Tutorial V - Economic Dispatch Problem

Energy System Optimization with Julia

1. Modelling the ED problem

Implement the ED problem from the lecture in Julia. Before we start, let's load the necessary packages and data.

```
using JuMP, HiGHS
using CSV
using DataFrames
using Plots
```

> **Tip**
>
> If you haven't installed the packages yet, you can do so by running `using Pkg` first and then `Pkg.add("JuMP")`, `Pkg.add("HiGHS")`, `Pkg.add("DataFrames")`, `Pkg.add("Plots")`, and `Pkg.add("StatsPlots")`.

Now, let's load the data. The generator data ($p_g^{\min}, p_g^{\max}, c_g^{var}, c_g^{fix}$), c_g^{fix} being fixed cost not used in the ED, the wind data (c_w^{var}), and the scenario data (p_w^f, d^f) are provided as CSV files.

```
# Get the directory of the current file
file_directory = "$(@__DIR__)/data"

# Load the data of the thermal generators
generators = CSV.read("$file_directory/generator.csv", DataFrame)
println("Number of generator: $(nrow(generators))")
println("First 5 rows of available genrator:")
println(generators[1:5, :])
```

```
Number of generator: 6
First 5 rows of available genrator:
5×5 DataFrame
 Row   name     min_power  max_power  variable_cost  fix_cost
       String3  Int64      Int64      Int64          Int64

   1   G1             100        500            50      1000
   2   G2              50        350            60      1200
   3   G3              40        250            55      1300
   4   G4              30        200            70      1500
   5   G5              30        200            60      1500
```

```
# Load the data of the wind turbines
windTurbines = CSV.read("$file_directory/windTurbine.csv", DataFrame)
println("Number of wind turbines: $(nrow(windTurbines))")
println("Variable cost per wind turbine:")
```

```
println(windTurbines)
```

```
Number of wind turbines: 1
Variable cost per wind turbine:
1×2 DataFrame
 Row   name     var_cost
       String3  Int64

   1   T1             50
```

```
# Load the sceanrio data about the demand and wind forecast
scenarios = CSV.read("$file_directory/scenario.csv", DataFrame)
println("First 5 rows of sceanios:")
println(scenarios[1:5, :])
```

```
First 5 rows of sceanios:
5×3 DataFrame
 Row   name     wind_forecast  demand_forecast
       String3  Int64          Int64

   1   S1               1000             1500
   2   S2               1000             1600
   3   S3               1000             1400
   4   S4               1000             1300
   5   S5               1000             1000
```

Next, you need to prepare the given data for the model. We will use 'function' to create a 'Named Tuple' which we can access with the dot notation:

```
# This function creates the Named Tuple ThermalGenerator
function ThermalGenerator(
    min::Int64,
    max::Int64,
    fixed_cost::Int64,
    variable_cost::Int64,
)
    return (
        min = min,
        max = max,
        fixed_cost = fixed_cost,
        variable_cost = variable_cost,
    )
end

# Add generators of the data to a dictionary of the generators
dictThermalGeneartors = Dict(row.name => ThermalGenerator(row.min_power, row.max_power,
 ↪  row.fix_cost, row.variable_cost) for row in eachrow(generators))

# Now a generator propety can be accessed
println(dictThermalGeneartors["G1"].variable_cost)
```

Analogously create a dictionary for the wind turbines and scenarios. Call them `dictWindTurbines` and `dictScenarios`.

```
# YOUR CODE BELOW
```

```
# Validate your solution
@assert length(dictThermalGeneartors) == nrow(generators) "Available time dictionary
 ↪  should have same length as input data"
@assert length(dictWindTurbines) == nrow(windTurbines) "Available time dictionary should
 ↪  have same length as input data"
@assert length(dictScenarios) == nrow(scenarios) "Scenario dictionary should have same
 ↪  length as input data"

# Check that all values are positive
@assert all(v -> all(x -> x >= 0, [v.min, v.max, v.fixed_cost, v.variable_cost]),
 ↪  values(dictThermalGeneartors)) "All thermal generator values must be positive"
@assert all(v -> v.variable_cost >= 0, values(dictWindTurbines)) "All wind turbine values
 ↪  must be positive"
@assert all(v -> all(x -> x >= 0, [v.wind_forecast, v.demand_forecast]),
 ↪  values(dictScenarios)) "All scenario values must be positive"

# Check that dictionaries contain all expected keys
@assert all(p -> haskey(dictThermalGeneartors, p), generators.name) "Missing names in
 ↪  dictionary"
@assert all(b -> haskey(dictWindTurbines, b), windTurbines.name) "Missing names in
 ↪  dictionary"
@assert all(b -> haskey(dictScenarios, b), scenarios.name) "Missing names in dictionary"
```

Next, we define the model instance for the ED problem.

```
# Prepare the model instance
dispatchModel = Model(HiGHS.Optimizer)
```

Now, create your variables. Please name them `p_g` for the power output of generators, `p_w` for the power injection of wind turbines.

