u;
152         }
153
154         //求解方程 (B^T B)C(G^T G) = B^T U G 即可得到系数矩阵C
155         Matrix temp = LinearEqUtil::solveByGauss((matB.transpose() * matB).toMatrix(),
156                                                  (matB.transpose() * matU * matG).toMatrix().getColumnVectors());
157         matC = Matrix(LinearEqUtil::solveByGauss((matG.transpose() * matG).transpose().toMatrix(),
158                                                  temp.transpose().getColumnVectors())).transpose();
159
160         delta = 0;
161         //计算误差值delta
162         for (int i = 0; i <= m; i++)
163             for (int j = 0; j <= n; j++) {
164                 double d = p(0.08 * i, 0.5 + 0.05 * j) - matU.at(i + 1, j + 1);
165                 delta += d * d;
166             }
167         cout << "k = " << k << " delta = " << toScientific(delta, 12) << endl;
168     } while (!ZeroRangeGuard::isZero(delta));
169     cout << "Final acceptable k = " << k << " delta (<=1E-7): " << toScientific(delta, 12) <<
    endl;
170     for (int i = 1; i <= k; i++) {
```

```
171         for (int j = 1; j <= k; j++)
172             cout << toScientific(matC.at(i, j), 12) << "\t";
173         cout << endl;
174     }
175
176     //打印数表 (x_i^*, y_i^*, f(x_i^*, y_i^*), p(x_i^*, y_i^*))
177     for (int i = 1; i <= 8; i++)
178         for (int j = 1; j <= 5; j++) {
179             double x = 0.1 * i, y = 0.5 + 0.2 * j;
180             cout << toScientific(x, 2) << "\t" << toScientific(y, 3) << "\t" << toScientific(f(x, y), 12
    ) << "\t"
181                  << toScientific(p(x, y), 12) << endl;
182         }
183
184     auto end = system_clock::now();
185     auto duration = duration_cast<microseconds>(end - start);
186 #ifdef NDEBUG
187     cout.clear();
188 #endif
189     //输出程序运行时间
190     cout << "time: " << duration.count() << " microseconds" << endl;
191     return 0;
192 }
193
```

```
1  //
2  // Created by 40461 on 2021/11/27.
3  //
4
5  #ifndef NUMERICALANALYSIST8_MATRIX_H
6  #define NUMERICALANALYSIST8_MATRIX_H
7
8
9  #include <vector>
10 #include <cassert>
11 #include "Vector.h"
12
13 /**
14  * @brief 矩阵类
15  * 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
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 Matrix {
20 public:
21     explicit Matrix(int _n);
22
23     Matrix(int _n, std::initializer_list<std::initializer_list<double>> list);
24
25     Matrix(const std::vector<Vector> &columnVectors);
26
27     inline double &at(int i, int j) {
28         assert(i > 0 && i <= n && j > 0 && j <= n);
29         return data[(i - 1) * n + j - 1];
30     }
31
32     inline const double &at(int i, int j) const {
33         return const_cast<Matrix *>(this)->at(i, j);
34     }
35
36     inline int size() const {
37         return n;
38     }
39
40     Matrix transpose() const;
41
42     Vector operator*(const Vector &v) const;
43
44     Matrix operator+(const Matrix &m) const;
45
46     Matrix operator-(const Matrix &m) const;
47
48     Matrix operator*(const Matrix &m) const;
49
50     Matrix operator*(double d) const;
51
52     friend Matrix operator*(double d, const Matrix &m);
53
54     Matrix operator+(double d) const;
55
56     Matrix operator-(double d) const;
57
58     Matrix indexMultiply(const Matrix &m) const;
59
```

```
60      Matrix subMatrix(int i, int j, int m) const;
61
62      double sum() const;
63
64      std::vector<Vector> getColumnVectors() const;
65
66      void print() const;
67
68  private:
69      friend class NonSqMatrix;
70
71      int n;
72      std::vector<double> data;
73  };
74
75  #endif //NUMERICALANALYSIST8_MATRIX_H
76
```

