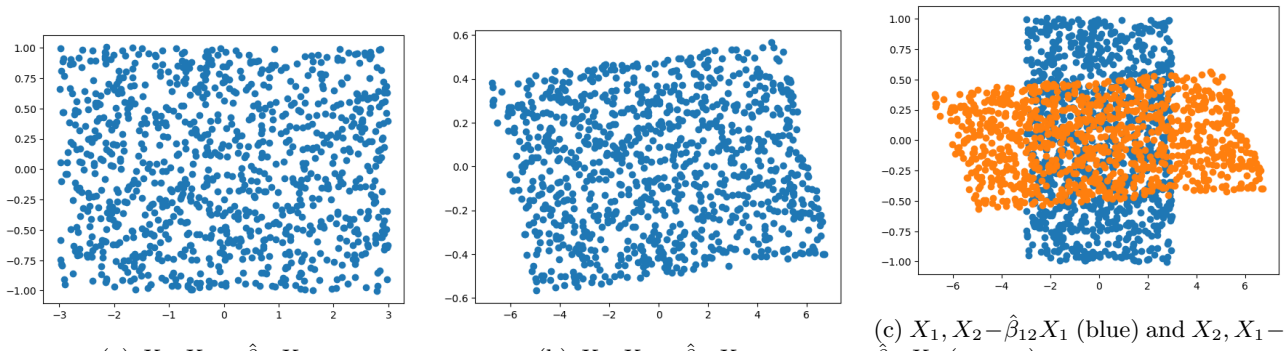
# 6.S091: Problem Set 2

**Hyewon Jeong**
hyewonj@mit.edu

## 1 Problem 1: Identifying direction by non-Gaussianity

**1. Plotting Regression Residuals**
**(a) Let $\hat{\beta}_{12}$ be the linear regression coefficient when regressing $X_2$ onto $X_1$ (without an intercept term). Report $\hat{\beta}_{12}$ and plot $(X_1, X_2 - \hat{\beta}_{12}X_1)$.**
The regression coefficient $\hat{\beta}_{12} = 2.00326517$. The process of reading out the file `nongaussian_samples.csv` and fitting $X_2$ onto $X_1$ using linear regression is in Figure 2. The intercept term was excluded using `interept=False`. The scatterplot between $X_1$ and $X_2 - \hat{\beta}_{12}X_1$ is in Figure 1a.

**(b) Let $\hat{\beta}_{21}$ be the linear regression coefficient when regressing $X_1$ onto $X_2$ (without an intercept term). Report $\hat{\beta}_{21}$ and plot $(X_2, X_1 - \hat{\beta}_{21}X_2)$.**
The regression coefficient $\hat{\beta}_{21} = 0.48538$. The scatterplot between $X_2$ and $X_1 - \hat{\beta}_{21}X_1$ is in Figure 1b. Code segment is in Figure 2.



(a) $X_1, X_2 - \hat{\beta}_{12}X_1$ (b) $X_2, X_1 - \hat{\beta}_{21}X_2$ (c) $X_1, X_2 - \hat{\beta}_{12}X_1$ (blue) and $X_2, X_1 - \hat{\beta}_{21}X_2$ (orange)

Figure 1: Plot of (a) $X_1, X_2 - \hat{\beta}_{12}X_1$, (b) $X_2, X_1 - \hat{\beta}_{21}X_2$, and (c) both.

**Causal Direction Inference [1 point]**
**(c) Which SCM is the data more likely to be generated from? Explain.**
The Figure 1 shows the relationship between the regression input variable for the linear regression model and the estimated parameter $\hat{\epsilon}$. From the SCM $M^a$, $X_1 = \epsilon_1$ and $\hat{\epsilon}_2 = X_2 - \hat{\beta}_{12}X_1$. Also, from the SCM $M^b$, $X_2 = \epsilon_2$ and $\hat{\epsilon}_1 = X_1 - \hat{\beta}_{21}X_2$.

