<u>**Name: Ahmed Nouralla**</u>

<u>**Group: BS19 – 02**</u>

Solving the Least Square Approximation problem, graphing the result using gnuplot.

Input format:

- m : the length m of data set
- t_i b_i : m lines with experimental data
- n : The degree of the polynomial

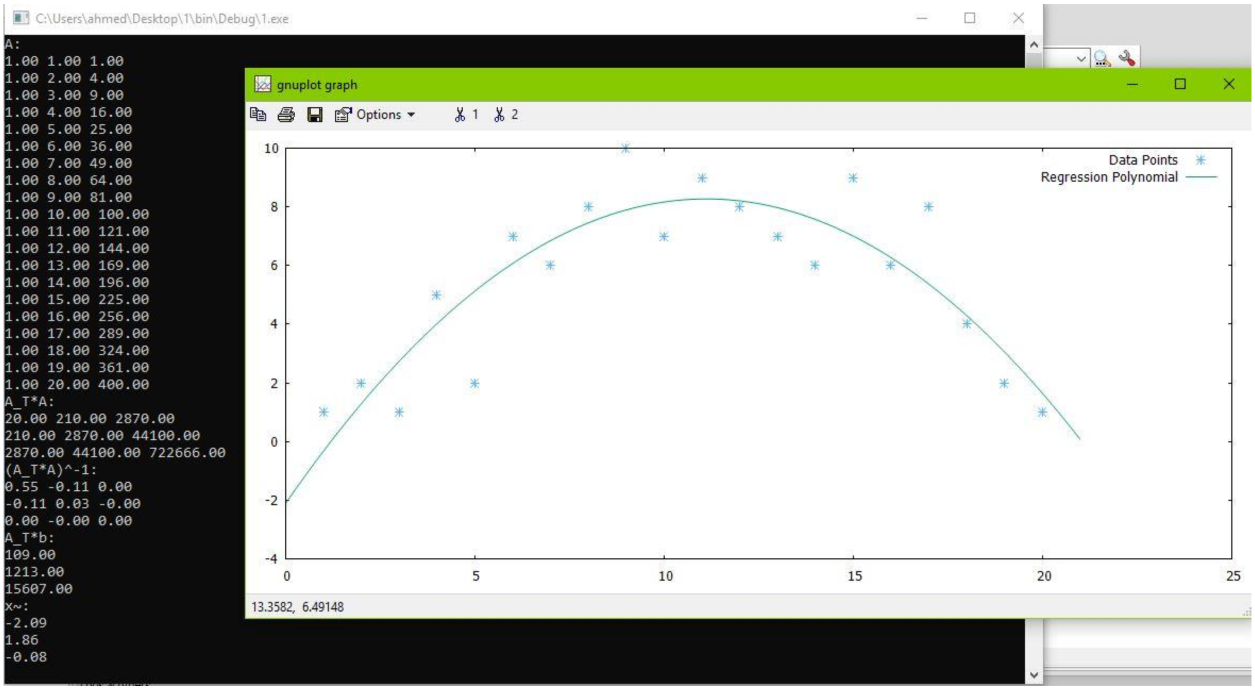
Output format:

- The matrix A itself after the line "A:"
- The matrix A_T*A: after the line "A_T*A:"
- The matrix (A_T*A)^-1 after the line "(A_T*A)^-1:"
- The matrix A_T*b after the line "A_T*b:"
- The answer itself after the line "x~:"

The program will calculate the equation and plot the points and the resulting polynomial x using gnuplot.

Example: Input/Output:

```
20
1 1
2 2
3 1
4 5
5 2
6 7
7 6
8 8
9 10
10 7
11 9
12 8
13 7
14 6
15 9
16 6
17 8
18 4
19 2
20 1
2
```



```
A:
1.00 1.00 1.00
1.00 2.00 4.00
1.00 3.00 9.00
1.00 4.00 16.00
1.00 5.00 25.00
1.00 6.00 36.00
1.00 7.00 49.00
1.00 8.00 64.00
1.00 9.00 81.00
1.00 10.00 100.00
1.00 11.00 121.00
1.00 12.00 144.00
1.00 13.00 169.00
1.00 14.00 196.00
1.00 15.00 225.00
1.00 16.00 256.00
1.00 17.00 289.00
1.00 18.00 324.00
1.00 19.00 361.00
1.00 20.00 400.00
A_T*A:
20.00 210.00 2870.00
210.00 2870.00 44100.00
2870.00 44100.00 722666.00
(A_T*A)^-1:
0.55 -0.11 0.00
-0.11 0.03 -0.00
0.00 -0.00 0.00
A_T*b:
109.00
1213.00
15607.00
x~:
-2.09
1.86
-0.08
```

