Tutorial VIII - Stochastic Energy System Design Problem

Energy System Optimization with Julia

Dr. Tobias Cors

Stochastic System Design

System Design Problem

The system design problem combines investment planning (sizing) and operational optimization to determine the optimal configuration of an energy system. This integrated approach allows us to:

1. Make optimal investment decisions
2. Account for operational flexibility
3. Balance investment and operational costs
4. Consider system-wide interactions

> **Note**
>
> The system design problem is a two-stage optimization problem:
> 1. First stage: Investment decisions (sizing)
> 2. Second stage: Operational decisions (dispatch)

Key Components

Investment Planning (Sizing)

- Component selection
- Capacity sizing
- Technology mix
- Investment costs

Operational Optimization (Dispatch)

- Power generation
- Storage operation
- Grid interaction
- Operational costs

Mathematical Structure

Two-Stage Problem

1. **First Stage (Here-and-Now)**:
   - Investment decisions
   - Component sizing
   - Fixed costs
2. **Second Stage (Wait-and-See)**:

- Operational decisions
- Power dispatch
- Variable costs

Objective Function

Minimize Investment Costs + Expected Operational Costs

> **Note**
>
> The objective function combines:
> 1. One-time investment costs (annualized)
> 2. Expected operational costs over the system lifetime

Solution Approaches

Deterministic Approach

- Single scenario
- Perfect foresight
- Simplified uncertainty handling

Stochastic Approach

- Multiple scenarios
- Probability-weighted costs
- Robust investment decisions

Applications

Energy System Planning

- Microgrid design
- Industrial energy systems
- Residential energy systems

Grid Integration

- Renewable energy integration
- Storage deployment
- Grid capacity planning

Market Participation

- Multi-market optimization
- Price arbitrage
- Ancillary services

> **Note**
>
> The system design problem provides a comprehensive framework for:
> 1. Optimal investment decisions
> 2. Efficient system operation

3. Cost-effective energy supply
4. Sustainable energy systems

Stochastic Formulation

The stochastic system design problem extends the deterministic model by considering multiple scenarios. This allows us to:

1. Account for uncertainty in renewable generation
2. Consider different demand patterns
3. Handle price variations
4. Make robust investment decisions

Sets

- $\mathcal{T}$ - Set of time periods indexed by $t \in \{1, 2, ..., |\mathcal{T}|\}$
- $\mathcal{S}$ - Set of storage systems indexed by $s \in \{1, 2, ..., |\mathcal{S}|\}$
- $\mathcal{W}$ - Set of wind parks indexed by $w \in \{1, 2, ..., |\mathcal{W}|\}$
- $\mathcal{V}$ - Set of PV parks indexed by $v \in \{1, 2, ..., |\mathcal{V}|\}$
- Ω - Set of scenarios indexed by $\omega \in \{1, 2, ..., |\Omega|\}$

Decision Variables

Investment Variables (First Stage)

- e_s^{nom} - Nominal energy capacity of storage s [MWh]
- $p_s^{ch,nom}$ - Nominal charging power of storage s [MW]
- $p_s^{dis,nom}$ - Nominal discharging power of storage s [MW]
- p_w^{nom} - Nominal power of wind park w [MW]
- p_v^{nom} - Nominal power of PV park v [MW]

Operational Variables (Second Stage)

- $p_{w,t,\omega}$ - Power output of wind park w at time t in scenario ω [MW]
- $p_{v,t,\omega}$ - Power output of PV park v at time t in scenario ω [MW]
- $p_{t,\omega}^{in}$ - Power inflow through market at time t in scenario ω [MW]
- $p_{t,\omega}^{out}$ - Power outflow through market at time t in scenario ω [MW]
- $p_{s,t,\omega}^{ch}$ - Charging power of storage s at time t in scenario ω [MW]
- $p_{s,t,\omega}^{dis}$ - Discharging power of storage s at time t in scenario ω [MW]
- $e_{s,t,\omega}$ - Energy level of storage s at time t in scenario ω [MWh]

Annual Cost Variables

- AC_s^{inv} - Annual investment cost for storage s [EUR/year]
- AC_w^{inv} - Annual investment cost for wind park w [EUR/year]
- AC_v^{inv} - Annual investment cost for PV park v [EUR/year]
- $AC_\omega^{grid,imp}$ - Annual grid electricity import cost in scenario ω [EUR/year]
- $AR_\omega^{grid,exp}$ - Annual grid electricity export revenue in scenario ω [EUR/year]

Parameters

Investment Costs (First Stage)

- C_s^E - Cost per MWh of energy capacity for storage s [EUR/MWh]
- $C_s^{P,ch}$ - Cost per MW of charging power capacity for storage s [EUR/MW]
- $C_s^{P,dis}$ - Cost per MW of discharging power capacity for storage s [EUR/MW]
- C_w^W - Cost per MW of wind park w [EUR/MW]
- C_v^{PV} - Cost per MW of PV park v [EUR/MW]
- F^{PVAF} - Present value annuity factor for investment costs
- B^{max} - Maximum investment budget [EUR]

Operational Parameters (Second Stage)

