NATCOR - Modeling with Xpress

The software that will be used in this course will be Xpress, a powerful solver for both linear and non-linear problems. We will use the Mosel language with IVE, an interface that is very friendly to use and very easy to learn. Although during the course we will use the full version of the software, there is a free student version with limited numbers of constraints and variables.

1. Our first model

Let us consider the following blending problem: A chemical firm can produce two products, A and B, from combining three elements: iron, lead, and tin. What is the blending that maximizes the benefit? Assume that all the production is sold.

Resources	Units used per kg of product		Available units
	Product A	Product B	
Iron	7	4	56
Lead	3	5	45
Tin	4	3	48
Net benefit (£/kg)	10	8	

First, we define the **decision variables:**

- $x_1 =$ "kilograms of product A".
- $x_2 =$ "kilograms of product B".

Second, we state the **objective function**: $10x_1 + 8x_2$.
Finally, we write the constraints:

- **Constraint on the iron availability:** $7x_1 + 4x_2 \leq 56$.
- **Constraint on the lead availability:** $3x_1 + 5x_2 \leq 45$.
- **Constraint on the tin availability:** $4x_1 + 3x_2 \leq 48$.
- **Variable non-negativity:** $x_1, x_2 \geq 0$.

Therefore, the model is:

$$\begin{cases} \text{Max.} & 10x_1 + 8x_2 & & \\ \text{s.t.} & 7x_1 + 4x_2 & \leq & 56, \\ & 3x_1 + 5x_2 & \leq & 45, \\ & 4x_1 + 3x_2 & \leq & 48, \\ & x_1, x_2 & \geq & 0. \end{cases}$$

Let us see now how to write it and solve it with Xpress.

2. Writing our first model

When we start using Xpress, the screen will look like in Figure 1.

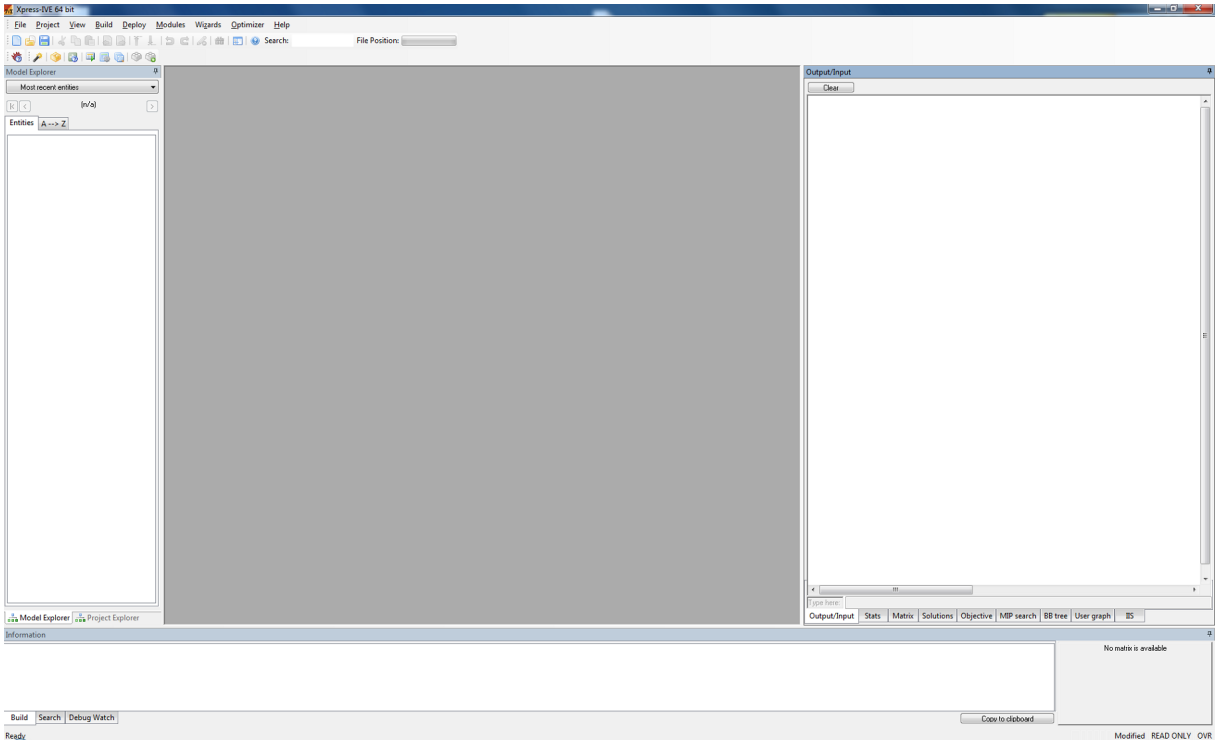

Figure 1: First screen.