Source code in the next pages.

```
/**
    Author: Ahmed Nouralla - BS19-02 - a.shaaban@innopolis university

    This code is tested on
        - A windows machine
        - With gnuplot installed in the directory C:\gnuplot
        - With GNU GCC Compiler following the 1999 ISO C language standard [-std=c99].
    And is not guaranteed to work on other machines having different properties.

*/

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

/// Plots n points given in array x[n], y[n].
/// In the same window, it plots the polynomial with coefficients given in v. Drawing range is [l, r] with s steps.

void plot(int n, double x[], double y[], vector<double> v, double l, double r, double s)
{
    FILE* pipe = _popen("C:\\gnuplot\\bin\\gnuplot -persist", "w");

    if(pipe != NULL) {

        /// The main 2 commands for the gnuplot, first one to plot the points, second one to draw the polynomial
        fprintf(pipe, "%s\n", "plot '-' w p ls 3 title 'Data Points', '-' title 'Regression Polynomial' with lines");

        for(int i = 0; i < n; i++){
            fprintf(pipe, "%f\t%f\n", x[i], y[i]);
        }

        fprintf(pipe, "%s\n", "e");

        for(double x = l; x <= r; x += s){

            double y = 0;
            for(int i = 0; i < v.size(); i++)
                y += v[i] * pow(x, i);

            fprintf(pipe, "%f\t%f\n", x, y);
        }

        fprintf(pipe, "%s\n", "e");
        fflush(pipe);
        _pclose(pipe);
    }

    else
        cout<<"Error\n";
}

class Matrix {
public:

    int n, m;
    map< pair<int,int>, double > x;

    Matrix(int r, int c)
    {
        this -> n = r;
        this -> m = c;
    }
```

```cpp

    Matrix operator * (Matrix t)
    {
        Matrix r(n, t.m);
        for(int k = 0; k < n; k++)
        {
            for(int i = 0; i < t.m; i++)
            {
                double sum = 0.0;
                for(int j = 0; j < m; j++)
                {
                    sum += x[{k,j}] * t.x[{j,i}];
                }
                r.x[{k,i}] = sum;
            }
        }
        return r;
    }

    Matrix Trn()
    {
        Matrix r(m, n);
        for(int i = 0; i < m; i++)
        {
            for(int j = 0; j < n; j++)
            {
                r.x[{i, j}] = x[{j, i}];
            }
        }
        return r;
    }

    Matrix Inv()
    {
        Matrix id(n, n);
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                if(i == j) id.x[{i, j}] = 1;
                else id.x[{i, j}] = 0;
            }
        }

        for(int t = 0; t < n - 1; t++)
        {
            /// --------------  Pivoting ---------------
            int l = t;
            double mx = x[{t, t}];
            for(int i = t + 1; i < n; i++)
            {
                if(abs(x[{i, t}]) > mx)
                {
                    mx = abs(x[{i, t}]);
                    l = i;
                }
            }
            if(l != t)
            {
                for(int j = 0; j < n; j++)
                {
                    swap(x[{t, j}], x[{l, j}]);
                    swap(id.x[{t, j}], id.x[{l, j}]);
                }
            }

            /// --------- Forward Elimination ----------
            for(int i = t + 1; i < n; i++)
            {
                double T = -x[{i, t}] / x[{t, t}];
                for(int j = 0; j < n; j++)
                {
                    x[{i ,j}] += T * x[{t, j}];
                    id.x[{i, j}] += T * id.x[{t, j}];
                }
            }
        }

        /// -------------- Way Back ---------------
        for(int t = n - 1; t >= 0; t--)
        {
            for(int i = t - 1; i >= 0; i--)
            {
                double T = -x[{i, t}] / x[{t, t}];
                for(int j = 0; j < n; j++)
                {
                    x[{i, j}] += T * x[{t, j}];
                    id.x[{i, j}] += T * id.x[{t, j}];
                }
            }
        }
        /// ------- Diagonal Normalization --------
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < n; j++)
            {
                id.x[{i, j}] /= x[{i, i}];
            }
            x[{i, i}] = 1.0;
        }

        return id;
    }

    friend ostream &operator << (ostream &output, Matrix& t)
    {
        int r;
        for(int i = 0; i < t.n; i++)
        {
            for(int j = 0; j < t.m - 1; j++)
            {
                r = t.x[{i, j}] * 1000;
                if(r % 10 == 5) r++; // To round up on 5.
                output << fixed << setprecision(2) << r/1000.0 << ' ' ;
            }
            r = t.x[{i, t.m-1}] * 1000;
            if(r % 10 == 5) r++;
            output << fixed << setprecision(2) << r/1000.0 << '\n' ;
        }
        return output;
    }
};

double pow(int b, int p) {
    ll r = 1;
    for(int i = 1; i <= p; i++) r *= b;
    return r;
}

int main()
{
    /// Uncomment to provide input from i.txt in the same directory as the project.
    // freopen("i.txt","r",stdin);
    int n, m;
    cin >> m;
    double t[m], b[m], mn = DBL_MAX, mx = -DBL_MAX;

    for(int i = 0; i < m; i++){
        cin >> t[i] >> b[i];
        mn = min(mn, t[i]);
        mx = max(mx, t[i]);
    }

    cin >> n;
    Matrix A(m, n+1);

    for(int i = 0; i < A.n; i++){
        for(int j = 0; j < A.m; j++){
            A.x[{i, j}] = pow(t[i], j);
        }
    }

    Matrix B(m, 1);
    for(int i = 0;i < m; i++){
        B.x[{i, 0}] = b[i];
    }

    Matrix R1 = A.Trn() * A ;
    Matrix R2 = A.Trn() * B;

    cout << "A:\n" << A << "A_T*A:\n" << R1;

    R1 = R1.Inv();
    Matrix R3 = R1 * R2;

    cout << "(A_T*A)^-1:\n" << R1 << "A_T*b:\n"  << R2 << "x~:\n" << R3;

    vector<double>v;
    for(int i = 0; i < R3.n; i++){
        v.push_back(R3.x[{i, 0}]);
    }

    plot(m, t, b, v, mn-1, mx+1, 0.01); /// Plotting the points and the regression polynomial.
}
```