Stochastic Programming - ATM Lab Assignment

Henrik Andreas Grenersen

June 2, 2024

1 Introduction and Problem Formulation

In this assignment I will consider the following optimization problem

$$
\begin{aligned}
\min_{x,y} \quad & cx + q\sum_{i=1}^{s} p_i y_i \\
\text{subject to} \quad & l \leq x \leq u \\
& x + y_i \geq \xi_i \quad i = 1, ..., s \\
& y_i \geq 0 \quad i = 1, ..., s
\end{aligned}
$$

.

Where this problem arises from a situation where a bank branch wants to determine how much money they should deposit in an ATM on a Friday (x), each euro having a unit cost of c. The demand of money is given by a discrete random variable ξ_i with corresponding probabilities p_i, for each of the scenarios denoted by $i \in \{1, ..., s\}$. The branch is also able to refill money, which here is the second stage decision, denoted by y_i. Thus, the problem states to minimize the total cost, given by the amount we deposit on Friday (cx) and the expected cost of refilling ($qE[\xi]$).

2 Formulation of Sub- and Master Problem

We can recognize the above problem as the extensive form of a stochastic program, and I have also solved it as this in AMPL. For now however, I will focus on solving the problem through Benders decomposition, by first presenting the formulation of the Benders master and subproblem, before implementing this in AMPL.

When formulating the problems mentioned above, I will try to follow the notation from the lectures as closely as possibly. To do this, I start by formulating the original problem, denoted by "(P)".

$$
(P): \quad
\begin{aligned}
\min_{x,y} \quad & \vec{c}^T\vec{x} + \vec{q}^T\vec{y} \\
\text{subject to} \quad & T\vec{x} + W\vec{y} = \vec{h} \\
& \vec{x} \in X \\
& \vec{y} \geq \vec{0}
\end{aligned}
$$

.

So, by comparing (1) and (P), we can recognize $\vec{c} = c$ and $\vec{x} = x$ while $X = \{x : l \leq x \leq u\}$. We also see that the demand that $\vec{y} \geq 0$ is present in both formulations, and this should be thought of as elementwise comparison. However, we see that (P) includes an equality constraint, not an inequality constraint as in (1). We can work around this by including slack variables the constraint for each scenario, by introducing slack variables denoted by sl_i. Thus we can update this constraint to

$$x + y_i - sl_i = \xi_i$$

while also including the constraint that $sl_i \geq 0$. Then, we see that the formulations are indeed equal by defining

$$\vec{y} = \begin{bmatrix} y_1 \\ sl_1 \\ y_2 \\ sl_2 \\ \vdots \\ y_s \\ sl_s \end{bmatrix}$$

To obtain the same objective function, we then need to define

$$\vec{q} = \begin{bmatrix} qp_1 \\ 0 \\ qp_2 \\ 0 \\ \vdots \\ qp_s \\ 0 \end{bmatrix}$$

since the slack variables make no contribution to the objective function. Additionally, T is a column vector of ones with s rows, while the matrix W has elements

$$W_{ij} = \begin{cases} 1 & \text{if} \quad i = j \\ -1 & \text{if} \quad j = i + 1 \\ 0 & \text{else} \end{cases}$$

So, in the case that $s = 3$ for instance, we have that

$$W = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}.$$

Lastly, the column vector $\vec{h}$ also has s elements, where each element is given as $h_i = \xi_i$.

Thus, to summarize, the formulation is given by

Element	**Expression**
$\vec{x}$	x
$\vec{c}$	c
$\vec{q_i}$	$\begin{cases} qp_i & \text{if i is odd} \\ 0 & \text{if i is even} \end{cases}$
$\vec{y_i}$	$\begin{cases} y_i & \text{if i is odd} \\ sl_{i-1} & \text{if i is even} \end{cases}$
T_i	1
W$_{ij}$	$\begin{cases} 1 & \text{if} \quad i = j \\ -1 & \text{if} \quad j = i + 1 \\ 0 & \text{else} \end{cases}$
$\vec{h_i}$	ξ_i

Table 1: Table with elements and expressions

Now, by considering only the part of (P) that depends on $\vec{y}$ and considering x as fixed, we can define the problem (Q), and its dual, called (Q_D). This problem is given as

$$(Q_D): \quad \begin{array}{ll} \max\limits_{\vec{u}} & \vec{u}^T(\vec{h} - T\vec{x}) \\ \text{subject to} & W^T\vec{u} \leq \vec{q} \\ & \vec{u} \in \mathbf{R}^s \end{array}$$

.

This formulation is what is known as the the subproblem in the Benders decomposition. When applying Benders algorithm, we will solve (Q_D) by fixing the current x we have found, and if this problem has a solution, we will either have found our optimal solution or need to add an optimality cut to the master problem. If (Q_D) is unbounded on the other hand, we will add a feasibility cut to the master problem.