```
1  //
2  // Created by 40461 on 2021/11/27.
3  //
4
5  #ifndef NUMERICALANALYSIST8_VECTOR_H
6  #define NUMERICALANALYSIST8_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     Vector(std::initializer_list<double> list);
24
25     inline double &at(int i) {
26         assert(i > 0 && i <= data.size());
27         return data[i - 1];
28     }
29
30     inline const double &at(int i) const {
31         return const_cast<Vector *>(this)->at(i);
32     }
33
34     inline int length() const {
35         return (int) data.size();
36     }
37
38     Vector operator/(double x) const;
39
40     Vector operator-(const Vector &v) const;
41
42     /**
43      * @brief 向量点乘
44      * @param v
45      * @return self^T * v
46      */
47     double dot(const Vector &v) const;
48
49     /**
50      * @brief 向量外积
51      * @param v
52      * @return self * v^T
53      */
54     Matrix outer(const Vector &v) const;
55
56     Vector operator*(double x) const;
57
58     friend Vector operator*(double x, const Vector &v);
59
```

```
60      void print() const;
61
62      double normInf() const;
63
64      Vector operator -() const;
65
66      Vector &operator+=(const Vector &v);
67
68  private:
69      std::vector<double> data;
70  };
71
72
73  #endif //NUMERICALANALYSIST8_VECTOR_H
74
```

```cpp
//
// Created by 40461 on 2021/11/27.
//

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

using namespace std;

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

Matrix::Matrix(int _n, std::initializer_list <std::initializer_list<double>> list) : n(_n), data(n * n) {
    int i = 0;
    for (auto &row: list) {
        ++i;
        int j = 0;
        for (auto &col: row) {
            ++j;
            at(i, j) = col;
        }
    }
}

Matrix::Matrix(const vector <Vector> &columnVectors) : n((int) columnVectors.size()), data(n * n
) {
    for (int j = 1; j <= n; j++) {
        assert(columnVectors[j - 1].length() == n);
        for (int i = 1; i <= n; i++)
            at(i, j) = columnVectors[j - 1].at(i);
    }
}


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;
}

```

```
 59  Matrix Matrix::operator*(const Matrix &m) const {
 60      assert(n == m.n);
 61      Matrix mat(n);
 62      for (int i = 1; i <= n; i++)
 63          for (int j = 1; j <= n; j++)
 64              for (int k = 1; k <= n; k++)
 65                  mat.at(i, j) += at(i, k) * m.at(k, j);
 66      return mat;
 67  }
 68
 69  Matrix Matrix::operator*(double d) const {
 70      Matrix mat(n);
 71      for (int i = 0; i < data.size(); i++)
 72          mat.data[i] = data[i] * d;
 73      return mat;
 74  }
 75
 76  Matrix operator*(double d, const Matrix &m) {
 77      return m * d;
 78  }
 79
 80  void Matrix::print() const {
 81      cout << fixed << setprecision(3);
 82      for (int i = 1; i <= n; i++) {
 83          for (int j = 1; j <= n; j++)
 84              cout << at(i, j) << "\t";
 85          cout << endl;
 86      }
 87      cout << endl;
 88  }
 89
 90  Matrix Matrix::operator+(const Matrix &m) const {
 91      assert(n == m.n);
 92      Matrix mat(n);
 93      for (int i = 0; i < data.size(); i++)
 94          mat.data[i] = data[i] + m.data[i];
 95      return mat;
 96  }
 97
 98  Matrix Matrix::operator+(double d) const {
 99      Matrix mat(*this);
100      for (int i = 1; i <= n; i++)
101          mat.at(i, i) += d;
102      return mat;
103  }
104
105  Matrix Matrix::operator-(double d) const {
106      return *this + (-d);
107  }
108
109  Matrix Matrix::indexMultiply(const Matrix &m) const {
110      assert(n == m.n);
111      Matrix mat(n);
112      for (int i = 1; i <= n; i++)
113          for (int j = 1; j <= n; j++)
114              mat.at(i, j) = at(i, j) * m.at(i, j);
115      return mat;
116  }
117
```

