Tutorial 2

Topic 1: Wagner-Whitin (dynamic programming) problem

Input Arguments:

```
K       = fixed cost [scalar or vector]
h       = holding cost per item per period [scalar or vector]
T       = number of periods in horizon [scalar or vector]
d       = demand in each period [vector or matrix]
I0      = initial inventory [scalar or vector]
```

Output Arguments:

```
Q       = optimal order quantities in each period [vector or matrix]
optcost = optimal cost [scalar or vector]
```

```
WW_DP (generic function with 1 method)
```

```
begin
    function WW_DP(K, h, T, d, I0)

        # how many sub-problem do you have
        m = length(K)

        # create the vector to store output
        Q = zeros(m, maximum(T))
        optcost = zeros(m, 1)

        for r in 1:m # iterate through all the sub-problem

            # If your initial inventory is not 0
            # => use these inventories to fulfill demand first
            # => stop the loop when initial inventory are all ordered
            # => for example, I0 = 500, d = [100, 200, 300, 400]
            # => your while loop should stop at period 3!

            # tt = store the period you will run out of inventory
            tt = 1
            while I0[r] > 0
                # do comparison first,
                # amt = the demand you can fulfill at period tt without further
order
                amt = min(I0[r], d[r,tt])

                d[r,tt] = d[r,tt] - amt
                I0[r] = I0[r] - amt

                tt = tt + 1
            end

            # store the period which the optimal cost can be realized
            opts = zeros(Int64, 1, maximum(T))
            # store cost calculated cost for each period
            theta = zeros(1, maximum(T)+1)

```

```
            # you need to order more stock after you run out of inventory
            # backward induction
            for tt in T[r]:-1:1
                theta[tt] = 1.0e300

                # cost = fixed cost + holding cost + stock cost
                for s in (tt+1):(T[r]+1)
                    tempcost = K[r] # fixed cost

                    for i in tt:(s-1)
                        tempcost = tempcost + h[r] * (i - tt) * d[r,i] # holding cost
                    end

                    tempcost = tempcost + theta[s]

                    if tempcost < theta[tt]
                        theta[tt] = tempcost
                        opts[tt] = s
                    end

                end

            end

            tt = 1
            while d[r,tt] == 0
                tt = tt + 1
            end

            optcost[r] = theta[tt]
            while tt < (T[r]+1)
                Q[r,tt] = sum( d[ r, tt:(opts[tt]-1) ] );
                tt = opts[tt]
            end

        end

        return Q, optcost
    end
end
```

```
Example 1: without initial inventory
```

```
(1×4 Matrix{Float64}:   ,  1×1 Matrix{Float64}:)
  210.0  0.0  150.0  0.0    1380.0
```

```
begin
    K_1 = 500
    h_1 = 2
    T_1 = 4
    I0_1 = 0
    d_1 = [90 120 80 70]

    Q_1, optcost1 = WW_DP(K_1, h_1, T_1, d_1, I0_1)
end
```

```
Example 2: having initial inventory
```

```
(1×4 Matrix{Float64}: ,  1×1 Matrix{Float64}:)
  0.0  160.0  0.0  0.0    940.0
```

```
begin
    K_2 = 500
    h_2 = 2
    T_2 = 4
    I0_2 = [200]
    d_2 = [90 120 80 70]

    Q_2, optcost2 = WW_DP(K_2, h_2, T_2, d_2, I0_2)
end
```

Topic 2: Linear Programming and Integer Programming

```
(1) Linear programming
```

$$max x_1 + 2x_2 + 5x_3$$

$$s.t. -x_1 + x_2 + 3x_3 \leq -5$$

$$x_1 + 3x_2 - 7x_3 \leq 10$$

$$0 \leq x_1 \leq 10$$

$$x_2 \geq 0$$

$$x_3 \geq 0$$

Step 1: Import package

```
using JuMP, GLPK
```

Step 2: Preparing the optimization model

```
m = A JuMP Model
    Feasibility problem with:
    Variables: 0
    Model mode: AUTOMATIC
    CachingOptimizer state: EMPTY_OPTIMIZER
    Solver name: GLPK
```

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

Step 3: Declaring decision variables

x3

```
begin
```

```
•     md"""   Rule: @variable(model's name, decision variables' names and its feasible
  region)"""
•     @variable(m, 0<= x1 <=10)
•     @variable(m, x2 >=0)
•     @variable(m, x3 >=0)
• end
```

Step 4: Setting the objective

$$x1 + 2\,x2 + 5\,x3$$

```
• begin
•     md"""   Rule: @objective(model's name, Min or Max, your objective)"""
•     @objective(m, Max, x1 + 2x2 + 5x3)
• end
```

Step 5: Add constraints

$$\text{constraint2} : x1 + 3x2 - 7x3 \leq 10.0$$

```
• begin
•     md""" Rule: @constraint(model's name, constraint's name, your constraint)"""
•     @constraint(m, constraint1, -x1 +  x2 + 3x3 <= -5)
•     @constraint(m, constraint2,  x1 + 3x2 - 7x3 <= 10)
• end
```