Now it seems natural to stop and ask what the master problem is exactly. This is a problem, which we solve iteratively in tandem with (Q_D) as described above, and it is given as

$$(BP_R): \quad \begin{array}{ll} \min\limits_{x,z} & z \\ \text{subject to} & z \geq cx + \vec{u}^{iT}(\vec{h} - Tx) \quad i \in I \\ & \vec{v}^{jT}(\vec{h} - Tx) \leq 0 \quad j \in J \\ & x \in X \end{array}$$

Where the vectors $\vec{u}^i$ and $\vec{v}^j$ correspond to optimality and feasibility constraints respectively.

3 Benders Algorithm

Now I aim to make use of the problem formulations presented in the last section through the following algorithm in order to solve the original problem. The algorithm is given as follows

1. Initializations: Consists of creating empty sets for the constraints in the master provlem
2. Solve (BP_R) to get z_R^* and x_R^*
3. Solve (Q_D) by fixing $x = x_R^*$, which gives $\vec{u}^*$ and z^*
   (a) If (Q_D) has a solution, compare z_R^* with $z^* = cx_R^* + \vec{u}^{*T}(\vec{h} - Tx_R^*)$
      i. If $z_R^* = z^*$, then stop, as the optimal solution is found
      ii. Else, add optimality cut $z + \vec{u}^{*T}(\vec{h} - Tx)$ to (BP_R)
   (b) If (Q_D) is unbounded, add a feasibility cut $\vec{v}^{*T}(\vec{h} - Tx) \leq 0$ to (BP_R)
4. Go to step 2

Where the feasibility cuts are added to ensure that the subproblem is not unbounded, so that the original problem involving only $\vec{y}$, is feasible. The optimality cuts on the other hand control the value of z in the master problem, as we want to minimize z while still having it greater than the right hand side in each optimality cut.

4 Implementation in AMPL

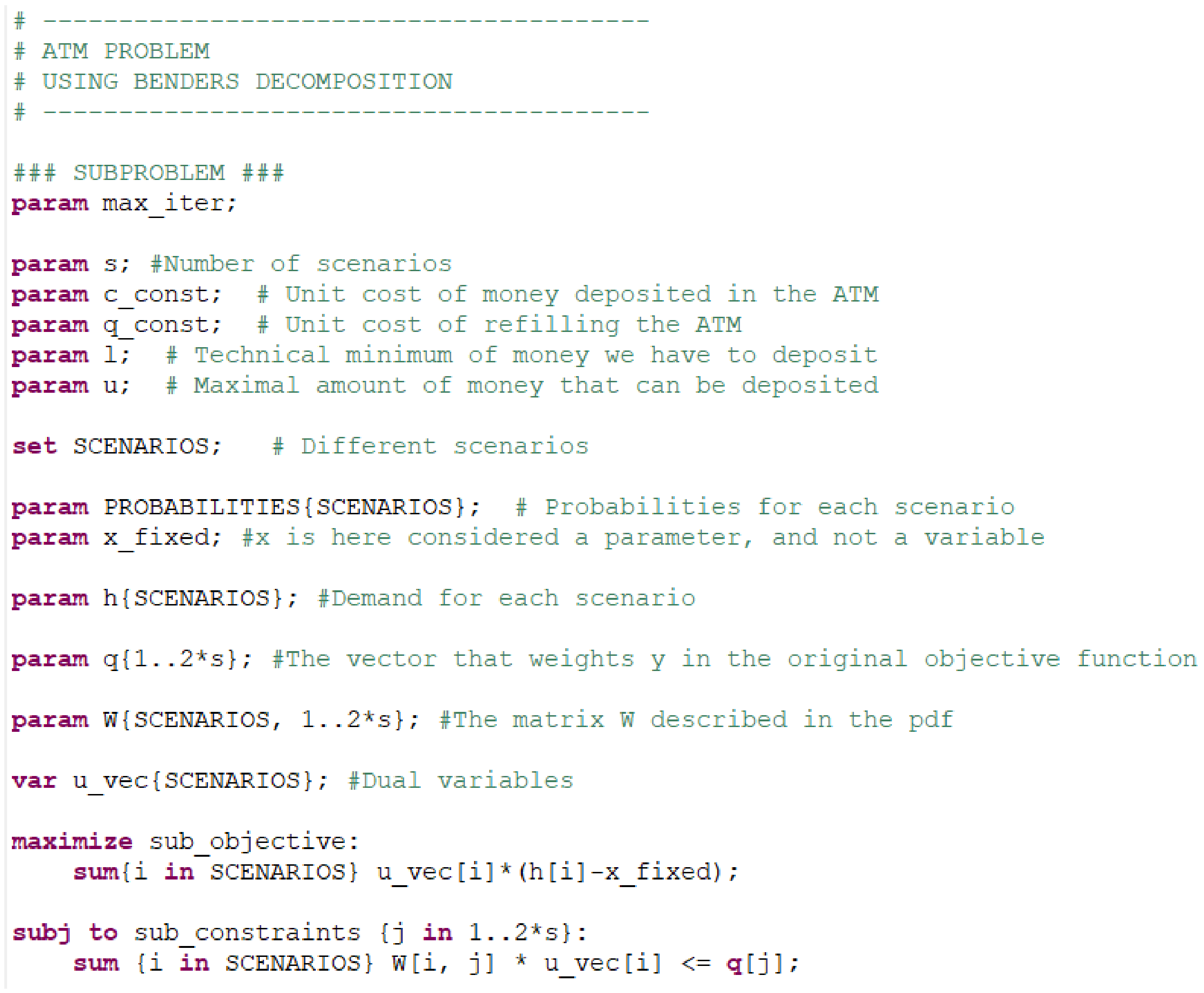
When implementing Benders algorithm in AMPL, I have made use of the example code provided by the lecturers. My approach differs from the example code mainly in that I have chosen to adapt the notation from the slides also in the code.

