power_system_optimization

April 8, 2022

1 Optimization of a power system

Student: Bruno Kiyoshi Ynumaru; Prof. Eduardo Camponogara

In this exercise, you will have the opportunity to approximate a nonlinear problem as a MILP by means of piecewise-linear models. The power system has 3 buses as depicted in Fig. 1. Buses 1 and 3 have generation units, whereas bus 2 is a power consumer. This Figures also gives the maximum power generation ($\bar{P_{gi}}$) and the power consumption ($\bar{P_{di}}$) of each bus i, under low and high demand. The properties of the transmission lines appear in Table 1, in pu using a 100 MVA basis, are indicated in the Figure. The parameters are the resistance $r_{i,j}$, the reactance $x_{i,j}$ and the number of lines installed between buses i and j.

Line	$r_{i,j}$ (pu)	$x_{i,j}$ (pu)	ni,j
1-2	0.030	0.23	2
1-3	0.035	0.25	1
2-3	0.025	0.20	1

A simplified model is adopted for the transmission network, in which:
•Lines and transformers are represented by their series impedances in per unit:
$z_{i,j} = r_{i,j} + jx_{i,j}(1)$
where ri;j is the resistance and xi;j is the reactance of line (i; j).
• Voltage magnitudes are fixed at 1:0 pu.
• Reactive power balance is supposed to be satisfied.
With these assumptions, active power flows are expressed as
$p.flowi \rightarrow j : P_{i,j} = g_{i,j} - (g_{i,j}cos\theta_{i,j} + b_{i,j}sin\theta_{i,j})(2a)$
$p.flowj \rightarrow i : P_{j,i} = g_{i,j} - (g_{i,j}cos\theta_{i,j} - b_{i,j}sin\theta_{i,j})(2b)$
where gij and b_{ij} are, respectively, the series conductance and series susceptance of line (i, j), $\theta_{i,j} = (\theta_i - \theta_j)$ and θ_i is the voltage angle of bus i.
Conductance and susceptance are calculated as follows

$$g_{i,j} = \frac{r_{i,j}}{(r_{i,j}^2 + x_{i,j}^2)}$$

$$b_{i,j} = -\frac{x_{i,j}}{(r_{i,j}^2 + x_{i,j}^2)}$$

The power injected into bus i is defined as

$$P_i = \sum_{j \in N_i} n_{i,j}P_{i,j} = \sum_{j \in N_i} n_{i,j}[g_{i,j} - (g_{i,j}cos\theta_{i,j} + b_{i,j}sin\theta_{i,j}])](3)$$

where N_i is the set of neighboring buses of bus i. To ensure energy conservation, the following equations must also be satisfied
$Pg_i = \bar{Pd}_i + P_i (4)$
Aiming to minimize the power loss in transmission, the power-flow optimization problem could be solved:

$$\begin{aligned} min \sum_{i \in N} |P_i| \\ s.t : Pg_i = \bar{Pd}_i + P_i, \\ P_i = \sum_{j \in N_i} n_{i,j} P_{i,j}, \\ 0 \leq Pg_i \leq \bar{Pg}_i, \\ \theta_i \in [\frac{-\pi}{2}, \frac{\pi}{2}, i \in N], \\ P_{i,j} = g_{i,j} - (g_{i,j} cos\theta_{i,j} + b_{i,j} sin\theta_{i,j}), \\ \theta_{i,j} = \theta_i - \theta_j, \\ \theta_{i,j} \in [-\pi, \pi], i \in N, j \in N_i \end{aligned}$$

Tasks:
• Reformulate the power-flow optimization problem in MILP using the following piecewise-linear models: CC and SOS2.
• Implement the models in AMPL, choosing a suitable number of breakpoints to induce a good approximation of the power-flow equations. You may plot the piecewise linear approximations for sin $\theta_{i,j}$ and cos $\theta_{i,j}$ in order to show the degree of approximation.
• Solve the problem for the low and high power demand cases. Present and illustrate the solutions.

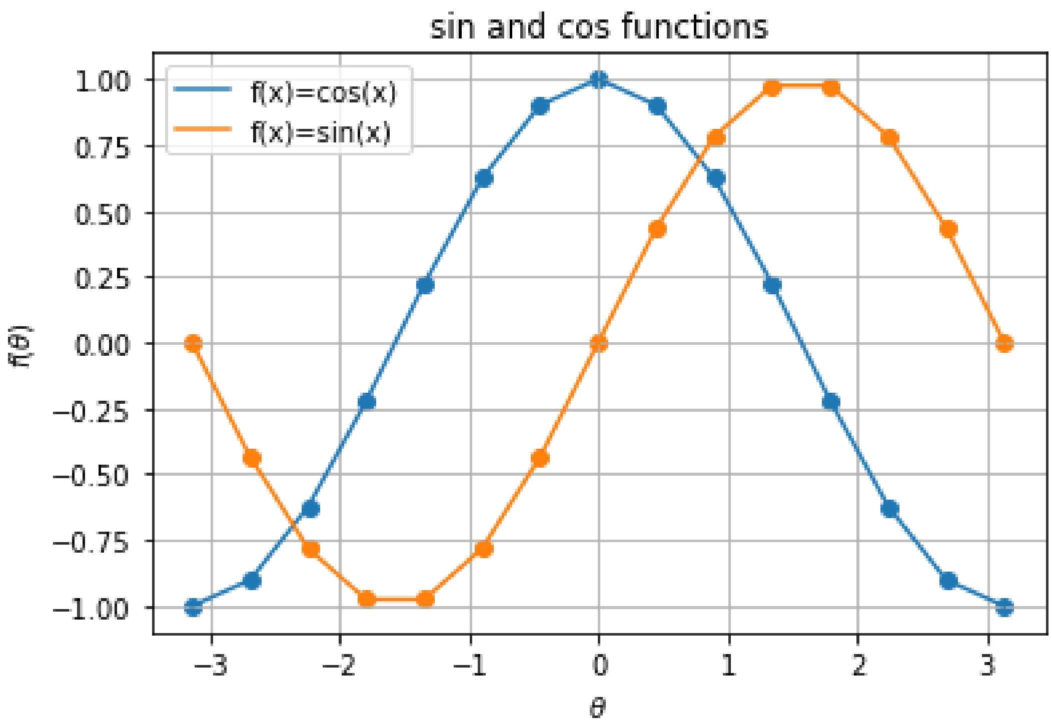
1.1 Optimization problem solution using Gurobi