> **Note**
>
> Consider the bounds for these variables. First, we only want to solve the model for sceanrio “S1”.

```
# YOUR CODE BELOW
```

```
# Validate your solution
# Check variable dimensions
@assert length(p_g) == length(dictThermalGeneartors) "Incorrect dimensions for p_g"
@assert length(p_w) == length(dictWindTurbines) "Incorrect dimensions for p_w"

# Check variable types
@assert all(x -> is_valid(dispatchModel, x), p_g) "p_g must be valid variables"
@assert all(x -> is_valid(dispatchModel, x), p_w) "p_w must be valid variables"
```

Next, define the objective function.

```
# YOUR CODE BELOW
```

```
# Validate your solution
# Check if the model has an objective
@assert objective_function(dispatchModel) !== nothing "Model must have an objective
 ↪  function"

# Check if it's a minimization problem
@assert objective_sense(dispatchModel) == MOI.MIN_SENSE "Objective should be
 ↪  minimization"

# Check if the objective function contains both cost components
obj_expr = objective_function(dispatchModel)
@assert contains(string(dispatchModel), "p_g") "Objective must include variable costs
 ↪  (p_g)"
@assert contains(string(dispatchModel), "p_w") "Objective must include variable costs
 ↪  (p_w)"
```

Now, we need to define all necessary constraints for the model, which is only the demand/production balance constraint as we considered min and max power limitations in the variable setup.

```
# YOUR CODE BELOW


```

Finally, implement the solve statement for your model instance and print the results.

```
# YOUR CODE BELOW


```

```
# Validate your solution
@assert objective_value(dispatchModel) == 76600 "Objective value should be 76600"
```

2. Solving scenarios of the ED problem

We now want to solve all sceanrios. To do so we wrap the model in a function that we then can call with different inputs.

> **Note**
>
> Copy your model into the function. The results should be stored in the dataframe.

```
# Create a function `solve_economic_dispatch`, which solves the economic
# dispatch problem for a given set of input parameters.

function solve_economic_dispatch(dictThermalGeneartors::Dict, dictWindTurbines::Dict,
 ↪  scenario)
    ## Define the economic dispatch (ED) model
    dispatchModel = Model(HiGHS.Optimizer)
    set_silent(dispatchModel)
    ## Define decision variables
    ## p_g power output of generators
    # YOUR CODE BELOW

    ## p_w wind power injection
    # YOUR CODE BELOW

    ## Define the objective function
    # YOUR CODE BELOW

    ## Define the power balance constraint
    # YOUR CODE BELOW

    ## Solve statement
    optimize!(dispatchModel)
    assert_is_solved_and_feasible(dispatchModel)

    ## return the optimal value of the objective function and variables
    return (
        p_g = value.(p_g),
        p_w = value.(p_w),
        wind_curtailment = scenario.wind_forecast - sum(value.(p_w)),
        total_cost = objective_value(dispatchModel),
    )
end
```

```
# Create a dataframe to store results
results_df = DataFrame(
    scenario = String[],
    total_cost = Float64[],
    wind_curtailment = Float64[]
)

# Loop over the scenarios and save the results to a dataframe
for (scenario_name, scenario_data) in dictScenarios
    solution = solve_economic_dispatch(dictThermalGeneartors, dictWindTurbines,
 ↪  scenario_data)
    push!(results_df, (scenario_name, solution.total_cost, solution.wind_curtailment))
end

# Print the dataframe
println("\nResults for all scenarios:")
println(results_df)
```

What is the problem in scenario “S5” with the assumptions made in the ED problem leading to an inefficient usage of wind turbines?

```
# YOUR ANSWER HERE
```

Solutions

You will likely find solutions to most exercises online. However, I strongly encourage you to work on these exercises independently without searching explicitly for the exact answers to the exercises. Understanding someone else's solution is very different from developing your own. Use the lecture notes and try to solve the exercises on your own. This approach will significantly enhance your learning and problem-solving skills.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions.

Later, you will find the solutions to these exercises online in the associated GitHub repository, but we will also quickly go over them in next week's tutorial. To access the solutions, click on the Github button on the lower right and search for the folder with today's lecture and tutorial. Alternatively, you can ask ChatGPT or Claude to explain them to you. But please remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities.