Julia Compendium

Mathias Milo Hauge

Based on Julia v 1.6.2 (Found by typing VERSION in Julias "REPL" i.e. the console/command line)

1 General Julia

1.1 Julia variables: Constants, vectors and matrices

1.1.1 Defining constants

Defining a constant with a value:

```
a = 2
```

$$\longrightarrow \quad a = 2$$

Defining a constant with an expression:

```
b = 2*4 + 3
```

$$\longrightarrow \quad b = 11$$

Defining a constant from other variables:

```
c = 2*a - b
```

$$\longrightarrow \quad c = -7$$

1.1.2 Defining vectors and matrices

Defining a row vector (separate values with spaces):

```
vr = [1 2 3]
```

$$\longrightarrow \quad vr = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Defining a column vector (separate values with commas):

```
vc = [1,2,3]
```

$$\longrightarrow \quad vc = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Defining a 2D matrix with 2 rows and 3 columns (separate values in rows with spaces and separate rows with semicolons):

```
m = [1 2 3 ; 4 5 6]
```

$$\longrightarrow \quad m = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Defining a vector/matrix with r rows and c columns with all entries equal to 0:

```
r = 2
c = 3
m0 = zeros(r,c)
```

$$\longrightarrow \quad m0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Defining a vector/matrix with r rows and c columns with all entries equal to 1:

```
r = 2
c = 3
m1 = ones(r,c)
```

$$\longrightarrow \quad m1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

1.2 Changing existing variables

Once you have defined your constant, vector or matrix, you might want to change its value or some of its values.

1.2.1 Constants

When changing constants you can both change the value to something entirely new, or you can modify the existing value using arithmetic.

Changing the value of a constant:

```
a = 2
a = 5
```

$$\longrightarrow \quad a = 5$$

Modifying the value of a constant with algebra:

```
a = 2
a = (a*4 + 1)/3
```

$$\longrightarrow \quad a = 3$$

Modifying the value of a constant with a single simple arithmetic operation:

```
a = 2
a += 1
```

$$\longrightarrow \quad a = 3$$

```
a *= 2
a -= 4
a /= 2
```

$$\longrightarrow \quad a = 1$$

1.2.2 Vectors and matrices

Adding an element to the end of a column vector:

```
vc = [1,2,3]
push!(v,4)
```

$$\longrightarrow \quad vc = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Adding columns to the end of a matrix or an element to the end of a row vector:

```
vr = [1 2 3]
m = [1 2 3 ; 4 5 6]
vr = hcat(vr,4)
m = hcat(m,[7,8])
```

$$\longrightarrow \quad vr = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \qquad m = \begin{bmatrix} 1 & 2 & 3 & 7 \\ 4 & 5 & 6 & 8 \end{bmatrix}$$

Adding a row at the bottom of a matrix:

```
m = [1 2 3 ; 4 5 6]
m = vcat(m,[7,8,9])
```

$$\longrightarrow \quad m = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Accessing positions in vectors and matrices and modifying the values on the positions:

```
m = [1 2 3 ; 4 5 6]
m[2,3] = 0
m[1,1] *= 10
```

$$\longrightarrow \quad m = \begin{bmatrix} 10 & 2 & 3 \\ 4 & 5 & 0 \end{bmatrix}$$

Accessing an entire row (or column) and modifying the values in the same way (Add "." before the operation):

```
m = [1 2 3 ; 4 5 6]
m[1,:] .*= 10
m[2,:] .= -1
```

$$\longrightarrow \quad m = \begin{bmatrix} 10 & 20 & 30 \\ -1 & -1 & -1 \end{bmatrix}$$

1.3 Loops, ifs and sums

When writing loops and if-statements, remember to end it with "end".

1.3.1 Loops

A for loop with $i = 1, \ldots, n$:

```
for i = 1:n
    x[i] = 1
end
```

Nested for loops with $i = 1, \ldots, n$ and $j = i, \ldots, n$:

```
for i = 1:n
    for j = i:n
        x[i,j] = 1
    end
end
```

A for loop iterating over the elements of a vector:

```
v = [1 2 3]
for i in v
    print(i)
end
```

A while loop:

```
i = 0
while i <= 10
    print(i)
    i += 1
end
```

1.3.2 If-statements

A general if statement:

```
if EXPR
    do something
elseif EXPR2
    do something else
else
    do something third
end
```

EXPR and EXPR2 can be any logical expression resulting in True or False. E.g.

```
a == b
a <= 6
a in V
(a == b) and (b <= 2)
```

These are only a small number of examples of logical expressions.

