Lab06-Linear Programming

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2021.

∗ If there is any problem, please contact TA Haolin Zhou.
∗ Name: Zirui Liu Student ID: 519021910343 Email: L.prime@sjtu.edu.cn

1. *Hirschberg Algorithm.* Recall the **String Similarity** problem in class, in which we calculate the edit distance between two strings in a sequence alignment manner.

   (a) Implement the algorithm combining **dynamic programming** and **divide-and-conquer** strategy in C/C++. Analyze the time complexity of your algorithm. (The template *Code-SequenceAlignment.cpp* is attached on the course webpage).

   (b) Given $\alpha(x, y) = |ascii(x) - acsii(y)|$, where $ascii(c)$ is the ASCII code of character c, and $\delta = 13$. Find the edit distance between the following two strings.

$$X[1..60] = CMQHZZRIQOQJOCFPRWOUXXCEMYSWUJ$$
$$TAQBKAJIETSJPWUPMZLNLOMOZNLTLQ$$

$$Y[1..50] = SUYLVMUSDROFBXUDCOHAATBKN$$
$$AAENXEVWNLMYUQRPEOCJOCIMZ$$

Solution. The results are here.

```
CMQHZZRIQOQJOCFPRWOUXXCEMYSWUJTAQBKAJIETSJPWUPMZLNLOMO-ZNLTLQ
--S-UY-LVMU-SD-RO-FBXUDCO-H--A-ATBKNAAENXEVWNLMYUQRPEOCJOCIMZ
TOTAL cost: 385
```

Figure 1: result1

And codes can be found in the appendices.
The time complexity is $O(mn)$.
The time used to calculate a value for $Dp(s1, s2)$ is $O(|s1| * |s2|)$. Considering the recursion, The two top-leveled recursive calls each solve a smaller problem. Together these two sub-problems add up to half of the size of the original problem, which is $O(|s1| * |s2|/2)$ time complexity in total. Then we consider the next recursive level, which has four small sub-recursive problems in total, each one takes one quarter of the original size. Keep doing this, then we can conclude that the total time taken for the algorithm is $O(|s1| * |s2| * (1 + 1/2 + 1/4 + ...))$ which is $O(2 * |s1| * |s2|)$. That is $O(|s1| * |s2|)$. This use of recursion reduces the space used from O(|s1|*|s2|) to O(|s2|) and roughly doubles the time taken.

□

2. *Travelling Salesman Problem.* Given a list of cities and the distances between each pair of cities ($G = (V, E, W)$), we want to find the shortest possible route that visits each city exactly once and returns to the origin city. Similar to **Maximum Independent Set** and **Dominating Set**, please turn the traveling salesman problem into an ILP form.

   Remark: W is the set of weights corresponds to the edges that connecting adjacent cities.

Solution. Let us suppose that x_{uv} be an indicator for whether there exists direct edge connecting two vertexes u and v. w_{uv} to be the weight of the directed edge from u to v. x_{uv} satisfies the following condition:

$$x_{uv} = \begin{cases} 1, & \text{if the salesman goes from } u \text{ to } v \text{ directly} \\ 0, & \text{if the salesman doesn't go from } u \text{ to } v \text{ directly} \end{cases}$$

Then we can transform the original problem to an ILP problem. We want to have: $\sum_{u,v \in V} w_{uv} * x_{uv}$ having the least sum.

□

3. *Investment Strategy.* A company intends to invest 0.3 million yuan in 2021, with a proper combination of the following 3 projects:

   - **Project 1:** Invest at the beginning of a year, and can receive a 20% profit of the investment in this project at the end of this year. Both the capital and profit can be invested at the beginning of next year;
   - **Project 2:** Invest at the beginning of 2021, and can receive a 50% profit of the investment in this project at the end of 2022. The investment in this project cannot exceed 0.15 million dollars;
   - **Project 3:** Invest at the beginning of 2022, and can receive a 40% profit of the investment in this project at the end of 2022. The investment in this project cannot exceed 0.1 million dollars.

   Assume that the company will invest *all* its money at the beginning of a year. Please design a scheme of investment in 2021 and 2022 which maximizes the overall sum of capital and profit at the end of 2022.

   (a) Formulate a linear programming with necessary explanations.

   (b) Transform your LP into its standard form and slack form.

   (c) Transform your LP into its dual form.

   (d) Use the simplex method to solve your LP.