My formulation of the subproblem, as described in Section 2, along with parameters needed throughout the program is given in Figure 1-3.

The implementation of the master problem is given in Figure 4.

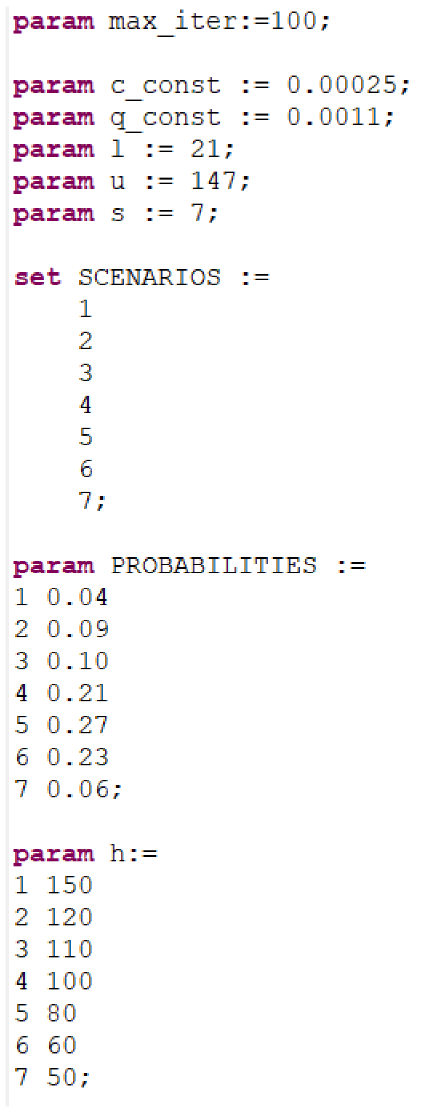
Finally, the algorithm described in section 3 is implemented in the code given in Figure 5.

Additionally, I have implemented and solved the extended problem in its initial form in AMPL, and the model is given in Figure 6.



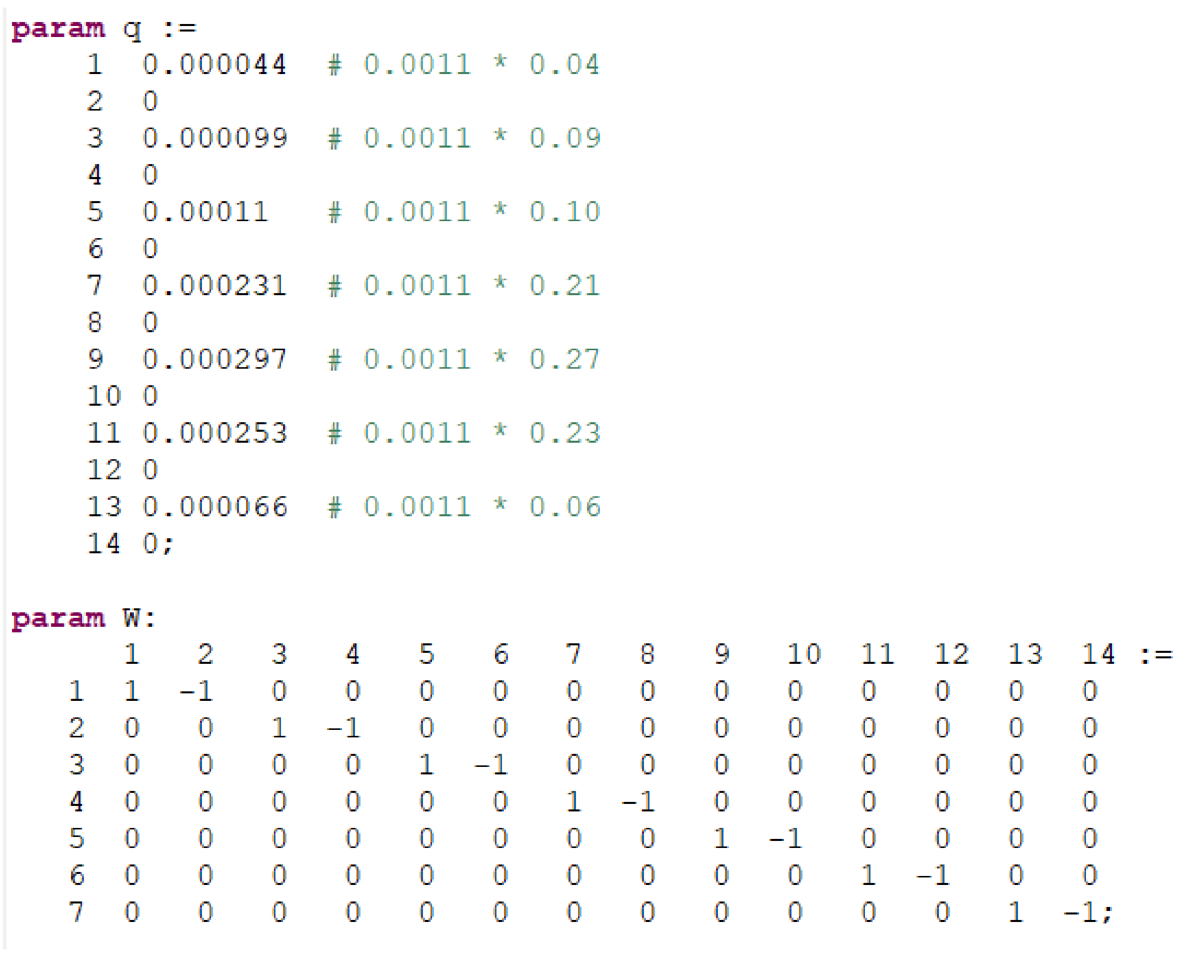
```
# ----------------------------------------
# ATM PROBLEM
# USING BENDERS DECOMPOSITION
# ----------------------------------------

### SUBPROBLEM ###
param max_iter;

param s; #Number of scenarios
param c_const;  # Unit cost of money deposited in the ATM
param q_const;  # Unit cost of refilling the ATM
param l;  # Technical minimum of money we have to deposit
param u;  # Maximal amount of money that can be deposited

set SCENARIOS;   # Different scenarios

param PROBABILITIES{SCENARIOS};  # Probabilities for each scenario
param x_fixed; #x is here considered a parameter, and not a variable

param h{SCENARIOS}; #Demand for each scenario

param q{1..2*s}; #The vector that weights y in the original objective function

param W{SCENARIOS, 1..2*s}; #The matrix W described in the pdf

var u_vec{SCENARIOS}; #Dual variables

maximize sub_objective:
    sum{i in SCENARIOS} u_vec[i]*(h[i]-x_fixed);

subj to sub_constraints {j in 1..2*s}:
    sum {i in SCENARIOS} W[i, j] * u_vec[i] <= q[j];
```

Figure 1: Implementation of Subproblem in AMPL



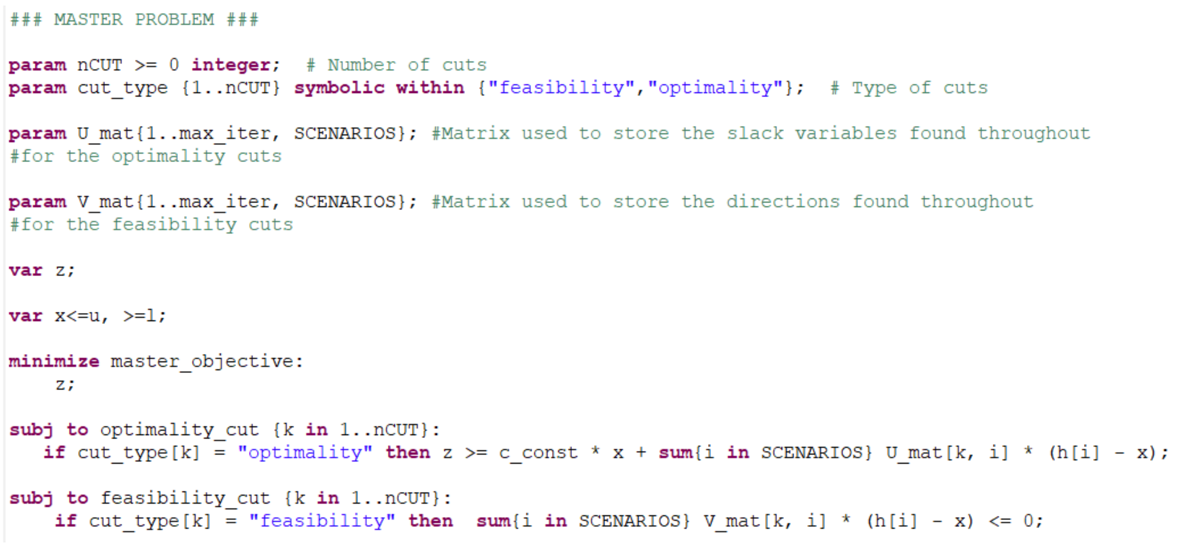
```
param max_iter:=100;

param c_const := 0.00025;
param q_const := 0.0011;
param l := 21;
param u := 147;
param s := 7;

set SCENARIOS :=
    1
    2
    3
    4
    5
    6
    7;

param PROBABILITIES :=
1 0.04
2 0.09
3 0.10
4 0.21
5 0.27
6 0.23
7 0.06;

param h:=
1 150
2 120
3 110
4 100
5 80
6 60
7 50;
```