1.3.3 Sums

A simple sum of the elements in a vector with length n, $\sum_{i=1}^{n} v_i$:

```
v = [1 2 3]
sum(v[i] for i = 1:n)
```

or

```
sum(i for i in v)
```

A double sum for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, $\sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j}$:

```
sum(x[i,j] for i = 1:n, j = 1:m)
```

Adding conditions to the sums is possible. I.e.:

```
sum(x[i,j] for i = 1:n, j = 1:m if i <= 2*j)
```

2 JuMP optimization modelling

Any parameters and constants you need in the model can be defined as described earlier.

2.1 Packages

For the optimization model we need the JuMP package and a solver (GLPK in our case).

```
using JuMP, GLPK
```

or

```
using JuMP
using GLPK
```

2.2 Model

To define the model itself you write:

```
ModelName = Model(GLPK.Optimizer)
```

The entire model is now created as the julia variable ModelName.

2.3 Variables

You then create the variables you need for your model. When creating variables you can specify an upper bound, a lower bound and whether it is continuous, integer or binary. Variables are continuous by default.

Creating a single continuous variable x without any bounds.:

```
@variable(ModelName, x)
```

Creating a single continuous variable x with a lower bound of 0 and an upper bound of 5:

```
@variable(ModelName, 0 <= x <= 5)
```

Creating n positive integer variables x_i for $i = 1, \ldots, n$:

```
@variable(ModelName, x[1:n] >= 0, Int)
```

or

```
@variable(ModelName, x[i=1:n] >= 0, Int)
```

Creating n*m binary variables x_{ij} for $i = 1, \ldots, n$, $j = 1, \ldots, m$:

```
@variable(ModelName, y[1:n,1:m], Bin)
```

2.4 Objective

Having defined the variables, you can now define the objective function for your model.

Simple objective function considering minimization:

```
@variable(ModelName, x[1:4] >= 0, Int)
@objective(ModelName, Min, x[1] + 2*x[2] + 25*x[3] - 2*x[4])
```

or using a sum as previously described:

```
w = [1 2 25 -2]
@variable(ModelName, x[1:4] >= 0, Int)
@objective(ModelName, Min, sum(w[i]*x[i] for i = 1:4))
```

Objective function maximizing the sum of the defined z variables:

```
@variable(ModelName, z[1:4,1:6] >= 0, Int)
@objective(ModelName, Max, sum(z[i,j] for i = 1:4, j = 1:6))
```

Note that Min/Max should be with capital M!

2.5 Constraints

You now add constraints to your model.

The general form of constraints is:

```
@constraint(ModelName, [i=...,j=...;COND], a SIGN b)
```

where you specify for instance "$\forall i = 1,\ldots,5, \forall j = 1,\ldots,3$" in the square brackets as $[i = 1:5, j = 1:3]$ i.e. separated by comma. If there are any conditional logical expressions they are added after a semicolon as "COND". In the constraint expression "a SIGN b" you can use <=, >= and ==.

The constraint $\sum_{j=1}^{m} x_{i,j} \leq y_i \quad \forall i = 1,\ldots,n$:

```
@constraint(ModelName, [i=1:n], sum(x[i,j] for j = 1:m) <= y[i])
```

The constraint $x_{i,j} \geq y_i \quad \forall i = 1,\ldots,n, \forall j = 1,\ldots,m; j \neq i$:

```
@constraint(ModelName, [i=1:n,j=1:m; j != i], x[i,j] >= y[i])
```

The constraint $\sum_{i=1}^{n}\sum_{j=1}^{m} x_{i,j} = \sum_{i=1}^{n} y_i$:

```
@constraint(ModelName, sum(x[i,j] for i = 1:n, j = 1:m) == sum(y[i] for i = 1:n))
```

2.6 Solving

Once the model has been constructed you solve it with JuMP:

```
JuMP.optimize!(ModelName)
```

2.7 Output

To print the objective value of the problem:

```
println(JuMP.objective_value(ModelName))
```

To print the values of some decision variables x_i:

```
println(JuMP.value.(x))
```

or

```
for i = 1:n
    println(JuMP.value(x[i]))
end
```

One way to print a 2D variable:

```
for i = 1:n
    for j = 1:m
        print(JuMP.value(x[i,j]),"  ")
    end
    println()
end
```