   Solution. (a):
   We assume that x_1 and x_2 separately to be investment for **Project 1** in year 2021 and 2022, y and z separately to be investment for **Project 2** and **Project 3**. The original problem can be transformed into the following linear programming problem:
   We want $max\,(1.2x_2 + 1.5y + 1.4z)$
   And we have following conditions:
   $x_1 + y = 0.3$
   $x_2 + z = 1.2x_1$
   $y \leq 0.15$
   $z \leq 0.1$
   $x_1, x_2, y, z \geq 0$

   Explanation :
   At the beginning of 2021, we can only invest in Project 1 and Project 2 and since all the money must be used, we can get the equation $x_1 + y = 0.3$ holds.Then the money available can be invested at the beginning of 2022 is the profit from then end of 2021, which is clear that we can have $x_2 + z = 1.2x_1$. $y \leq 0.15$, $z \leq 0.1$, $x_1, x_2, y, z \geq 0$ are basic requirements for **Project 2** and **Project 3**.

(b):
Standard form:
We want $max\,(1.2x_2 + 1.5y + 1.4z)$
And we have following conditions:
$x_1 + y \leq 0.3$
$x_2 + z \leq 1.2x_1$
$y \leq 0.15$
$z \leq 0.1$
$x_1, x_2, y, z \geq 0$

Slack form:
We want $max\,(1.2x_2 + 1.5y + 1.4z)$
We also assume w_1 and w_2 to be slack variables.
And we have following conditions:
$x_1 + y = 0.3$
$x_2 + z = 1.2x_1$
$y + w_1 = 0.15$
$z + w_2 = 0.1$
$x_1, x_2, y, z \geq 0$

(c):
Dual form:
We want $min\,(0.3a_1 - 0.3a_2 + 0.15a_5 + 0.1a_6)$
And we have following conditions:
$a_1 - a_2 - 1.2a_3 + 1.2a_4 \geq 0$
$a_3 - a_4 \geq 1.2$
$a_1 - a_2 + a_5 \geq 1.5$
$a_3 - a_4 + a_6 \geq 1.4$
$a_1, a_2, a_3, a_4, a_5, a_6 \geq 0$

(d):
Final solution:

$\bar{x} = (0.15, 0.1, 0, 0)$
We can solve that the result is 0.461.

□

4. *Factory Production.* An engineering factory makes seven products (PROD 1 to PROD 7) on the following machines: four grinders, two vertical drills, three horizontal drills, one borer and one planer. Each product yields a certain contribution to profit (in £/unit). These quantities (in £/unit) together with the unit production times (hours) required on each process are given below. A dash indicates that a product does not require a process.

   There are marketing limitations on each product in each month, given in the following table:

   It is possible to store up to 100 of each product at a time at a cost of £0.5 per unit per month (charged at the end of each month according to the amount held at that time). There are no stocks at present, but it is desired to have a stock of exactly 50 of each type of product at the end of June. The factory works six days a week with two shifts of 8h each day. It may be

	PROD 1	PROD 2	PROD 3	PROD 4	PROD 5	PROD 6	PROD 7
Contribution to profit	10	6	8	4	11	9	3
Grinding	0.5	0.7	-	-	0.3	0.2	0.5
Vertical drilling	0.1	0.2	-	0.3	-	0.6	-
Horizontal drilling	0.2	-	0.8	-	-	-	0.6
Boring	0.05	0.03	-	0.07	0.1	-	0.08
Planing	-	-	0.01	-	0.05	-	0.05

	PROD 1	PROD 2	PROD 3	PROD 4	PROD 5	PROD 6	PROD 7
January	500	1000	300	300	800	200	100
February	600	500	200	0	400	300	150
March	300	600	0	0	500	400	100
April	200	300	400	500	200	0	100
May	0	100	500	100	1000	300	0
June	500	500	100	300	1100	500	60

assumed that each month consists of only 24 working days. Each machine must be down for maintenance in one month of the six. No sequencing problems need to be considered.

When and what should the factory make in order to maximize the total net profit?

(a) Use *CPLEX Optimization Studio* to solve this problem. Describe your model in *Optimization Programming Language* (OPL). Remember to use a separate data file (.dat) rather than embedding the data into the model file (.mod).