```
118  Matrix Matrix::subMatrix(int i, int j, int m) const {
119      assert(i > 0 && i + m - 1 <= n && j > 0 && j + m - 1 <= n);
120      Matrix mat(m);
121      for (int k = 1; k <= m; k++)
122          for (int l = 1; l <= m; l++)
123              mat.at(k, l) = at(i + k - 1, j + l - 1);
124      return mat;
125  }
126
127  double Matrix::sum() const {
128      double sum = 0;
129      for (double i: data)
130          sum += i;
131      return sum;
132  }
133
134  std::vector <Vector> Matrix::getColumnVectors() const {
135      std::vector <Vector> vec;
136      vec.reserve(n);
137      for (int j = 1; j <= n; j++) {
138          vec.emplace_back(n);
139          auto &t = vec.back();
140          for (int i = 1; i <= n; i++)
141              t.at(i) = at(i, j);
142      }
143      return vec;
144  }
145
```

```cpp
//
// Created by 40461 on 2021/11/27.
//

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

using namespace std;

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

Vector::Vector(std::initializer_list<double> list) : data(list) {}

Vector Vector::operator/(double x) const {
    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);
```

```
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  Vector Vector::operator-() const {
73      Vector v(length());
74      for (int i = 0; i < data.size(); ++i)
75          v.data[i] = -data[i];
76      return v;
77  }
78
79  Vector &Vector::operator+=(const Vector &v) {
80      assert(length() == v.length());
81      for (int i = 0; i < data.size(); ++i)
82          data[i] += v.data[i];
83      return *this;
84  }
85
86
```

```
1  //
2  // Created by 40461 on 2021/11/28.
3  //
4
5  #ifndef NUMERICALANALYSIST8_NONSQMATRIX_H
6  #define NUMERICALANALYSIST8_NONSQMATRIX_H
7
8  #include <vector>
9  #include <cassert>
10
11 class Matrix;
12
13 /**
14  * @brief 非方阵类
15  * 实现了非方阵的矩阵乘法，矩阵的转置等运算
16  */
17
18 class NonSqMatrix {
19 public:
20
21     explicit NonSqMatrix(int _m, int _n);
22
23     inline double &at(int i, int j) {
24         assert(i>0 && i<=m && j>0 && j<=n);
25         return data[(i - 1) * n + j - 1];
26     }
27
28     inline const double &at(int i, int j) const {
29         return const_cast<NonSqMatrix *>(this)->at(i, j);
30     }
31
32     NonSqMatrix operator*(const NonSqMatrix &other) const;
33
34     /**
35      * @brief 当n=m时，可以将矩阵转换为方阵
36      * @return 方阵
37      */
38     Matrix toMatrix() const &;
39
40     Matrix toMatrix() &&;
41
42     NonSqMatrix transpose() const;
43
44 private:
45     friend class Matrix;
46
47     int m, n;
48     std::vector<double> data;
49 };
50
51
52 #endif //NUMERICALANALYSIST8_NONSQMATRIX_H
53
```

```
cmake_minimum_required(VERSION 3.16)
project(NumericalAnalysisT8)

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(NumericalAnalysisT8 main.cpp NonLinFormula.cpp NonLinFormula.h
NonLinItemMatrix.cpp NonLinItemMatrix.h NonLinearEqUtil.cpp NonLinearEqUtil.h Vector.
cpp Vector.h Matrix.cpp Matrix.h LinearEqUtil.cpp LinearEqUtil.h ZeroRangeGuard.h
InterpolationUtil.cpp InterpolationUtil.h NonSqMatrix.cpp NonSqMatrix.h)

```

```
1  //
2  // Created by 40461 on 2021/11/27.
3  //
4
5  #ifndef NUMERICALANALYSIST8_LINEAREQUTIL_H
6  #define NUMERICALANALYSIST8_LINEAREQUTIL_H
7
8  #include "Vector.h"
9  #include <vector>
10
11 /**
12  * @brief 求解线性方程组的解
13  * 实现了列主元高斯消去法
14  */
15
16 class LinearEqUtil {
17 public:
18     /**
19      * @brief 列主元高斯消去法，求出每一个A*x=b_i的解向量x_i，其中A为n*n的矩阵，
    b为n的向量组成的集合
20      * @param matA n*n的矩阵
21      * @param b 长度为n的向量组成的集合
22      * @return {x_i} \forall b_i
23      */
24     static std::vector<Vector> solveByGauss(Matrix &matA, std::vector<Vector> &b);
25
26     //右值重载
27     static std::vector<Vector> solveByGauss(Matrix &&matA, std::vector<Vector> &&b) {
28         return solveByGauss(matA, b);
29     }
30 };
31
32 #endif //NUMERICALANALYSIST8_LINEAREQUTIL_H
33
```

```
1  //
2  // Created by 40461 on 2021/11/26.
3  //
4
5  #ifndef NUMERICALANALYSIST8_NONLINFORMULA_H
6  #define NUMERICALANALYSIST8_NONLINFORMULA_H
7
8  #include <vector>
9  #include <cassert>
10
11 enum BasicType {
12     CONSTANT,
13     LINEAR,
14     SIN,
15     COS,
16     NONE
17 };
18
19 /**
20  * @brief NonLinItem
21  * 非线性项，包含一个基本类型和一个系数
22  * 支持线性项，常数项，sin，cos
23  * 允许括号调用
24  */
25
26 struct NonLinItem {
27     double coef;
28     BasicType type;
29
30     double operator()(double v) const;
31
32     NonLinItem derivative() const;
33 };
34
35 /**
36  * @brief 非线性表达式
37  * 带有非线性项的函数
38  */
39
40 class NonLinFormula {
41 public:
42
43     NonLinFormula(std::initializer_list<NonLinItem> list);
44
45     double operator()(std::initializer_list<double> values) const;
46
47     inline NonLinItem &at(int i) {
48         assert(i > 0 && i <= data.size());
49         return data[i - 1];
50     }
51
52     inline const NonLinItem &at(int i) const {
53         return const_cast<NonLinFormula *>(this)->at(i);
54     }
55
56 private:
57     std::vector<NonLinItem> data;
58 };
59
```

```
60
61  #endif //NUMERICALANALYSIST8_NONLINFORMULA_H
62
```

```cpp
//
// Created by 40461 on 2021/11/28.
//

#include "NonSqMatrix.h"
#include <cassert>
#include "Matrix.h"

NonSqMatrix::NonSqMatrix(int _m, int _n) : m(_m), n(_n), data(m * n) {}

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

Matrix NonSqMatrix::toMatrix() const &{
    assert(m == n);
    Matrix result(m);
    result.data = data;
    return result;
}

Matrix NonSqMatrix::toMatrix() &&{
    assert(m == n);
    Matrix result(0);
    result.data = std::move(data);
    result.n = m;
    return result;
}

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

```cpp
//
// Created by 40461 on 2021/11/27.
//

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

using namespace std;

vector <Vector> LinearEqUtil::solveByGauss(Matrix &matA, vector <Vector> &b) {
    int n = matA.size();
    for (int k = 1; k < n; k++) {
        int i = k;
        //选择第k列中第k行以下元素最大的行
        for (int j = k + 1; j < n; j++)
            if (abs(matA.at(j, k)) > abs(matA.at(i, k)))
                i = j;
        //交换第k行和第i行
        if (i != k) {
            for (int j = k; j <= n; j++)
                swap(matA.at(k, j), matA.at(i, j));
            for (auto &bt: b)
                swap(bt.at(k), bt.at(i));
        }
        //消元
        for (i = k + 1; i <= n; i++) {
            double m = matA.at(i, k) / matA.at(k, k);
            for (int j = k; j <= n; j++)
                matA.at(i, j) -= m * matA.at(k, j);
            for (auto &bt: b)
                bt.at(i) -= m * bt.at(k);
        }
    }
    vector <Vector> res;
    res.reserve(b.size());
    //求解
    for (auto & bt : b) {
        res.emplace_back(bt.length());
        auto &x = res.back();
        for (int k = n; k > 0; k--) {
            double s = 0;
            for (int j = k + 1; j <= n; j++)
                s += matA.at(k, j) * x.at(j);
            x.at(k) = (bt.at(k) - s) / matA.at(k, k);
        }
    }
    return res;
}