- η_s^{ch} - Charging efficiency of storage s
- η_s^{dis} - Discharging efficiency of storage s
- sdr_s - Self-discharge rate of storage s per time step
- DoD_s - Depth of discharge limit for storage s [%]
- $f_{w,t,\omega}$ - Wind capacity factor at time t in scenario ω for wind park w
- $f_{v,t,\omega}$ - Solar capacity factor at time t in scenario ω for PV park v
- $d_{t,\omega}$ - Electric demand at time t in scenario ω [MW]
- $c_{t,\omega}^{MP}$ - Grid electricity market price at time t in scenario ω [EUR/MWh]
- c^{TaL} - Grid electricity taxes and levies (including Netzentgelt) [EUR/MWh]
- π_ω - Probability of scenario ω

Objective Function

Minimize $\sum_{s\in\mathcal{S}} AC_s^{inv} + \sum_{w\in\mathcal{W}} AC_w^{inv} + \sum_{v\in\mathcal{V}} AC_v^{inv} + \sum_{\omega\in\Omega} \pi_\omega (AC_\omega^{grid,imp} - AR_\omega^{grid,exp})$

> **Note**
>
> The objective function minimizes: 1. First-stage costs (deterministic): - Investment costs for all components 2. Second-stage costs (stochastic): - Expected grid electricity costs/revenues - Weighted by scenario probabilities

Annual Cost Constraints

Investment Costs (First Stage)

$AC_s^{inv} = \frac{C_s^E}{F^{PVAF}} e_s^{nom} + C_s^{P,ch} p_s^{ch,nom} + C_s^{P,dis} p_s^{dis,nom} \quad \forall s \in \mathcal{S}$ $AC_w^{inv} = \frac{C_w^W}{F^{PVAF}} p_w^{nom} \quad \forall w \in \mathcal{W}$ $AC_v^{inv} = \frac{C_v^{PV}}{F^{PVAF}} p_v^{nom} \quad \forall v \in \mathcal{V}$

Investment Budget

$\sum_{s\in\mathcal{S}} (C_s^E e_s^{nom} + C_s^{P,ch} p_s^{ch,nom} + C_s^{P,dis} p_s^{dis,nom}) + \sum_{w\in\mathcal{W}} C_w^W p_w^{nom} + \sum_{v\in\mathcal{V}} C_v^{PV} p_v^{nom} \leq B^{max}$

> **Note**
>
> The investment budget constraint ensures that: 1. Total investment costs do not exceed the maximum budget 2. Includes all component investments: - Storage systems (energy and power capacity) - Wind parks - PV parks 3. Applies to first-stage decisions only

Grid Electricity Costs (Second Stage)

$AC_{\omega}^{grid,imp} = \sum_{t \in \mathcal{T}} (c_{t,\omega}^{MP} + c^{TaL}) p_{t,\omega}^{in} \quad \forall \omega \in \Omega \; AR_{\omega}^{grid,exp} = \sum_{t \in \mathcal{T}} c_{t,\omega}^{MP} p_{t,\omega}^{out} \quad \forall \omega \in \Omega$

Constraints

Power Balance

$\sum_{w \in \mathcal{W}} p_{w,t,\omega} + \sum_{v \in \mathcal{V}} p_{v,t,\omega} + (p_{t,\omega}^{in} - p_{t,\omega}^{out}) + \sum_{s \in \mathcal{S}} (p_{s,t,\omega}^{dis} - p_{s,t,\omega}^{ch}) = d_{t,\omega} \quad \forall t \in \mathcal{T}, \omega \in \Omega$

Component Limits

Wind Parks

$0 \leq p_{w,t,\omega} \leq f_{w,t,\omega} p_{w}^{nom} \quad \forall w \in \mathcal{W}, t \in \mathcal{T}, \omega \in \Omega$

PV Parks

$0 \leq p_{v,t,\omega} \leq f_{v,t,\omega} p_{v}^{nom} \quad \forall v \in \mathcal{V}, t \in \mathcal{T}, \omega \in \Omega$

Storage Systems

$0 \leq p_{s,t,\omega}^{ch} \leq p_{s}^{ch,nom} \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, \omega \in \Omega \; 0 \leq p_{s,t,\omega}^{dis} \leq p_{s}^{dis,nom} \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, \omega \in \Omega \; DoD_{s} e_{s}^{nom} \leq e_{s,t,\omega} \leq e_{s}^{nom} \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, \omega \in \Omega$

Storage Energy Balance

$e_{s,t,\omega} = (1 - sdr_{s}) e_{s,t-1,\omega} + \eta_{s}^{ch} p_{s,t,\omega}^{ch} - \frac{p_{s,t,\omega}^{dis}}{\eta_{s}^{dis}} \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, \omega \in \Omega$

> **Note**
>
> The stochastic formulation: 1. First-stage decisions (here-and-now): - Component sizing - Investment costs 2. Second-stage decisions (wait-and-see): - Operational decisions - Scenario-dependent costs 3. Key differences from deterministic model: - Time-dependent parameters become scenario-dependent - Operational variables become scenario-dependent - Objective includes expected costs