(b) Solve your model and give the following results.

i. For each machine:

A. the month for maintenance.
This can be checked out in the next chart, where the month for maintenance has zero amount to make.

ii. For each product:

A. The amount to make in each month.

	PROD 1	PROD 2	PROD 3	PROD 4	PROD 5	PROD 6	PROD 7
Jan	500.0	1000.0	300.0	300.0	800.0	200.0	100.0
Feb	600.0	500.0	200.0	0.0	400.0	300.0	150.0
Mar	400.0	700.0	100.0	100.0	600.0	400.0	200.0
Apr	0.0	0.0	0.0	0.0	0.0	0.0	0.0
May	0.0	100.0	500.0	100.0	1000.0	300.0	0.0
Jun	550.0	550.0	150.0	350.0	1150.0	550.0	110.0

B. The amount to sell in each month.

	PROD 1	PROD 2	PROD 3	PROD 4	PROD 5	PROD 6	PROD 7
Jan	500.0	1000.0	300.0	300.0	800.0	200.0	100.0
Feb	600.0	500.0	200.0	0.0	400.0	300.0	150.0
Mar	300.0	600.0	0.0	0.0	500.0	400.0	100.0
Apr	100.0	100.0	100.0	100.0	100.0	0.0	100.0
May	0.0	100.0	500.0	100.0	1000.0	300.0	0.0
Jun	500.0	500.0	100.0	300.0	1100.0	500.0	60.0

C. The amount to hold at the end of each month.

	PROD 1	PROD 2	PROD 3	PROD 4	PROD 5	PROD 6	PROD 7
Jan	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Feb	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Mar	100.0	100.0	100.0	100.0	100.0	0.0	100.0
Apr	0.0	0.0	0.0	0.0	0.0	0.0	0.0
May	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Jun	50.0	50.0	50.0	50.0	50.0	50.0	50.0

iii. The total selling profit.
\$109330

iv. The total holding cost.
\$475

v. The total net profit (selling profit minus holding cost).
\$108855.00

Remark: You can choose to use the attached .dat file or write it yourself.

Solution. For personal reasons that I have set-up Gurobi in MCM, so I used Gurobi instead of cplex. The codes can be seen in appendices.

□

1 First appendix: code-test.cpp

Input C++ source1:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int total=0;

int alpha(char c1, char c2) {
        return abs(c1 - c2);
}

vector<int> grade(string X, string Y, int delta, bool rev)
{
        int m = static_cast<int>(X.length());
        int n = static_cast<int>(Y.length());
        vector<int> result(n + 1, 0);
        for (int i = 0; i <= n; ++i){
                result[i]=i*delta;
        }

        if (rev)
        {
                reverse(X.begin(), X.end());
                reverse(Y.begin(), Y.end());
        }

        for (int i = 1; i <= m; ++i)
        {
                int prev = result[0];
                result[0] += delta;
                for (int j = 1; j <= n; ++j)
                {
                        int cost = 0;
                        cost=prev+alpha(X[i-1],Y[j-1]);
                        cost = min(cost, delta + result[j]);
                        cost = min(cost, delta + result[j - 1]);
                        prev = result[j];
                        result[j] = cost;
                }
        }
        return result;
}

pair<string, string> DP(string X, string Y, int delta,int &total)
{
        int m = X.length();
```

```
int n = Y.length();
vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
for (int i = 0; i <= m; ++i)
        dp[i][0] = i * delta;
for (int j = 0; j <= n; ++j)
        dp[0][j] = j * delta;

for (int i = 1; i <= m; ++i)
{
        //dp[i][0] = dp[i - 1][0] + delta;
        for (int j = 1; j <= n; ++j) {
                dp[i][j] = dp[i - 1][j - 1] + alpha(X[i-1 ], Y[j-1]);
                dp[i][j] = min(dp[i][j], dp[i - 1][j] + delta);
                dp[i][j] = min(dp[i][j], dp[i][j - 1] + delta);
        }
}
//cout<<"delta:  "<<delta<<endl;
//cout<<dp[m][n]<<endl;