```

```h
//
// Created by 40461 on 2021/11/27.
//

#ifndef NUMERICALANALYSIST8_ZERORANGEGUARD_H
#define NUMERICALANALYSIST8_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-12};
};

#endif //NUMERICALANALYSIST8_ZERORANGEGUARD_H

```

```
1  //
2  // Created by 40461 on 2021/11/27.
3  //
4
5  #ifndef NUMERICALANALYSIST8_NONLINEAREQUTIL_H
6  #define NUMERICALANALYSIST8_NONLINEAREQUTIL_H
7
8  #include "NonLinItemMatrix.h"
9  #include "Vector.h"
10
11 class NonLinearEqUtil {
12 public:
13     /**
14      * @brief 用牛顿迭代法求解非线性方程组
15      * @param mat 非线性矩阵
16      * @param c 常量向量
17      * @param initX 迭代初始值
18      * @return 返回mat*x=c的解
19      */
20     static Vector solveByNewtonMethod(const NonLinItemMatrix &mat, const Vector &c, const
    Vector &initX);
21 };
22
23
24 #endif //NUMERICALANALYSIST8_NONLINEAREQUTIL_H
25
```

```cpp
//
// Created by 40461 on 2021/11/26.
//

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

using namespace std;

NonLinFormula::NonLinFormula(initializer_list<NonLinItem> list) : data(list) {}

double NonLinFormula::operator()(initializer_list<double> values) const {
    assert(values.size() == data.size());
    double res = 0;
    int i = 0;
    for (auto &p: data)
        res += p(values.begin()[i++]);
    return res;
}

double NonLinItem::operator()(double v) const {
    switch (type) {
        case CONSTANT:
            return coef;
        case LINEAR:
            return coef * v;
        case SIN:
            return coef * sin(v);
        case COS:
            return coef * cos(v);
        default:;
    }
    return 0;
}

NonLinItem NonLinItem::derivative() const {
    switch (type) {
        case CONSTANT:
            return {0, CONSTANT};
        case LINEAR:
            return {coef, CONSTANT};
        case SIN:
            return {coef, COS};
        case COS:
            return {-coef, SIN};
        default:;
    }
    return {0, NONE};
}