Figure 2: First Set of Parameters



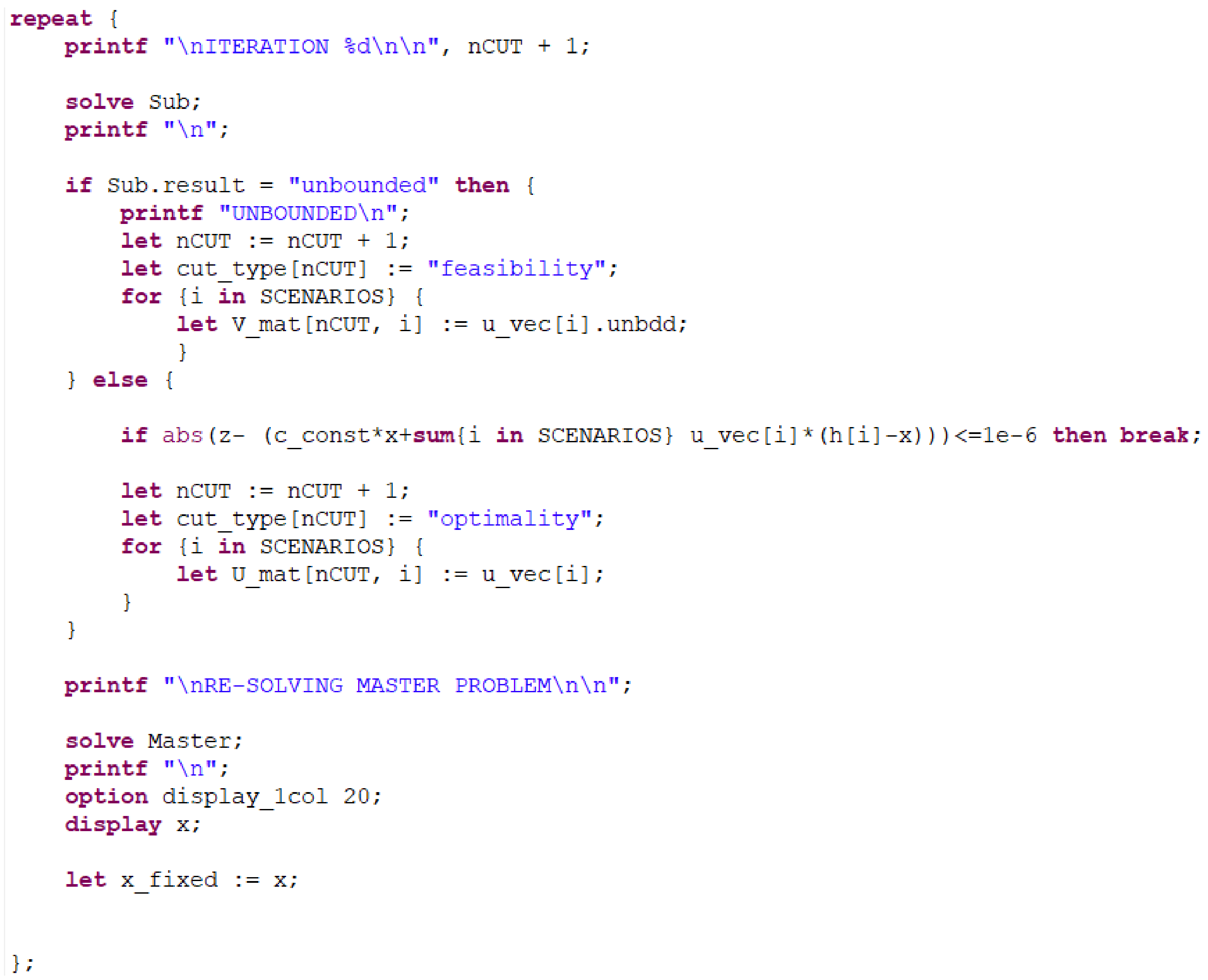
```
param q :=
     1  0.000044  # 0.0011 * 0.04
     2  0
     3  0.000099  # 0.0011 * 0.09
     4  0
     5  0.00011   # 0.0011 * 0.10
     6  0
     7  0.000231  # 0.0011 * 0.21
     8  0
     9  0.000297  # 0.0011 * 0.27
     10 0
     11 0.000253  # 0.0011 * 0.23
     12 0
     13 0.000066  # 0.0011 * 0.06
     14 0;

param W:
        1   2   3   4   5   6   7   8   9   10  11  12  13  14 :=
    1   1  -1   0   0   0   0   0   0   0   0   0   0   0   0
    2   0   0   1  -1   0   0   0   0   0   0   0   0   0   0
    3   0   0   0   0   1  -1   0   0   0   0   0   0   0   0
    4   0   0   0   0   0   0   1  -1   0   0   0   0   0   0
    5   0   0   0   0   0   0   0   0   1  -1   0   0   0   0
    6   0   0   0   0   0   0   0   0   0   0   1  -1   0   0
    7   0   0   0   0   0   0   0   0   0   0   0   0   1  -1;
```

Figure 3: Second Set of Parameters



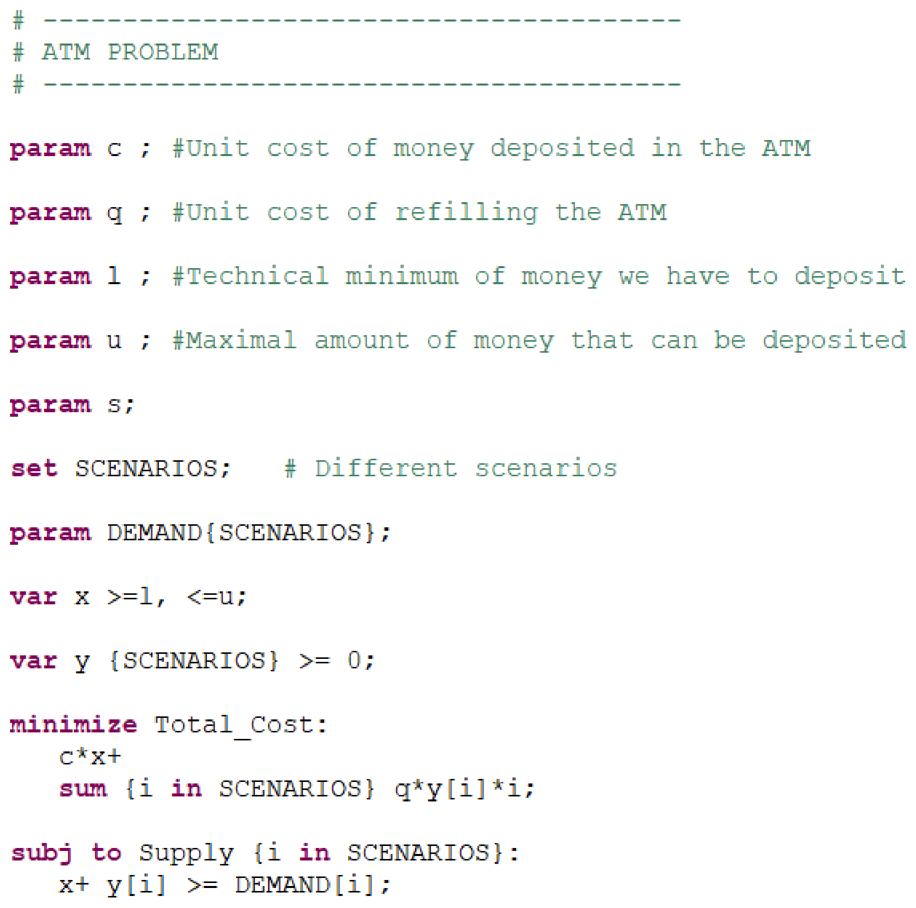
```
### MASTER PROBLEM ###

param nCUT >= 0 integer;  # Number of cuts
param cut_type {1..nCUT} symbolic within {"feasibility","optimality"};  # Type of cuts

param U_mat{1..max_iter, SCENARIOS}; #Matrix used to store the slack variables found throughout
#for the optimality cuts

param V_mat{1..max_iter, SCENARIOS}; #Matrix used to store the directions found throughout
#for the feasibility cuts

var z;

var x<=u, >=l;

minimize master_objective:
    z;

subj to optimality_cut {k in 1..nCUT}:
   if cut_type[k] = "optimality" then z >= c_const * x + sum{i in SCENARIOS} U_mat[k, i] * (h[i] - x);

subj to feasibility_cut {k in 1..nCUT}:
    if cut_type[k] = "feasibility" then  sum{i in SCENARIOS} V_mat[k, i] * (h[i] - x) <= 0;
```

