Homework 2

Name: Naman Makkar

Student ID:nbm49

- Reasoning and work must be shown to gain partial/full credit
- Please include the cover-page on your homework PDF with your name and student ID. Failure of doing so is considered bad citizenship.

In this homework, you are going to run a power flow simulation and estimate the state of a synthetic power test case, the IEEE-118 system. The IEEE 118-bus test case represents a simple approximation of the American Electric Power transmission system (in the U.S. Midwest). This test case is widely used for research purposes and you may find plenty of information online[1]. All the data you need to run these studies is provided to you in the datasets folder that contains the following

1. *branches_ieee118_subset.csv*. This file contains information about the edges of the IEEE-118 network. The file contains the following features.
   - **fbus**: The "from" bus ID
   - **tbus**" The "to" bus ID
   - **r**: The resistance in p.u.
   - **x**: The reactance in p.u.
   - **ratio**: transformer off nominal turns ratio
2. *buses_ieee118_subset.csv*. This file contains information about the nodes of the IEEE-118 network. The file contains the following features.
   - **bus_i**: ID of the bus
   - **Pd**: Real power demand in MW
   - **Vm**: Voltage magnitude in p.u.
   - **Va**: Voltage angle in degrees
   - **baseKV**: base voltage in kV
3. *generators_ieee118_subset.csv*. This file contains information about the nodes of the IEEE-118 network. The file contains the following features.
   - **bus**: ID of the bus that the generator is connected to
   - **Pg**: Real power output in MW
   - **mBase**: Total MVA base of machine

[1] https://icseg.iti.illinois.edu/ieee-118-bus-system/

1. (1–4 points) **DC Power Flow**: In this problem, we are going to use the DC approximation to model the flow of electricity through transmission lines. Power networks can be denoted as a graph $\mathcal{G} := \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Each edge $e := \{i, j\} \in \mathcal{E}$ connects two buses $i, j \in \mathcal{V}$. Also, $N := |\mathcal{V}|$ denotes the number of buses and $E := |\mathcal{E}|$ is the number of deges (the operator $|.|$ denotes the cardinality of a set). The DC model works under the following assumptions

   - Voltage magnitudes are 1 p.u., $\forall i \in \mathcal{V}$
   - Voltage angle differences between two buses $i, j \in \mathcal{V}$ are small, thus $\sin(\theta_i - \theta_j) \approx \theta_i - \theta_j$
   - The resistance of the line is neglected, i.e. $r_e \ll x_e, \forall e \in \mathcal{E}$

   The DC power flow can be expressed by the following equation

$$\mathbf{p} = \mathbf{B}\boldsymbol{\theta}, \tag{1}$$

   where $\mathbf{p} \in \mathbb{R}^N$ is the vector of net power injections, i.e. $\mathbf{p} := \mathbf{p}_{\text{inj}} - \mathbf{p}_{\text{d}}$ where $\mathbf{p}_{\text{inj}}$ is the power injected by the generators and $\mathbf{p}_{\text{d}}$ is the power demand (from the loads). Also, $\mathbf{B} \in \mathbb{R}^{N \times N}$ is the susceptance matrix and $\boldsymbol{\theta} \in \mathbb{R}^N$ is the vector of bus angles. In the DC power flow problem, all the decisions regarding the setpoints of the generators or the loads that will be served have been made, i.e. $\mathbf{p}$ is given. In addition, voltage magnitudes are 1 p.u. by definition. Thus, the state of your system and your variable of interest is $\boldsymbol{\theta}$.

   (a) (**0.25 pts**) What are you trying to calculate when solving a DC power flow problem?

   Solution: We are trying to calculate the vector of voltage angles $\boldsymbol{\theta}$ when solving a DC power flow problem which is our variable of interest. Calculating the voltage angles across all buses we can accurately calculate the power flow and losses across transmission lines.

   (b) (**0.5 pts**) Calculate the directed incidence matrix $\mathbf{M} \in \{-1, 0, 1\}^{E \times N}$ of the graph $\mathcal{G}$ defined by the edges provided in *branches_ieee118_subset.csv* (**Note**: Even though power can flow on either direction of an edge, $\mathbf{M}$ is the incidence matrix of a directed graph.)

   Solution:

In [1]:

```python

# Incidence Matrix Calculation
import numpy as np
import pandas as pd
branches = pd.read_csv('/kaggle/input/energy-systems-hw2/
    datasets/branches_ieee118_subset.csv')
buses = pd.read_csv('/kaggle/input/energy-systems-hw2/datasets/
    buses_ieee118_subset.csv')
generators = pd.read_csv('/kaggle/input/energy-systems-hw2/
    datasets/generators_ieee118_subset.csv')

E = len(branches['fbus'])
N = max(branches['tbus'])
M = [[0 for _ in range(N)] for _ in range(E)]
fbus = list(branches['fbus'])
tbus = list(branches['tbus'])

for edge_num in range(E):
    node1 = fbus[edge_num]
    node2 = tbus[edge_num]
    M[edge_num][node1-1] = -1
    M[edge_num][node2-1] = 1
M = np.array(M)
print(M)
```

Out[1]:

```

[[-1  1  0 ...  0  0  0]
 [-1  0  1 ...  0  0  0]
 [ 0  0  0 ...  0  0  0]
 ...
 [ 0  0  0 ...  0  1  0]
 [ 0  0  0 ...  0  0  1]
 [ 0  0  0 ...  0  0  1]]
```

(c) (**0.25 pts**) Calculate the vector of susceptances $\mathbf{b} \in \mathbb{R}^E$. Each entry $b_e, \forall e \in \mathcal{E}$ in $\mathbf{b}$ can be obtained as follows

$$b_e := \frac{1}{\tau_e x_e} \tag{2}$$

where τ_e are the transformer turn ratios, and x_e is the reactance of the line in p.u.

Solution:

In [2]:

```python

# b_e calculation
transformer_turn_ratios = list(branches['ratio'])
reactance = list(branches['x'])
susceptances = [0 for _ in range(E)]

for edge_num in range(E):
    susceptances[edge_num] = 1/(transformer_turn_ratios[
    edge_num]*reactance[edge_num])
b = np.array(susceptances)

print(b)
```

Out[2]:

```


[ 10.01001001  23.58490566 125.31328321   9.25925926  18.51851852
  48.07692308  32.78688525  38.02353657  31.05590062  14.53488372
  14.6627566   51.02040816  16.23376623   6.25        29.41176471
  13.67989056  14.14427157   4.09165303   5.12820513  11.99040767
  22.88329519   5.55247085  19.8019802   20.28397566   8.54700855
  25.38071066  11.77856302  10.30927835   6.28930818  20.32520325
  12.5         27.26876091   6.13496933  11.69590643  10.60445387
  26.84707904  19.84126984  11.62790698   6.39795266  30.21148036
   8.67302689  10.15228426  13.24503311   8.03858521   4.048583
  98.03921569  20.12072435   7.04225352  37.31343284 106.38297872
  28.52049911   9.43396226   5.95238095  18.51851852  16.52892562
  20.5338809    5.46448087   7.40740741   4.07497963   5.94883998
  11.09877913   7.37463127   7.87401575   5.29100529  16.
   3.09597523   3.09597523   5.37634409  19.8019802   13.29787234
   7.29927007  17.00680272   6.11620795   8.19672131   3.46020761
   3.43642612  14.14427157 104.71204188  66.22516556  10.35196687
   7.46268657  10.35196687  13.90820584   4.361099     3.98406375
   4.18410042   4.6339203    6.89655172   6.66666667  74.07407407
  17.82531194  26.59574468  26.98618307  50.          37.8816577
  10.14198783  33.11258278  10.88139282  10.88139282   4.58715596
   8.54700855  28.90591126   9.85221675  62.5          3.59971202
   3.08641975  28.90591126   7.87401575   2.43013366  28.16901408
   5.10204082   5.55555556  22.02643172   7.55857899   7.09219858
   8.19672131  24.63054187   6.75675676   9.9009901    5.00250125
  80.64516129  40.98360656  20.6185567    9.52380952  14.20454545
  49.5049505   28.90591126  11.72332943  27.2851296    7.57575758
   6.75675676  15.60062402   8.1300813    4.82160077   9.80392157
   5.78034682  14.04494382   5.31914894  10.03009027  11.96172249
  19.8019802    6.32511069   7.86163522  11.79245283   6.32911392
  13.66120219  23.04147465   5.49450549  18.86792453  11.50747986
  10.70663812   9.25925926   4.85436893   3.38983051  17.24137931
  18.28153565  11.29943503   5.58659218  12.300123     7.92393027
  17.88908766   8.92857143  19.04761905   4.90196078   6.31313131
   6.15384615   4.36681223  26.45502646  18.28153565   5.46448087
  14.22475107   5.46448087  34.72222222   5.5157198   13.12335958
  13.24503311  15.625       33.22259136   4.92610837  16.33986928
  13.49527665  96.15384615 246.91358025   7.14285714  20.79002079
  18.38235294]
```

(d) (**0.5 pts**) Calculate the susceptance matrix $\mathbf{B} := \mathbf{M}^\intercal \mathrm{diag}\left(\mathbf{b}\right) \mathbf{M}$. (Note: $\mathbf{B}$ is a weighted laplacian matrix where the weights are the susceptances of the lines)

Solution:

In [3]:

```python

# Susceptance Matrix calculation
B = M.T @ np.diag(b) @ M
print(B)
```

Out[3]:

```


[[ 33.59491567 -10.01001001 -23.58490566 ...   0.           0.
    0.        ]
 [-10.01001001  26.24377624   0.         ...   0.           0.
    0.        ]
 [-23.58490566   0.          39.09416492 ...   0.           0.
    0.        ]
 ...
 [  0.           0.           0.         ... 246.91358025   0.
    0.        ]
 [  0.           0.           0.         ...   0.
   7.14285714
    0.        ]
 [  0.           0.           0.         ...   0.           0.
  39.17237373]]
```

(e) (**0.25 pts**) Now, change the direction of the edge you assumed in part (b). Do the results change? Comment your results.

In [4]:

```python
M_changed_directions = [[0 for _ in range(N)] for _ in range(E)
    ]

for edge_num in range(E):
    node1 = fbus[edge_num]
    node2 = tbus[edge_num]
    M[edge_num][node1-1] = 1
    M[edge_num][node2-1] = -1
M_changed_directions = np.array(M_changed_directions)
# showing that changing the directions in part b does not
    change the lagrangian
B_changed_directions = M_changed_directions.T @ np.diag(b) @
    M_changed_directions
B.all() == B_changed_directions.all()
```

Out[4]:

```
True
```

(f) (**0.25 pts**) Calculate the condition number of the matrix **B**. What do you observe? Does this have any implications when solving Eq. (1)? Comment your results.

In [5]:

```python

#Calculating the condition number of B
#The condition number seems to be approaching infinity which
    indicates to us that the matrix B is non-invertible since
    cond(B) = norm(B)*norm(inv(B))
#What this means is that solving equation (1) would not be
    possible since B is non-invertible and we would not be able
    to calculate inv(B)*p
condition_number = np.linalg.cond(B)
condition_number
```

Out[5]:

```
8.251707700206938e+17
```

(g) (**0.5 pts**) Calculate the vector of net power injections **p**. (**Note**: To match the per unit system used for the reactance and the voltage, you must divide the quantities in MWs by 100. See below[2])

In [6]:

```
merged_df = buses.merge(generators, left_on='bus_i', right_on='bus', how='left')

merged_df['Pg'] = merged_df['Pg'].fillna(0)

merged_df['mBase'] = merged_df['mBase'].fillna(100.0)

merged_df['Pd'] = merged_df['Pd']
merged_df['Pg'] = merged_df['Pg']
p = merged_df['Pg'].values - merged_df['Pd'].values / 100
p
```

[2] https://en.wikipedia.org/wiki/Per-unit_system

Out[6]:

```

array([-5.1000e-01, -2.0000e-01, -3.9000e-01, -3.9000e-01,
    0.0000e+00,
       -5.2000e-01, -1.9000e-01, -2.8000e-01,  0.0000e+00,
    4.5000e+02,
       -7.0000e-01,  8.4530e+01, -3.4000e-01, -1.4000e-01,
    -9.0000e-01,
       -2.5000e-01, -1.1000e-01, -6.0000e-01, -4.5000e-01,
    -1.8000e-01,
       -1.4000e-01, -1.0000e-01, -7.0000e-02, -1.3000e-01,
    2.2000e+02,
        3.1400e+02, -7.1000e-01, -1.7000e-01, -2.4000e-01,
    0.0000e+00,
        6.5700e+00, -5.9000e-01, -2.3000e-01, -5.9000e-01,
    -3.3000e-01,
       -3.1000e-01,  0.0000e+00,  0.0000e+00, -2.7000e-01,
    -6.6000e-01,
       -3.7000e-01, -9.6000e-01, -1.8000e-01, -1.6000e-01,
    -5.3000e-01,
        1.8720e+01, -3.4000e-01, -2.0000e-01,  2.0313e+02,
    -1.7000e-01,
       -1.7000e-01, -1.8000e-01, -2.3000e-01,  4.6870e+01,
    -6.3000e-01,
       -8.4000e-01, -1.2000e-01, -1.2000e-01,  1.5223e+02,
    -7.8000e-01,
        1.6000e+02, -7.7000e-01,  0.0000e+00,  0.0000e+00,
    3.9100e+02,
        3.9161e+02, -2.8000e-01,  0.0000e+00,  5.1640e+02,
    -6.6000e-01,
        0.0000e+00, -1.2000e-01, -6.0000e-02, -6.8000e-01,
    -4.7000e-01,
       -6.8000e-01, -6.1000e-01, -7.1000e-01, -3.9000e-01,
    4.7570e+02,
        0.0000e+00, -5.4000e-01, -2.0000e-01, -1.1000e-01,
    -2.4000e-01,
       -2.1000e-01,  4.0000e+00, -4.8000e-01,  6.0700e+02,
    -1.6300e+00,
       -1.0000e-01, -6.5000e-01, -1.2000e-01, -3.0000e-01,
    -4.2000e-01,
       -3.8000e-01, -1.5000e-01, -3.4000e-01, -4.2000e-01,
    2.5163e+02,
       -2.2000e-01, -5.0000e-02,  3.9770e+01, -3.8000e-01,
    -3.1000e-01,
       -4.3000e-01, -5.0000e-01, -2.0000e-02, -8.0000e-02,
    -3.9000e-01,
        3.6000e+01, -6.8000e-01, -6.0000e-02, -8.0000e-02,
    -2.2000e-01,
       -1.8400e+00, -2.0000e-01, -3.3000e-01])
```

(h) (**1 pts**) Finally, solve Eq. (1). Since **B** is non-invertible, you must fix the voltage angle of one of the buses, and use that information to compute your inverse. You should solve a system of the form

$$\boldsymbol{\theta}' = \left(\mathbf{B}'\right)^{-1}\left(\mathbf{p}' - \mathbf{b}_0\theta_0\right) \tag{3}$$

where $\boldsymbol{\theta}', \mathbf{p}'$ are the corresponding vectors without the i-th entry, $\mathbf{B}'$ is the susceptance matrix without the i-th row and j-th column, $\mathbf{b}_0$ is the j-th column of $\mathbf{B}$ without the i-th entry, and θ_0 is the voltage angle at the reference bus. For this problem, we set $i = j := 69$ (that is, we use Bus ID 69 as the reference bus). The voltage angle at that bus θ_0 can be obtained from the *Va* column in *buses ieee118 subset.csv* (**Hint**: $\theta_0 := 30$ degrees but you must use radians. Also, please be careful with 0-indexed lists in Python)

Solution:

In [7]:

```python

#Fixing the voltage angle
i = 68
j = 68
angle_buses = np.array([np.radians(deg) for deg in list(buses['
    Va'])])
theta_0 = angle_buses[i]
p_new = np.concatenate((p[0:i],p[i+1:]), axis=0)
b_0 = np.concatenate((B[:,j][:i], B[:,j][i+1:]), axis=0)
temp = np.concatenate((B[:i], B[i+1:]), axis=0)
B_new = np.concatenate((temp[:,:j], temp[:,j+1:]), axis=1)
theta_prime = np.linalg.inv(B_new) @ (p_new - b_0*theta_0)
print(theta_prime)
```

Out[7]:

```

[0.2566869  0.26844121 0.27332208 0.33961817 0.34790097 0.30082574
 0.29350907 0.43667087 0.57392087 0.71882087 0.29503963 0.28800911
 0.26840658 0.27394613 0.26245856 0.28032816 0.30876636 0.26773569
 0.25726001 0.2698412  0.29425262 0.33572317 0.41960067 0.40919044
 0.55518049 0.58775332 0.32804443 0.29987361 0.28483434 0.39410028
 0.28749945 0.31748798 0.24865268 0.25677521 0.24875968 0.24867906
 0.26555355 0.35060505 0.20528037 0.18721417 0.17803056 0.20252293
 0.24942367 0.28286356 0.30955721 0.35539118 0.39125132 0.38291815
 0.40037323 0.362064   0.31407454 0.29759914 0.28121742 0.29705376
 0.29222229 0.29538314 0.3165802  0.30115243 0.36825617 0.43146208
 0.44664503 0.43658062 0.42456645 0.45495844 0.51036111 0.51554751
 0.46364692 0.50067874 0.40639826 0.40266383 0.39452869 0.39993983
 0.38499066 0.40602914 0.38687334 0.4827733  0.47751049 0.48447862
 0.5320394  0.51223974 0.49787969 0.519013   0.56040044 0.58754942
 0.56663942 0.57493542 0.64356992 0.71685042 0.60890113 0.606649
 0.6159423  0.56396844 0.52788821 0.50774216 0.50532471 0.51150593
 0.50699993 0.50348329 0.52635933 0.54775024 0.59137425 0.46659136
 0.41610579 0.39799277 0.39328652 0.34988965 0.38094997 0.37454399
 0.36369085 0.39087085 0.32017085 0.30831978 0.31223513 0.31217449
 0.49322674 0.26000911 0.38861562]
```

(i) (**0.5 pts**) Install the python package *pandapower* and run a DC and AC power flow. Run the code below and compare your results. You should make a scatter plot where you compare the solution you obtained for $\boldsymbol{\theta}$ to the DC and AC results you obtained from pandapower. Do they match? You should provide the Mean Absolute Percent Error of the voltage angle and voltage magnitude (**Note**: The DC solutions should match. **Hint**: The voltage magnitude of the DC solution is 1 p.u. by definition)

In [8]:

```python

# !pip install pandapower
import pandapower
import pandapower.networks

net_dc = pandapower.networks.case118()
pandapower.rundcpp(net_dc)

net_ac = pandapower.networks.case118()
pandapower.runpp(net_ac)

net_dc.res_bus # DC results
net_ac.res_bus # AC results
```

Solution:

In [9]:

```python
!pip install pandapower
import pandapower
import pandapower.networks
theta_powerflow = np.insert(theta_prime,68,theta_0)
net_dc = pandapower.networks.case118()
pandapower.rundcpp(net_dc)
net_ac = pandapower.networks.case118()
pandapower.runpp(net_ac)
df = net_dc.res_bus # DC results
df2 = net_ac.res_bus # AC results

dc_theta = np.array([np.radians(deg) for deg in df['va_degree'
    ]])
ac_theta = np.array([np.radians(deg) for deg in df2['va_degree'
    ]])

import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10,10))
plt.scatter(theta_powerflow, dc_theta)
plt.title('Comparison of Voltage Angles Power Flow vs DC')
plt.xlabel('Voltage Angle Power Flow (Radians)')
plt.ylabel('Voltage Angle DC (Radians)')
fig.savefig('scatter_plot_voltage_angle_powerflow_vs_dc.png')
plt.show()
```

In [10]:

```python
fig = plt.figure(figsize=(10,10))
plt.scatter(theta_powerflow, ac_theta, color='r')
plt.title('Comparison of Voltage Angles for Power Flow vs AC')
plt.xlabel('Voltage Angle Power Flow (Radians)')
plt.ylabel('Voltage Angle AC (Radians)')
fig.savefig('scatter_plot_voltage_angle_powerflow_vs_ac.png')
plt.show()
```