[1]:
```
import gurobipy as gp
from gurobipy import GRB, Model
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math
from itertools import combinations
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', None)
```

1.2 Generate data points

[2]:
```
n_points = 15
theta_array = np.linspace(-math.pi,math.pi, n_points)
cos_array = np.cos(theta_array)
sin_array = np.sin(theta_array)
plt.figure()
```

```
plt.scatter(theta_array,cos_array)
plt.scatter(theta_array,sin_array)
plt.plot(theta_array,cos_array,label="f(x)=cos(x)")
plt.plot(theta_array,sin_array,label="f(x)=sin(x)")
plt.title("sin and cos functions")
plt.xlabel(r'$\theta$')
plt.ylabel(r'$f(\theta)$')
plt.legend()
plt.grid(which="both")
plt.show()
```



[3]:
```
n_busses = 3
bus_set = [i for i in range(1,n_busses + 1)]
print(bus_set)
```

```
[1, 2, 3]
```

[4]:
```
# Create dictionaries containing the data provided in problem statement
Pd_dict_high = {bus_set[0]:1.0,bus_set[1]:3.5,bus_set[2]:5.0} # Power demand -␣
↪high
Pd_dict_low = {bus_set[0]:1.0,bus_set[1]:2.0,bus_set[2]:3.0} # Power demand - low
Pgmax_dict = {bus_set[0]:7.0,bus_set[1]:0.0,bus_set[2]:4.0} # Max power␣
↪generation
```

```
print(f'Pd_dict_high={Pd_dict_high}')
print(f'Pd_dict_low={Pd_dict_low}')
print(f'Pgmax_dict={Pgmax_dict}')
```

```
Pd_dict_high={1: 1.0, 2: 3.5, 3: 5.0}
Pd_dict_low={1: 1.0, 2: 2.0, 3: 3.0}
Pgmax_dict={1: 7.0, 2: 0.0, 3: 4.0}
```

[5]:
```
b_combs = combinations(bus_set,2) # connections between busses
b_combs = list(b_combs)
print(f"bus pairs:{b_combs}")
```

```
bus pairs:[(1, 2), (1, 3), (2, 3)]
```

[6]:
```
# More data from problem statement
r_dict = {b_combs[0]:0.030, b_combs[1]:0.035, b_combs[2]:0.025}
x_dict = {b_combs[0]:0.23, b_combs[1]:0.25, b_combs[2]:0.20}
n_dict = {b_combs[0]:2, b_combs[1]:1, b_combs[2]:1}
print(f'r_dict={r_dict}')
print(f'x_dict={x_dict}')
print(f'n_dict={n_dict}')

# Calculate z values and assign values to a dict:
z = lambda r,x : r + x*1j
z_dict = {pair:z(r_dict[pair], x_dict[pair]) for pair in b_combs}
print(f'z_dict={z_dict}')
```

```
r_dict={(1, 2): 0.03, (1, 3): 0.035, (2, 3): 0.025}
x_dict={(1, 2): 0.23, (1, 3): 0.25, (2, 3): 0.2}
n_dict={(1, 2): 2, (1, 3): 1, (2, 3): 1}
z_dict={(1, 2): (0.03+0.23j), (1, 3): (0.035+0.25j), (2, 3): (0.025+0.2j)}
```

[7]:
```
# Calculate g and b values and assign to dict:
g = lambda r,x: r / (r**2+x**2)
b = lambda r,x: -x / (r**2+x**2)

g_dict = {pair:g(r_dict[pair], x_dict[pair]) for pair in b_combs}
b_dict = {pair:b(r_dict[pair], x_dict[pair]) for pair in b_combs}
print(f'g_dict={g_dict}')
print(f'b_dict={b_dict}')
```

```
g_dict={(1, 2): 0.5576208178438662, (1, 3): 0.5492349941153394, (2, 3):
0.6153846153846153}
b_dict={(1, 2): -4.275092936802974, (1, 3): -3.9231071008238523, (2, 3):
-4.9230769230769225}
```

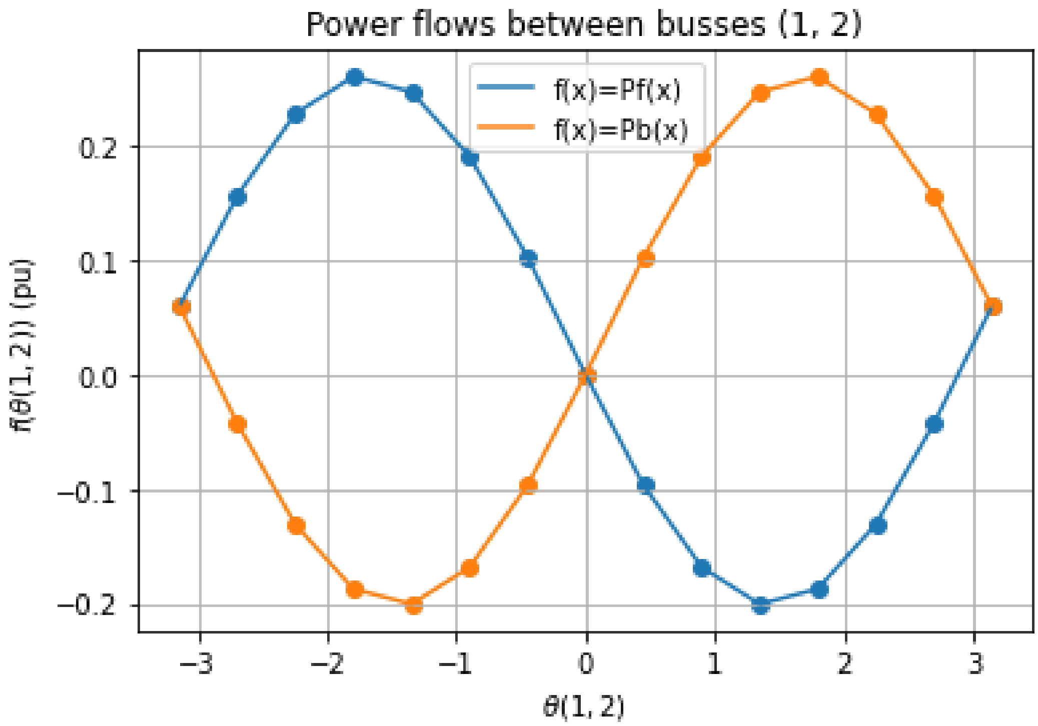
[8]:
```
# power flow (P) has different values for forwards (f-Pij) and backwards (b-Pji)␣
↪directions:
Pf = lambda g,b,theta: (g - (g*np.cos(theta) + b*np.sin(theta))) # Pij
```