Implementation of the Stochastic System Design Problem

1. Load Packages

```
using Pkg
Pkg.add("JuMP")
Pkg.add("HiGHS")
Pkg.add("CSV")
Pkg.add("DataFrames")
Pkg.add("CairoMakie")
Pkg.add("Dates")

# Required packages
using Random
using Statistics
using DataFrames
using CSV
using Dates
using JuMP
using HiGHS
using CairoMakie

# Set up CairoMakie
set_theme!(theme_light())
```

2. Data Generation

```
# Set random seed for reproducibility
Random.seed!(42)

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

# Number of scenarios and time periods
n_scenarios = 5
n_hours = 168  # One week

# Generate scenario data
```

```
scenario_data = DataFrame(
    scenario = String[],
    datetime = DateTime[],
    demand = Float64[],
    wind_cf = Float64[],
    pv_cf = Float64[],
    market_price = Float64[],
    probability = Float64[]
)

# Base datetime
base_datetime = DateTime(2024, 1, 1)

# Generate data for each scenario
for s in 1:n_scenarios
    # Generate base demand profile (daily pattern)
    base_demand = 60 .+ 40 .* sin.(2  .* (0:n_hours-1) ./ 24)

    # Add random variations for each scenario
    demand = base_demand .+ randn(n_hours) .* 10
    demand = max.(20, min.(100, demand))  # Clamp between 20 and 100 MW

    # Generate wind capacity factors
    wind_cf = rand(n_hours)  # Random between 0 and 1

    # Generate PV capacity factors (daily pattern)
    hour_of_day = mod.(0:n_hours-1, 24)
    pv_cf = max.(0, sin.(  .* hour_of_day ./ 12)) .+ randn(n_hours) .* 0.1
    pv_cf = max.(0, min.(1, pv_cf))  # Clamp between 0 and 1

    # Generate market prices
    base_price = 200 .+ 100 .* sin.(2  .* (0:n_hours-1) ./ 24)
    market_price = base_price .+ randn(n_hours) .* 100
    market_price = max.(-500, min.(1000, market_price))  # Clamp between -500 and 1000

    # Add data to DataFrame
    for h in 1:n_hours
        push!(scenario_data, (
            "S$s",
            base_datetime + Hour(h-1),
            demand[h],
            wind_cf[h],
            pv_cf[h],
            market_price[h],
            1/n_scenarios  # Equal probability for each scenario
        ))
    end
end

# Save scenario data
CSV.write("$file_directory/scenario.csv", scenario_data)

# Create grid data
```

```
grid_data = DataFrame(
    name = ["grid"],
    taxes_levies = [50.0]  # 50 EUR/MWh for taxes and levies
)

# Save grid data
CSV.write("$file_directory/grid.csv", grid_data)

# Create storage data
storage_data = DataFrame(
    name = ["storage"],
    energy_cost = [100000.0],  # 100,000 EUR/MWh
    power_cost = [50000.0],    # 50,000 EUR/MW
    lifetime = [10],           # 10 years
    discount_rate = [0.05],    # 5% discount rate
    charge_efficiency = [0.95],
    discharge_efficiency = [0.95],
    self_discharge_rate = [0.001]  # 0.1% per hour
)

# Save storage data
CSV.write("$file_directory/storage.csv", storage_data)

# Create wind turbine data
wind_data = DataFrame(
    name = ["wind"],
    power_cost = [1000000.0],  # 1,000,000 EUR/MW
    lifetime = [20],           # 20 years
    discount_rate = [0.05]     # 5% discount rate
)

# Save wind data
CSV.write("$file_directory/windTurbine.csv", wind_data)

# Create PV data
pv_data = DataFrame(
    name = ["pv"],
    power_cost = [500000.0],   # 500,000 EUR/MW
    lifetime = [25],           # 25 years
    discount_rate = [0.05]     # 5% discount rate
)

# Save PV data
CSV.write("$file_directory/pv.csv", pv_data)
```

3. Load and Process Data

```
# Load and process data into dictionaries
dfStorage = CSV.read("$file_directory/storage.csv", DataFrame)
dfWindTurbines = CSV.read("$file_directory/windTurbine.csv", DataFrame)
dfPV = CSV.read("$file_directory/pv.csv", DataFrame)
```

```
dfScenarios = CSV.read("$file_directory/scenario.csv", DataFrame)
dfGrid = CSV.read("$file_directory/grid.csv", DataFrame)

# Process storage data
dictStorage = Dict(
    row.name => (
        energy_cost = row.energy_cost,
        power_cost = row.power_cost,
        lifetime = row.lifetime,
        discount_rate = row.discount_rate,
        charge_efficiency = row.charge_efficiency,
        discharge_efficiency = row.discharge_efficiency,
        self_discharge_rate = row.self_discharge_rate
    ) for row in eachrow(dfStorage)
)

# Process wind turbine data
dictWindTurbines = Dict(
    row.name => (
        power_cost = row.power_cost,
        lifetime = row.lifetime,
        discount_rate = row.discount_rate
    ) for row in eachrow(dfWindTurbines)
)

# Process PV data
dictPV = Dict(
    row.name => (
        power_cost = row.power_cost,
        lifetime = row.lifetime,
        discount_rate = row.discount_rate
    ) for row in eachrow(dfPV)
)

# Process scenario data
dictScenarios = Dict()
for scenario in unique(dfScenarios.scenario)
    scenario_data = dfScenarios[dfScenarios.scenario .== scenario, :]
    dictScenarios[scenario] = (
        datetime = scenario_data.datetime,
        demand = scenario_data.demand,
        wind_cf = scenario_data.wind_cf,
        pv_cf = scenario_data.pv_cf,
        market_price = scenario_data.market_price,
        probability = scenario_data.probability[1]
    )
end

# Process grid data
dictGrid = Dict(
    row.name => (
        taxes_levies = row.taxes_levies,
    ) for row in eachrow(dfGrid)
```

```
)
```

