CHALMERS
UNIVERSITY OF TECHNOLOGY

Project in TMA947 / MMG621 – Nonlinear optimization

Planning of electricity production and transmission

Data Science and Ai, Master Program

Yingtian Huang (huangyi@chalmers.se)
Tharinrath Jatupattrapiboorn (thajat@chalmers.se)
Pirapon Supakkeittikul (pirapon@chalmers.se)
Department of Computer Science and Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Project in TMA947 / MMG621 – Nonlinear optimization

Planning of electricity production and transmission

Yingtian Huang
Tharinrath Jatupattrapiboorn
Pirapon Supakkeittikul

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Project in TMA947 / MMG621 – Nonlinear optimization
Planning of electricity production and transmission
Yingtian Huang
Tharinrath Jatupattrapiboorn
Pirapon Supakkeittikul

Lecturer and examiner: Axel Ringh, Department of Mathematical Sciences

Project Report 2023
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Project in TMA947 / MMG621 – Nonlinear optimization
Planning of electricity production and transmission
Yingtian Huang
Tharinrath Jatupattrapiboorn
Pirapon Supakkeittikul
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

This report presents a model for optimizing electricity production and transmission with the objective of minimizing the cost of production with various constraints. The model was formulated and then implemented in Julia using the JuMP and Ipopt package and included with the objective function and all constraints to ensure that there is no power shortage occurred. The result is acceptable, the model can address the most efficient generator and utilize it most. Which resulted in the optimal cost of 183SEK, while satisfied all constraints present. Future work could be done to explore more on the real-world situation constrains such as the length between node and the maximum power flow limitation on each edges.

Keywords: Power System Optimization, Linear Programming, Electricity Production, Transmission Network, JuMP, Julia Programming, Cost Minimization, Power Balance, Mathematical Modeling.

Contents

List of Figures

List of Tables

1

Introduction

In this project, we derived a simplified model for planning production and transmission of electric power. While the real-world electricity grid may appear as a large-scale complex system, this model considers only a few components. Despite its simplicity, this model serves as an illustrative example of solving real-world problems. We have a figure of transmission network shown in Figure 1.1. There are 11 nodes in this network, and each node can be generator, consumer or both.

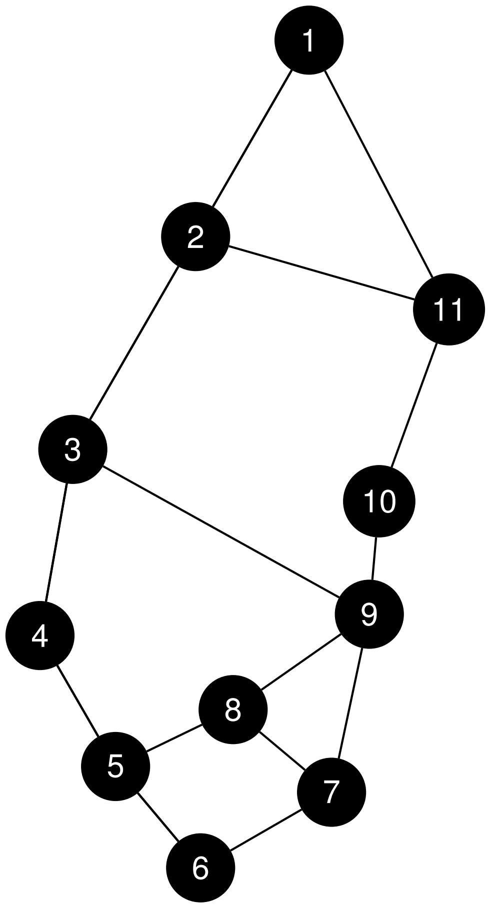

Figure 1.1: Illustration of the transmission as a network

2

Objective

Our objective is to plan the electric power production and transmission. The primary goal is to minimize the overall cost of power production in the grid, considering the constraints and parameters provided. The grid comprises 9 generators and 7 consumers, with their data provided in Table 2.1 and Table 2.2, including the location node, maximum capacity, and energy production cost for each generator, and location node and demand active power for each consumer.

Generator	Location Node	Maximum capacity [pu]	Energy production cost [pu]
G_1	2	0.02	175
G_2	2	0.15	100
G_3	2	0.08	150
G_4	3	0.07	150
G_5	4	0.04	300
G_6	5	0.17	350
G_7	7	0.17	400
G_8	9	0.26	300
G_9	9	0.05	200

Table 2.1: Parameters related to the 9 generators

Consumer	Location Node	Demand Active Power [pu]
C_1	1	0.10
C_2	4	0.19
C_3	6	0.11
C_4	8	0.09
C_5	9	0.21
C_6	10	0.05
C_7	11	0.04

Table 2.2: Parameters related to the 7 consumers

To optimize this problem, we have several constraints as follows:

1. **Active Power conservation:**
For each node with generators, the total active power produced by all generators in the node must be equal to the sum of the consumer's demand in the node combine with active power flow transmitted to adjacent nodes.

2. **Reactive Power conservation:**
For each node with generators, the total reactive power produced or absorb by all generators in the node must be equal to reactive power flow transmitted to adjacent nodes.

3. **Active/ Reactive Power Equation:**
Calculating power flow between nodes depends on associated voltage amplitude and voltage phase angle at each node. The voltage amplitude must be kept in the range of 0.98 normalized voltage units (vu) and 1.02 vu. The voltage phase angles are given radians and must be kept in the range of $-\pi$ and π . Besides these two variables, we have two more parameters b and g which describe the edges between the nodes and gives values to the formulation of power flow. All the values of b and g can be found in Table 2.3.

4. **Generation Capacity:**
Each generator can produce nonnegative active power up to a specific maximum capacity given in table x. In addition, the reactive power that each generator can produce or absorb is within plus/minus 0.3 percent of its maximum capacity.

These constraints are vital for achieving an optimal solution.

Edge (k, l)	(1,2)	(1,11)	(2,3)	(2,11)	(3,4)	(3,9)	(4,5)	(5,6)
b_{kl}	-20.1	-22.3	-16.8	-17.2	-11.7	-19.4	-10.8	-12.3
g_{kl}	4.12	5.67	2.41	2.78	1.98	3.23	1.59	1.71

Edge (k, l)	(5,8)	(6,7)	(7,8)	(7,9)	(8,9)	(9,10)	(10,11)
b_{kl}	-9.2	-13.9	-8.7	-11.3	-14.7	-13.5	-26.7
g_{kl}	1.26	1.11	1.32	2.01	2.41	2.14	5.06

Table 2.3: Values of parameters describing the edges. Note that $b_{kl} = b_{lk}$ and $g_{kl} = g_{lk}$

3

Mathematical formulation

3.1 Input

- N = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11} ; Set of all nodes in the network.
- E = $\{(k, l) \in$ N× N} ; Set of edges between each nodes.
- G = {1, 2, 3, 4, 5, 6, 7, 8, 9} ; Set of all generators in the network.
- D = {1, 2, 3, 4, 5, 6, 7} ; Set of all consumers in the network.

3.2 Decision Variables

- P_g: Active power generated by generator g
- Q_g: Reactive power generated by generator g
- v_k: Voltage amplitude at node k
- θ_k: Voltage phase angle at node k

3.3 Parameters

- Cap_g: Maximum capacity of generator g
- G_k: Generators that exist in node k
- D_k: Demands that exist in node k
- c_g: Cost of energy production per unit of generator g
- d_i: Demand of consumer i
- p_{kl}: Active power flow from node k to node l
- q_{kl}: Reactive power flow from node k to node l
- b_{kl}: Parameter for active power flow from node k to node l
- q_{kl}: Parameter for reactive power flow from node k to node l

3.4 Equation

$$p_{kl} = v_k^2 g_{kl} - v_k v_l g_{kl} \cos(\theta_k - \theta_l) - v_k v_l b_{kl} \sin(\theta_k - \theta_l) \tag{3.1}$$

$$q_{kl} = -v_k^2 b_{kl} + v_k v_l b_{kl} \cos(\theta_k - \theta_l) - v_k v_l g_{kl} \sin(\theta_k - \theta_l) \tag{3.2}$$

3.5 Objective Function

Minimize the total generation cost across the network.

$$\text{Minimize} \sum_{g \in G} c_g \cdot P_g \tag{3.3}$$

3.6 Constraints

1. **Active Power conservation:**

$$\sum_{g \in G_k} P_g = \sum_{i \in D_k} d_i + \sum_{(k,l) \in E} p_{kl}, \quad \forall k \in N \tag{3.4}$$

2. **Reactive Power conservation:**

$$\sum_{g \in G_k} Q_g = \sum_{(k,l) \in E} q_{kl}, \quad \forall k \in N \tag{3.5}$$

3. **Generation Capacity:**

$$0 \leq P_g \leq Cap_g, \quad \forall g \in G \tag{3.6}$$

$$-0.003Cap_g \leq Q_g \leq 0.003Cap_g, \quad \forall g \in G \tag{3.7}$$

4. **Voltage Magnitude:**

$$0.98 \leq v_k \leq 1.02, \quad \forall k \in N \tag{3.8}$$

5. **Phase Angle:**

$$-\pi \leq \theta_k \leq \pi, \quad \forall k \in N \tag{3.9}$$

4

Result and analysis

4.1 Result

The total cost of power production is evaluated to be 186.29 SEK per Unit. The amount of active and reactive power generated by each generator is detailed in Table 4.1, which displays the location node, generator, and the corresponding generated power. The voltage amplitudes and the voltage phase angles at each nodes are provided in Table 4.2, organizing the location node, voltage amplitude, and voltage phase angle for each node. The flow of active and reactive power along the edges of the network is depicted in Figure 4.1 and outlined in Table 4.3.

Regarding the optimality of the solution, it is found to be locally optimal. Given that neither the function nor the constraints exhibit convexity, the problem itself is non-convex. As a result, we cannot assert that the solution is globally optimal.

Generator	Location Node	Active Power [pu]	Reactive Power [pu]
G_1	2	0.00487	0.00006
G_2	2	0.15000	0.00045
G_3	2	0.08000	0.00024
G_4	3	0.07000	0.00021
G_5	4	0.04000	0.00012
G_6	5	0.13886	0.00051
G_7	7	0.00308	0.00051
G_8	9	0.25368	0.00078
G_9	9	0.05000	0.00015

Table 4.1: Optimal Active Power and Reactive Power Generated by Each Generator

Location Node	Voltage Amplitude [vu]	Phase Angle [radians]
1	1.01913	0.00181
2	1.02000	0.00626
3	1.01949	0.00301
4	1.01814	-0.00484
5	1.01887	-0.00031
6	1.01821	-0.00530
7	1.01853	-0.00219
8	1.01852	-0.00256
9	1.01922	0.00157
10	1.01901	0.00064
11	1.01923	0.00192

Table 4.2: Optimal Voltage Amplitude and Phase Angle for Each Node

Edges	Active Power Flow [pu]	Reactive Power Flow [pu]
(1, 2)	-0.09677	0.00153
(1, 11)	-0.00323	-0.00153
(2, 1)	0.09685	-0.00110
(2, 3)	0.05814	0.00072
(2, 11)	0.07988	0.00113
(3, 2)	-0.05811	-0.00053
(3, 4)	0.09815	0.00027
(3, 9)	0.02997	0.00047
(4, 3)	-0.09802	0.00050
(4, 5)	-0.05198	-0.00038
(5, 4)	0.05202	0.00062
(5, 6)	0.06487	-0.00048
(5, 8)	0.02197	0.00038
(6, 5)	-0.06482	0.00081
(6, 7)	-0.04518	-0.00081
(7, 6)	0.04519	0.00095
(7, 8)	0.00336	-0.00040
(7, 9)	-0.04547	-0.00004
(8, 5)	-0.02196	-0.00033
(8, 7)	-0.00336	0.00040
(8, 9)	-0.06468	-0.00007
(9, 3)	-0.02996	-0.00043
(9, 7)	0.04550	0.00021
(9, 8)	0.06472	0.00034
(9, 10)	0.01342	0.00081
(10, 9)	-0.01342	-0.00080
(10, 11)	-0.03658	0.00080
(11, 1)	0.00323	0.00153
(11, 2)	-0.07982	-0.00078
(11, 10)	0.03659	-0.00075

Table 4.3: Power Flow along the Edge

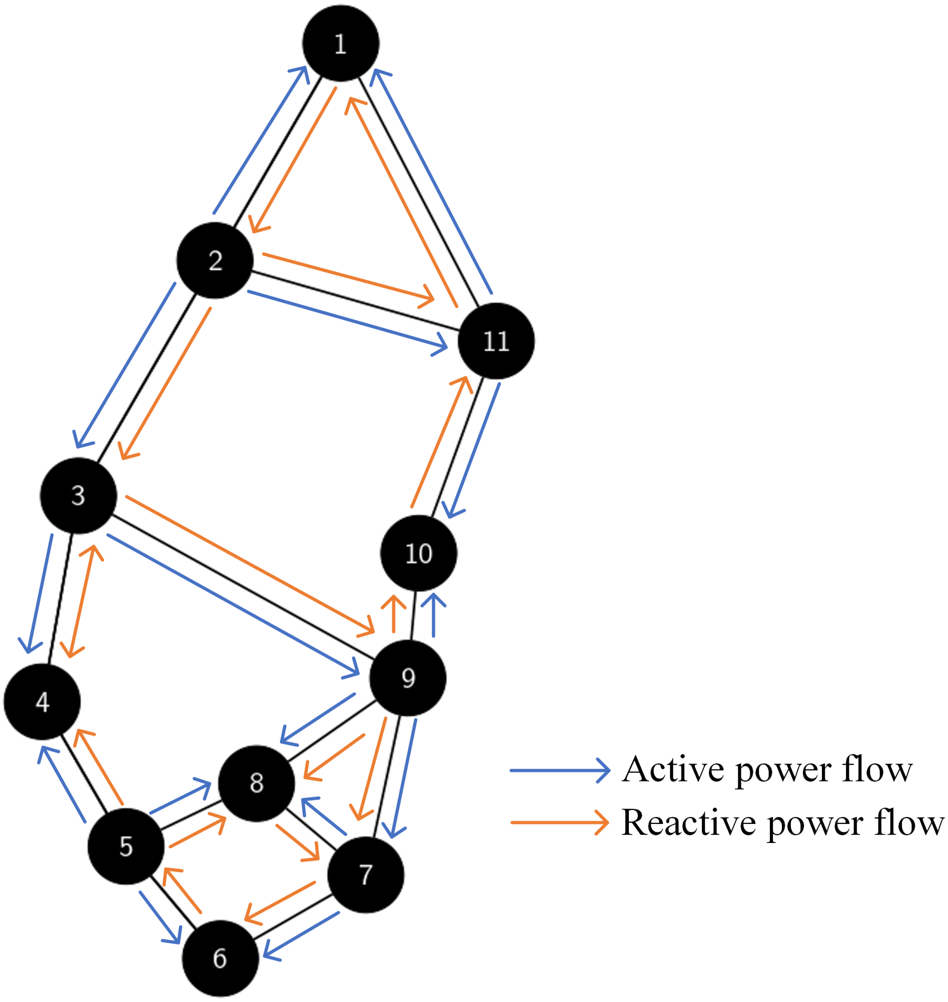

Figure 4.1: The direction of the arrows indicate the direction of active and reactive power flow

4.2 Analysis

In order to determine which generator would be most effective for increasing in capacity of 0.01 power unit with the goal of minimizing the cost of power production. We should take a look at the Lagrange multipliers corresponding to each constraints of the maximum capacity. Because by picking the generator with the most negative Lagrange multipliers, It will resulted in the largest cost reduction. And as seen from the result in table 4.4, we choose Generator 5 in this case.

Gen1	Gen2	Gen3	Gen4	Gen5	Gen6	Gen7	Gen8	Gen9
0.0	-75.0	-25.0	-111.665	-170.666	-2.561e-7	0.0	-2.259e-6	-100.0

Table 4.4: Dual variables/Lagrange multipliers corresponding to active power

5

Summary and conclusions

For the objective of minimize the cost of energy production for the Swedish international power grid. We have formulated a model, which contains an objective function and derived the total cost of power production with the prescribed constraints.

The project gave us a hands-on experiences on how to formulate a model and constraints for the real-world situation.

The solution is locally optimal and not globally optimal because of the fact that some constraints are not convex and achieving global optimality maybe computationally intricate. However, the suggestion is to try another solver and simplifies the model.

Overall, this model is acceptable since it prioritized the utilization of the generator by its efficiency. But we do need to keep in mind, that there's so much more constraints in the real-world which include limitation of power flow in the transmission line, the production of unplannable generator, etc.

A

Appendix 1

```
using JuMP
using Ipopt

# input node information
node_info = Dict(
    1 => Dict(
        "generators" => Dict(
            )
        ,
        "customers" => Dict(
            "Cust1" => 0.10
        )
    ),
    2 => Dict(
        "generators" => Dict(
            "Gen1" => Dict(
                "capacity" => 0.02,
                "cost" => 175
            ),
            "Gen2" => Dict(
                "capacity" => 0.15,
                "cost" => 100
            ),
            "Gen3" => Dict(
                "capacity" => 0.08,
                "cost" => 150
            )
        ),
        "customers" => Dict(
        )
    ),
    3 => Dict(
        "generators" => Dict(
            "Gen4" => Dict(
                "capacity" => 0.07,
                "cost" => 150
            )
        ),
        "customers" => Dict(
        )
    ),
    4 => Dict(
        "generators" => Dict(
            "Gen5" => Dict(
```

```
                "capacity" => 0.04,
                "cost" => 300
            )
        ),
        "customers" => Dict(
            "Cust2" => 0.19
        )
    ),
    5 => Dict(
        "generators" => Dict(
            "Gen6" => Dict(
                "capacity" => 0.17,
                "cost" => 350
            )
        ),
        "customers" => Dict(
        )
    ),
    6 => Dict(
        "generators" => Dict(
        ),
        "customers" => Dict(
            "Cust3" => 0.11
        )
    ),
    7 => Dict(
        "generators" => Dict(
            "Gen7" => Dict(
                "capacity" => 0.17,
                "cost" => 400
            )
        ),
        "customers" => Dict(
        )
    ),
    8 => Dict(
        "generators" => Dict(
        ),
        "customers" => Dict(
            "Cust4" => 0.09
        )
    ),
    9 => Dict(
        "generators" => Dict(
            "Gen8" => Dict(
                "capacity" => 0.26,
                "cost" => 300
            ),
            "Gen9" => Dict(
                "capacity" => 0.05,
                "cost" => 200
            )
        ),
        "customers" => Dict(
            "Cust5" => 0.21
        )
```

```
    ),
    10 => Dict(
        "generators" => Dict(
        ),
        "customers" => Dict(
            "Cust6" => 0.05
        )
    ),
    11 => Dict(
        "generators" => Dict(
        ),
        "customers" => Dict(
            "Cust7" => 0.04
        )
    )
)

edges = Dict{Tuple{Int, Int}, Dict{String, Float64}}()

# create an array for edges
for k in (1:length(node_info))
    for l in (1:length(node_info))
        # Set default value for b,g to 0.0 for all edges
        edges[(k, l)] = Dict("b" => 0.0,"g" => 0.0)
    end
end

# input value of b,g for existing edges
edges[1, 2] = Dict("b" => -20.1,"g" => 4.12)
edges[2, 1] = Dict("b" => -20.1,"g" => 4.12)
edges[1, 11] = Dict("b" => -22.3,"g" => 5.67)
edges[11, 1] = Dict("b" => -22.3,"g" => 5.67)
edges[2, 3] = Dict("b" => -16.8,"g" => 2.41)
edges[3, 2] = Dict("b" => -16.8,"g" => 2.41)
edges[2, 11] = Dict("b" => -17.2,"g" => 2.78)
edges[11, 2] = Dict("b" => -17.2,"g" => 2.78)
edges[3, 4] = Dict("b" => -11.7,"g" => 1.98)
edges[4, 3] = Dict("b" => -11.7,"g" => 1.98)
edges[3, 9] = Dict("b" => -19.4,"g" => 3.23)
edges[9, 3] = Dict("b" => -19.4,"g" => 3.23)
edges[4, 5] = Dict("b" => -10.8,"g" => 1.59)
edges[5, 4] = Dict("b" => -10.8,"g" => 1.59)
edges[5, 6] = Dict("b" => -12.3,"g" => 1.71)
edges[6, 5] = Dict("b" => -12.3,"g" => 1.71)
edges[5, 8] = Dict("b" => -9.2,"g" => 1.26)
edges[8, 5] = Dict("b" => -9.2,"g" => 1.26)
edges[6, 7] = Dict("b" => -13.9,"g" => 1.11)
edges[7, 6] = Dict("b" => -13.9,"g" => 1.11)
edges[7, 8] = Dict("b" => -8.7,"g" => 1.32)
edges[8, 7] = Dict("b" => -8.7,"g" => 1.32)
edges[7, 9] = Dict("b" => -11.3,"g" => 2.01)
edges[9, 7] = Dict("b" => -11.3,"g" => 2.01)
edges[8, 9] = Dict("b" => -14.7,"g" => 2.41)
edges[9, 8] = Dict("b" => -14.7,"g" => 2.41)
edges[9, 10] = Dict("b" => -13.5,"g" => 2.14)
edges[10, 9] = Dict("b" => -13.5,"g" => 2.14)
```

```
edges[10, 11] = Dict("b" => -26.7,"g" => 5.06)
edges[11, 10] = Dict("b" => -26.7,"g" => 5.06)

# Define a function for flow of active power
function pkl(k, l, v, theta)
    return v[k]^2 * edges[k, l]["g"] - v[k] * v[l]
    * edges[k, l]["g"] * cos(theta[k] - theta[l]) -
    v[k] * v[l] * edges[k, l]["b"] * sin(theta[k] - theta[l])
  end

# Define a function for flow of reactive power
  function qkl(k, l, v, theta)
    return -v[k]^2 * edges[k, l]["b"] + v[k] * v[l]
    * edges[k, l]["b"] * cos(theta[k] - theta[l]) -
    v[k] * v[l] * edges[k, l]["g"] * sin(theta[k] - theta[l])
  end

# Define the optimization model
model = Model(optimizer_with_attributes(Ipopt.Optimizer, "print_level" => 0))

# Decision Variables

# Active Power generated
@variable(model, 0 <= active_gen[node = 1:length(node_info),
gen_id = keys(node_info[node]["generators"])]
<= node_info[node]["generators"][gen_id]["capacity"])
# Reactive Power generated
@variable(model, -0.003 * node_info[node]["generators"][gen_id]["capacity"]
<= reactive_gen[node = 1:length(node_info), gen_id =
keys(node_info[node]["generators"])]
<= 0.003 * node_info[node]["generators"][gen_id]["capacity"])
# Voltage Amplitude in each node
@variable(model, 0.98 <= v[node = 1:length(node_info)] <= 1.02)
# Voltage Angles in each node
@variable(model, -pi <= theta[node = 1:length(node_info)] <= pi)


# Objective Function
@objective(model, Min, sum(active_gen[node, gen_id] *
get(node_info[node]["generators"][gen_id], "cost", 0)
for (node, gen_id) in eachindex(active_gen)))

# Active power Constraints
for k in (1:length(node_info))
    generators = node_info[k]["generators"]
    demands = 0
    for value in values(get(node_info[k],"customers",Dict()))
        demands += value
    end
    @NLconstraint(model,
    sum(active_gen[k, gen_id] for gen_id in keys(generators)) ==
    (sum(v[k]^2 * edges[k, l]["g"] - v[k] * v[l] * edges[k, l]["g"] *
    cos(theta[k] - theta[l]) - v[k] * v[l] * edges[k, l]["b"]
    * sin(theta[k] - theta[l]) for l in (1:length(node_info))) + demands)
    )
end
```

```
# Reactive Power Constraints
for k in (1:length(node_info))
    generators = node_info[k]["generators"]
    @NLconstraint(model,
    sum(reactive_gen[k, gen_id] for gen_id in keys(generators)) ==
    sum(-v[k]^2 * edges[k, l]["b"] + v[k] * v[l] * edges[k, l]["b"]
    * cos(theta[k] - theta[l]) - v[k] * v[l] * edges[k, l]["g"]
    * sin(theta[k] - theta[l]) for l in (1:length(node_info)))
    )
end

# Solve the optimization problem
optimize!(model)

# print the optimzation problem in the terminal
println("Model info:")
show(model)

println("\n----------------------------------------")

println("Model details:")
println(model)

println("\n----------------------------------------")

println("Termination status: ", JuMP.termination_status(model))
println("Objectjective function value: ", JuMP.objective_value(model))

println("\n----------------------------------------")

println("Optimal voltage amplitude for each node: ")
println(JuMP.value.(v))
println("Optimal voltage phase angle for each node: ")
println(JuMP.value.(theta))
println("Optimal active power generated by each generator: ")
println(JuMP.value.(active_gen))
println("Optimal rective power generated by each generator: ")
println(JuMP.value.(reactive_gen))

println("\n----------------------------------------")

# print the amount of active power and reactive power flowing in each edge
println("Active power flowing in each edge:
**Will not show the edge that has no active power flow")
for k in (1:length(node_info))
    for l in (1:length(node_info))
        if pkl(k, l, value.(v), value.(theta)) != 0
            println("p[", k, ",", l, "] =", pkl(k, l, value.(v), value.(theta)))
        else
            continue
        end
    end
end

println("\n----------------------------------------")
```

```
println("Reactive power flowing in each edge:
**Will not show the edge that has no reactive power flow")
for k in (1:length(node_info))
    for l in (1:length(node_info))
        if qkl(k, l, value.(v), value.(theta)) != 0
            println("q[", k, ",", l, "] =", qkl(k, l, value.(v), value.(theta)))
        else
            continue
        end
    end
end

println("\n------------------------------------------------")

println("Dual variables/Lagrange multipliers corresponding to active power
constraints: ")
println(JuMP.dual.(JuMP.UpperBoundRef.(active_gen)))
```

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY