Introduction to Optimization

Programming Task 1 Report

Innopolis University, Fall 2024

Team information

1. Gleb Popov — team leader
2. Daniil Mayorov
3. Andrew Pavlov

Contribution

The tasks were evenly distributed; each team member did their part perfectly.

Gleb Popov	5/5
Daniil Mayorov	5/5
Andrew Pavlov	5/5

Product Information

- Programming language — C++
- The product is available at the GitHub via the link.

Linear Programming Model

Our program solves the problem of **maximization** an expression under specific constraints each in form of $\leq$.

Input

Objective function as a vector of coefficients	`vector<double>`
Constraint function as a matrix of coefficients	`vector<vector<double>>`
Right-Hand Side as a vector of RHSs	`vector<double>`
Approximation accuracy as decimal places	`int`

Output

Information about solvability of the problem	`bool`
Vector of decision variables (if solvable)	`vector<double>`
Maximum value of the objective function (if solvable)	`double`

Example

Problem	Input format

Maximize:
$z = 9x_1 + 10x_2 + 16x_3$
Restrictions:
$6x_1 + 4x_2 + 8x_3 \leq 192$
$18x_1 + 15x_2 + 12x_3 \leq 360$
$5x_1 + 3x_2 + 3x_3 \leq 180$
Accuracy: 3 decimal places
	`vector<double> z = {9, 10, 16};`
	`vector<vector<double>> constrFun = {`
	`  {6, 4, 8},`
	`  {18, 15, 12},`
	`  {5, 3, 3}`
	`};`
	`vector<double> RHS = {192, 360, 180};`
	`int accur = 3;`
	`simplexMethod(z, constrFun, RHS, accur);`

Code

```
#include <iostream>
#include <vector>
#include <iomanip>

#define DEFAULT_ACCURACY 3

using namespace std;

int findKeyColumn(const vector<vector<double>>& table, const int& n, const int&
accur) {
    double minn = 0;
    int ind = -1;
    for(int col = 0; col < n; col++) {
        if (table[0][col] < minn) {
            minn = table[0][col];
            ind = col;
        }
    }
    for (int i = 0; i < accur; i++) {
        minn *= 10;
    }
    if (minn > -1) {
        return -1;
    }
    return ind;
}

int findKeyRow(const vector<vector<double>>& table, const int& n, const int& vars) {
    double minn = 1e308;
    int ind = -1;
    for (int i = 0; i < n + 1; i++) {
        if (table[i][vars + n + 1] > 0 && table[i][vars + n + 1] < minn) {
            minn = table[i][vars + n + 1];
            ind = i;
        }
    }
    return ind;
}

struct SimplexResult {
```

```
    bool solved;
    vector<double> res_x;
    double res_z;
    int accur;

    void print() const {
        cout << fixed << setprecision(accur);
        if (!solved) {
            cout << "The method is not applicable!" << endl;
        } else {
            cout << "The maximum value of z is " << res_z << "." << endl;
            cout << "The vector of decision varibales is {";
            for (int i = 0; i < res_x.size(); i++) {
                if (i > 0) {
                    cout << ", ";
                }
                cout << res_x[i];
            }
            cout << "}." << endl;
        }
        cout << endl;
    }
};

SimplexResult simplexMethod(vector<double>& objFun, vector<vector<double>>&
constrFun, vector<double>& RHS, int accur = DEFAULT_ACCURACY) {
    int vars = objFun.size();          // number of variables
    int n = constrFun.size();          // number of equations (exluding the objective)
    bool solvable = true;              // does function have a solution
    vector<double> pivots(vars, 0);    // pivots[variable_index] = index of its line

    // creating a table
    vector<vector<double>> table(n + 1, vector<double>(vars + n + 2, 0));

    // filling the first row (with the objective function)
    for (int col = 0; col < vars; col++) {
        table[0][col] = -objFun[col];
    }
    table[0][vars + n] = 0; // RHS = 0

    // filling the entire table
    for (int r = 1; r < n + 1; r++) {
        for (int c = 0; c < vars; c++) {
            table[r][c] = constrFun[r - 1][c];
        }
        table[r][vars + n] = RHS[r - 1];  // filling RHS
        table[r][vars - 1 + r] = 1;       // filling slack variables
    }

    while (true) {

        // findind key column
        int kk = findKeyColumn(table, vars + n);
        if (kk == -1) {
            break;
        }

        // computing Ratio
```

```
        for (int i = 0; i < n + 1; i++) {
            if (table[i][kk] == 0) {
                table[i][vars + n + 1] = -1;
            } else {
                table[i][vars + n + 1] = table[i][vars + n] / table[i][kk];
            }
        }

        // finding key row
        int kr = findKeyRow(table, n, vars);
        if (kr == -1) {
            solvable = false;
            break;
        }

        // dividing the key row by key element
        double keyelem = table[kr][kk];
        for (int i = 0; i < vars + n + 1; i++) {
            table[kr][i] = table[kr][i] / keyelem;
        }

        // creating new table
        for (int i = 0; i < n + 1; i++) {
            keyelem = table[i][kk];
            if (keyelem == table[kr][kk]) {
                continue;
            }
            for (int j = 0; j < vars + n + 1; j++) {
                table[i][j] = table[i][j] - (keyelem * table[kr][j] / table[kr][kk]);
            }
        }
        pivots[kk] = kr;
    }

    // transferring data from a table to a result
    // if (i = 0), leave it 0, if not, replace with the RHS of its line
    for (int i = 0; i < vars; i++) {
        if (pivots[i] != 0) {
            pivots[i] = table[pivots[i]][vars + n];
        }
    }

    SimplexResult res;
    if (!solvable) {
        res.solved = false;
    } else {
        res.solved = true;
        res.accur = accur;
        res.res_z = table[0][vars + n];
        res.res_x = pivots;
    }
    return res;
}

int main() {
    vector<double> z, RHS;
    vector<vector<double>> constrFun;
    int accur;
```

```
    // TEST 1 – solvable
    z = {9, 10, 16};
    constrFun = {
        {6, 4, 8},
        {18, 15, 12},
        {5, 3, 3}
    };
    RHS = {192, 360, 180};
    accur = 3;
    simplexMethod(z, constrFun, RHS, accur).print();

    // TEST 2 – solvable
    z = {2, 3, 0, -1, 0, 0};
    constrFun = {
        {2, -1, 0, -2, 1, 0},
        {3, 2, 1, -3, 0, 0},
        {-1, 3, 0, 4, 0, 1},
    };
    RHS = {16, 18, 24};
    accur = 5;
    simplexMethod(z, constrFun, RHS, accur).print();

    // TEST 3 – solvable
    z = {1, 2, 1};
    constrFun = {
        {1, 2, 1},
        {3, 1, 1},
        {1, 1, 2},
        {1, 1, 1}
    };
    RHS = {2, 4, 4, 2};
    accur = 3;
    simplexMethod(z, constrFun, RHS, accur).print();

    // TEST 4 – not solvable since one of the RHS < 0
    z = {3, 2};
    constrFun = {
        {1, -1},
        {-2, 1},
    };
    RHS = {1, -2};
    accur = 3;
    simplexMethod(z, constrFun, RHS, accur).print();

    // TEST 5 – not solvable but initial RHS >= 0
    z = {3, 0};
    constrFun = {
        {-1, 1},
        {-1, 1},
    };
    RHS = {1, 2};
    accur = 3;
    simplexMethod(z, constrFun, RHS, accur).print();
}
```