Once we create a new file (File → New), we start to write the model.

1. First, there is a line giving a name to the model:

   ```
   model blending
   ```

   We have given it the name blending, but we can use any name we like.

2. Now, we add a line that calls the solver:

   ```
   uses "mmxprs"
   ```

3. Next, we declare the variables of the model:

   ```
   declarations
       x1,x2:mpvar
   end-declarations
   ```

   A decision variable is an object of type `mpvar`.

4. Now, we write the objective function. We have also added a comment line (begins with the symbol "!") which is not read at all, but it helps to have the code better organized.

```
! Objective function
obj:= 10*x1 + 8*x2
```

5. Then, we define the constraints, one per line.

   ```
   ! Constraints
   7*x1 + 4*x2 <= 56
   3*x1 + 5*x2 <= 45
   4*x1 + 3*x2 <= 48
   ```

   Note that we write the multiplication symbol "*" between the coefficient and the variable (unless it is 0 or 1, in which case we can omit the coefficient and the multiplication symbol). Remember also that the constraints that we will use are only of the type "$\leq$", "$\geq$", and "=". We will *never* have inequalities with "$<$" or "$>$".

   An important remark is that we do not need to add explicitly the non-negativity constraints. Unless stated otherwise, Xpress assumes that all the variables are non-negative and continuous.

6. Finally, we tell the solver that we are going to maximize the objective function:

   ```
   maximize(obj)
   ```

7. The last line says that the model is completed:

   ```
   end-model
   ```

This model can be compiled (which automatically saves the file) with F7. Or we can run it with F6 (which saves and compiles before that). Alternatively, you can use the buttons shown in Figure 2.

Once we have run the model, we can see some relevant information in the output window. Figure 3 shows one of the several tabs. In this one, you can see the algorithm used (simplex dual), the number of iterations, the final objective value, the status (the solution is optimal) and the time used to solve the model. In other tabs you can find even more information (value of the decision variables, slack of each constraint, dual variables, etc.).

Actually, we may prefer to print some of these values in the main output window. Add the following lines before the last line of the model and run the model again:

```
writeln("x1: ",getsol(x1))
writeln("x2: ",getsol(x2))
writeln("Objective value: ",getobjval)
```

3. The blending problem with general data

The model we have written has a serious deficiency: if we change the data, we will have to modify individually each entry, which can be highly time consuming for problems with many variables and constraints. Therefore, we are going to use arrays and read the data from files so that we can solve *any problem with the same structure*.

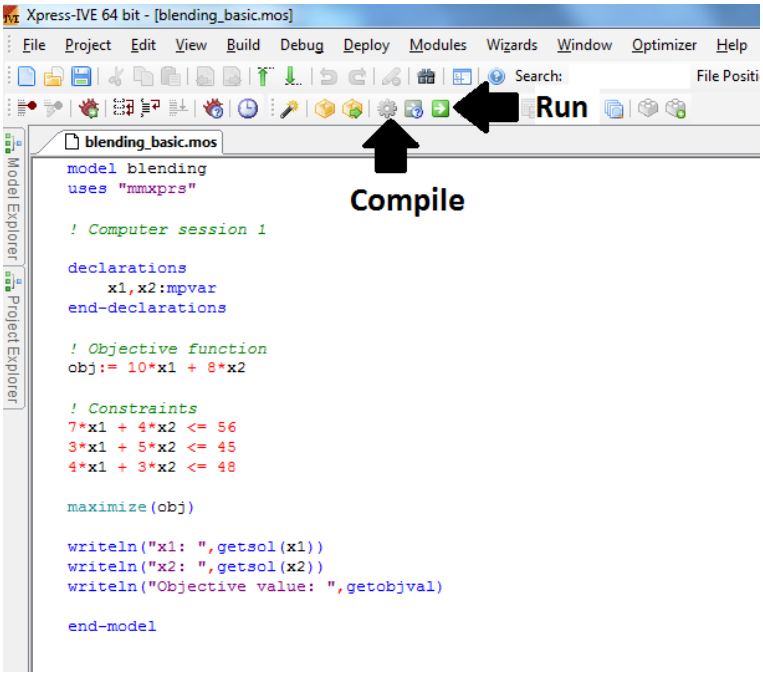

Figure 2: Blending model.

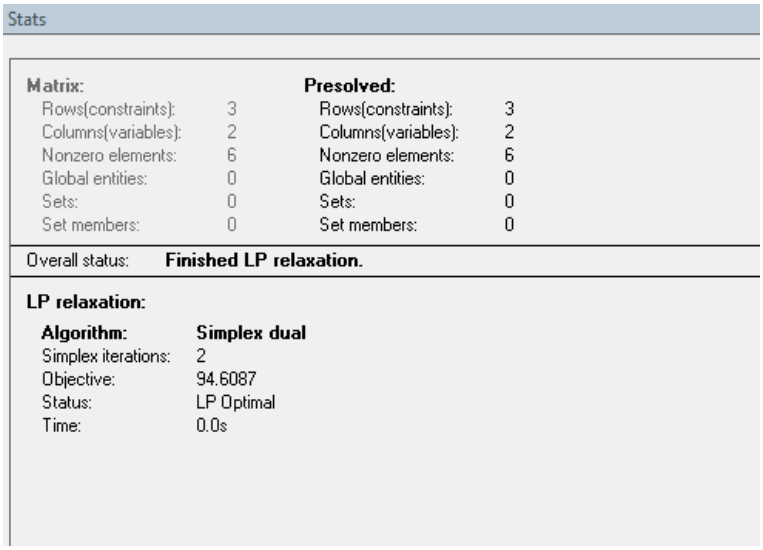

Figure 3: Output for the blending model.

3.1 Using sets

A first approach is to define sets of indices. We replace the previous declarations block with:

```
declarations
    products = 1..2
    resources = 1..3
    benefit: array(products) of real
    limit: array(resources) of real
    use:array(resources,products) of real
```

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

We have defined two sets of indices: products, which has two elements, and resources, which has three. Then, we define several unidimensional arrays. We also define a bidimensional matrix (use). But this only provides information about the dimensions. Next, we fill them with the data:

```
benefit:: [10,8]
limit:: [56,45,48]
use:: [7,4,3,5,4,3]
```

Now, we have to rewrite the model taking into account that all the elements are indexed with the previous sets. The objective function is:

```
total_benefit:= sum(p in products) benefit(p)*make(p)
```

And the constraints are written as follows:

```
forall(r in resources) do
    sum(p in products) use(r,p)*make(p) <= limit(r)
end-do
```

Finally, the code displaying the solution needs to be modified too:

```
forall(p in products) do
    writeln("Amount produced of product ",p,": ",getsol(make(p))," kgs.")
end-do
writeln
writeln("The net benefit is £",getobjval,".")
```

Figure 4 shows the full code.

3.2 Initializing from files

Instead of having the data explicitly in the code, there is the option of having external files and reading this information from them. In order to so, we need first to have a file blending.dat where all the information is stored. The file will look like this:

```
benefit: [10 8]
limit: [56 45 48]
use: [7 4 3 5 4 3]
```

And we need to replace the lines defining the values of the arrays with the following code:

```
initializations from "blending.dat"
    benefit limit use
end-initializations
```

Moreover, we can use sets of strings to give names to the indices:

blending_sets.mos

```
model blending
uses "mmxprs"

! Computer session 1

declarations
    products = 1..2
    resources = 1..3
    benefit: array(products) of real
    limit: array(resources) of real
    use:array(resources,products) of real
    make: array(products) of mpvar
end-declarations

! Initialize values
benefit:: [10,8]
limit:: [56,45,48]
use:: [7,4,3,5,4,3]

! Objective function
total_benefit:= sum(p in products) benefit(p)*make(p)

! Constraints
forall(r in resources) do
    sum(p in products) use(r,p)*make(p) <= limit(r)
end-do

maximize(total_benefit)

forall(p in products) do
    writeln("Amount produced of product ",p,": ",getsol(make(p)),".")
end-do
writeln
writeln("The net benefit is ",getobjval,".")

end-model
```

Figure 4: Blending model using sets.

```
declarations
    products, resources: set of string
    benefit: array(products) of real
    limit: array(resources) of real
    use:array(resources,products) of real
    make: array(products) of mpvar
end-declarations

! Initialize values
initializations from "blending2.dat"
    products resources benefit limit use
end-initializations
```

File blending2.dat is file blending.dat with two new lines:

```
products: ["A" "B"]
resources: ["iron" "lead" "tin"]
```

You can see the full code in Figure 5.

blending_file2.mos

```
model blending
uses "mmxprs"

declarations
    products, resources: set of string
    benefit: array(products) of real
    limit: array(resources) of real
    use:array(resources,products) of real
    make: array(products) of mpvar
end-declarations

! Initialize values
initializations from "blending2.dat"
    products resources benefit limit use
end-initializations

! Objective function
total_benefit:= sum(p in products) benefit(p)*make(p)

! Constraints
forall(r in resources) do
    sum(p in products) use(r,p)*make(p) <= limit(r)
end-do

maximize(total_benefit)

forall(p in products) do
    writeln("Amount produced of product ",p,": ",getsol(make(p))," kgs.")
end-do
writeln
writeln("The net benefit is £",getobjval,".")

end-model
```

Figure 5: Blending model using sets.

4. Multi-period models and inventory

The blending problem that we have studied is *single-period*. However, production plans usually have a *multi-period* horizon (for example, year divided in months) because the decisions at one period have consequences on later decisions. We are going to see now how to model this with Xpress.

Let us consider the following problem: Sailco Corporation must decide how many sailboats and surfboards to produce at each quarter. The demand is:

Quarter	Spring	Summer	Autumn	Winter
Sailboats demand	40	60	75	25
Surfboards demands	190	350	130	20

Each sailboat needs 20 hours of work and each surfboard needs 3 hours. Sailco have 1860 hours of work available per quarter. Besides, the company has a warehouse for which the cost of storing from one quarter to the next is £50 per sailboat and £2 per surfboard. If the cost of producing one boat is £400 and the cost of producing a surfboard is £35, what is the production planning that meets the demand of the whole year at minimum cost? Assume that what is produced in one quarter can be used to meet the demand on that quarter.

In order to model the problem, we we will use the index p for the product (1 for sailboats and 2 for surfboards) and the index t for the period (1 for spring, 2 for summer, and so on). Next, we define the following variables:

- x_{pt} = "units of product p made on quarter $p = 1, 2$, $t = 1, 2, 3, 4$.
- y_{pt} = "units of product p sold on quarter $p = 1, 2$, $t = 1, 2, 3, 4$.
- i_{pt} = "units of product p in inventory at the end of quarter $p = 1, 2$, $t = 1, 2, 3, 4$.

Now, we write the different parts of the objective function:

- The production cost for sailboats is: $400x_{11} + 400x_{12} + 400x_{13} + 400x_{14}$.
- The production cost for surfboards is: $35x_{21} + 35x_{22} + 35x_{23} + 35x_{24}$.
- The holding cost for sailboats is: $50i_{11} + 50i_{12} + 50i_{13} + 50i_{14}$.
- The holding cost for surfboards is: $2i_{21} + 2i_{22} + 2i_{23} + 2i_{14}$.

Next, we start to write the constraints:

- The constraints that limit the hours of work are: $20x_{1t} + 3x_{2t} \leq 1860,\ t = 1, 2, 3, 4$.
- The demand requirements are: $y_{11} \geq 40$, $y_{12} \geq 60$, $y_{13} \geq 75$, $y_{14} \geq 25$, $y_{21} \geq 190$, $y_{22} \geq 350$, $y_{23} \geq 130$, $y_{24} \geq 20$.