Figure 4: Implementation of Master problem in AMPL



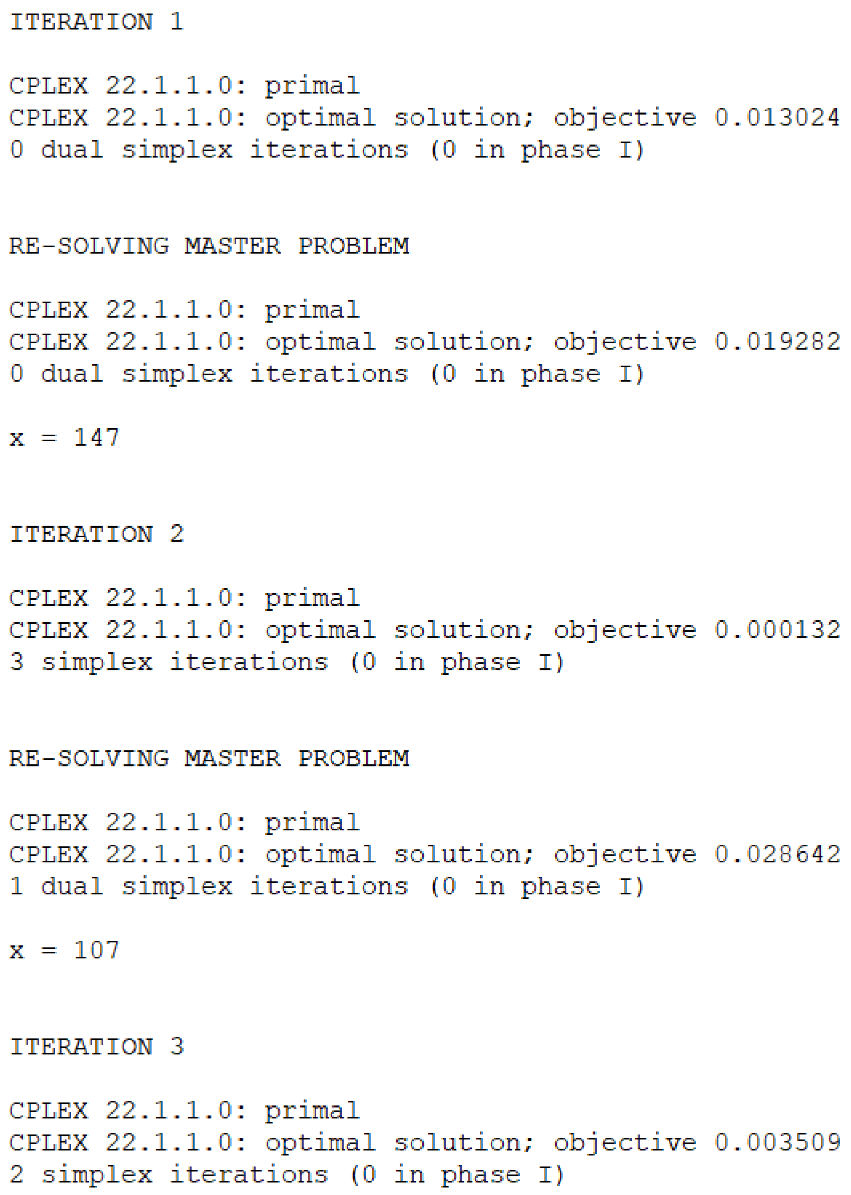
```
repeat {
    printf "\nITERATION %d\n\n", nCUT + 1;

    solve Sub;
    printf "\n";

    if Sub.result = "unbounded" then {
        printf "UNBOUNDED\n";
        let nCUT := nCUT + 1;
        let cut_type[nCUT] := "feasibility";
        for {i in SCENARIOS} {
            let V_mat[nCUT, i] := u_vec[i].unbdd;
            }
    } else {

        if abs(z- (c_const*x+sum{i in SCENARIOS} u_vec[i]*(h[i]-x)))<=1e-6 then break;

        let nCUT := nCUT + 1;
        let cut_type[nCUT] := "optimality";
        for {i in SCENARIOS} {
            let U_mat[nCUT, i] := u_vec[i];
        }
    }

    printf "\nRE-SOLVING MASTER PROBLEM\n\n";

    solve Master;
    printf "\n";
    option display_1col 20;
    display x;

    let x_fixed := x;


};
```

Figure 5: Benders Algorithm for Solving the ATM Problem



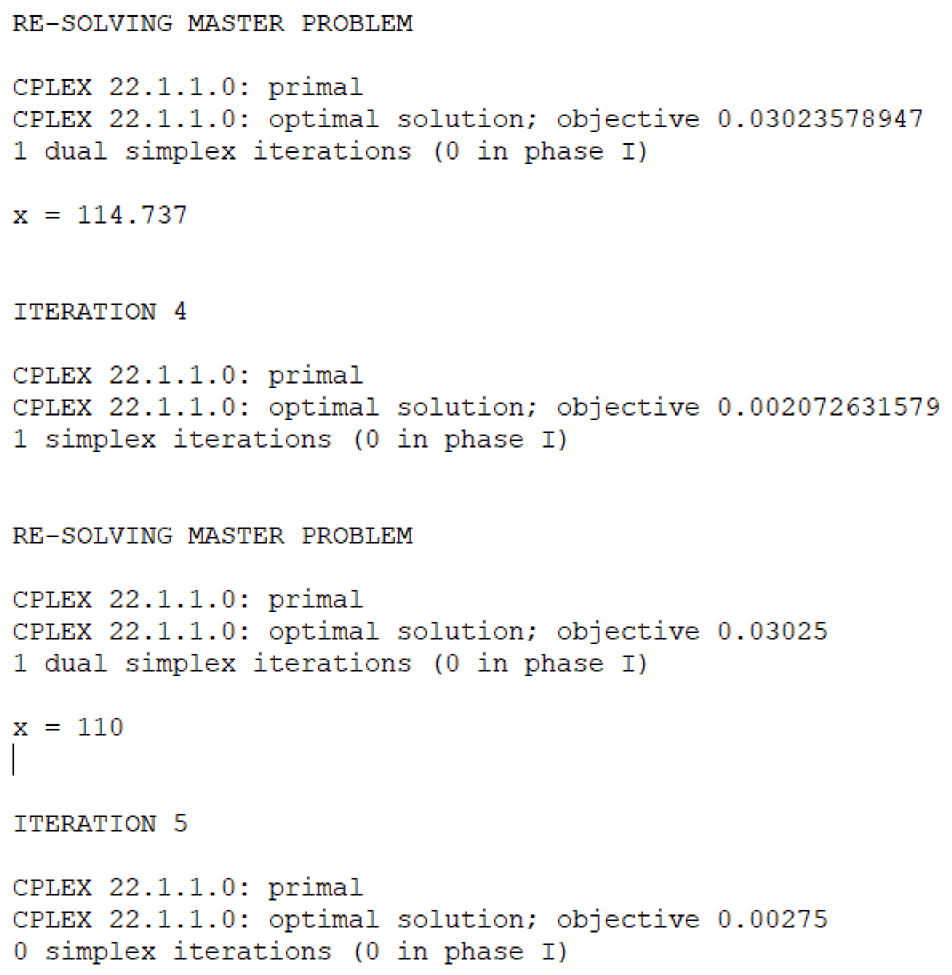
```
# ----------------------------------------
# ATM PROBLEM
# ----------------------------------------

param c ; #Unit cost of money deposited in the ATM

param q ; #Unit cost of refilling the ATM

param l ; #Technical minimum of money we have to deposit

param u ; #Maximal amount of money that can be deposited

param s;

set SCENARIOS;   # Different scenarios

param DEMAND{SCENARIOS};

var x >=l, <=u;

var y {SCENARIOS} >= 0;

minimize Total_Cost:
   c*x+
   sum {i in SCENARIOS} q*y[i]*i;

subj to Supply {i in SCENARIOS}:
   x+ y[i] >= DEMAND[i];
```