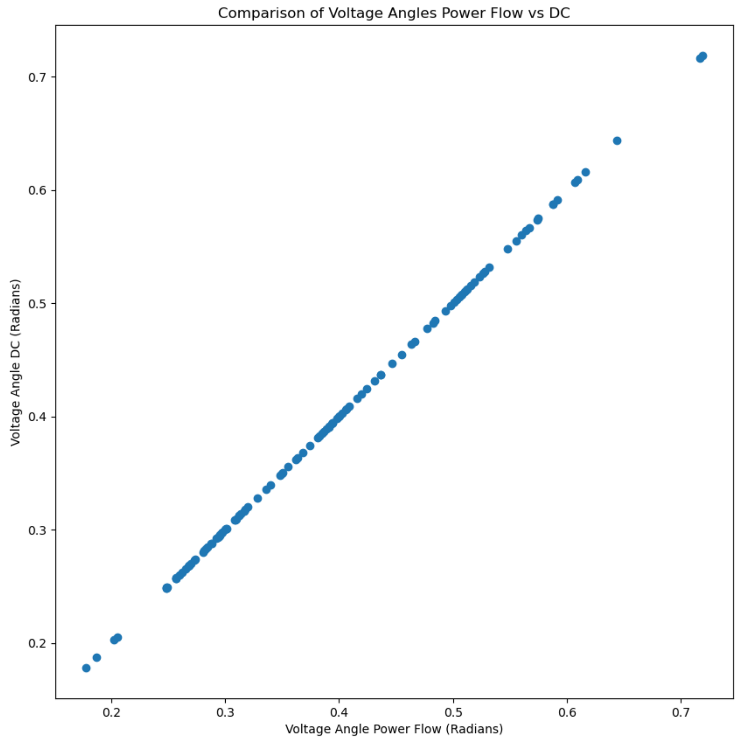

Figure 1: Comparison of Voltage Angles DC vs Power Flow

In [11]:

```

# Mean Absolute Percentage Error for Voltage Magnitudes
voltage_magnitude_ac = np.array(df2.vm_pu)
voltage_magnitude_dc = np.ones(shape=(118,))
voltage_magnitude_power_flow = np.array(buses.Vm)

mape_vm_ac = np.mean(np.abs(voltage_magnitude_ac -
    voltage_magnitude_power_flow)/voltage_magnitude_ac)*100
mape_vm_dc = np.mean(np.abs(voltage_magnitude_dc -
    voltage_magnitude_power_flow)/voltage_magnitude_dc)*100

print(f'Mean Absolute Percentage Error for Voltage Magnitudes
    for DC vs Power Flow is {mape_vm_dc}%')
print(f'Mean Absolute Percentage Error for Voltage Magnitudes
    for AC vs Power Flow is {mape_vm_ac}%')
```

Out[11]:

```

Mean Absolute Percentage Error for Voltage Magnitudes for DC vs
    Power Flow is 2.277118644067797%
Mean Absolute Percentage Error for Voltage Magnitudes for AC vs
    Power Flow is 0.05366115320849697%
```

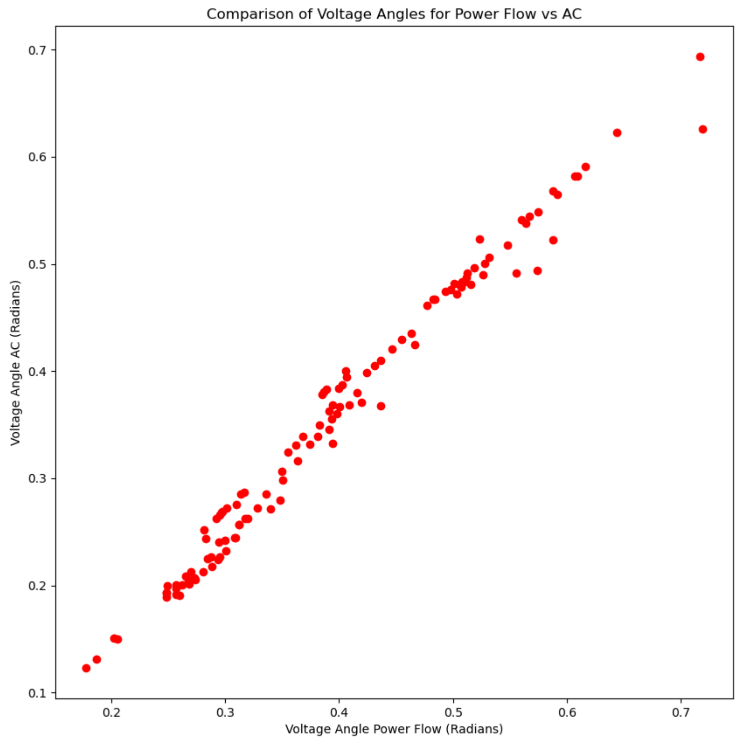

Figure 2: Comparison of Voltage Angles AC vs Power Flow

In [12]:
```python

#Mean Absolute Percentage Error for Voltage Angles
mape_va_dc = np.mean(np.abs(dc_theta-theta_powerflow)/dc_theta)
    *100
print(f'Mean Absolute Percentage Error for Voltage Angles for
    DC vs Power Flow is {mape_va_dc}%')

mape_va_ac = np.mean(np.abs(ac_theta-theta_powerflow)/ac_theta)
    *100
print(f'Mean Absolute Percentage Error for Voltage Angles for
    AC vs Power Flow is {mape_va_ac}%')
```

Out[12]:
```

Mean Absolute Percentage Error for Voltage Angles for DC vs Power
    Flow is 0.0033763106551810453%
Mean Absolute Percentage Error for Voltage Angles for AC vs Power
    Flow is 14.777733670273927%
```

In [13]:

```
fig = plt.figure(figsize=(10,10))
plt.scatter(voltage_magnitude_power_flow, voltage_magnitude_ac,
     color='r')
plt.title('Comparison of Voltage Magnitudes for Power Flow vs
    AC')
plt.xlabel('Voltage Magnitude Power Flow (p.u)')
plt.ylabel('Voltage Magnitude AC (p.u)')
fig.savefig('scatter_plot_voltage_magnitudes_powerflow_vs_ac.
    png')
plt.show()
```

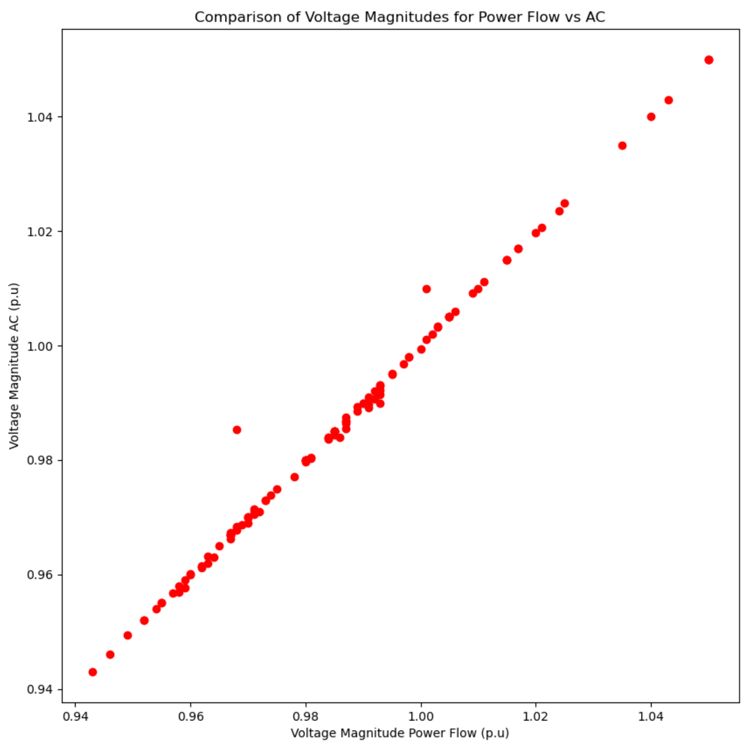

Figure 3: Comparison of Voltage Magnitudes AC vs Power Flow

In [14]:

```
fig = plt.figure(figsize=(10,10))
plt.scatter(voltage_magnitude_power_flow, voltage_magnitude_dc)
plt.title('Comparison of Voltage Magnitudes for Power Flow vs
    DC')
plt.xlabel('Voltage Magnitude Power Flow (p.u)')
plt.ylabel('Voltage Magnitude DC (p.u)')
fig.savefig('scatter_plot_voltage_magnitudes_powerflow_vs_dc.
    png')
plt.show()
```

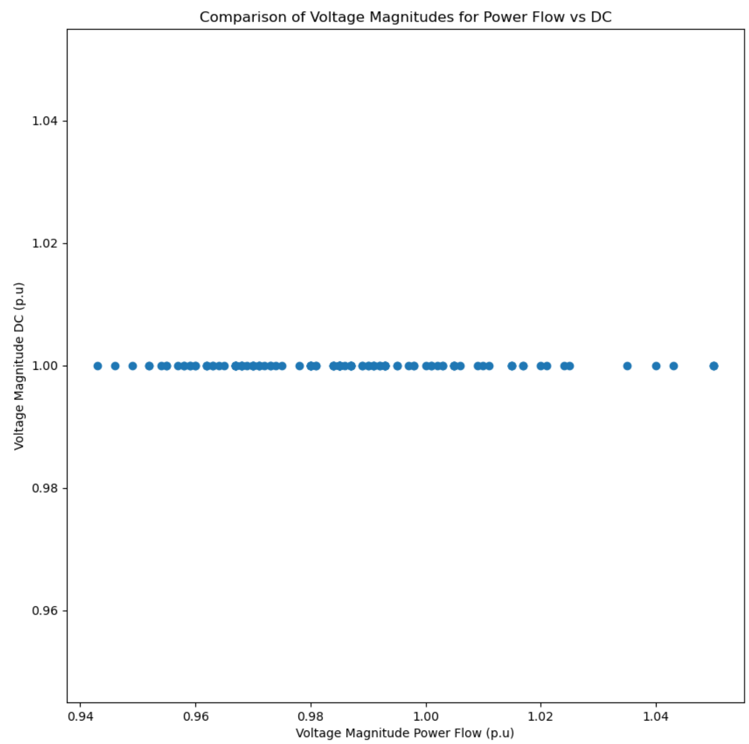

Figure 4: Comparison of Voltage Magnitudes DC vs Power Flow

2. (1–4 points) **DC state estimation**: The power flow problem that you solved in Problem 1 is an integral part of any power market mechanism in North America. It is used to understand how the flows of electricity will be distributed throughout the network. The power flow problem (and in the particular the DC power flow model) is used by market operators for multiple applications, e.g. to dispatch generation (a.k.a. economic dispatch), to run contingency analysis (to ensure that the system can sail through any unplanned event), or for long-term planning (to understand how congestion in the lines may impact future flows in a specific area). In real-time operations, running a power flow is not possible. Instead, the market operator monitors the conditions of the system (in a subset of nodes) and estimates the state of the system (i.e., the voltages), which is the goal of this problem. DC state estimation is similar to other estimation problems you may have seen in your detection and estimation theory course (e.g. Maximum Likelihood Estimation). Usually, in DC state estimation, the quantities that are measured are the following

   1. Voltage magnitude and angle
   2. Active and reactive power flows at the "from" and "to" bus, i.e. on both sides of the line.
   3. Currents at the "from" and "to" bus
   4. Power injections at the generator buses

In this problem, we will only use active power flows at the "from" and "to" bus (See Fig. 5 for reference). Thus, we should have models of the form.

$$\mathbf{w}_m = \mathbf{p}_{ij} + \boldsymbol{\epsilon}_w, \quad \boldsymbol{\epsilon}_w \sim \mathcal{N}\left(0, \boldsymbol{\Sigma}_w\right) \tag{4}$$

$$\mathbf{w}'_m = \mathbf{p}_{ji} + \boldsymbol{\epsilon}_{w'}, \quad \boldsymbol{\epsilon}_{w'} \sim \mathcal{N}\left(0, \boldsymbol{\Sigma}_{w'}\right) \tag{5}$$

where $\mathbf{w}_m$, $\mathbf{w}'_m$ are the measurements of the power flow at the "from" and "to" nodes, respectively, $\mathbf{p}_{ij}$ and $\mathbf{p}_{ji}$ are the power flows at the "from" and "to" node, and $\boldsymbol{\epsilon}_w$, $\boldsymbol{\epsilon}_{w'}$ are the vectors of noise. Noise is assumed to be Gaussian i.i.d. (i.e. independent and identically distributed)

(a) (**0.25 pts**) What are you trying to estimate when solving a DC state estimation?

Solution: The DC state estimation allows us to come up with an estimate for our variables of interest $\boldsymbol{\theta}$ with the help of certain measurements of power flow at the "from" and "to" node. DC state estimation is important due to the inability to solve the DC Power Flow equations in the time required to respond to the demands of the market.

(b) (**0.5 pts**) Using Eq. (1), derive a linear expression for $\mathbf{p}_{ij}$ and $\mathbf{p}_{ji}$ as a function of $\boldsymbol{\theta}$ and calculate the numerical values using the $\boldsymbol{\theta}$ from Problem 1. (**Hint**: $\left[\mathbf{p}_{ij}\right]_e = b_e\left(\theta_i - \theta_j\right)$ and $\left[\mathbf{p}_{ji}\right]_e = b_e\left(\theta_j - \theta_i\right)$. $\left[\mathbf{p}_{ij}\right]_e$ denotes the e-th entry of vector $\mathbf{p}_{ij}$. You need to use $\mathbf{M}$ to express the equations in matrix-vector form.)

Figure 5: Edge variables. The scalar p_{ij} denotes the power flow going from bus i to bus j. Similarly, p_{ji} denotes the power flow going from bus j to bus i

Solution:

$$\mathbf{p_{ij}} = \mathbf{diag(b)M}\boldsymbol{\theta}$$

$$\mathbf{p_{ji}} = -\mathbf{diag(b)M}\boldsymbol{\theta}$$

In [15]:

```
p_ij = np.diag(b) @ M @ theta_powerflow
p_ij
```

Out[15]:

```
array([-1.17660783e-01, -3.92339217e-01, -1.03794398e+00,
    -6.90545256e-01,
        8.71763363e-01,  3.51763363e-01, -4.50000000e+00,
    3.37534555e+00,-4.50000000e+00,  6.47943985e-01,
    7.75092949e-01,  3.58699730e-01, -3.17660783e-01,
    -9.17939609e-02,  1.61763363e-01,  3.64337204e-01,
    1.98910605e-01,  2.43372042e-02,  5.89106046e-02,
    9.20977434e-02, -1.05967488e+00, -1.57902257e-01,
    8.12488401e-01,  2.12488401e-01, -1.07531467e-01,
    1.31942898e-01, -2.87531467e-01, -4.27531467e-01,
    -5.27531467e-01,  2.11590174e-01, -1.69474774e+00,
    8.88220538e-01, 1.39347280e+00,  3.29483217e-01,
    1.59483217e-01,  2.29096664e+00, 8.44654448e-01,
    2.25177946e+00,  1.36064665e-01, -8.05167828e-02,
    8.85626102e-01, -3.04452118e-01,  1.39820441e-01,
    1.10979791e-01, 1.96276648e-03,  7.90469756e-03,
    -3.37904698e-01, -1.19020209e-01, 3.02095302e-01,
    -9.33865668e-01,  2.42571127e+00,  5.68614878e-01,
        4.66305817e-01,  8.05467269e-01,  2.98614878e-01,
    1.88575094e-01, -8.36543998e-02, -1.81424906e-01,
    -1.36266868e-01,  4.37331319e-02, -2.96266868e-01,
    -3.38008651e-01, -2.82363327e-01, -1.45645325e-01,
       -1.45950604e-01, -6.12539653e-01, -6.12539653e-01,
    -4.88258217e-01, -3.45645325e-01,  5.09431313e-01,
    6.29917449e-01,  2.80193979e-01, 1.00193979e-01,
    -1.29806021e-01,  3.57506842e-01,  3.55049750e-01,
        6.83375339e-02,  1.74933827e-01, -2.09327523e-01,
    -2.19431313e-01, 3.39431313e-01, -5.97234709e-02,
    1.79723471e-01, -3.10520789e-01, -2.90330817e-01,
    -3.04908096e-01, -3.52334943e-01, -4.35902783e-01,
    -5.22592371e-01, -1.12466311e+00, -9.12396717e-02,
    2.67670384e-01, 1.51959949e+00, -1.51959949e+00,
    3.14925866e-01, -1.62024400e+00,
       -1.83452536e+00, -1.25325653e+00, -1.25325653e+00,
    -3.62233424e-01,
...
        8.60374373e-02,  2.62858601e-01,  2.42429643e-01,
    2.37141399e-01,
        2.22429643e-01,  5.67570357e-01,  1.42429643e-01,
    -3.60000000e-01,
        6.80000000e-01,  1.48364371e-02,  4.51635629e-02,
    8.58308627e-02,
        2.14169137e-01,  5.83086265e-03,  1.84000000e+00,
    2.00000000e-01,
        3.62027270e-01, -3.20272698e-02])
```

In [16]:

```
p_ij = - np.diag(b) @ M @ theta_powerflow
p_ij
```

Out[16]:

```
array([ 1.17660783e-01,  3.92339217e-01,  1.03794398e+00,
    6.90545256e-01,
       -8.71763363e-01, -3.51763363e-01,  4.50000000e+00,
    -3.37534555e+00,
        4.50000000e+00, -6.47943985e-01, -7.75092949e-01,
    -3.58699730e-01,
        3.17660783e-01,  9.17939609e-02, -1.61763363e-01,
    -3.64337204e-01,
       -1.98910605e-01, -2.43372042e-02, -5.89106046e-02,
    -9.20977434e-02,
        1.05967488e+00,  1.57902257e-01, -8.12488401e-01,
    -2.12488401e-01,
        1.07531467e-01, -1.31942898e-01,  2.87531467e-01,
    4.27531467e-01,
        5.27531467e-01, -2.11590174e-01,  1.69474774e+00,
    -8.88220538e-01,
       -1.39347280e+00, -3.29483217e-01, -1.59483217e-01,
    -2.29096664e+00,
       -8.44654448e-01, -2.25177946e+00, -1.36064665e-01,
    8.05167828e-02,
       -8.85626102e-01,  3.04452118e-01, -1.39820441e-01,
    -1.10979791e-01,
       -1.96276648e-03, -7.90469756e-03,  3.37904698e-01,
    1.19020209e-01,
       -3.02095302e-01,  9.33865668e-01, -2.42571127e+00,
    -5.68614878e-01,
       -4.66305817e-01, -8.05467269e-01, -2.98614878e-01,
    -1.88575094e-01,
        8.36543998e-02,  1.81424906e-01,  1.36266868e-01,
    -4.37331319e-02,
        2.96266868e-01,  3.38008651e-01,  2.82363327e-01,
    1.45645325e-01,
        1.45950604e-01,  6.12539653e-01,  6.12539653e-01,
    4.88258217e-01,
        3.45645325e-01, -5.09431313e-01, -6.29917449e-01,
    -2.80193979e-01,
       -1.00193979e-01,  1.29806021e-01, -3.57506842e-01,
    -3.55049750e-01,
       -6.83375339e-02, -1.74933827e-01,  2.09327523e-01,
    2.19431313e-01,
       -3.39431313e-01,  5.97234709e-02, -1.79723471e-01,
    3.10520789e-01,
        2.90330817e-01,  3.04908096e-01,  3.52334943e-01,
    4.35902783e-01,
        5.22592371e-01,  1.12466311e+00,  9.12396717e-02,
    -2.67670384e-01,
       -1.51959949e+00,  1.51959949e+00, -3.14925866e-01,
    1.62024400e+00,
        1.83452536e+00,  1.25325653e+00,  1.25325653e+00,
    3.62233424e-01,
...
        -8.60374373e-02,  -2.62858601e-01,  -2.42429643e-01,
    -2.37141399e-01,
        -2.22429643e-01,  -5.67570357e-01,  -1.42429643e-01,
    3.60000000e-01,
        -6.80000000e-01,  -1.48364371e-02,  -4.51635629e-02,
    -8.58308627e-02,
        -2.14169137e-01,  -5.83086265e-03,  -1.84000000e+00,
    -2.00000000e-01,
        -3.62027270e-01, 3.20272698e-02])
```

(c) (**0.5 pts**) Using the expression you obtained in part (b), and stacking Eqs. (4),(5), you may arrive at an expression like the following

$$\mathbf{w} = \mathbf{H}\boldsymbol{\theta} + \boldsymbol{\epsilon} \tag{6}$$

where $\mathbf{w} := [\mathbf{w}_m^\intercal, (\mathbf{w}'_m)^\intercal]^\intercal$ and $\boldsymbol{\epsilon} := \left[\boldsymbol{\epsilon}_w^\intercal, \boldsymbol{\epsilon}_{w'}^\intercal\right]^\intercal$. Provide an expression for $\mathbf{H}$

Solution:

$$\mathbf{w} = \begin{pmatrix} \mathbf{w}_m \\ \mathbf{w}'_m \end{pmatrix} = \begin{pmatrix} \mathbf{p_{ij}} \\ \mathbf{p_{ji}} \end{pmatrix} \theta + \boldsymbol{\epsilon} = \begin{pmatrix} diag(b) \times \mathbf{M} \\ -diag(b) \times \mathbf{M} \end{pmatrix} \boldsymbol{\theta} + \boldsymbol{\epsilon} \tag{7}$$

Therefore, we can conclude that

$$\mathbf{H} = \begin{pmatrix} diag(b) \times \mathbf{M} \\ -diag(b) \times \mathbf{M} \end{pmatrix} \tag{8}$$

Thus we obtain H which is a (372,118) matrix

In [17]:

```

#Calculating H for 2.c
m1 = - np.diag(b) @ M
m2 = np.diag(b) @ M
H = np.concatenate((m1,m2), axis=0)
H
```

Out[17]:

```

array([[-10.01001001,  10.01001001,   0.        , ...,   0.        ,
          0.        ,   0.        ],
       [-23.58490566,   0.        ,  23.58490566, ...,   0.        ,
          0.        ,   0.        ],
       [  0.        ,   0.        ,   0.        , ...,   0.        ,
          0.        ,   0.        ],
       ...,
       [  0.        ,   0.        ,   0.        , ...,   0.        ,
         -7.14285714,   0.        ],
       [  0.        ,   0.        ,   0.        , ...,   0.        ,
          0.        , -20.79002079],
       [  0.        ,   0.        ,   0.        , ...,   0.        ,
          0.        , -18.38235294]])
```

(d) (**1 pts**) Using Eq. (6), formulate a weighted least squares and provide a closed-form solution in **matrix-vector form** to calculate $\hat{\boldsymbol{\theta}}$. (**Hint**: Use the Maximum Likelihood Estimate.)

We are provided with the measurement -

$$\mathbf{w} = \mathbf{H}\theta + \epsilon, \quad \epsilon \sim \mathbf{N}(\mathbf{0}, \mathbf{I}\sigma^{\mathbf{2}})$$

We make use of MLE to minimize the Gaussian noise. Maximum Likelihood estimation on PMU measurements yields an unconstrained convex quadratic Weighted Least Squares fit:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} ||\mathbf{w} - \mathbf{H}\theta||_2^2 \tag{9}$$

Which provides us with the following closed form solution in matrix-vector form:

$$\hat{\boldsymbol{\theta}} = (\mathbf{H^T H})^{-1}\mathbf{H^T w} \tag{10}$$

(e) (**0.5 pts**) Generate synthetic measurements by adding Gaussian noise ($\sigma^2 := 0.01$) to the power flow values you calculated in part(b). We will use these values as a proxy to model $\mathbf{w}_m, \mathbf{w}'_m$. However, in the real-world, these measurements would come from sensors.

In [18]:
```
np.random.seed(42)
epsilon_w = np.random.normal(0,0.1,186)
epsilon_w_prime = np.random.normal(0,0.1,186)
w_m = p_ij + epsilon_w
w_m_prime = p_ji + epsilon_w_prime
w = np.concatenate((w_m.T,w_m_prime.T), axis=0).T
np.round(w,3)
```

Out[18]:
```

array([-6.800e-02, -4.060e-01, -9.730e-01, -5.380e-01,  8.480e-01,
        3.280e-01, -4.342e+00,  3.452e+00, -4.547e+00,  7.020e-01,
        7.290e-01,  3.120e-01, -2.930e-01, -2.830e-01, -1.100e-02,
        3.080e-01,  9.800e-02,  5.600e-02, -3.200e-02, -4.900e-02,
       -9.130e-01, -1.800e-01,  8.190e-01,  7.000e-02, -1.620e-01,
        1.430e-01, -4.030e-01, -3.900e-01, -5.880e-01,  1.820e-01,
       -1.755e+00,  1.073e+00,  1.392e+00,  2.240e-01,  2.420e-01,
        2.169e+00,  8.660e-01,  2.056e+00,  3.000e-03, -6.100e-02,
        9.590e-01, -2.870e-01,  1.280e-01,  8.100e-02, -1.460e-01,
       -6.400e-02, -3.840e-01, -1.300e-02,  3.360e-01, -1.110e+00,
        2.458e+00,  5.300e-01,  3.990e-01,  8.670e-01,  4.020e-01,
        2.820e-01, -1.680e-01, -2.120e-01, -1.030e-01,  1.410e-01,
       -3.440e-01, -3.570e-01, -3.930e-01, -2.650e-01, -6.500e-02,
       -4.770e-01, -6.200e-01, -3.880e-01, -3.090e-01,  4.450e-01,
        6.660e-01,  4.340e-01,  9.700e-02,  2.700e-02,  9.600e-02,
        4.370e-01,  7.700e-02,  1.450e-01, -2.000e-01, -4.180e-01,
        3.170e-01, -2.400e-02,  3.280e-01, -3.620e-01, -3.710e-01,
       -3.550e-01, -2.610e-01, -4.030e-01, -5.760e-01, -1.073e+00,
       -8.200e-02,  3.650e-01,  1.449e+00, -1.552e+00,  2.760e-01,
       -1.767e+00, -1.805e+00, -1.227e+00, -1.253e+00, -3.860e-01,
       -3.730e-01, -1.920e-01,  4.770e-01,  5.250e-01, -4.930e-01,
       -3.400e-01, -4.740e-01,  9.400e-01,  3.300e-02,  9.800e-02,
       -1.170e-01,  4.300e-02,  6.600e-02,  4.080e-01, -1.700e-02,
        9.940e-01, -5.220e-01, -7.650e-01,  5.180e-01, -3.090e-01,
        5.040e-01, -3.770e-01, -8.760e-01, -6.090e-01, -6.170e-01,
...
       -2.880e-01, -2.750e-01, -4.950e-01, -4.950e-01, -8.800e-02,
       -3.630e-01, -2.440e-01, -2.660e-01, -1.900e-01, -6.500e-01,
       -9.000e-02,  5.130e-01, -6.910e-01,  2.500e-02,  2.400e-02,
       -1.260e-01, -1.920e-01, -5.000e-03, -1.830e+00, -2.770e-01,
       -3.600e-01,  8.200e-02])
```

(f) (**0.75 pts**) Using $\mathbf{w}$ from part (e), the $\mathbf{H}$ you obtained from (c) and the expression you derived in (d), provide the estimate for $\hat{\boldsymbol{\theta}}$. Plot your results, and compare them to the real values you previously calculated (before adding noise). How good is your estimate?

In [19]:

```
theta_cap = np.linalg.inv(H.T @ H) @ H.T @ w
mape_theta_prime_theta = np.mean((np.abs(theta_powerflow -
    theta_cap)/theta_powerflow))*100
print(f'Mean Absolute Percentage Error between real valued
    theta vs estimate theta_prime for powerflow is: {
    mape_theta_prime_theta}%')
```

Out[19]:

```
Mean Absolute Percentage Error between real valued theta vs
    estimate theta_prime for powerflow is: 77.19595502631205%
```

In [20]:

```
fig = plt.figure(figsize=(10,10))
plt.scatter(theta_cap, theta_powerflow)
plt.xlabel('Theta from DC State Estimation')
plt.ylabel('Voltage Angles DC')
plt.title('Voltage Angle Comparison')
fig.savefig('voltage_angle_comparison_powerflow_vs_estimate.png
    ')
plt.show()
```

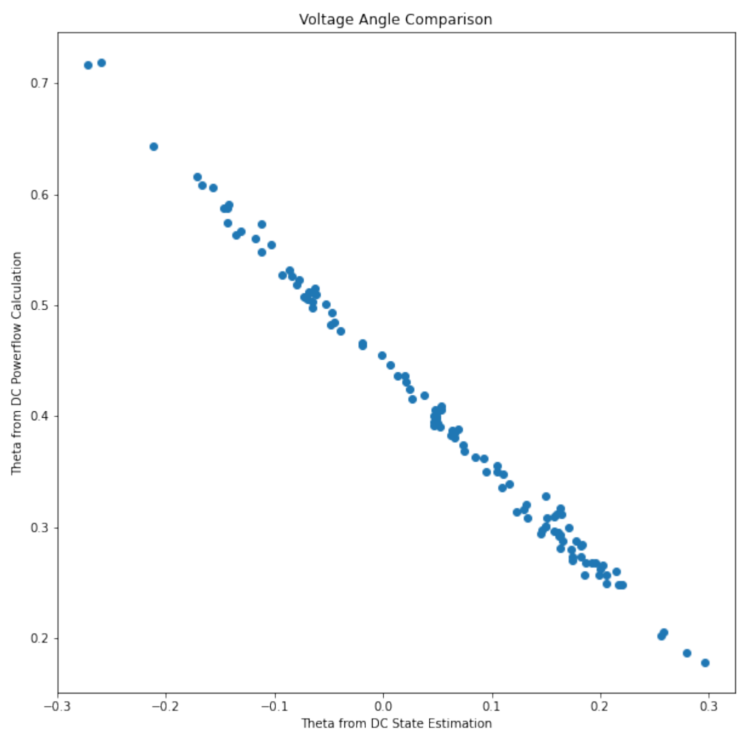

Figure 6: Comparison of Voltage Angles from DC State Estimation vs Power Flow

In [21]:

```python
fig = plt.figure(figsize=(10,10))
plt.scatter(theta_cap, dc_theta, color='r')
plt.xlabel('Theta from DC State Estimation')
plt.ylabel('Voltage Angles DC (Radians)')
plt.title('Voltage Angle Comparison')
fig.savefig('theta_estimate_vs_dc_voltage_angles.png')
plt.show()
```

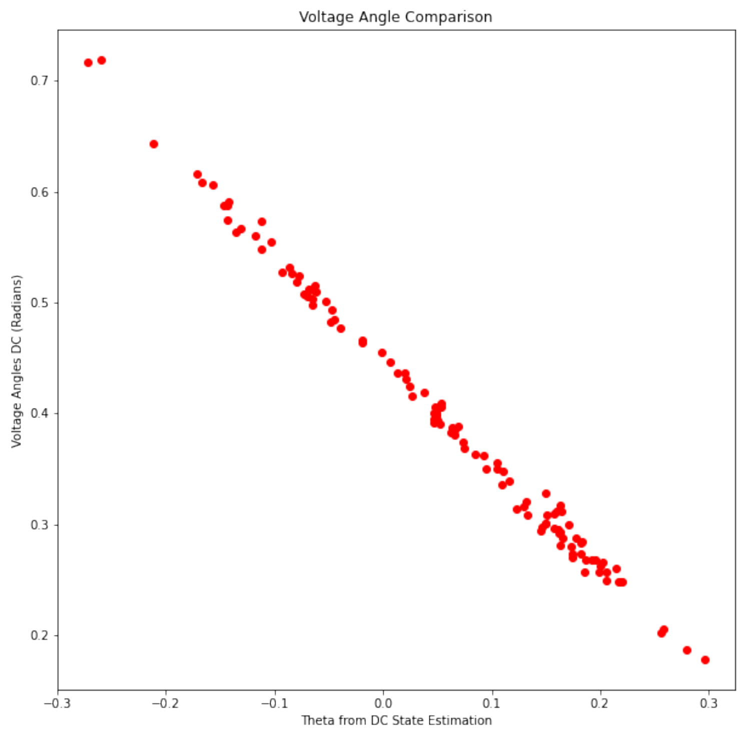

Figure 7: Comparison of Voltage Angles from State Estimation vs DC

In [22]:

```python
fig = plt.figure(figsize=(10,10))
plt.scatter(theta_cap, ac_theta, color='g')
plt.xlabel('Theta from DC State Estimation')
plt.ylabel('Voltage Angles AC (Radians)')
plt.title('Voltage Angle Comparison')
fig.savefig('theta_estimate_vs_ac_voltage_angles.png')
plt.show()
```

(g) (**0.5 pts**) In part (f) you used a model with complete information about the system (you had measurements for every single line in the system). Now, select a subset of measurements (any subset) containing 70% of the measurements and repeat part (f). Plot your results and compare them to the results from part (f). Why did your results change? Are they better or worse? Comment your results.

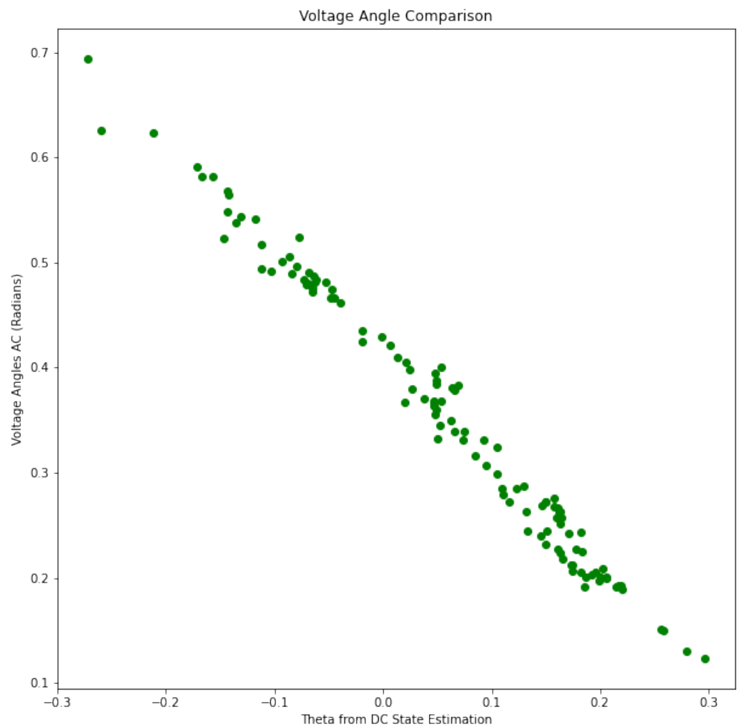

Figure 8: Comparison of Voltage Angles from State Estimation vs AC

Solution:

In [23]:
```python

#Q2.g
mask = np.random.choice([0, 1], size=H.shape, p=[0.3, 0.7])
H_mask = mask*H
theta_cap_new = np.linalg.inv(H_mask.T @ H_mask) @ H_mask.T @ w
mape_theta_prime_theta_cap = np.mean((np.abs(theta_powerflow -
    theta_cap_new)/theta_powerflow))*100
print(f'Mean Absolute Percentage Error between real valued
    theta vs estimate theta_prime after dropping 30 percent of
    estimates for powerflow is: {mape_theta_prime_theta_cap}%')
```

Out[23]:
```
Mean Absolute Percentage Error between real valued theta vs
    estimate theta_prime after dropping 30 percent of estimates for
    powerflow is: 97.65899652776238%
```

It can be observed that the Mean Absolute Percentage Error between $\boldsymbol{\theta}$ and $\hat{\boldsymbol{\theta}}$ increased from 77.2% to 97.65% after dropping 30% of the measurements. This was to be expected since dropping measurements from $\mathbf{H}$ results in a larger residual value $\mathbf{w} - \mathbf{H}\boldsymbol{\theta}$ due to which the final estimate for $\hat{\boldsymbol{\theta}}$ is more noisy than the one calculated before resulting in a greater Mean Absolute Percent Error. It can be observed that dropping measurements from $\mathbf{H}$ results in a more noisy $\mathbf{w}$ due to the greater contribution of $\boldsymbol{\epsilon}$ than before, leading to a more noisy estimate of $\hat{\boldsymbol{\theta}}$. Since $\hat{\boldsymbol{\theta}} = (\mathbf{H^T H})^{-1}\mathbf{H^T w}$

In [24]:

```
mean_absolute_residual_new = np.mean(np.abs(w - H_mask @
    theta_cap_new))
mean_absolute_residual_new
```

Out[24]:

```
0.31729084338563596
```

In [25]:

```
mean_absolute_residual_old = np.mean(np.abs(w - H @ theta_cap))
mean_absolute_residual_old
```

Out[25]:

```
0.11200852761906159
```

On calculating the mean absolute residual before and after removing 30% of the measurements we can also observe that the residual $\mathbf{r} = \mathbf{w} - \mathbf{H}\hat{\boldsymbol{\theta}}$ has increased.