int i = m, j = n;
vector<string> alignment(2);
// getting back to check the path of connecting couples
while (i >= 1 || j >= 1)
{
        if ((i > 0 && dp[i][j] == delta + dp[i - 1][j]) || j == 0)
        {
                cout<<"X[i-1]: "<<X[i-1]<<"     "<<"Y[j-1]: "<<"-"<<en
                cout<<"total= "<<total<<"+"<<delta<<endl;
                total+=delta;
                alignment[0].push_back(X[i - 1]);
                alignment[1].push_back('-');
                --i;
        }
        else if ((j > 0 && dp[i][j] == delta + dp[i][j - 1]) || i ==
        {
                cout<<"X[i-1]: "<<"-"<<"     "<<"Y[j-1]: "<<Y[j-1]<<en
                cout<<"total= "<<total<<"+"<<delta<<endl;
                total+=delta;
                alignment[0].push_back('-');
                alignment[1].push_back(Y[j - 1]);
                --j;
        }
        else if (dp[i][j] == dp[i - 1][j - 1] + alpha(X[i-1],Y[j-1])
        {
                //cout<<"i: "<<i<<"     "<<"j: "<<j<<endl;
                cout<<"X[i-1]: "<<X[i-1]<<"     "<<"Y[j-1]: "<<Y[j-1]<
                cout<<"total = "<<total<<"+"<<alpha(X[i-1],Y[j-1])<<e
                total+=alpha(X[i-1],Y[j-1]);
                alignment[0].push_back(X[i - 1]);
                alignment[1].push_back(Y[j - 1]);
```

```
                --i, --j;
            }
        }
        reverse(alignment[0].begin(), alignment[0].end());
        reverse(alignment[1].begin(), alignment[1].end());

        return {alignment[0], alignment[1]};
}
//    ATC-GCGTA
//    ACCTGACT-

int argmax_(vector<int> scoreL, vector<int> scoreR)
{
        int n = static_cast<int>(scoreL.size());
        int minimum = INT_MAX, index = 0;
        //  ymid = arg max ScoreL + rev(ScoreR)
        reverse(scoreR.begin(), scoreR.end());

        for (int i = 0; i < n; ++i)
        {
            if (scoreL[i] + scoreR[i] < minimum)
            {
                minimum = scoreL[i] + scoreR[i];
                index = i;
            }
        }
        return index;
}

pair<string, string> Hirschberg(string X, string Y, int delta,int &total)
{
        int m = static_cast<int>(X.length());
        int n = static_cast<int>(Y.length());

        string Sub_1, Sub_2;

        if (m == 0)
        {
            cout<<"X[i-1]: "<<"-"<<"    "<<"Y[j-1]: "<<Y<<endl;
            cout<<"total= "<<total<<"+"<<n*delta<<endl;
            total+=(delta*n);
            return{ string(n, '-'), Y };  //return n times - .
        }
        else if (n == 0)
        {
            cout<<"X[i-1]: "<<X<<"    "<<"Y[j-1]: "<<"-"<<endl;
            cout<<"total= "<<total<<"+"<<delta*m<<endl;
            total+=(delta*m);
            return{ X, string(m, '-') };  // same
        }
```

```
        else if (m == 1 || n == 1)
        {
                //cout<<"total= "<<total<<"+"<<alpha(X[0],Y[0])<<endl;
                //total+=alpha(X[0],Y[0]);
                return DP(X, Y, delta,total);
        }
        else
        {
                int xmid = m / 2;
                string X_left = X.substr(0, xmid);    // substr( start_point,
                string X_right = X.substr(xmid, m - xmid);
                int ymid = argmax_(grade(X_left, Y, delta, false), grade(X_r
                string Y_left = Y.substr(0, ymid);
                string Y_right = Y.substr(ymid, n - ymid);

                pair<string, string> result1 = Hirschberg(X_left, Y_left, del
                pair<string, string> result2 = Hirschberg(X_right, Y_right, d

                Sub_1 = result1.first + result2.first;
                Sub_2 = result1.second + result2.second;
        }
        return { Sub_1, Sub_2 };
}


int main()
{
//      string X = "CTACCG";
//      string Y = "TACATG";

    //   ATCGCGTA
    //   ACCTGACT

    string X="ATCGCGTA";
    string Y="ACCTGACT";

        string X1 = "AGTACGCA";
        string Y1 = "TATGC";

        string X2="CMQHZZRIQOQJOCFPRWOUXXCEMYSWUJTAQBKAJIETSJPWUPMZLNLOMOZNLT
        string Y2="SUYLVMUSDROFBXUDCOHAATBKNAAENXEVWNLMYUQRPEOCJOCIMZ";
        //int total;
        auto result1 = Hirschberg(X, Y, 13,total);
        cout << "X: " << X << endl;
        cout << "Y: " << Y << endl;
        cout << "the alignment for X and Y:" << endl;
        cout << result1.first << endl;
        cout << result1.second << endl;
        cout<<"TOTAL cost: "<<total<<endl;
```