4. Model Implementation

Now, let's implement the Stochastic System Design model using the dictionary format:

```
function solve_stochastic_design(dictStorage, dictWindTurbines, dictPV, dictScenarios,
 ↪  dictGrid, max_budget)
    # Create model
    model = Model(HiGHS.Optimizer)
    set_silent(model)

    # Define sets
    T = 1:168  # Time periods (hours)
    S = keys(dictStorage)  # Set of storage systems
    W = keys(dictWindTurbines)  # Set of wind parks
    V = keys(dictPV)  # Set of PV parks
    Ω = keys(dictScenarios)  # Set of scenarios

    # Calculate PVAF for each component type
    pvaf = Dict(
        "storage" => (1 +
         ↪  dictStorage["storage"].discount_rate)^dictStorage["storage"].lifetime /
                  (dictStorage["storage"].discount_rate * (1 +
 ↪  dictStorage["storage"].discount_rate)^dictStorage["storage"].lifetime),
        "wind" => (1 +
         ↪  dictWindTurbines["wind"].discount_rate)^dictWindTurbines["wind"].lifetime /
                (dictWindTurbines["wind"].discount_rate * (1 +
 ↪  dictWindTurbines["wind"].discount_rate)^dictWindTurbines["wind"].lifetime),
        "pv" => (1 + dictPV["pv"].discount_rate)^dictPV["pv"].lifetime /
               (dictPV["pv"].discount_rate * (1 +
 ↪  dictPV["pv"].discount_rate)^dictPV["pv"].lifetime)
    )

    # Decision Variables
    # First stage (investment)
    @variable(model, e_nom[s in S] >= 0)  # Nominal energy capacity
    @variable(model, p_ch_nom[s in S] >= 0)  # Nominal charging power
    @variable(model, p_dis_nom[s in S] >= 0)  # Nominal discharging power
    @variable(model, p_w_nom[w in W] >= 0)  # Nominal wind power
    @variable(model, p_v_nom[v in V] >= 0)  # Nominal PV power

    # Second stage (operation)
    @variable(model, p_w[w in W, t in T,  in Ω] >= 0)  # Wind power output
    @variable(model, p_v[v in V, t in T,  in Ω] >= 0)  # PV power output
    @variable(model, p_in[t in T,  in Ω] >= 0)  # Grid import
    @variable(model, p_out[t in T,  in Ω] >= 0)  # Grid export
    @variable(model, p_ch[s in S, t in T,  in Ω] >= 0)  # Storage charging
    @variable(model, p_dis[s in S, t in T,  in Ω] >= 0)  # Storage discharging
    @variable(model, e[s in S, t in T,  in Ω] >= 0)  # Storage energy level

    # Annual cost variables
```

```
    @variable(model, AC_inv_s[s in S] >= 0)  # Annual storage investment cost
    @variable(model, AC_inv_w[w in W] >= 0)  # Annual wind investment cost
    @variable(model, AC_inv_v[v in V] >= 0)  # Annual PV investment cost
    @variable(model, AC_grid_imp[  in Ω] >= 0)  # Annual grid import cost
    @variable(model, AR_grid_exp[  in Ω] >= 0)  # Annual grid export revenue

    # Objective Function
    @objective(model, Min,
        sum(AC_inv_s[s] for s in S) +
        sum(AC_inv_w[w] for w in W) +
        sum(AC_inv_v[v] for v in V) +
        sum(dictScenarios[ ].probability * (AC_grid_imp[ ] - AR_grid_exp[ ]) for  in Ω) *
    ↪  52.1429
    )

    # Investment cost constraints
    @constraint(model, [s in S],
        AC_inv_s[s] == dictStorage[s].energy_cost/pvaf["storage"] * e_nom[s] +
                       dictStorage[s].power_cost/pvaf["storage"] * (p_ch_nom[s] +
↪   p_dis_nom[s])
    )

    @constraint(model, [w in W],
        AC_inv_w[w] == dictWindTurbines[w].power_cost/pvaf["wind"] * p_w_nom[w]
    )

    @constraint(model, [v in V],
        AC_inv_v[v] == dictPV[v].power_cost/pvaf["pv"] * p_v_nom[v]
    )

    # Investment budget constraint
    @constraint(model,
        sum(dictStorage[s].energy_cost * e_nom[s] +
            dictStorage[s].power_cost * (p_ch_nom[s] + p_dis_nom[s]) for s in S) +
        sum(dictWindTurbines[w].power_cost * p_w_nom[w] for w in W) +
        sum(dictPV[v].power_cost * p_v_nom[v] for v in V) <= max_budget
    )

    # Grid electricity costs
    @constraint(model, [  in Ω],
        AC_grid_imp[ ] == sum(
            (dictScenarios[ ].market_price[t] + dictGrid["grid"].taxes_levies) * p_in[t, ]
            for t in T
        )
    )

    @constraint(model, [  in Ω],
        AR_grid_exp[ ] == sum(
            dictScenarios[ ].market_price[t] * p_out[t, ]
            for t in T
        )
    )
```

```
# Power balance
@constraint(model, [t in T,  in Ω],
    sum(p_w[w,t,] for w in W) +
    sum(p_v[v,t,] for v in V) +
    (p_in[t,] - p_out[t,]) +
    sum(p_dis[s,t,] - p_ch[s,t,] for s in S) ==
    dictScenarios[].demand[t]
)

# Component limits
@constraint(model, [w in W, t in T,  in Ω],
    p_w[w,t,] <= dictScenarios[].wind_cf[t] * p_w_nom[w]
)

@constraint(model, [v in V, t in T,  in Ω],
    p_v[v,t,] <= dictScenarios[].pv_cf[t] * p_v_nom[v]
)

@constraint(model, [s in S, t in T,  in Ω],
    p_ch[s,t,] <= p_ch_nom[s]
)

@constraint(model, [s in S, t in T,  in Ω],
    p_dis[s,t,] <= p_dis_nom[s]
)

@constraint(model, [s in S, t in T,  in Ω],
    e[s,t,] <= e_nom[s]
)

# Storage energy balance
@constraint(model, [s in S, t in 2:length(T),  in Ω],
    e[s,t,] == (1 - dictStorage[s].self_discharge_rate) * e[s,t-1,] +
               dictStorage[s].charge_efficiency * p_ch[s,t,] -
               p_dis[s,t,] / dictStorage[s].discharge_efficiency
)

@constraint(model, [s in S,  in Ω],
    e[s,1,] == dictStorage[s].charge_efficiency * p_ch[s,1,] -
               p_dis[s,1,] / dictStorage[s].discharge_efficiency
)

# Solve the model
optimize!(model)

# Assert that the solution is feasible
if termination_status(model) != MOI.OPTIMAL
    ts = termination_status(model)
    @info "Optimization finished. The model was not solved correctly. Termination
     ↪  Status: $ts"
    # Helpful resource: https://jump.dev/JuMP.jl/stable/manual/solutions/#Conflicts
end
```

```
    # Return results
    return (
        e_nom = value.(e_nom),
        p_ch_nom = value.(p_ch_nom),
        p_dis_nom = value.(p_dis_nom),
        p_w_nom = value.(p_w_nom),
        p_v_nom = value.(p_v_nom),
        total_cost = objective_value(model),
        investment_costs = Dict(
            "storage" => value.(AC_inv_s["storage"]),
            "wind" => value.(AC_inv_w["wind"]),
            "pv" => value.(AC_inv_v["pv"])
        ),
        operational_costs = Dict(
            "grid_import" => sum(dictScenarios[].probability * value.(AC_grid_imp[]) for
            ↪    in Ω) * 52.1429,
            "grid_export" => sum(dictScenarios[].probability * value.(AR_grid_exp[]) for
            ↪    in Ω) * 52.1429
        ),
        operational_variables = Dict(
            "storage_energy" => value.(e),
            "storage_charge" => value.(p_ch),
            "storage_discharge" => value.(p_dis),
            "wind_power" => value.(p_w),
            "pv_power" => value.(p_v),
            "grid_import" => value.(p_in),
            "grid_export" => value.(p_out)
        )
    )
end
```