```

```
1  //
2  // Created by 40461 on 2021/11/26.
3  //
4
5  #ifndef NUMERICALANALYSIST8_NONLINITEMMATRIX_H
6  #define NUMERICALANALYSIST8_NONLINITEMMATRIX_H
7
8
9  #include "NonLinFormula.h"
10 #include "Vector.h"
11 #include <vector>
12
13 /**
14  * @brief 非线性项矩阵
15  * 其中每一个元素是一个非线性项
16  */
17
18 class NonLinItemMatrix {
19 public:
20     explicit NonLinItemMatrix(int n);
21
22     inline NonLinItem &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 NonLinItem &at(int i, int j) const {
28         return const_cast<NonLinItemMatrix *>(this)->at(i, j);
29     }
30
31     inline int size() const {
32         return n;
33     }
34
35     /**
36      * @brief 计算非线性项矩阵的值
37      * @param v 每一行未知数的值所组成的向量
38      * @return 每一行作为一个非线性表达式，求出的值组成的向量
39      */
40     Vector operator*(const Vector &v) const;
41
42     /**
43      * @brief 计算每一项的函数求导组成的新矩阵
44      * @return 求导之后的新矩阵
45      */
46     NonLinItemMatrix derivative() const;
47
48 private:
49     int n;
50     std::vector<NonLinItem> data;
51 };
52
53
54 #endif //NUMERICALANALYSIST8_NONLINITEMMATRIX_H
55
```

```
1  //
2  // Created by 40461 on 2021/11/28.
3  //
4
5  #ifndef NUMERICALANALYSIST8_INTERPOLATIONUTIL_H
6  #define NUMERICALANALYSIST8_INTERPOLATIONUTIL_H
7
8  #include <algorithm>
9  #include "Vector.h"
10
11 /**
12  * @brief 插值工具类
13  * 实现了二维的分片拉格朗日插值法
14  */
15
16 class InterpolationUtil {
17 public:
18     /**
19      * @brief 分片拉格朗日插值法，求出给定点的值
20      * @param x 给定点的x坐标
21      * @param y 给定点的y坐标
22      * @param p 插值点的x坐标列表（length = 3）
23      * @param q 插值点的y坐标列表（length = 3）
24      * @param m 插值点的原函数值
25      * @return 给定点的插值拟合值
26      */
27     static double twoDimQuadLagrangeInterpolation(double x, double y, const Vector &p, const
    Vector &q, const Matrix &m);
28 private:
29     /**
30      * @brief 求一维拉格朗日插值函数l_k对应的值l_k(x)
31      * @param x 给定的坐标
32      * @param p 插值点坐标
33      * @return 返回{l_k(x) \forall k}组成的向量
34      */
35     static Vector valueOfLagrangeFunc(double x, const Vector &p);
36 };
37
38
39 #endif //NUMERICALANALYSIST8_INTERPOLATIONUTIL_H
40
```

```cpp
//
// Created by 40461 on 2021/11/27.
//

#include "NonLinearEqUtil.h"
#include "Matrix.h"
#include "LinearEqUtil.h"
#include "ZeroRangeGuard.h"
#include <iostream>

using namespace std;

Vector NonLinearEqUtil::solveByNewtonMethod(const NonLinItemMatrix &mat, const Vector &c, const Vector &initX) {
    auto n = mat.size();
    Vector x= initX;
    auto dMat = mat.derivative();
    // F(x)函数
    auto f = [&](const Vector &x) -> auto {
        return mat * x - c;
    };

    // F'(x)函数矩阵，矩阵中每一项都是一个函数，F'(x)_{ij}为\partial F_i(x_j) /\partial x_j
    auto fd = [&](const Vector &x) -> auto {
        Matrix r(n);
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                r.at(i, j) = dMat.at(i, j)(x.at(j));
        return r;
    };

    Vector deltaX(n);
    // 迭代主循环
    int k = 0;
    do {
        x += deltaX;
        deltaX = LinearEqUtil::solveByGauss(fd(x), {-f(x)}).front();
        ++k;
        assert(("Too many iterations Newton Method!", k < 1000));
    } while (!ZeroRangeGuard::isZero(deltaX.normInf() / x.normInf()));
    //cout << "Newton Method: " << k << " iterations" << endl;
    return x;
}

```

```cpp
//
// Created by 40461 on 2021/11/26.
//

#include <cassert>
#include "NonLinItemMatrix.h"

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

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

NonLinItemMatrix NonLinItemMatrix::derivative() const {
    NonLinItemMatrix res(n);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            res.at(i, j) = at(i, j).derivative();
    return res;
}

```

```cpp
//
// Created by 40461 on 2021/11/28.
//

#include "InterpolationUtil.h"
#include "Matrix.h"
using namespace std;

Vector InterpolationUtil::valueOfLagrangeFunc(double x, const Vector &p) {
    assert(p.length() == 3);
    return {(x - p.at(2)) * (x - p.at(3)) / ((p.at(1) - p.at(2)) * (p.at(1) - p.at(3))),
            (x - p.at(1)) * (x - p.at(3)) / ((p.at(2) - p.at(1)) * (p.at(2) - p.at(3))),
            (x - p.at(1)) * (x - p.at(2)) / ((p.at(3) - p.at(1)) * (p.at(3) - p.at(2)))};
}

double InterpolationUtil::twoDimQuadLagrangeInterpolation(double x, double y, const Vector &p, const Vector &q,
                                                          const Matrix &m) {
    assert(p.length() == q.length() && p.length() == m.size());
    // \sum_{k=1}^3 \sum_{r=1}^3 l_k(x) l'_r(y) f_{x_k,y_r}
    return valueOfLagrangeFunc(x, p).outer(valueOfLagrangeFunc(y, q)).indexMultiply(m).sum();
}

```