```
        cout << "------------------------------------------------------------" << endl;
        total=0;
        auto result2 = Hirschberg(X1, Y1, 13,total);
        cout << "X1: " << X1 << endl;
        cout << "Y1: " << Y1 << endl;
        cout << "the alignment for X1 and Y1:" << endl;
        cout << result2.first << endl;
        cout << result2.second << endl;
        cout<<"TOTAL cost: "<<total<<endl;

        cout << "------------------------------------------------------------" << endl;
        total=0;
        auto result3 = Hirschberg(X2, Y2, 13,total);
        cout << "X1: " << X2 << endl;
        cout << "Y1: " << Y2 << endl;
        cout << "the alignment for X1 and Y1:" << endl;
        cout << result3.first << endl;
        cout << result3.second << endl;
        cout<<"TOTAL cost: "<<total<<endl;
    //
        //DP(X,Y,13);


        return 0;
}
```

2 Second appendix: gurobi.py

Input python source2:

```
import numpy as np
import pandas as pd
import gurobipy as gp
from gurobipy import GRB

products = ["Prod1", "Prod2", "Prod3", "Prod4", "Prod5", "Prod6", "Prod7"]
machines = ["grinder", "vertDrill", "horiDrill", "borer", "planer"]
months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]

profit = {"Prod1":10, "Prod2":6, "Prod3":8, "Prod4":4, "Prod5":11, "Prod6":9,

time_require = {
    "grinder": {    "Prod1": 0.5, "Prod2": 0.7, "Prod5": 0.3,
                    "Prod6": 0.2, "Prod7": 0.5 },
    "vertDrill": {  "Prod1": 0.1, "Prod2": 0.2, "Prod4": 0.3,
                    "Prod6": 0.6 },
    "horiDrill": {  "Prod1": 0.2, "Prod3": 0.8, "Prod7": 0.6 },
    "borer": {      "Prod1": 0.05,"Prod2": 0.03,"Prod4": 0.07,
                    "Prod5": 0.1, "Prod7": 0.08 },
    "planer": {     "Prod3": 0.01,"Prod5": 0.05,"Prod7": 0.05 }
}

# number of each machine available
machines_installed = {"grinder":4, "vertDrill":2, "horiDrill":3, "borer":1, "

# number of machines that need to be under maintenance
down_require = {"grinder":2, "vertDrill":2, "horiDrill":3, "borer":1, "planer

# market limitation of sells
max_sales = {
    ("Jan", "Prod1") : 500,
    ("Jan", "Prod2") : 1000,
    ("Jan", "Prod3") : 300,
    ("Jan", "Prod4") : 300,
    ("Jan", "Prod5") : 800,
    ("Jan", "Prod6") : 200,
    ("Jan", "Prod7") : 100,
    ("Feb", "Prod1") : 600,
    ("Feb", "Prod2") : 500,
    ("Feb", "Prod3") : 200,
    ("Feb", "Prod4") : 0,
    ("Feb", "Prod5") : 400,
    ("Feb", "Prod6") : 300,
    ("Feb", "Prod7") : 150,
    ("Mar", "Prod1") : 300,
    ("Mar", "Prod2") : 600,
    ("Mar", "Prod3") : 0,
```

```
    ("Mar", "Prod4") : 0,
    ("Mar", "Prod5") : 500,
    ("Mar", "Prod6") : 400,
    ("Mar", "Prod7") : 100,
    ("Apr", "Prod1") : 200,
    ("Apr", "Prod2") : 300,
    ("Apr", "Prod3") : 400,
    ("Apr", "Prod4") : 500,
    ("Apr", "Prod5") : 200,
    ("Apr", "Prod6") : 0,
    ("Apr", "Prod7") : 100,
    ("May", "Prod1") : 0,
    ("May", "Prod2") : 100,
    ("May", "Prod3") : 500,
    ("May", "Prod4") : 100,
    ("May", "Prod5") : 1000,
    ("May", "Prod6") : 300,
    ("May", "Prod7") : 0,
    ("Jun", "Prod1") : 500,
    ("Jun", "Prod2") : 500,
    ("Jun", "Prod3") : 100,
    ("Jun", "Prod4") : 300,
    ("Jun", "Prod5") : 1100,
    ("Jun", "Prod6") : 500,
    ("Jun", "Prod7") : 60,
}

holding_cost = 0.5
max_inventory = 100
store_target = 50
hours_per_month = 2*8*24

factory = gp.Model('Factory Planning')

make = factory.addVars(months, products, name="Make")
store = factory.addVars(months, products, ub=max_inventory, name="Store")
sell = factory.addVars(months, products, ub=max_sales, name="Sell")
repair = factory.addVars(months, machines, vtype=GRB.INTEGER, ub=down_require