Figure 6: Implementation of the Problem in its Original Form



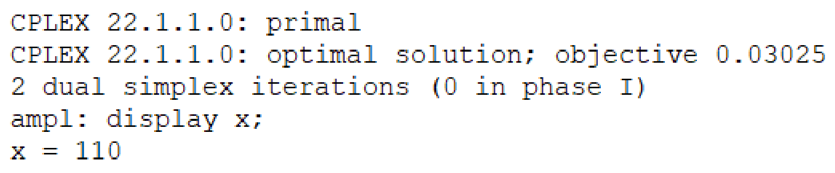
```
ITERATION 1

CPLEX 22.1.1.0: primal
CPLEX 22.1.1.0: optimal solution; objective 0.013024
0 dual simplex iterations (0 in phase I)


RE-SOLVING MASTER PROBLEM

CPLEX 22.1.1.0: primal
CPLEX 22.1.1.0: optimal solution; objective 0.019282
0 dual simplex iterations (0 in phase I)

x = 147


ITERATION 2

CPLEX 22.1.1.0: primal
CPLEX 22.1.1.0: optimal solution; objective 0.000132
3 simplex iterations (0 in phase I)


RE-SOLVING MASTER PROBLEM

CPLEX 22.1.1.0: primal
CPLEX 22.1.1.0: optimal solution; objective 0.028642
1 dual simplex iterations (0 in phase I)

x = 107


ITERATION 3

CPLEX 22.1.1.0: primal
CPLEX 22.1.1.0: optimal solution; objective 0.003509
2 simplex iterations (0 in phase I)
```

Figure 7: First Set of Results



```
RE-SOLVING MASTER PROBLEM

CPLEX 22.1.1.0: primal
CPLEX 22.1.1.0: optimal solution; objective 0.03023578947
1 dual simplex iterations (0 in phase I)

x = 114.737


ITERATION 4

CPLEX 22.1.1.0: primal
CPLEX 22.1.1.0: optimal solution; objective 0.002072631579
1 simplex iterations (0 in phase I)


RE-SOLVING MASTER PROBLEM

CPLEX 22.1.1.0: primal
CPLEX 22.1.1.0: optimal solution; objective 0.03025
1 dual simplex iterations (0 in phase I)

x = 110


ITERATION 5

CPLEX 22.1.1.0: primal
CPLEX 22.1.1.0: optimal solution; objective 0.00275
0 simplex iterations (0 in phase I)
```

Figure 8: Second Set of Results



```
CPLEX 22.1.1.0: primal
CPLEX 22.1.1.0: optimal solution; objective 0.03025
2 dual simplex iterations (0 in phase I)
ampl: display x;
x = 110
```

Figure 9: Results from the Extended Model

5 Comparison of Results

After numerours rounds of tweaking and fixing bugs, I have managed to make the results of the two approaches coincide, as they also should. The results from running my .run file are given in Figures 7 and 3.

So we need five iterations, before we can conclude that $z_R^* = z^*$ (or at least with an accuracy of 10^{-6}), and we see that the values x takes before convergence are $(147, 107, 114.737, 110, 110)$. Now, by running the extended model, we get the results presented in Figure 9.

So, the results of the two approaches coincide, as hoped. It should however be noted that the Benders approach needs some more time, but might be more effective in situations where the problem in its extensive form is more complex.

6 Conclusion

To summarize, I have in this project learned that Benders decomposition can be used to solve stochastic programs, and yield the same results as the straight forward approach we have learned in the course. Formulating the sub and master problem has however been challenging, and especially implementing the solver in AMPL, compared to solving the extensive form directly as a linear optimization problem. The justification for why we can apply Benders algorithm here is that we have an "L"-structure in the matrix that form the constraints in the extensive form, with x being present in all constraints, and since none of the second-stage variables (y) appear in more than one constraint each. It might however be easier to see the advantages of implementing Benders decomposition in even larger stochastic programs than here.