Diet Problem Julia/JuMP Walk-Through

Julia Version 0.4.6, JuMP Version 0.13.2

Problem Setup and Formulation

Consider the problem of diet optimization based on cost and different nutritional factors.

Find a minimum-cost diet that contains:

- At least 500 calories
- At least 6 grams of chocolate
- At least 10 grams of sugar
- At least 8 grams of fat

The nutritional information and cost of each food item is shown below.

	Brownies	Ice Cream	Cola	Cheese Cake
Calories	400	200	150	500
Chocolate	3	2	0	0
Sugar	2	2	4	4
Fat	2	4	1	5
Cost	$0.50	$0.20	$0.30	$0.80

Problem Formulation:

Decision Variables

$x_{brownies}$	units of Brownies to consume
$x_{ice\ cream}$	units of Ice Cream to consume
x_{cola}	units of Cola to consume
$x_{cheese\ cake}$	units of Cheese Cake to consume

Constraints

$400x_{brownies} + 200x_{ice\ cream} + 150x_{cola} + 500x_{cheese\ cake} \geq 500$	At least 500 calories
$3x_{brownies} + 2x_{ice\ cream} \geq 6$	At least 6 grams of chocolate
$2x_{brownies} + 2x_{ice\ cream} + 4x_{cola} + 4x_{cheese\ cake} \geq 10$	At least 10 grams of sugar
$2x_{brownies} + 4x_{ice\ cream} + 1x_{cola} + 5x_{cheese\ cake} \geq 8$	At least 8 grams of fat

Objective Function

Minimize the total cost of the units of food to consume

$$Min \quad 0.5x_{brownies} + 0.2x_{ice\ cream} + 0.3x_{cola} + 0.8x_{cheese\ cake}$$

Julia/JuMP Walk-Through

Model Setup

Begin by ***specifying the packages and/or solver*** that you will be using in your Julia program. In this class, begin by calling the package JuMP.

```
In [ ]: using JuMP
```

Next, ***define and name your model***. For our Diet Problem example, we will define a base model named **mymodel** that uses the default solver.

```
In [ ]: using JuMP

        mymodel = Model()
```

Data Setup

For many optimization problems, you will be given starting information or data. Before you can define decision variables and constraints, you will need to ***create the data*** in Julia. Data can be stored in arrays (lists), matrices, tuples, or dictionaries, etc.

In our Diet Problem example, we choose to store information about the different foods in a 1D-array.

```
# Foods available
foods = ["brownies", "ice cream", "cola", "cheese cake"]
```

Decision Variables

After setting up the starting data for our LP, we can now ***define our decision variables*** using the **`@variable`** macro. You must specify the model, the decision variable name and indices (if applicable), any upper and lower bounds, and other variable classifications (binary/integer).

Below we define four decision variables. The decision variables are named **x** with one index. We specify the indices by writing the name of the decision variable followed by brackets containing the array of indices (which is the array of different foods). We also specify that the decision variables must be non-negative.

Decision Variables: $x_{brownies}$ $x_{ice\ cream}$ x_{cola} $x_{cheese\ cake}$

```
# Define decision variables
@variable(mymodel, x[foods] >= 0)
```

Constraints

To ***add constraints***, use the **`@constraint`** macro. You must specify the model name, followed by the constraint expression. Below we add the following constraints to our Julia program.

$400x_{brownies} + 200x_{ice\ cream} + 150x_{cola} + 500x_{cheese\ cake} \geq 500$	At least 500 calories
$3x_{brownies} + 2x_{ice\ cream} \geq 6$	At least 6 grams of chocolate
$2x_{brownies} + 2x_{ice\ cream} + 4x_{cola} + 4x_{cheese\ cake} \geq 10$	At least 10 grams of sugar
$2x_{brownies} + 4x_{ice\ cream} + 1x_{cola} + 5x_{cheese\ cake} \geq 8$	At least 8 grams of fat

```
# Constraints
# At least 500 calories
@constraint(mymodel, 400x["brownies"] + 200x["ice cream"] + 150x["cola"] + 500x["cheese cake"] >= 500)

# At least 6 grams of chocolate
@constraint(mymodel, 3x["brownies"] + 2x["ice cream"] >= 6)

# At least 10 grams of sugar
@constraint(m, 2x["brownies"] + 2x["ice cream"] + 4x["cola"] + 4x["cheese cake"] >= 10)

# At least 8 grams of fat
@constraint(m, 2x["brownies"] + 4x["ice cream"] + 1x["cola"] + 5x["cheese cake"] >= 8)
```

