Linear optimization (Linear Programming) simplex

NO ESCRITO POR MI.

Este programa resuelve problemas de optimización lineal de la forma:

$$\text{Maximiza } c^T \cdot x$$

$$\text{Sujeto a } Ax \leq b$$

$$x \geq 0$$

```cpp
// This program was written by jaehyunp and taken from:
// https://github.com/jaehyunp/stanfordacm/blob/master/code/Simplex.cc

// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).

#include <cmath>
#include <iomanip>
#include <iostream>
#include <limits>
#include <vector>

using namespace std;

using DOUBLE = long double;
using Row = vector<DOUBLE>;
using Matrix = vector<Row>;
using VI = vector<int>;

const DOUBLE EPS = 1e-9;
```

```
struct LPSolver
{
    int m, n;
    VI B, N;
    Matrix D;

    LPSolver(const Matrix& A, const Row& b, const Row& c)
        : m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, Row(n + 2))
    {
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                D[i][j] = A[i][j];
        for (int i = 0; i < m; i++)
        {
            B[i] = n + i;
            D[i][n] = -1;
            D[i][n + 1] = b[i];
        }
        for (int j = 0; j < n; j++)
        {
            N[j] = j;
            D[m][j] = -c[j];
        }
        N[n] = -1;
        D[m + 1][n] = 1;
    }

    void Pivot(int r, int s)
    {
        double inv = 1.0 / D[r][s];
        for (int i = 0; i < m + 2; i++)
            if (i != r)
                for (int j = 0; j < n + 2; j++)
                    if (j != s)
                        D[i][j] -= D[r][j] * D[i][s] * inv;
        for (int j = 0; j < n + 2; j++)
            if (j != s)
                D[r][j] *= inv;
        for (int i = 0; i < m + 2; i++)
            if (i != r)
                D[i][s] *= -inv;
        D[r][s] = inv;
        swap(B[r], N[s]);
    }
```

```
bool Simplex(int phase)
{
    int x = phase == 1 ? m + 1 : m;
    while (true)
    {
        int s = -1;
        for (int j = 0; j <= n; j++)
        {
            if (phase == 2 && N[j] == -1)
                continue;
            if (s == -1 || D[x][j] < D[x][s] ||
                (D[x][j] == D[x][s] && N[j] < N[s]))
                s = j;
        }
        if (D[x][s] > -EPS)
            return true;
        int r = -1;
        for (int i = 0; i < m; i++)
        {
            if (D[i][s] < EPS)
                continue;
            if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
                ((D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) &&
                 B[i] < B[r]))
                r = i;
        }
        if (r == -1)
            return false;
        Pivot(r, s);
    }
}

DOUBLE Solve(Row& x)
{
    int r = 0;
    for (int i = 1; i < m; i++)
        if (D[i][n + 1] < D[r][n + 1])
            r = i;
    if (D[r][n + 1] < -EPS)
    {
        Pivot(r, n);
        if (!Simplex(1) || D[m + 1][n + 1] < -EPS)
            return -numeric_limits<DOUBLE>::infinity();
        for (int i = 0; i < m; i++)
            if (B[i] == -1)
```

```
                {
                    int s = -1;
                    for (int j = 0; j <= n; j++)
                        if (s == -1 || D[i][j] < D[i][s] ||
                            (D[i][j] == D[i][s] && N[j] < N[s]))
                            s = j;
                    Pivot(i, s);
                }
        }
        if (!Simplex(2))
            return numeric_limits<DOUBLE>::infinity();
        x = Row(n);
        for (int i = 0; i < m; i++)
            if (B[i] < n)
                x[B[i]] = D[i][n + 1];
        return D[m][n + 1];
    }
};

int main()
{

    const int m = 4;
    const int n = 3;
    DOUBLE _A[m][n] = {{6, -1, 0}, {-1, -5, 0}, {1, 5, 1}, {-1, -5, -1}};
    DOUBLE _b[m] = {10, -4, 5, -5};
    DOUBLE _c[n] = {1, -1, 0};

    Matrix A(m);
    Row b(_b, _b + m);
    Row c(_c, _c + n);
    for (int i = 0; i < m; i++)
        A[i] = Row(_A[i], _A[i] + n);

    LPSolver solver(A, b, c);
    Row x;
    DOUBLE value = solver.Solve(x);

    cerr << "VALUE: " << value << endl; // VALUE: 1.29032
    cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
    for (size_t i = 0; i < x.size(); i++)
        cerr << " " << x[i];
    cerr << endl;
    return 0;
}
```

Output:

```
VALUE: 1.29032
SOLUTION: 1.74194 0.451613 1
```