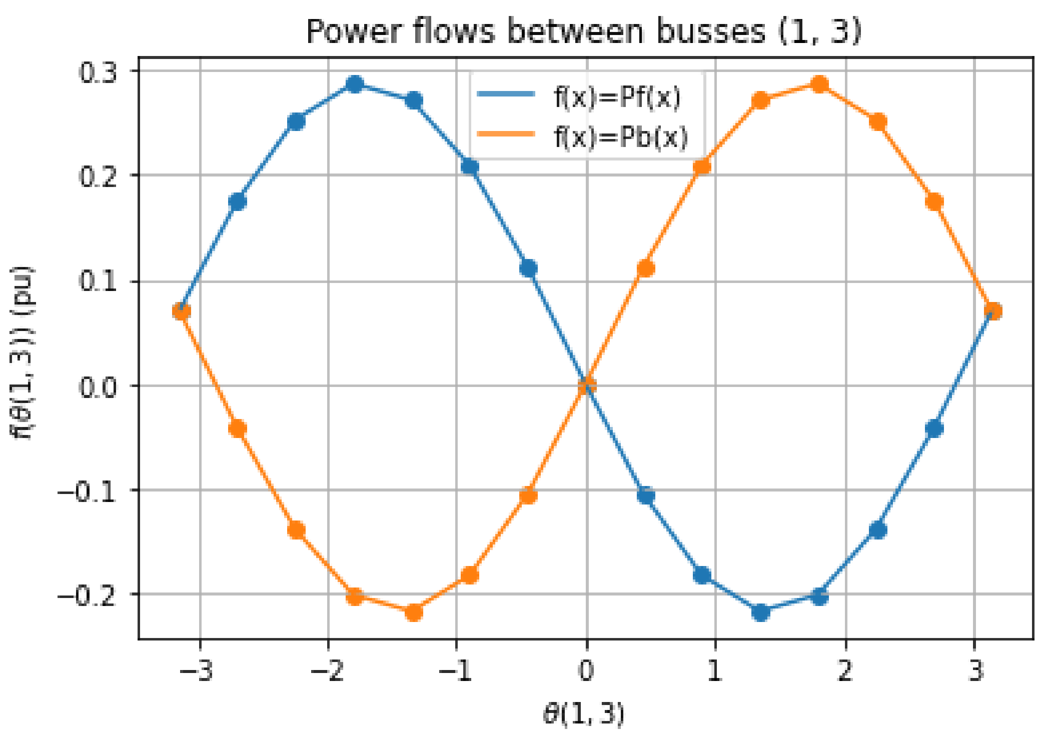
```
Pb = lambda g,b,theta: (g - (g*np.cos(theta) - b*np.sin(theta))) # Pji

Pf_dict = {pair:Pf(r_dict[pair], x_dict[pair], theta_array) for pair in b_combs}
print(f'Pf_dict={Pf_dict}')
Pb_dict = {pair:Pb(r_dict[pair], x_dict[pair], theta_array) for pair in b_combs}
print(f'Pb_dict={Pb_dict}')
```

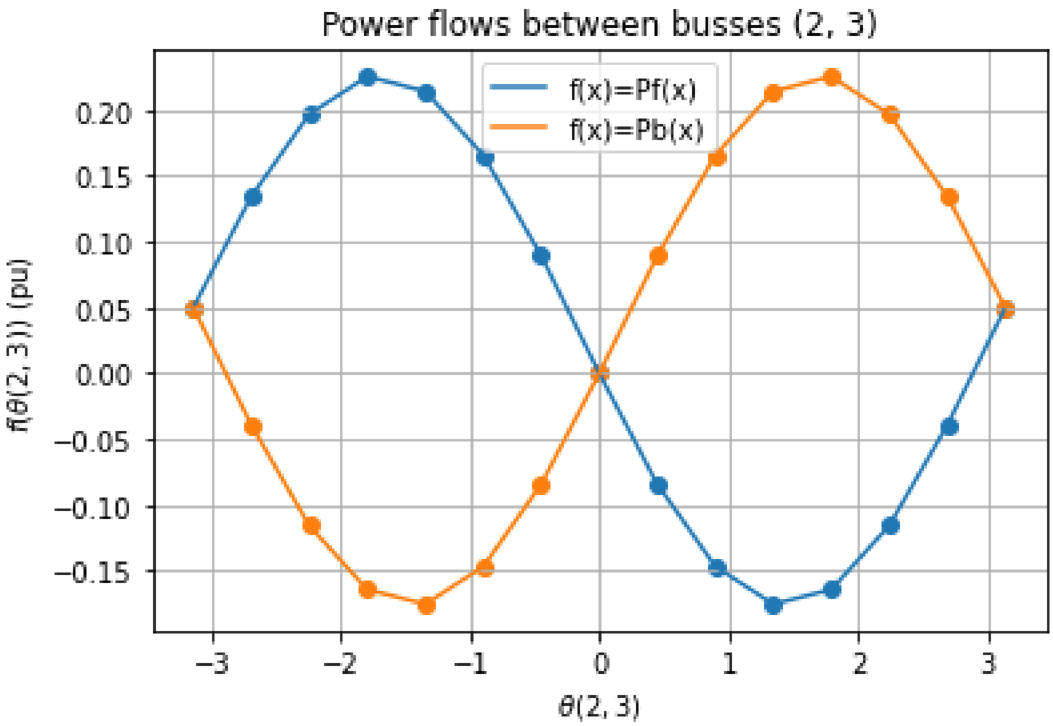
```
Pf_dict={(1, 2): array([ 0.06      ,  0.15682233,  0.22852594,  0.26090905,
0.24755779,
        0.19111655,  0.10276419,  0.        , -0.09682233, -0.16852594,
       -0.20090905, -0.18755779, -0.13111655, -0.04276419,  0.06      ]), (1,
3): array([ 0.07      ,  0.17500485,  0.25228001,  0.28652021,  0.27094375,
        0.20863573,  0.11193702,  0.        , -0.10500485, -0.18228001,
       -0.21652021, -0.20094375, -0.13863573, -0.04193702,  0.07      ]), (2,
3): array([ 0.05      ,  0.13430097,  0.19695354,  0.22554861,  0.21442256,
        0.16577905,  0.08925253,  0.        , -0.08430097, -0.14695354,
       -0.17554861, -0.16442256, -0.11577905, -0.03925253,  0.05      ])}
Pb_dict={(1, 2): array([ 0.06      , -0.04276419, -0.13111655, -0.18755779,
-0.20090905,
       -0.16852594, -0.09682233,  0.        ,  0.10276419,  0.19111655,
        0.24755779,  0.26090905,  0.22852594,  0.15682233,  0.06      ]), (1,
3): array([ 0.07      , -0.04193702, -0.13863573, -0.20094375, -0.21652021,
       -0.18228001, -0.10500485,  0.        ,  0.11193702,  0.20863573,
        0.27094375,  0.28652021,  0.25228001,  0.17500485,  0.07      ]), (2,
3): array([ 0.05      , -0.03925253, -0.11577905, -0.16442256, -0.17554861,
       -0.14695354, -0.08430097,  0.        ,  0.08925253,  0.16577905,
        0.21442256,  0.22554861,  0.19695354,  0.13430097,  0.05      ])}
```