Balance0 = factory.addConstrs((make[months[0], product] == sell[months[0], pr
                + store[months[0], product] for product in products), name=

#2. Balance
Balance = factory.addConstrs((store[months[months.index(month) -1], product]
              make[month, product] == sell[month, product] + store[month, p
              for product in products for month in months
              if month != months[0]), name="Balance")

TargetInv = factory.addConstrs((store[months[-1], product] == store_target fo
name="End_Balance")
```

```
MachineCap = factory.addConstrs((gp.quicksum(time_require[machine][product] *
                                 for product in time_require[machine])
                    <= hours_per_month * (machines_installed[machine] - repai
                     for machine in machines for month in months),
                    name = "Capacity")

Maintenance = factory.addConstrs((repair.sum('*', machine) == down_require[m

obj = gp.quicksum(profit[product] * sell[month, product] -  holding_cost * st
                  for month in months for product in products)

factory.setObjective(obj, GRB.MAXIMIZE)

factory.optimize()

rows = months.copy()
columns = products.copy()
make_plan = pd.DataFrame(columns=columns, index=rows, data=0.0)

for month, product in make.keys():
    if (abs(make[month, product].x) > 1e-6):
        make_plan.loc[month, product] = np.round(make[month, product].x, 1)
make_plan

rows = months.copy()
columns = products.copy()
sell_plan = pd.DataFrame(columns=columns, index=rows, data=0.0)

for month, product in sell.keys():
    if (abs(sell[month, product].x) > 1e-6):
        sell_plan.loc[month, product] = np.round(sell[month, product].x, 1)
sell_plan

rows = months.copy()
columns = products.copy()
store_plan = pd.DataFrame(columns=columns, index=rows, data=0.0)

for month, product in store.keys():
    if (abs(store[month, product].x) > 1e-6):
        store_plan.loc[month, product] = np.round(store[month, product].x, 1)
store_plan

rows = months.copy()
columns = machines.copy()
repair_plan = pd.DataFrame(columns=columns, index=rows, data=0.0)

for month, machine in repair.keys():
    if (abs(repair[month, machine].x) > 1e-6):
        repair_plan.loc[month, machine] = repair[month, machine].x
```

```
repair_plan

#factory.write("factory-planning-1-output.sol")
```

Appendix

A. FactoryPlanning.dat

```
NbMonths = 6;

Prod = {Prod1, Prod2, Prod3, Prod4, Prod5, Prod6, Prod7};
Process = {Grind, VDrill, HDrill, Bore, Plane};

// profitProd[j] is profit per unit for product j
ProfitProd = [10 6 8 4 11 9 3];

// processProd[i][j] gives hours of process i required by product j
ProcessProd = [[0.5 0.7 0.0 0.0 0.3 0.2 0.5 ]
[0.1 0.2 0.0 0.3 0.0 0.6 0.0 ]
[0.2 0.0 0.8 0.0 0.0 0.0 0.6 ]
[0.05 0.03 0.0 0.07 0.1 0.0 0.08]
[0.0 0.0 0.01 0.0 0.05 0.0 0.05]];

// marketProd[i][j] gives marketing limitation on product j for month i
MarketProd = [[500 1000 300 300 800 200 100]
[600 500 200 0 400 300 150]
[300 600 0 0 500 400 100]
[200 300 400 500 200 0 100]
[0 100 500 100 1000 300 0 ]
[500 500 100 300 1100 500 60 ]];

CostHold = 0.5;
StartHold = 0;
EndHold = 50;
MaxHold = 100;

// process capacity
HoursMonth = 384; // 2 eight hour shifts per day, 24 working days per month;

// number of each type of machine
NumProcess = [4 2 3 1 1];

// how many machines must be down over 6 month period
NumDown = [4 2 3 1 1];
```

Remark: You need to include your .cpp, .mod, .dat, .pdf and .tex files in your uploaded .zip file.