Thus, Figure 1a shows the relationship between $\epsilon_1 = X_1$ and $\hat{\epsilon}_2 = X_2 - \hat{\beta}_{12}X_1$ and Figure 1b shows the relationship betwen $\epsilon_2$ and $\hat{\epsilon}_1$. We can see that $\epsilon_2$ and $\hat{\epsilon}_1$ have some amount of correlation, whereas there seems no correlation between $\epsilon_1$ and $\hat{\epsilon}_2$. We want in SCM M that $\epsilon_1 \perp\!\!\!\perp \epsilon_2$, thus SCM $M^a$ is more probable SCM to be generated from the dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
from sklearn.linear_model import LinearRegression

nongaussian_samples = pd.read_csv('./nongaussian_samples.csv', delimiter=' ', names=['X1', 'X2'])
x_1 = np.asarray(nongaussian_samples['X1']).reshape(-1, 1)
x_2 = np.asarray(nongaussian_samples['X2']).reshape(-1, 1)
```

```
UsageError: Line magic function `%` not found.
```

```
nongaussian_samples.head(10)
```

| | X1 | X2 |
|---|---|---|
| 0 | -1.727048 | -3.599960 |
| 1 | -1.445891 | -3.270109 |
| 2 | -0.458869 | -1.148513 |
| 3 | 1.317966 | 2.516560 |
| 4 | 1.153506 | 2.006051 |
| 5 | 1.369049 | 1.859680 |
| 6 | -1.274701 | -2.909195 |
| 7 | -1.475500 | -2.355980 |
| 8 | 2.765574 | 5.275598 |
| 9 | 0.391240 | -0.158557 |

```
reg_12 = LinearRegression(fit_intercept=False).fit(x_1, x_2)
beta_12 = reg_12.coef_
print (reg_12.score(x_1, x_2), beta_12)
```

```
0.9723446422280946 [[2.00326517]]
```

```
reg_21 = LinearRegression(fit_intercept=False).fit(x_2, x_1)
beta_21 = reg_21.coef_
print (reg_21.score(x_2, x_1), beta_21)
```

```
0.9723448012402138 [[0.48538]]
```

Figure 2: Linear Regression fitting and coefficient calculation for question 1a, 1b.

# 2 The PC Algorithm

## 2.1 Partial correlation

**(a) Report the value of** $\rho(X1, X4, \phi) =$ `compute_partial_correlation(pcalg_samples, 1, 4, [])`
$\rho(X1, X4, \phi) = 0.1851510816056242$

**(b) Report the value of** $\rho(X1, X4, \phi) =$ `compute_partial_correlation(pcalg_samples, 1, 4, [2,3])`
$\rho(X1, X4, \phi) = 0.009334086556151497$
The code segment for 2(a) and (b) is in Figure 3.

## 2.2 Fisher's z-transformation

**(c) Report the value of** $\hat{z}(X_1, X_4, X_2, X_3) =$ `compute_test_statistic(pcalg_samples, 1, 4, [2, 3])`
$\hat{z}(X_1, X_4, X_2, X_3) = 0.9332023767104407$

The code segment used to calculate the fisher z transformation is in Figure 4a.

## 2.3 p-values

**(d) Report the value of** `compute_pvalue(pcalg_samples, 1, 4, [2, 3])`
`compute_pvalue(pcalg_samples, 1, 4, [2, 3])` $= 0.35071548783635986$
The code segment used to calculate the p value of test statistics is in 4b.

```python
def compute_partial_correlation(samples, i, j, S):
    x_i = samples.to_numpy()[:, i-1]
    x_j = samples.to_numpy()[:, j-1]

    if len(S) == 0:
        rho = np.corrcoef(x_i, x_j)
        return rho[0][1]

    S_list = [str(x) for x in S]
    S_sample = np.asarray(samples[S_list])
    reg_is = LinearRegression().fit(S_sample, x_i)
    reg_js = LinearRegression().fit(S_sample, x_j)

    beta_is = reg_is.coef_
    beta_js = reg_js.coef_

    ri = x_i - np.matmul(S_sample, np.transpose(beta_is))
    rj = x_j - np.matmul(S_sample, np.transpose(beta_js))

    rho = pearsonr(np.squeeze(ri), np.squeeze(rj))[0]

    return rho
```

```python
compute_partial_correlation(pcalg_samples, 1, 7, [])
```

```
-0.2916695821495249
```

```python
compute_partial_correlation(pcalg_samples, 1, 7, [3, 4])
```

```
0.013246672696374633
```

```python
#(a)
compute_partial_correlation(pcalg_samples, 1, 4, [])
```

```
0.18515108160562416
```

```python
#(b)
compute_partial_correlation(pcalg_samples, 1, 4, [2,3])
```

```
0.009334086556151497
```

Figure 3: Partial correlation calculation for question 2a, 2b.

**Fisher's z-transformation**

```python
def fisherz(r):
    z = 0.5*np.log((1 + r)/(1 - r))
    return z

def compute_test_statistic(samples, i, j, S):
    n = len(samples)
    s_len = len(S)

    z = np.sqrt(n-s_len-3)*fisherz(compute_partial_correlation(samples, i, j, S))
    return z
```

```python
compute_test_statistic(pcalg_samples, 1, 7, [3, 4])
```

```
1.324413531901477
```

```python
compute_test_statistic(pcalg_samples, 1, 7, [])
```

```
-30.03450548819237
```

```python
#c
compute_test_statistic(pcalg_samples, 1, 4, [2, 3])
```

```
0.9332023767104407
```

(a) Compute Fisher Z statistic

**P value**

```python
from scipy.stats import norm

def compute_pvalue(samples, i, j, S):
    return 2*(1-norm.cdf(np.abs(compute_test_statistic(samples, i, j, S))))
```

```python
compute_pvalue(pcalg_samples, 1, 7, [3, 4])
```

```
0.18536574289813657
```

```python
compute_pvalue(pcalg_samples, 1, 7, [])
```

```
0.0
```

```python
#d
compute_pvalue(pcalg_samples, 1, 4, [2, 3])
```

```
0.35071548783635986
```

(b) Compute P value

Figure 4: Code segment for Fisher Z transformation, Test statistic, and P value.

## 2.4 Skeleton phase

**(e) Report the number of edges in the estimated skeleton when $\alpha = 0.2$ and using only the first 500 rows of samples, i.e., in Python, the number of edges in the skeleton output by** `pcalg_skeleton(samples[:500], 0.2)`
The number of edges with the half of the data samples and the threshold 0.2 is **11** (Figure 5). The code segment used to calculate the p value of test statistics is in 4b.

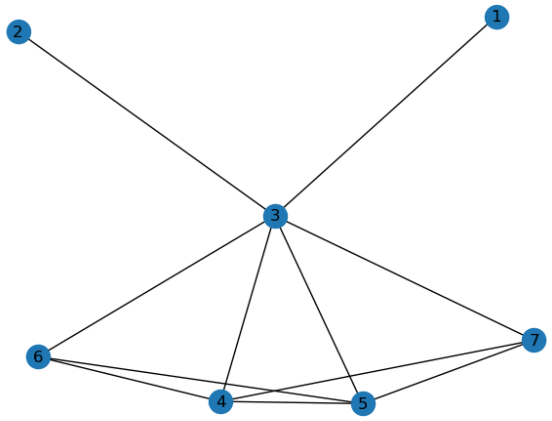



Figure 5: The plot of the graph generated with the Skeleton phase of PC algorithm with 500 samples with p value threshold 0.2

(f) **Report the number of edges in the estimated skeleton when $\alpha = 0.001$ and using only the first 500 rows of samples, i.e., in Python, the number of edges in the skeleton output by** `pcalg_skeleton(samples[:500], 0.001)`
The number of edges with the half of the data samples and the threshold 0.2 is **9** (Figure 6). The code segment used to calculate the p value of test statistics is in 7a.

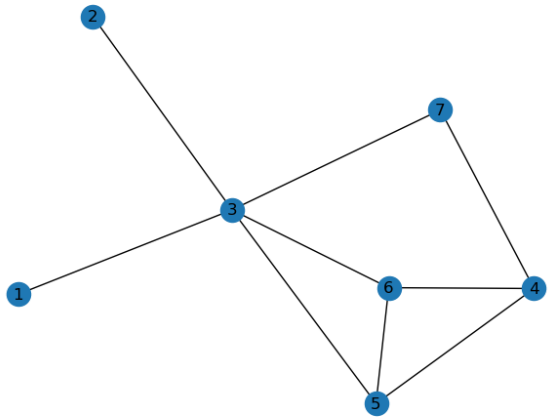



Figure 6: Probability Calculation for question 1a, 1b.

## 2.5 Orientation phase

**(g) Let estimated skeleton, estimated separator function be the output of pcalg skeleton(pcalg samples, 0.05), i.e., the output that you used to check your implementation of pcalg skeleton. What are the unshielded colliders output by `pcalg_orient(estimated skeleton, estimated separator function)`?**
With the code from Figure 7b, we get the nodes $X_1 \to X_3 \leftarrow X_2$ as unshielded colliders for the graph generated by the whole samples and the threshold 0.05.

**(h) Show that, by the Meek rules, you can orient $X_3 \to X_4, X_3 \to X_5, X_3 \to X_6, X_3 \to X_7$. Can you orient any more edges? If not, explain why.**
From orientation phase of PC algorithm, we know that there exists collider $X_1 \to X_3 \leftarrow X_2$ in the graph generated by the dataset and the threshold 0.05. From the first rule of the Meek rule, we can orient $X_3 \to X_4, X_3 \to X_5, X_3 \to X_6, X_3 \to X_7$ as $X_1 \to X_3$ and $X_1$ is not adjacent to $X_4, X_5, X_6, X_7$ (which applies the same for $X_2$).

We cannot apply Rule 2-4 as we don't have the condition required for the rules, and thus the orientation $X_3 \to X_4, X_3 \to X_5, X_3 \to X_6, X_3 \to X_7$ is all that we can add using the Meek rule. For instance, $X_i \to X_3 \to X_j$ with $i = 1, 2$ or $j = 4, 5, 6, 7$ are the candidates for $X_i \to X_k \to X_j$ in rule 2, but the rule cannot be applied as $X_i$, $X_j$ does not have connection beteween them. Also, $X_u \to X_v \to X_j$ can be with the nodes $u = 1, 2, v = 3, j = 4, 5, 6, 7$ in rule 4 but we don't have $X_i$ that connects $X_u$ and $X_v$ thus this rule cannot also be applied here. Finally, $X_1 \to X_3 \leftarrow X_2$ is the only collider pathway we have in the graph and $X_1$ and $X_2$ do not have parents like in rule 3, thus cannot apply this rule here.

```python
import networkx as nx
from more_itertools import powerset

def existence(G, d):
    for i,j in G.edges:
        adj_i = [n for n in G.neighbors(i)]
        adj_i.remove(j)

        if len(adj_i) >= d:
            return True
    return False

def pcalg_skeleton(samples, alpha):
    nodes = [int(x) for x in samples.columns]
    G = nx.complete_graph(len(nodes))
    mapping = {}
    separator = {}
    for i in range(len(nodes)):
        mapping[i] = i+1
    G = nx.relabel_nodes(G, mapping)

    d = 0
    while existence(G, d):
        for i,j in G.edges:
            s_i = [n for n in G.neighbors(i)]
            s_i.remove(j)
            S_i = powerset(s_i)

            for s_item in S_i:
                if len(s_item) == d:
                    if compute_pvalue(samples, i, j, s_item) > alpha:
                        if G.has_edge(i, j):
                            G.remove_edge(i,j)
                            separator[(i,j)] = s_item
                            print(compute_pvalue(samples, i, j, s_item), i, j, s_i, s_item)

            s_j = [n for n in G.neighbors(j)]
            if i in s_j:
                s_j.remove(i)
            S_j = powerset(s_j)

            for s_item in S_j:
                if len(s_item) == d:
                    if compute_pvalue(samples, i, j, s_item) > alpha:
                        if G.has_edge(i, j):
                            G.remove_edge(i,j)
                            separator[(i,j)] = s_item
                            print(compute_pvalue(samples, i, j, s_item), i, j, s_i, s_item)


        d += 1
    print(G.number_of_edges())
    return G, separator
```

(a) Skeleton Phase

```python
def pcalg_orient(G, s):
    unshielded = []
    for k in G.nodes:
        neighbors = [n for n in G.neighbors(k)]

        if len(neighbors) < 2:
            continue

        else:
            print(k)
            colliders = list(itertools.combinations(neighbors, 2))
            print(colliders)

            for i, j in colliders:
                if not G.has_edge(i, j):
                    if k not in s[(i, j)]:
                        unshielded.append((i, k, j))

    return unshielded
```

(b) Orientation Phase

Figure 7: Code segment for the PC algorithm.