[9]:

```
for k, bc in enumerate(b_combs):

    plt.figure()
    plt.scatter(theta_array,Pf_dict[bc])
    plt.scatter(theta_array,Pb_dict[bc])
    plt.plot(theta_array,Pf_dict[bc],label="f(x)=Pf(x)")
    plt.plot(theta_array,Pb_dict[bc],label="f(x)=Pb(x)")
    plt.title(f"Power flows between busses {bc}")
    plt.xlabel(fr'$\theta{bc}$')
    plt.ylabel(fr'$f(\theta{bc})$ (pu)')
    plt.legend()
    plt.grid(which="both")
    plt.show()
```


Power flows between busses (1, 2)
f(x)=Pf(x)
f(x)=Pb(x)
$f\theta(1,2))$ (pu)
$\theta(1,2)$


Power flows between busses (1, 3)
f(x)=Pf(x)
f(x)=Pb(x)
$f\theta(1,3))$ (pu)
$\theta(1,3)$


Power flows between busses (2, 3)
f(x)=Pf(x)
f(x)=Pb(x)
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−3
−2
−1
0
1
2
3
$\theta(2,3)$
$f(\theta(2,3))$ (pu)

1.3 Piecewise linear reformulations and solving

1.3.1 Convex-combination - CC

In convex combination approximations, the dependent variable is given by a linear combination of two consecutive points from the available data: For a given pair of buses (i, j), the CC piecewise linear reformulation is:

$$
\begin{aligned}
(i, j) \in \{1, \ldots, N\}^2, i \neq j \\
Givens^{(i,j)} : \{(\theta_{k=0}^{(i,j)}, P_{k=0}^{(i,j)}), \ldots, (\theta_{(M-1)}^{(i,j)}, P_{(M-1)}^{(i,j)})\} \\
\theta^{(i,j)} = \sum_{k=0}^{M-1} \lambda_k^{(i,j)} \theta_k^{(i,j)} \\
P^{(i,j)} = \sum_{k=0}^{M-1} \lambda_k^{(i,j)} P_k^{(i,j)} \\
\lambda_k^{(i,j)} \geq 1, k = 0, ..., (M-1) \\
1 = \sum_{k=0}^{M-1} \lambda_k^{(i,j)} \\
1 = \sum_{k=1}^{M-1} z_k^{(i,j)} \\
z_k^{(i,j)} \in \{0, 1\}, k = 1, ..., (M-1) \\
\lambda_0^n \leq z_1^{(i,j)} \\
\lambda_{(M-1)}^{(i,j)} \leq z_{(M-1)}^{(i,j)} \\
\lambda_k^{(i,j)} \leq z_k^{(i,j)} + z_{k+1}, k = 1, ..., (M-1) \\
z_k^{(i,j)} = 1 \text{ if } P^{(i,j)} \in [P_{k-1}^{(i,j)}, P_k^{(i,j)}]
\end{aligned}
$$

Where M is the number of available data points for bus pair (i, j) and N is the number of buses in the system.

