Methodology, Modelling, and Consulting Skills
School of Mathematics
The University of Edinburgh
Year: 2022/2023

THE UNIVERSITY of EDINBURGH

Computer Session 2 — Supplementary document: Multi-period and multi-index models

For the modelling of multi-period and multi-product models two Xpress Mosel constructs are important: `if`-statements and multi-index notation.

1 Conditional statements

The Mosel language has a number of features to express conditional statements. The most important one is the `if`-statement. We will use the example of the inventory balance to describe how it works.

In the lecture, we had written the constraints as follows:

- $i_1 = (i_0) + x_1 - y_1$,
- $i_t = i_{t-1} + x_t - y_t$, for all $t = 2, 3, 4$.

The important point is that the case $t = 1$ is special. Let us look at the corresponding Mosel code

```
forall(t in periods) do
        if (t>1) then
                inventory(t) = inventory(t-1) + make(t) - sell(t)
        else
                inventory(1) = make(1) - sell(1)
        end-if
end-do
```

The `if`-statement is used to distinguish between the case $t > 1$ and the other case ($t = 1$). The syntax is

```
        if (condition)
        then
        ...
        else
        ...
        end-if
```

One important point: *Xpress must be able to evaluate the condition using only information that was defined previously in the file!* That means if you use `if`, as here, to build a constraint, you can only use information that is available before you have solved the model.

A common error is to try to use `if`-conditions to depend on solution values. This is only allowed *after* the `minimize` resp. `maximize` statement, which means you cannot use solution values guide the construction of your model.

Let us look at an alternative: The `|`-operator. This operator allows us to restrict `sum` or `forall` expressions. An equivalent formulation for our inventory model is

```
inventory(1) = make(1) - sell(1)
forall(t in periods | t > 1) do
inventory(t) = inventory(t-1) + make(t) - sell(t)
end-do
```

Mosel has a number of other conditional expressions (eg `case`). If you are interested, have a look at the Mosel Language Quick Reference.

2 Multi-Index notation

In multi-period multi-product models you will encounter the need to sum or iterate not only over one but over two sets. In other contexts you will need more than two indices, but the notation we learn here generalizes to this case.

For example, you need to write the objective function for a multi-period multi-product model. Assume we have defined a set `products` and a set `periods`. We look at two parts of the objective: How to define variables for the amount of how much of each product we made and how to write the sum in the objective.

Let us first look at the variable declaration: You declare the variables as an array over both products and periods. The notation for this is

```
declarations
...
make: array(products, periods) of mpvar
...
end-declarations
```

One note about notation: Try to be consistent in the order of sets. So, here I prefer first `products` then `periods`. That means, whenever I have multiple indices `products` should come before `periods`.

What about the sum in the objective? We need to sum over all products and over all periods. The preferred way of writing such a double sum is

```
sum(p in products, t in periods) cost(p) * make(p,t)
```

It is, however, also allowed to write

```
sum(p in products) sum(t in periods) cost(p) * make(p,t)
```

The same rules apply also to arrays of other types and to `forall` constructs.

We will learn how to model with Xpress the production planning problems that we have studied in Lecture 2. We will not just model them but also, for each of the models, we will write a general code that can solve problems with the same structure but different data. The guide of this week is deliberately shorter so that you have time in class to work on Problem Set 2.