The inventory balance constraints are:

- Sailboats, quarter 1: $i_{11} = x_{11} - y_{11}$.
- Sailboats, quarter t, $t = 2, 3, 4$: $i_{1t} = i_{1,t-1} + x_{1t} - y_{1t}$.
- Surfboards, quarter 1: $i_{21} = x_{21} - y_{21}$.
- Surfboards, quarter t, $t = 2, 3, 4$: $i_{2t} = i_{2,t-1} + x_{2t} - y_{2t}$.

Therefore, the full model is:

$$
\begin{array}{lll}
\text{Min.} & \displaystyle\sum_{t=1}^{4} 400x_{1t} + \sum_{t=1}^{4} 35x_{2t} + \sum_{t=1}^{4} 50i_{1t} + \sum_{t=1}^{4} 2i_{2t} & \\
\text{s.t.} & 20x_{1t} + 3x_{2t} \leq 1860, & t = 1, 2, 3, 4, \\
& y_{11} \geq 40,\ y_{12} \geq 60, & \\
& y_{13} \geq 75,\ y_{14} \geq 25, & \\
& y_{21} \geq 190,\ y_{22} \geq 350, & \\
& y_{23} \geq 130,\ y_{24} \geq 20, & \\
& i_{p1} = x_{p1} - y_{p1}, & p = 1, 2, \\
& i_{pt} = i_{p,t-1} + x_{pt} - y_{pt}, & p = 1, 2,\ t = 2, 3, 4, \\
& i_{pt}, x_{pt}, y_{pt} \geq 0, & p = 1, 2,\ t = 1, 2, 3, 4.
\end{array}
$$

Now we show how to write the model with Xpress. As we have seen before, first we need to define indices and structures at the declarations block:

```
model sailboat_inventory_multiproduct
uses "mmxprs"

declarations
number_of_periods = 4
periods = 1..number_of_periods
period_names: array(periods) of string
number_of_products = 2
products = 1..number_of_products
product_names: array(products) of string
demand: array(products,periods) of real
cost, holding_cost, hours_needed: array(products) of real
production_limit:real
make,sell,inventory: array(products,periods) of mpvar
end-declarations

initialisations from "sailboat_inventory_multiproduct.dat"
period_names product_names demand cost production_limit holding_cost
    hours_needed
end-initialisations
```

periods is the set of indices of size number_of_periods and the names of these periods are stored in array period_names. The meaning of the rest of the structures is obvious. We create file sailboat_inventory_multiproduct.dat with the following information:

```
period_names: ["Spring" "Summer" "Autumn" "Winter"]
product_names: ["Sailboats" "Surfboards"]
cost: [400 35]
holding_cost: [50 2]
production_limit: 1860
demand: [40 60 75 25 190 350 130 20]
hours_needed: [20 3]
```

Defining the objective function and the constraints is trivial:

```
! Objective function
total_cost:= sum(p in products, t in periods) cost(p)*make(p,t) +
 sum(p in products, t in periods) holding_cost(p)*inventory(p,t)

! Constraints

! Production limit
forall(t in periods) sum(p in products) hours_needed(p)*make(p,t) <=
 production_limit

! Demand satisfaction
forall(p in products, t in periods) sell(p,t) >= demand(p,t)

! Inventory balance
forall(p in products, t in periods) do
if (t>1) then
```

```
inventory(p,t) = inventory(p,t-1) + make(p,t) - sell(p,t)
else
inventory(p,1) = make(p,1) - sell(p,1)
end-if
end-do
```

Finally, we solve the model and display the solution information:

```
minimize(total_cost)

forall(t in periods) do
writeln("In ", period_names(t),":")
forall(p in products) do
write("  * ",product_names(p),": make ",getsol(make(p,t))," and sell ",
getsol(sell(p,t)),".")
if(getsol(inventory(p,t))>0) then
writeln(" Inventory level at the end of the quarter: ",getsol(inventory(p,t)))
else
writeln
end-if
end-do
writeln
end-do

end-model
```

5. Final remarks

Some other good practices are:

- Divide the model clearly in blocks.
- Add comments that give you some information about the blocks.
- Use meaningful names.
- Backup your files.