Step 6: check your model and run it

```
A JuMP Model
Maximization problem with:
Variables: 3
Objective function type: AffExpr
`JuMP.AffExpr`-in-`MathOptInterface.LessThan{Float64}`: 2 constraints
`JuMP.VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
`JuMP.VariableRef`-in-`MathOptInterface.LessThan{Float64}`: 1 constraint
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: GLPK
Names registered in the model: constraint1, constraint2, x1, x2, x3
```

```
• m
```

```
• JuMP.optimize!(m)
```

Optimal Solutions:

x1 = 10.0, x2 = 2.1875, x3 = 0.9375

```
• begin
•     a1 = JuMP.value(x1)
•     a2 = JuMP.value(x2)
•     a3 = JuMP.value(x3)
•     md"""
•     ##### Optimal Solutions:
•     x1 = $a1,
•     x2 = $a2,
•     x3 = $a3
•     """
•
```

```
end
```

A compacter way to formulate this problem

```
begin
    m2 = Model(GLPK.Optimizer)

    c = [ 1; 2; 5]
    A = [-1  1  3;
          1  3 -7]
    b = [-5; 10]

    index_y = 1:3
    index_constraints = 1:2

    @variable(m2, y[index_y] >= 0)
    @objective(m2, Max, sum( c[i]*y[i] for i in index_y) )

    @constraint(m2, constraint[j in index_constraints],
                   sum( A[j,i]*y[i] for i in index_y ) <= b[j] )

    @constraint(m2, bound, y[1] <= 10)

    JuMP.optimize!(m2)
end
```

Optimal Solutions:

y1 = 10.0, y2 = 2.1875, y3 = 0.9375

```
begin
    a4 = JuMP.value(y[1])
    a5 = JuMP.value(y[2])
    a6 = JuMP.value(y[3])
    md"""
    ##### Optimal Solutions:
    y1 = $a4,
    y2 = $a5,
    y3 = $a6
    """
end
```

(2) Mixed Integer Linear programming

$$max x_1 + 2x_2 + 5x_3$$

$$s.t. -x_1 + x_2 - 7x_3 \leq -5$$

$$x_1 + 3x_2 - 7x_3 \leq 10$$

$$0 \leq x_1 \leq 10$$

$$x_2 \geq 0, x_2 \in \mathbb{N}$$

$$x_3 \in \{0, 1\}$$

```
begin
    m3 = Model(GLPK.Optimizer)

    @variable(m3, 0<= x1 <=10)
    @variable(m3, x2 >=0, Int) #restrict x2 to an integer
    @variable(m3, x3, Bin) # restrict x3 to a binary number

    @objective(m3, Max, x1 + 2x2 + 5x3)

    @constraint(m3, constraint1, -x1 +  x2 + 3x3 <= -5)
    @constraint(m3, constraint2,  x1 + 3x2 - 7x3 <= 10)

    # Solving the optimization problem
    JuMP.optimize!(m3)
end
```

Optimal Solutions:

x1 = 10.0, x2 = 2.0, x3 = 1.0

```
begin
    a7 = JuMP.value(x1)
    a8 = JuMP.value(x2)
    a9 = JuMP.value(x3)
    md"""
    ##### Optimal Solutions:
    x1 = $a7,
    x2 = $a8,
    x3 = $a9
    """
end
```

Topic 3: Wagner-Whitin Reformulation

Formulation

$$min \sum_{t=1}^{T}(Ky_t + hx_t)$$

$$s.t. x_t = x_{t-1} + q_t - d_t, \forall t = 1, \ldots, T$$

$$q_t \leq My_t, \forall t = 1, \ldots, T$$

$$x_t \geq 0, \forall t = 1, \ldots, T$$

$$q_t \geq 0, \forall t = 1, \ldots, T$$

$$y_t \in \{0, 1\}, \forall t = 1, \ldots, T$$

variable definition

```
begin
    K_3 = 500
```

```
    h_3 = 2
    T_3 = 4
    I0_3 = 0
    d_3 = [90 120 80 70]

    md"##### variable definition"
end
```

```
begin
    m4 = Model(GLPK.Optimizer)

    var_index = 1:T_3
    @variable(m4, a[var_index] >= 0)
    @variable(m4, b[var_index], Bin)
    @variable(m4, q[var_index] >= 0)

    @objective(m4, Min, sum(K_3*b[i] + h_3*a[i] for i in var_index) )

    @constraint(m4, cons1[1], a[1] == (I0_3 + q[1] - d_3[1]) )
    @constraint(m4, cons2[j in 2:T_3], a[j] == (a[j-1] + q[j] - d_3[j]) )

    M = 100000
    @constraint(m4, cons3[i in var_index], q[i] <= M*b[i])

    JuMP.optimize!(m4)
end
```

In period 1, we order 210.0. In period 2, we order 0.0. In period 3, we order 150.0. In period 4, we order 0.0.