Objective Function

To add an ***objective function***, use the **`@objective`** macro. You must specify the model name, whether it is **`Min`** or **`Max`**, and the objective function expression. Below we add the objective that we want to minimize the total cost of our food.

$$Min \quad 0.5x_{brownies} + 0.2x_{ice\ cream} + 0.3x_{cola} + 0.8x_{cheese\ cake}$$

```
@objective(mymodel, Min, 0.5x["brownies"] + 0.2x["ice cream"] + 0.3x["cola"] + 0.8x["cheese cake"])
```

Solving the Model and Model Output

To ***solve the model***, use the **`solve()`** function and write the name of the model you wish to solve within the parentheses.

```
# Solve the optimization problem
solve(mymodel)
```

There are various ways to ***print the model's output***. Below are a few examples of how to print model output.

To print all decision variable values and objective value. Here we use the functions **`getvalue()`** and **`getobjectivevalue()`** where in the parentheses we write the decision variable and model respectively.

```
# Print the consumption amounts
println("variable values: ", getvalue(x))

# Print the objective value
println("Objetive value: ", getobjectivevalue(mymodel))
```

```
variable values: x: 1 dimensions:
[   brownies] = 0.0
[  ice cream] = 3.0000000000000004
[       cola] = 0.9999999999999998
[cheese cake] = 0.0

Objetive value: 0.9
```

To print only the decision variable values that are non-zero and the final objective value. Here we use a **`for`** loop and a nested **`if`** statement within the for loop.

```
# Print the consumption amounts
for f in foods                                  # Iterate through each index (food)
    if getvalue(x[f]) != 0.0                    # If the value of the decision variable isn't 0
        println(f, " = ", getvalue(x[f]))       # Print the decision variable and its value
    end
end

# Print the objective value
println("Objetive value: ", getobjectivevalue(mymodel))
```

```
ice cream = 3.0000000000000004
cola = 0.9999999999999998
Objetive value: 0.9
```

Full Julia Code and Output for the Diet Problem

Note that the full .ipynb file can be found on Stellar.

In [15]:

```
# Initialize JuMP
using JuMP

# Define model
mymodel = Model()

# Create a foods available array
foods = ["brownies", "ice cream", "cola", "cheese cake"]

# Define decision variables, non-negative
@variable(mymodel, x[foods] >= 0)

# Constraints
# At least 500 calories
@constraint(mymodel, 400x["brownies"] + 200x["ice cream"] + 150x["cola"] + 500x["cheese cake"] >= 500)

# At least 6 grams of chocolate
@constraint(mymodel, 3x["brownies"] + 2x["ice cream"] >= 6)

# At least 10 grams of sugar
@constraint(mymodel, 2x["brownies"] + 2x["ice cream"] + 4x["cola"] + 4x["cheese cake"] >= 10)

# At least 8 grams of fat
@constraint(mymodel, 2x["brownies"] + 4x["ice cream"] + 1x["cola"] + 5x["cheese cake"] >= 8)

# Set objective function
@objective(mymodel, Min, 0.5x["brownies"] + 0.2x["ice cream"] + 0.3x["cola"] + 0.8x["cheese cake"])

# Solve the optimization problem
solve(mymodel)

# Print the consumption amounts
println("Variable values: ", getvalue(x))

# Print the objective value
println("Objetive value: ", getobjectivevalue(mymodel))
```

```
Variable values: x: 1 dimensions:
[   brownies] = 0.0
[  ice cream] = 3.0000000000000004
[       cola] = 0.9999999999999998
[cheese cake] = 0.0

Objetive value: 0.9
```