In [1]:
```
import pyro
import pyro.distributions as dist
from pyro.infer import Importance, EmpiricalMarginal
import matplotlib.pyplot as plt
import torch
import numpy as np
import pandas as pd
import random
```

1. 4

In [2]:
```
def model():
    Nx = pyro.sample('Nx', dist.Bernoulli(torch.tensor(0.5)))
    Ny = pyro.sample('Ny', dist.Bernoulli(torch.tensor(0.5)))
    Nz = pyro.sample('Nz', dist.Bernoulli(torch.tensor(0.5)))
    X = pyro.sample('X', dist.Delta(Nx))
    Y = pyro.sample('Y', dist.Delta(Ny))
    ZVal = (Nz * (min((X+Y), torch.tensor(1.0)))) + ((torch.tensor(1.
0) - Nz) * min((X*Y), torch.tensor(1.0)))
    Z = pyro.sample('Z', dist.Delta(ZVal))
    return X, Y, Z
```

1.5

In [3]:
```
conditoned_model = pyro.condition(model, data = {"X": torch.tensor(1.
), "Z": torch.tensor(1.)})
```

In [4]:
```
posterior = Importance(conditoned_model, num_samples=1000).run()
```

Abduction Step (1.6.1)

In [7]:
```
for _ in range(1):
    trace = posterior()
    print(f"Nx is {trace.nodes['Nx']['value']}")
    print(f"Ny is {trace.nodes['Ny']['value']}")
    print(f"Nz is {trace.nodes['Nz']['value']}")
```

```
Nx is 1.0
Ny is 0.0
Nz is 1.0
```

Action Step (1.6.2)

In [8]:
```
intervention_model = pyro.do(model, data = {"X": torch.tensor(0.)})
```

Prediction Step (1.6.3)

In [9]:
```
prediction_model = pyro.condition(intervention_model, data={"Nx": tor
ch.tensor(0.), "Ny": torch.tensor(1.), "Nz": torch.tensor(0.)})
```

In [10]:
```
trace_prediction = pyro.poutine.trace(prediction_model)
for _ in range(1):
    trace = trace_prediction.get_trace()
    print(f"Z is {trace.nodes['Z']['value']}")
```

```
Z is 0.0
```

1.6.d

In [11]:
```
def counterfactual(posterior):
    # Abduction Step
    #Nx = torch.tensor(0.)
    #Ny = torch.tensor(0.)
    #Nz = torch.tensor(0.)
    # Get abduction samples
    for _ in range(1):
        trace = posterior()
        Nx = trace.nodes['Nx']['value']
        Ny = trace.nodes['Ny']['value']
        Nz = trace.nodes['Nz']['value']

    # Intervention Step
    intervention_model = pyro.do(model, data = {"X": torch.tensor(0.
)})

    # Condition Step
    prediction_model = pyro.condition(intervention_model, data={"Nx":
Nx, "Ny": Ny, "Nz": Nz})
    trace_prediction = pyro.poutine.trace(prediction_model)
    for _ in range(1):
        trace = trace_prediction.get_trace()
        z = trace.nodes['Z']['value']
        return z
```

In [12]:
```
conditoned_model = pyro.condition(model, data = {"X": torch.tensor(1.
), "Z": torch.tensor(1.)})
posterior = Importance(conditoned_model, num_samples=1000).run()
counterfactual_samples = [counterfactual(posterior) for _ in range(10
00)]
```

In [13]:
```
PZ1_X1Z1 = sum(counterfactual_samples)/len(counterfactual_samples)
```

In [14]:
```
pZ0_x1Z1 = 1 - PZ1_X1Z1
```

In [15]:
```
pZ0_x1Z1
```

Out[15]:
```
tensor(0.6710)
```

In [16]:
```
PZ1_X1Z1
```

Out[16]:
```
tensor(0.3290)
```

2.1

In [17]:
```
def scm():
    Nx = pyro.sample('Nx', dist.Bernoulli(torch.tensor(0.5)))
    Nq = pyro.sample('Nq', dist.Bernoulli(torch.tensor(0.9)))
    Ny = pyro.sample('Ny', dist.Bernoulli(torch.tensor(0.2)))
    X = pyro.sample('X', dist.Delta(Nx))
    Q = pyro.sample('Q', dist.Delta(Nq))
    YVal = (X and Q) or Ny
    Y = pyro.sample('Y', dist.Delta(YVal))
    return X, Q, Y
```

In [18]:
```
def counterfactualscm(posterior, intervention_model):

    Nx = torch.tensor(0.)
    Ny = torch.tensor(0.)
    Nq = torch.tensor(0.)
    for _ in range(1):
        trace = posterior()
        Nx = trace.nodes['Nx']['value']
        Ny = trace.nodes['Ny']['value']
        Nq = trace.nodes['Nq']['value']

    # Prediction Step
    prediction_model = pyro.condition(intervention_model, data={"Nx":
Nx, "Ny": Ny, "Nq": Nq})
    trace_prediction = pyro.poutine.trace(prediction_model)
    for _ in range(1):
        trace = trace_prediction.get_trace()
        y = trace.nodes['Y']['value']
        return y
```

In [31]:
```
condition_pn = pyro.condition(scm, data = {"X": torch.tensor(1.), "Y"
: torch.tensor(1.)})
# Intervention Step
intervention_model = pyro.do(scm, data = {"X": torch.tensor(0.)})
posterior = Importance(condition_pn, num_samples=1000).run()
pn = [counterfactualscm(posterior, intervention_model) for _ in range
(1000)]
```

In [32]:
```
pn = 1 - (sum(pn)/len(pn))
```

In [33]:
```
pn
```

Out[33]:
```
tensor(0.7920)
```

In [43]:
```
condition_ps = pyro.condition(scm, data = {"X": torch.tensor(0.), "Y"
: torch.tensor(0.)})
# Intervention Step
intervention_model_ps = pyro.do(scm, data = {"X": torch.tensor(1.)})
posterior_ps = Importance(condition_ps, num_samples=1000).run()
ps = [counterfactualscm(posterior_ps, intervention_model_ps) for _ in
range(1000)]
```

In [44]:
```
ps = sum(ps)/len(ps)
```

In [45]:
```
ps
```

Out[45]:
```
tensor(0.9000)
```