5. Solve and Analyze Results

Let's solve the model with a maximum budget of 50 million EUR:

```
# Solve model
max_budget = 50_000_000  # 50 million EUR
results = solve_stochastic_design(dictStorage, dictWindTurbines, dictPV, dictScenarios,
 ↪  dictGrid, max_budget)

# Create results DataFrame
results_df = DataFrame(
    component = String[],
    capacity = Float64[],
    investment_cost = Float64[]
)

# Add storage results
push!(results_df, ("Storage", results.e_nom["storage"],
 ↪  results.investment_costs["storage"]))

# Add wind results
```

```
push!(results_df, ("Wind", results.p_w_nom["wind"], results.investment_costs["wind"]))

# Add PV results
push!(results_df, ("PV", results.p_v_nom["pv"], results.investment_costs["pv"]))

# Calculate total costs
total_investment_cost = sum(values(results.investment_costs))
total_operational_cost = results.operational_costs["grid_import"] -
 ↪  results.operational_costs["grid_export"]
total_cost = total_investment_cost + total_operational_cost

# Print results
println("Optimal System Design:")
println("Storage Energy Capacity: $(round(results.e_nom["storage"], digits=2)) MWh")
println("Storage Charging Power: $(round(results.p_ch_nom["storage"], digits=2)) MW")
println("Storage Discharging Power: $(round(results.p_dis_nom["storage"], digits=2)) MW")
println("Wind Power Capacity: $(round(results.p_w_nom["wind"], digits=2)) MW")
println("PV Power Capacity: $(round(results.p_v_nom["pv"], digits=2)) MW")
println("\nCosts:")
println("Total Annual Cost: $(round(total_cost, digits=2)) EUR/year")
println("Annual Investment Cost: $(round(total_investment_cost, digits=2)) EUR/year")
println("Annual Operational Cost: $(round(total_operational_cost, digits=2)) EUR/year")
println("\nInvestment Costs by Component:")
println("Storage: $(round(results.investment_costs["storage"], digits=2)) EUR/year")
println("Wind: $(round(results.investment_costs["wind"], digits=2)) EUR/year")
println("PV: $(round(results.investment_costs["pv"], digits=2)) EUR/year")
println("\nOperational Costs:")
println("Grid Import: $(round(results.operational_costs["grid_import"], digits=2))
 ↪  EUR/year")
println("Grid Export Revenue: $(round(results.operational_costs["grid_export"],
 ↪  digits=2)) EUR/year")

# Create operational results DataFrame
operational_df = DataFrame(
    scenario = String[],
    hour = Int[],
    storage_energy = Float64[],
    storage_charge = Float64[],
    storage_discharge = Float64[],
    wind_power = Float64[],
    pv_power = Float64[],
    grid_import = Float64[],
    grid_export = Float64[],
    demand = Float64[]
)

# Add operational data for each scenario
for   in keys(dictScenarios)
    for t in 1:168
        push!(operational_df, (
             ,
            t,
            results.operational_variables["storage_energy"]["storage", t,  ],
```

```
            results.operational_variables["storage_charge"]["storage", t, ],
            results.operational_variables["storage_discharge"]["storage", t, ],
            results.operational_variables["wind_power"]["wind", t, ],
            results.operational_variables["pv_power"]["pv", t, ],
            results.operational_variables["grid_import"][t, ],
            results.operational_variables["grid_export"][t, ],
            dictScenarios[ ].demand[t]
        ))
    end
end
```

6. Create Figures

```
# Create figure for component sizes and costs
fig1 = Figure(size=(800, 800))

# Component sizes
ax1 = Axis(fig1[1, 1], title="Optimal Component Sizes")
x_pos = 1:length(results_df.component)
barplot!(ax1, x_pos, results_df.capacity)
ax1.xticks = (x_pos, results_df.component)
ax1.xticklabelrotation =  /4
ax1.ylabel = "Capacity [MWh/MW]"

# Cost breakdown
ax2 = Axis(fig1[1, 2], title="Annual Cost Breakdown", aspect=1)  # Force square aspect
 ↪  ratio
hidespines!(ax2)  # Hide the axis spines
hidedecorations!(ax2)  # Hide all decorations (ticks, labels, etc.)
costs = [results.investment_costs["storage"], results.investment_costs["wind"],
 ↪  results.investment_costs["pv"]]
colors = [:blue, :green, :orange]
labels = ["Storage", "Wind", "PV"]

# Create pie chart
pie!(ax2, costs, color=colors, radius=0.8)

# Create legend below the pie chart
Legend(fig1[2, 2], [PolyElement(color=c) for c in colors], labels, "Components",
    orientation=:horizontal)

# Display figure
display(fig1)

# Create operational plots
fig2 = Figure(size=(1200, 1600))

# Storage operation
ax1 = Axis(fig2[1, 1], title="Storage Energy Level")
for   in keys(dictScenarios)
    lines!(ax1, operational_df[operational_df.scenario .==  , :].hour,
```

```
            operational_df[operational_df.scenario .==  , :].storage_energy,
            label= )
end
ax1.xlabel = "Hour"
ax1.ylabel = "Energy [MWh]"
axislegend(ax1)

# Storage power
ax2 = Axis(fig2[1, 2], title="Storage Power")
for   in keys(dictScenarios)
    lines!(ax2, operational_df[operational_df.scenario .==  , :].hour,
            operational_df[operational_df.scenario .==  , :].storage_charge,
            label="Charge - $ ")
    lines!(ax2, operational_df[operational_df.scenario .==  , :].hour,
            operational_df[operational_df.scenario .==  , :].storage_discharge,
            label="Discharge - $ ")
end
ax2.xlabel = "Hour"
ax2.ylabel = "Power [MW]"
axislegend(ax2)

# Wind power
ax3 = Axis(fig2[2, 1], title="Wind Power Generation")
for   in keys(dictScenarios)
    lines!(ax3, operational_df[operational_df.scenario .==  , :].hour,
            operational_df[operational_df.scenario .==  , :].wind_power,
            label= )
end
ax3.xlabel = "Hour"
ax3.ylabel = "Power [MW]"
axislegend(ax3)

# PV power
ax4 = Axis(fig2[2, 2], title="PV Power Generation")
for   in keys(dictScenarios)
    lines!(ax4, operational_df[operational_df.scenario .==  , :].hour,
            operational_df[operational_df.scenario .==  , :].pv_power,
            label= )
end
ax4.xlabel = "Hour"
ax4.ylabel = "Power [MW]"
axislegend(ax4)

# Grid interaction
ax5 = Axis(fig2[3, 1], title="Grid Import")
for   in keys(dictScenarios)
    lines!(ax5, operational_df[operational_df.scenario .==  , :].hour,
            operational_df[operational_df.scenario .==  , :].grid_import,
            label= )
end
ax5.xlabel = "Hour"
ax5.ylabel = "Power [MW]"
axislegend(ax5)
```

```
ax6 = Axis(fig2[3, 2], title="Grid Export")
for   in keys(dictScenarios)
    lines!(ax6, operational_df[operational_df.scenario .==  , :].hour,
           operational_df[operational_df.scenario .==  , :].grid_export,
           label= )
end
ax6.xlabel = "Hour"
ax6.ylabel = "Power [MW]"
axislegend(ax6)

# Demand
ax7 = Axis(fig2[4, 1:2], title="System Demand")
for   in keys(dictScenarios)
    lines!(ax7, operational_df[operational_df.scenario .==  , :].hour,
           operational_df[operational_df.scenario .==  , :].demand,
           label= )
end
ax7.xlabel = "Hour"
ax7.ylabel = "Power [MW]"
axislegend(ax7)

# Display figure
display(fig2)
```

7. Sensitivity Analysis Example

Let's analyze how the optimal solution changes with different maximum budgets:

```
# Test different budgets
budgets = [25_000_000, 50_000_000, 75_000_000, 100_000_000]
results_by_budget = Dict()

for budget in budgets
    results_by_budget[budget] = solve_stochastic_design(dictStorage, dictWindTurbines,
 ↪  dictPV, dictScenarios, dictGrid, budget)
end

# Create figure
fig3 = Figure(size=(1200, 800))

# Sort budgets and get corresponding results
sorted_budgets = sort(budgets)
sorted_results = [results_by_budget[b] for b in sorted_budgets]

# Component sizes vs budget
ax1 = Axis(fig3[1, 1], title="Component Sizes vs Budget")
lines!(ax1, sorted_budgets, [r.e_nom["storage"] for r in sorted_results], label="Storage
 ↪  Energy")
lines!(ax1, sorted_budgets, [r.p_w_nom["wind"] for r in sorted_results], label="Wind")
lines!(ax1, sorted_budgets, [r.p_v_nom["pv"] for r in sorted_results], label="PV")
ax1.xlabel = "Maximum Budget [EUR]"
```

```
ax1.ylabel = "Capacity [MWh or MW]"
axislegend(ax1)

# Costs vs budget
ax2 = Axis(fig3[1, 2], title="Costs vs Budget")
lines!(ax2, sorted_budgets, [r.total_cost for r in sorted_results], label="Total Annual
 ↪  Cost")
lines!(ax2, sorted_budgets, [r.investment_costs["storage"] for r in sorted_results],
 ↪  label="Storage")
lines!(ax2, sorted_budgets, [r.investment_costs["wind"] for r in sorted_results],
 ↪  label="Wind")
lines!(ax2, sorted_budgets, [r.investment_costs["pv"] for r in sorted_results],
 ↪  label="PV")

# Add initial investment cost line
initial_investment = [sum([
    dictStorage["storage"].energy_cost * r.e_nom["storage"] +
    dictStorage["storage"].power_cost * (r.p_ch_nom["storage"] + r.p_dis_nom["storage"])
 ↪  +
    dictWindTurbines["wind"].power_cost * r.p_w_nom["wind"] +
    dictPV["pv"].power_cost * r.p_v_nom["pv"]
]) for r in sorted_results]
lines!(ax2, sorted_budgets, initial_investment, label="Initial Investment",
 ↪  linestyle=:dash)

# Add budget line for reference
lines!(ax2, sorted_budgets, sorted_budgets, label="Maximum Budget", linestyle=:dot,
 ↪  color=:black)

ax2.xlabel = "Maximum Budget [EUR]"
ax2.ylabel = "Annual Cost [EUR/year]"
axislegend(ax2)

# Display figure
display(fig3)
```

8. Sensitivity Analysis of the Electricity Price

TASK: Watch for **##YOUR CODE HERE** and implement the missing code.

- Implement a sensitivity analysis of the electricity price.
- Plot the results in a figure.
- Interpret the results and make a recommendation in section 9.

```
# Test different price scaling factors
price_scales = 1.0:0.1:2.0  # From 100% to 200% in 10% steps
results_by_price = Dict()

# Create a copy of the original scenarios to modify
modified_scenarios = deepcopy(dictScenarios)


for scale in price_scales
```

```
    # HINT: Scale the market prices in each scenario. Use the modified_scenarios
     ↪ dictionary.
    ## YOUR CODE HERE

    # Solve the model with modified prices
    ## YOUR CODE HERE
end

# Create figure
fig4 = Figure(size=(1200, 800))

# Sort price scales and get corresponding results
sorted_scales = sort(collect(keys(results_by_price)))
sorted_results = [results_by_price[s] for s in sorted_scales]

# Calculate initial investment costs
initial_investment = [sum([
    dictStorage["storage"].energy_cost * r.e_nom["storage"] +
    dictStorage["storage"].power_cost * (r.p_ch_nom["storage"] + r.p_dis_nom["storage"])
 ↪  +
    dictWindTurbines["wind"].power_cost * r.p_w_nom["wind"] +
    dictPV["pv"].power_cost * r.p_v_nom["pv"]
]) for r in sorted_results]

# Component sizes vs price scale
ax1 = Axis(fig4[1, 1], title="Component Sizes vs Electricity Price")
lines!(ax1, sorted_scales, [r.e_nom["storage"] for r in sorted_results], label="Storage
 ↪  Energy")
lines!(ax1, sorted_scales, [r.p_w_nom["wind"] for r in sorted_results], label="Wind")
lines!(ax1, sorted_scales, [r.p_v_nom["pv"] for r in sorted_results], label="PV")
ax1.xlabel = "Price Scale Factor"
ax1.ylabel = "Capacity [MWh or MW]"
axislegend(ax1)

# Costs vs price scale
ax2 = Axis(fig4[1, 2], title="Costs vs Electricity Price")
lines!(ax2, sorted_scales, [r.total_cost for r in sorted_results], label="Total Annual
 ↪  Cost")
lines!(ax2, sorted_scales, [r.investment_costs["storage"] for r in sorted_results],
 ↪  label="Storage")
lines!(ax2, sorted_scales, [r.investment_costs["wind"] for r in sorted_results],
 ↪  label="Wind")
lines!(ax2, sorted_scales, [r.investment_costs["pv"] for r in sorted_results],
 ↪  label="PV")

# Add operational costs
operational_costs = [r.operational_costs["grid_import"] -
 ↪  r.operational_costs["grid_export"] for r in sorted_results]
lines!(ax2, sorted_scales, operational_costs, label="Operational Cost", linestyle=:dash)

ax2.xlabel = "Price Scale Factor"
ax2.ylabel = "Annual Cost [EUR/year]"
axislegend(ax2)
```

```
# Display figure
display(fig4)

# Print key findings
println("\nKey Findings from Price Sensitivity Analysis:")
println("1. Impact on Component Sizes:")
println("   - Storage capacity changes from $(round(sorted_results[1].e_nom["storage"],
 ↪  digits=2)) to $(round(sorted_results[end].e_nom["storage"], digits=2)) MWh")
println("   - Wind capacity changes from $(round(sorted_results[1].p_w_nom["wind"],
 ↪  digits=2)) to $(round(sorted_results[end].p_w_nom["wind"], digits=2)) MW")
println("   - PV capacity changes from $(round(sorted_results[1].p_v_nom["pv"],
 ↪  digits=2)) to $(round(sorted_results[end].p_v_nom["pv"], digits=2)) MW")

println("\n2. Impact on Costs:")
println("   - Total cost changes from $(round(sorted_results[1].total_cost, digits=2)) to
 ↪  $(round(sorted_results[end].total_cost, digits=2)) EUR/year")
println("   - Operational cost changes from $(round(operational_costs[1], digits=2)) to
 ↪  $(round(operational_costs[end], digits=2)) EUR/year")
println("   - Investment cost changes from $(round(sorted_results[1].total_cost -
 ↪  operational_costs[1], digits=2)) to $(round(sorted_results[end].total_cost -
 ↪  operational_costs[end], digits=2)) EUR/year")
```

9. Recommendations for Hamburg ChemPark GmbH

TASK: Place your recommendations here: X, Y, Z, W, A, B, C, D, E, F, G.

Based on our analysis, we can provide the following recommendations for Hamburg ChemPark GmbH for the expected case of a price scaled by 200%:

1. **Optimal System Configuration**:

```
   - Storage system with [X] MWh energy capacity and [Y] MW power capacity
   - Wind park with [Z] MW capacity
   - PV system with [W] MW capacity
```

2. **Economic Performance**:

```
   - Total annual cost: [A] EUR/year
   - Investment cost: [B] EUR/year
   - Operational cost: [C] EUR/year
```

3. **Findings**:

```
    - Significant storage capacity is added at an electricity price scaled by [D]% of the
     ↪  base case.
    - The annual investment cost increases from base case to 200% case from [E] EUR/year
     ↪  to [F] EUR/year, because the [G] for storage is much lower than for PV.
```

In a comprehensive sensitivity analysis, the following aspects should be considered:

- The solution is most sensitive to [parameter]
- Key uncertainties include [uncertainty]
- Robustness can be improved by [action]

> **Note**
>
> The model provides a foundation for decision-making, but additional factors should be considered:
>
> 1. Site-specific constraints
> 2. Grid connection capacity
> 3. Environmental ambitions
> 4. Maintenance requirements and costs
> 5. Future expansion possibilities

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.