1.3.2 Convex Combination in Gurobi

[10]:
```
cc = Model()
cc.Params.LogFile = "cc_log.txt"
# each bus i is in a certain theta
thetai_vars = cc.addVars(bus_set, lb=-math.pi/2, ub=math.pi/2, vtype=GRB.
    ↪CONTINUOUS, name="thetai")

# power generated in each bus i, constrained by values given in problem statement
Pgi_vars = cc.addVars(bus_set, lb=0,ub=Pgmax_dict,vtype=GRB.CONTINUOUS,␣
    ↪name="Pgi")
cc.update()
print([(f"Pg{Pgi}",Pgi_vars[Pgi].ub) for Pgi in Pgi_vars])
# Pgi_vars = cc.addVars(bus_set, lb=0,vtype=GRB.CONTINUOUS, name="Pgi") # This␣
    ↪would be a relaxation
```

```python
# total power flow into each bus i
Pi_vars = cc.addVars(bus_set, vtype=GRB.CONTINUOUS, name="Pi")

# auxiliary variables which represents each |Pi|
Pi_abs_vars = cc.addVars(bus_set, vtype=GRB.CONTINUOUS, name="Pi_abs")

lambda_vars = gp.tupledict() # 1 value per data interval per pair ((n_points-1) * len(b_combs))
z_vars = gp.tupledict() # 1 value per data point per pair (n_points * len(b_combs))
lag_vars = gp.tupledict() # 1 value per pair (len(b_combs))
Pf_vars = gp.tupledict() # 1 value per pair (len(b_combs))
Pb_vars = gp.tupledict() # 1 value per pair (len(b_combs))

data_indexes = range(n_points)

for pair in b_combs:
    # create lambdas for current pair
    lambda_vars_ = cc.addVars(data_indexes, lb=0.0, vtype=GRB.CONTINUOUS, name=f"lambdas{pair}")
    cc.addConstr(gp.quicksum(lambda_vars_)==1)
    lambda_vars[pair] = lambda_vars_

    # z variables indicate in which piece of the piecewise linear function the decision variable is
    # create z variables for current pair
    z_vars_ = cc.addVars(data_indexes[1:], vtype=GRB.BINARY, name=f"z{pair}") # note that first index is removed
    cc.addConstr(gp.quicksum(z_vars_)==1)
    z_vars[pair] = z_vars_

    cc.addConstr(lambda_vars_[0]<=z_vars_[1]) # ensure first lambda =0 if decision variable outside of first piece
    cc.addConstr(lambda_vars_[n_points-1]<=z_vars_[n_points-1]) # ensure last lambda =0 if decision variable outside of last piece

    for i in range(1,n_points-1):
        cc.addConstr(lambda_vars_[i]<=z_vars_[i]+z_vars_[i+1]) # ensure lambda=0 if decision variable not in any of the neighboring pieces

    # create lag variable for current pair
    lag_var = cc.addVar(vtype=GRB.CONTINUOUS, name=f"lag{pair}")
    cc.addConstr(lag_var==(gp.quicksum([lambda_vars_[i]*theta_array[i] for i in data_indexes])), name=f"lag{pair} calculation")
    lag_vars[pair] = lag_var
```

```
    # create power flow variable for current pair
    Pf_data = Pf_dict[pair] # power flows calculated before for current pair (Pij)
    Pf_var = cc.addVar(vtype=GRB.CONTINUOUS, name=f"Pf{pair}") # Power flow variable for current pair
    cc.addConstr(Pf_var==(gp.quicksum([lambda_vars_[i]*Pf_data[i] for i in data_indexes])), name=f"Pf{pair} calculation")
    Pf_vars[pair] = Pf_var

    # create power flow variable for current pair, reverse
    Pb_data = Pb_dict[pair] # power flows calculated before for current pair (Pij)
    Pb_var = cc.addVar(vtype=GRB.CONTINUOUS, name=f"Pb{pair}") # Power flow variable for current pair
    cc.addConstr(Pb_var==(gp.quicksum([lambda_vars_[i]*Pb_data[i] for i in data_indexes])), name=f"Pb{pair} calculation")
    Pb_vars[pair] = Pb_var

    # not quite necessary, but create theta for each individual bus
    i = pair[0]
    j = pair[1]
    cc.addConstr(lag_var==thetai_vars[i]-thetai_vars[j], name=f"thetas_i=({pair})")
cc.update()
for bus in bus_set:
    # energy conservation constraint
    # calculate power inflow at each bus
    print(f"Power inflow at bus {bus}:")
    Pi_value = 0
    for pair in b_combs:
        if bus == pair[0]: # bus is i in pair (i,j), power inflow at i is Pb
            inflow = Pb_vars[pair]
        elif bus == pair[1]: # bus is j in pair (i,j), power inflow at i is Pf
            inflow = Pf_vars[pair]
        else:
            inflow = 0
        Pi_value += n_dict[pair] * inflow

    print(Pi_value)

    Pi_injected_constr = cc.addConstr(Pi_vars[bus]==Pi_value)

    # en_cons_constr = cc.addConstr(Pgi_vars[bus]==Pd_dict_low[bus]+Pi_vars[bus], f"Energy balance {bus}")
```

```python
    en_cons_constr = cc.addConstr(Pgi_vars[bus]==Pd_dict_high[bus]+Pi_vars[bus],␣
↪f"Energy balance {bus}")

    abs_constr = cc.addGenConstrAbs(Pi_abs_vars[bus], Pi_vars[bus], name="ABS")

    cc.update()
    print(Pi_injected_constr)


cc.setObjective(gp.quicksum(Pi_abs_vars))
cc.update()

# cc.display()
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2022-05-20
Set parameter LogFile to value "cc_log.txt"
[('Pg1', 7.0), ('Pg2', 0.0), ('Pg3', 4.0)]
Power inflow at bus 1:
<gurobi.LinExpr: 2.0 Pb(1, 2) + Pb(1, 3)>
<gurobi.Constr R63>
Power inflow at bus 2:
<gurobi.LinExpr: 2.0 Pf(1, 2) + Pb(2, 3)>
<gurobi.Constr R65>
Power inflow at bus 3:
<gurobi.LinExpr: Pf(1, 3) + Pf(2, 3)>
<gurobi.Constr R67>
```

[11]:

```python
cc.optimize()
```

```
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (win64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 69 rows, 108 columns and 375 nonzeros
Model fingerprint: 0xd6415f2b
Model has 3 general constraints
Variable types: 66 continuous, 42 integer (42 binary)
Coefficient statistics:
  Matrix range     [4e-02, 3e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 7e+00]
  RHS range        [1e+00, 5e+00]
Presolve removed 0 rows and 3 columns
Presolve time: 0.00s

Explored 0 nodes (0 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 1 (of 4 available processors)

Solution count 0
```

```
Model is infeasible
Best objective -, best bound -, gap -
```

[12]:
```
cc.getVars()
```

[12]:
```
[<gurobi.Var thetai[1]>,
 <gurobi.Var thetai[2]>,
 <gurobi.Var thetai[3]>,
 <gurobi.Var Pgi[1]>,
 <gurobi.Var Pgi[2]>,
 <gurobi.Var Pgi[3]>,
 <gurobi.Var Pi[1]>,
 <gurobi.Var Pi[2]>,
 <gurobi.Var Pi[3]>,
 <gurobi.Var Pi_abs[1]>,
 <gurobi.Var Pi_abs[2]>,
 <gurobi.Var Pi_abs[3]>,
 <gurobi.Var lambdas(1, 2)[0]>,
 <gurobi.Var lambdas(1, 2)[1]>,
 <gurobi.Var lambdas(1, 2)[2]>,
 <gurobi.Var lambdas(1, 2)[3]>,
 <gurobi.Var lambdas(1, 2)[4]>,
 <gurobi.Var lambdas(1, 2)[5]>,
 <gurobi.Var lambdas(1, 2)[6]>,
 <gurobi.Var lambdas(1, 2)[7]>,
 <gurobi.Var lambdas(1, 2)[8]>,
 <gurobi.Var lambdas(1, 2)[9]>,
 <gurobi.Var lambdas(1, 2)[10]>,
 <gurobi.Var lambdas(1, 2)[11]>,
 <gurobi.Var lambdas(1, 2)[12]>,
 <gurobi.Var lambdas(1, 2)[13]>,
 <gurobi.Var lambdas(1, 2)[14]>,
 <gurobi.Var z(1, 2)[1]>,
 <gurobi.Var z(1, 2)[2]>,
 <gurobi.Var z(1, 2)[3]>,
 <gurobi.Var z(1, 2)[4]>,
 <gurobi.Var z(1, 2)[5]>,
 <gurobi.Var z(1, 2)[6]>,
 <gurobi.Var z(1, 2)[7]>,
 <gurobi.Var z(1, 2)[8]>,
 <gurobi.Var z(1, 2)[9]>,
 <gurobi.Var z(1, 2)[10]>,
 <gurobi.Var z(1, 2)[11]>,
 <gurobi.Var z(1, 2)[12]>,
 <gurobi.Var z(1, 2)[13]>,
 <gurobi.Var z(1, 2)[14]>,
```

```
<gurobi.Var lag(1, 2)>,
<gurobi.Var Pf(1, 2)>,
<gurobi.Var Pb(1, 2)>,
<gurobi.Var lambdas(1, 3)[0]>,
<gurobi.Var lambdas(1, 3)[1]>,
<gurobi.Var lambdas(1, 3)[2]>,
<gurobi.Var lambdas(1, 3)[3]>,
<gurobi.Var lambdas(1, 3)[4]>,
<gurobi.Var lambdas(1, 3)[5]>,
<gurobi.Var lambdas(1, 3)[6]>,
<gurobi.Var lambdas(1, 3)[7]>,
<gurobi.Var lambdas(1, 3)[8]>,
<gurobi.Var lambdas(1, 3)[9]>,
<gurobi.Var lambdas(1, 3)[10]>,
<gurobi.Var lambdas(1, 3)[11]>,
<gurobi.Var lambdas(1, 3)[12]>,
<gurobi.Var lambdas(1, 3)[13]>,
<gurobi.Var lambdas(1, 3)[14]>,
<gurobi.Var z(1, 3)[1]>,
<gurobi.Var z(1, 3)[2]>,
<gurobi.Var z(1, 3)[3]>,
<gurobi.Var z(1, 3)[4]>,
<gurobi.Var z(1, 3)[5]>,
<gurobi.Var z(1, 3)[6]>,
<gurobi.Var z(1, 3)[7]>,
<gurobi.Var z(1, 3)[8]>,
<gurobi.Var z(1, 3)[9]>,
<gurobi.Var z(1, 3)[10]>,
<gurobi.Var z(1, 3)[11]>,
<gurobi.Var z(1, 3)[12]>,
<gurobi.Var z(1, 3)[13]>,
<gurobi.Var z(1, 3)[14]>,
<gurobi.Var lag(1, 3)>,
<gurobi.Var Pf(1, 3)>,
<gurobi.Var Pb(1, 3)>,
<gurobi.Var lambdas(2, 3)[0]>,
<gurobi.Var lambdas(2, 3)[1]>,
<gurobi.Var lambdas(2, 3)[2]>,
<gurobi.Var lambdas(2, 3)[3]>,
<gurobi.Var lambdas(2, 3)[4]>,
<gurobi.Var lambdas(2, 3)[5]>,
<gurobi.Var lambdas(2, 3)[6]>,
<gurobi.Var lambdas(2, 3)[7]>,
<gurobi.Var lambdas(2, 3)[8]>,
<gurobi.Var lambdas(2, 3)[9]>,
<gurobi.Var lambdas(2, 3)[10]>,
<gurobi.Var lambdas(2, 3)[11]>,
```

```
<gurobi.Var lambdas(2, 3)[12]>,
<gurobi.Var lambdas(2, 3)[13]>,
<gurobi.Var lambdas(2, 3)[14]>,
<gurobi.Var z(2, 3)[1]>,
<gurobi.Var z(2, 3)[2]>,
<gurobi.Var z(2, 3)[3]>,
<gurobi.Var z(2, 3)[4]>,
<gurobi.Var z(2, 3)[5]>,
<gurobi.Var z(2, 3)[6]>,
<gurobi.Var z(2, 3)[7]>,
<gurobi.Var z(2, 3)[8]>,
<gurobi.Var z(2, 3)[9]>,
<gurobi.Var z(2, 3)[10]>,
<gurobi.Var z(2, 3)[11]>,
<gurobi.Var z(2, 3)[12]>,
<gurobi.Var z(2, 3)[13]>,
<gurobi.Var z(2, 3)[14]>,
<gurobi.Var lag(2, 3)>,
<gurobi.Var Pf(2, 3)>,
<gurobi.Var Pb(2, 3)>]
```

1.3.3 Special Ordered Set of Variables Type 2 - SOS2

$$
\begin{aligned}
(i,j) &\in \{1,\ldots,N\}^2, i \neq j \\
Givens^{(i,j)} : \{(\theta^{(i,j)}_{k=0}, P^{(i,j)}_{k=0}), \ldots, (\theta^{(i,j)}_{(M-1)}, P^{(i,j)}_{(M-1)})\} \\
\theta^{(i,j)} &= \sum_{k=0}^{M-1} \lambda_k^{(i,j)} \theta_k^{(i,j)} \\
P^{(i,j)} &= \sum_{k=0}^{M-1} \lambda_k^{(i,j)} P_k^{(i,j)} \\
\lambda_k^{(i,j)} \geq 0, k = 0, ..., (M-1) \\
1 &= \sum_{k=0}^{M-1} \lambda_k^{(i,j)} \\
\{\lambda_k^{(i,j)}\}_{k=0}^{(M-1)} &\in \text{SOS2}
\end{aligned}
$$

Where **SOS2** (Special Ordered Set of type 2) is the set of sets that comply to the following criteria:

At most 2 elements are positive.

In the case 2 elements in the set are positive, they must be consecutive in the ordered set.

Modeling the SOS2 constraint in linear terms

$$
\begin{aligned}
&\lambda_k^{(i,j)} \in [0, +\infty], k = 0, ..., M-1 \\
&z_k^{(i,j)} \in \{0, 1\}, k = 0, ..., M-1 \\
&\lambda_k^{(i,j)} \leq z_k, k = 0, ..., M-1 \\
&\sum_{k=0}^{M-1} z_k \leq 2 \\
&z_k^{(i,j)} + z_l^{(i,j)} \leq 1, l \in \{k+2, ..., M-1\}
\end{aligned}
$$

1.3.4 Special Ordered Set of Variables Type 2 in Gurobi

[13]:
```
sos2 = Model()
sos2.Params.LogFile = "sos2_log.txt"
# each bus i is in a certain theta
thetai_vars = sos2.addVars(bus_set, lb=-math.pi/2, ub=math.pi/2, vtype=GRB.
 ↪CONTINUOUS, name="thetai")

# power generated in each bus i, constrained by values given in problem statement
Pgi_vars = sos2.addVars(bus_set, lb=0,ub=Pgmax_dict,vtype=GRB.CONTINUOUS,␣
 ↪name="Pgi")
sos2.update()
print([(f"Pg{Pgi}",Pgi_vars[Pgi].ub) for Pgi in Pgi_vars])
# Pgi_vars = sos2.addVars(bus_set, lb=0,vtype=GRB.CONTINUOUS, name="Pgi") # This␣
 ↪would be a relaxation
```

```python
# total power flow into each bus i
Pi_vars = sos2.addVars(bus_set, vtype=GRB.CONTINUOUS, name="Pi")

# auxiliary variables which represents each |Pi|
Pi_abs_vars = sos2.addVars(bus_set, vtype=GRB.CONTINUOUS, name="Pi_abs")

lambda_vars = gp.tupledict() # 1 value per data interval per pair ((n_points-1)␣
↪* len(b_combs))
z_vars = gp.tupledict() # 1 value per data point per pair (n_points *␣
↪len(b_combs))
lag_vars = gp.tupledict() # 1 value per pair (len(b_combs))
Pf_vars = gp.tupledict() # 1 value per pair (len(b_combs))
Pb_vars = gp.tupledict() # 1 value per pair (len(b_combs))

data_indexes = range(n_points)

for pair in b_combs:
    # create lambdas for current pair
    lambda_vars_ = sos2.addVars(data_indexes, lb=0.0, vtype=GRB.CONTINUOUS,␣
↪name=f"lambdas{pair}")
    sos2.addConstr(gp.quicksum(lambda_vars_)==1)
    lambda_vars[pair] = lambda_vars_

    z_vars_ = sos2.addVars(data_indexes, vtype=GRB.BINARY, name=f"z{pair}")
    sos2.addConstr(gp.quicksum(z_vars_)<=2)
    z_vars[pair] = z_vars_

    for k in range(0,n_points):
        sos2.addConstr(lambda_vars_[k]<=z_vars_[k])
        for l in range(k+2,n_points):
            sos2.addConstr(lambda_vars_[k]+z_vars_[l]<=1)

    # create lag variable for current pair
    lag_var = sos2.addVar(vtype=GRB.CONTINUOUS, name=f"lag{pair}")
    sos2.addConstr(lag_var==(gp.quicksum([lambda_vars_[i]*theta_array[i] for i␣
↪in data_indexes])), name=f"lag{pair} calculation")
    lag_vars[pair] = lag_var

    # create power flow variable for current pair
    Pf_data = Pf_dict[pair] # power flows calculated before for current pair␣
↪(Pij)
    Pf_var = sos2.addVar(vtype=GRB.CONTINUOUS, name=f"Pf{pair}") # Power flow␣
↪variable for current pair
    sos2.addConstr(Pf_var==(gp.quicksum([lambda_vars_[i]*Pf_data[i] for i in␣
↪data_indexes])), name=f"Pf{pair} calculation")
```

```python
    Pf_vars[pair] = Pf_var

    # create power flow variable for current pair, reverse
    Pb_data = Pb_dict[pair] # power flows calculated before for current pair (Pij)
    Pb_var = sos2.addVar(vtype=GRB.CONTINUOUS, name=f"Pb{pair}") # Power flow variable for current pair
    sos2.addConstr(Pb_var==(gp.quicksum([lambda_vars_[i]*Pb_data[i] for i in data_indexes])), name=f"Pb{pair} calculation")
    Pb_vars[pair] = Pb_var

    # not quite necessary, but create theta for each individual bus
    i = pair[0]
    j = pair[1]
    sos2.addConstr(lag_var==thetai_vars[i]-thetai_vars[j], name=f"thetas_i=({pair})")
sos2.update()
for bus in bus_set:
    # energy conservation constraint
    # calculate power inflow at each bus
    print(f"Power inflow at bus {bus}:")
    Pi_value = 0
    for pair in b_combs:
        if bus == pair[0]: # bus is i in pair (i,j), power inflow at i is Pb
            inflow = Pb_vars[pair]
        elif bus == pair[1]: # bus is j in pair (i,j), power inflow at i is Pf
            inflow = Pf_vars[pair]
        else:
            inflow = 0
        Pi_value += n_dict[pair] * inflow

    print(Pi_value)

    Pi_injected_constr = sos2.addConstr(Pi_vars[bus]==Pi_value)

    # en_cons_constr = sos2.addConstr(Pgi_vars[bus]==Pd_dict_low[bus]+Pi_vars[bus], f"Energy balance {bus}")
    en_cons_constr = sos2.addConstr(Pgi_vars[bus]==Pd_dict_high[bus]+Pi_vars[bus], f"Energy balance {bus}")

    abs_constr = sos2.addGenConstrAbs(Pi_abs_vars[bus], Pi_vars[bus], name="ABS")

    sos2.update()
    print(Pi_injected_constr)
```

```
sos2.setObjective(gp.quicksum(Pi_abs_vars))
sos2.update()

# sos2.display()
```

```
Set parameter LogFile to value "sos2_log.txt"
[('Pg1', 7.0), ('Pg2', 0.0), ('Pg3', 4.0)]
Power inflow at bus 1:
<gurobi.LinExpr: 2.0 Pb(1, 2) + Pb(1, 3)>
<gurobi.Constr R336>
Power inflow at bus 2:
<gurobi.LinExpr: 2.0 Pf(1, 2) + Pb(2, 3)>
<gurobi.Constr R338>
Power inflow at bus 3:
<gurobi.LinExpr: Pf(1, 3) + Pf(2, 3)>
<gurobi.Constr R340>
```

[14]:

```
sos2.optimize()
```

```
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (win64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 342 rows, 111 columns and 885 nonzeros
Model fingerprint: 0x36dcab36
Model has 3 general constraints
Variable types: 66 continuous, 45 integer (45 binary)
Coefficient statistics:
  Matrix range     [4e-02, 3e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 7e+00]
  RHS range        [1e+00, 5e+00]
Presolve removed 0 rows and 3 columns
Presolve time: 0.00s

Explored 0 nodes (0 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 1 (of 4 available processors)

Solution count 0

Model is infeasible
Best objective -, best bound -, gap -
```

1.3.5 Using Gurobi built-in PWL constraints

[15]:
```
m = Model()
m.Params.LogFile = "m_log.txt"
# each bus i is in a certain theta
thetai_vars = m.addVars(bus_set, lb=-math.pi/2, ub=math.pi/2, vtype=GRB.
 ↪CONTINUOUS, name="thetai")

# power generated in each bus i, constrained by values given in problem statement
Pgi_vars = m.addVars(bus_set, lb=0,ub=Pgmax_dict,vtype=GRB.CONTINUOUS,␣
 ↪name="Pgi")
m.update()
print([(f"Pg{Pgi}",Pgi_vars[Pgi].ub) for Pgi in Pgi_vars])
# Pgi_vars = m.addVars(bus_set, lb=0,vtype=GRB.CONTINUOUS, name="Pgi") # This␣
 ↪would be a relaxation

# total power flow into each bus i
Pi_vars = m.addVars(bus_set, vtype=GRB.CONTINUOUS, name="Pi")

# auxiliary variables which represents each |Pi|
Pi_abs_vars = m.addVars(bus_set, vtype=GRB.CONTINUOUS, name="Pi_abs")

lambda_vars = gp.tupledict() # 1 value per data interval per pair ((n_points-1)␣
 ↪* len(b_combs))
z_vars = gp.tupledict() # 1 value per data point per pair (n_points *␣
 ↪len(b_combs))
lag_vars = gp.tupledict() # 1 value per pair (len(b_combs))
Pf_vars = gp.tupledict() # 1 value per pair (len(b_combs))
Pb_vars = gp.tupledict() # 1 value per pair (len(b_combs))

data_indexes = range(n_points)
pwlff = {}
pwlfb = {}
for pair in b_combs:

    Pf_vars[pair] = m.addVar(vtype=GRB.CONTINUOUS, name=f"Pf{pair}")
    Pb_vars[pair] = m.addVar(vtype=GRB.CONTINUOUS, name=f"Pb{pair}")
    lag_vars[pair] = m.addVar(vtype=GRB.CONTINUOUS, name=f"lag{pair}")
    pwlff[pair] = m.addGenConstrPWL(lag_vars[pair], Pf_vars[pair], theta_array,␣
 ↪Pf_dict[pair], f"Pf(lag) pair {pair}")
    pwlfb[pair] = m.addGenConstrPWL(lag_vars[pair], Pb_vars[pair], theta_array,␣
 ↪Pb_dict[pair], f"Pb(lag) pair {pair}")

    # not quite necessary, but create theta for each individual bus
    i = pair[0]
    j = pair[1]
```

```python
    m.addConstr(lag_vars[pair]==thetai_vars[i]-thetai_vars[j], name=f"thetas_i=({pair})")
m.update()
for bus in bus_set:
    # energy conservation constraint
    # calculate power inflow at each bus
    print(f"Power inflow at bus {bus}:")
    Pi_value = 0
    for pair in b_combs:
        if bus == pair[0]: # bus is i in pair (i,j), power inflow at i is Pb
            inflow = Pb_vars[pair]
        elif bus == pair[1]: # bus is j in pair (i,j), power inflow at i is Pf
            inflow = Pf_vars[pair]
        else:
            inflow = 0
        Pi_value += n_dict[pair] * inflow

    print(Pi_value)

    Pi_injected_constr = m.addConstr(Pi_vars[bus]==Pi_value)

    # en_cons_constr = m.addConstr(Pgi_vars[bus]==Pd_dict_low[bus]+Pi_vars[bus], f"Energy balance {bus}")
    en_cons_constr = m.addConstr(Pgi_vars[bus]==Pd_dict_high[bus]+Pi_vars[bus], f"Energy balance {bus}")

    abs_constr = m.addGenConstrAbs(Pi_abs_vars[bus], Pi_vars[bus], name="ABS")

    m.update()
    print(Pi_injected_constr)


m.setObjective(gp.quicksum(Pi_abs_vars))
m.update()

# m.display()
```

```
Set parameter LogFile to value "m_log.txt"
[('Pg1', 7.0), ('Pg2', 0.0), ('Pg3', 4.0)]
Power inflow at bus 1:
<gurobi.LinExpr: 2.0 Pb(1, 2) + Pb(1, 3)>
<gurobi.Constr R3>
Power inflow at bus 2:
<gurobi.LinExpr: 2.0 Pf(1, 2) + Pb(2, 3)>
<gurobi.Constr R5>
Power inflow at bus 3:
<gurobi.LinExpr: Pf(1, 3) + Pf(2, 3)>
```

```
<gurobi.Constr R7>
```

[16]:
```
m.optimize()
```

```
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (win64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 9 rows, 21 columns and 24 nonzeros
Model fingerprint: 0x70ab9279
Model has 9 general constraints
Variable types: 21 continuous, 0 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [2e+00, 7e+00]
  RHS range        [1e+00, 5e+00]
  PWLCon x range   [0e+00, 3e+00]
  PWLCon y range   [0e+00, 3e-01]
Presolve removed 0 rows and 3 columns
Presolve time: 0.00s

Explored 0 nodes (0 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 1 (of 4 available processors)

Solution count 0

Model is infeasible
Best objective -, best bound -, gap -
```

[17]:
```
m.getVars()
```

[17]:
```
[<gurobi.Var thetai[1]>,
 <gurobi.Var thetai[2]>,
 <gurobi.Var thetai[3]>,
 <gurobi.Var Pgi[1]>,
 <gurobi.Var Pgi[2]>,
 <gurobi.Var Pgi[3]>,
 <gurobi.Var Pi[1]>,
 <gurobi.Var Pi[2]>,
 <gurobi.Var Pi[3]>,
 <gurobi.Var Pi_abs[1]>,
 <gurobi.Var Pi_abs[2]>,
 <gurobi.Var Pi_abs[3]>,
 <gurobi.Var Pf(1, 2)>,
 <gurobi.Var Pb(1, 2)>,
 <gurobi.Var lag(1, 2)>,
 <gurobi.Var Pf(1, 3)>,
 <gurobi.Var Pb(1, 3)>,
 <gurobi.Var lag(1, 3)>,
```

```
 <gurobi.Var Pf(2, 3)>,
 <gurobi.Var Pb(2, 3)>,
 <gurobi.Var lag(2, 3)>]
```

[ ]:

[ ]: