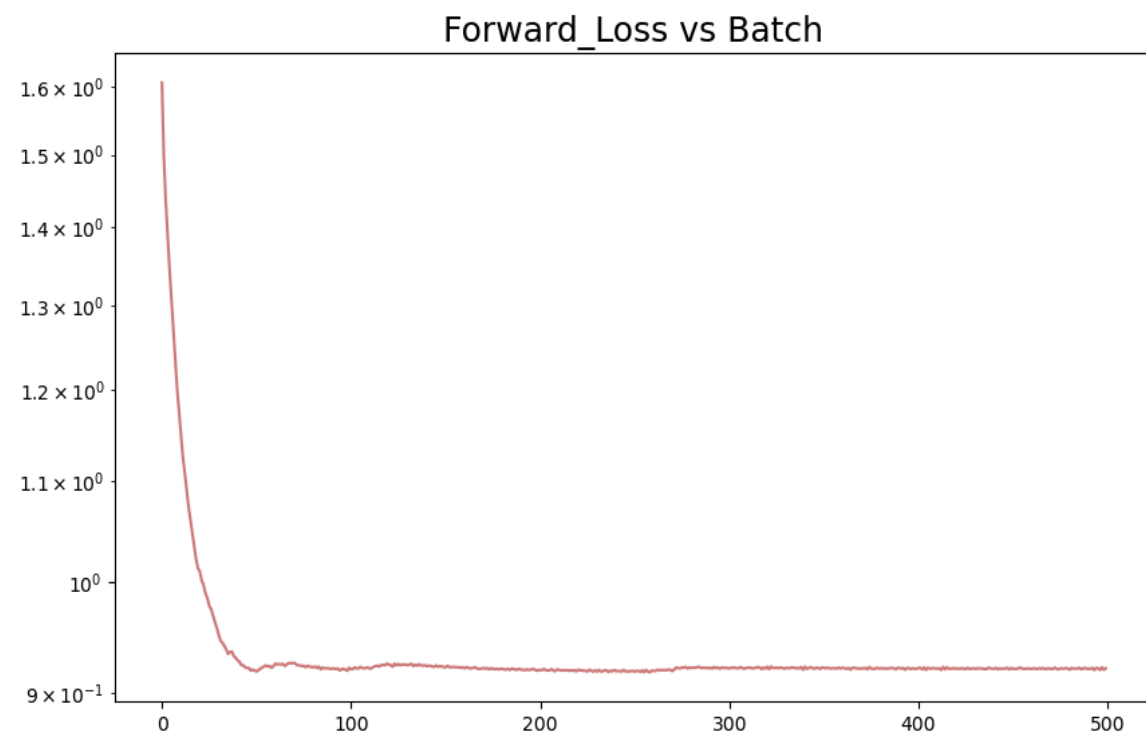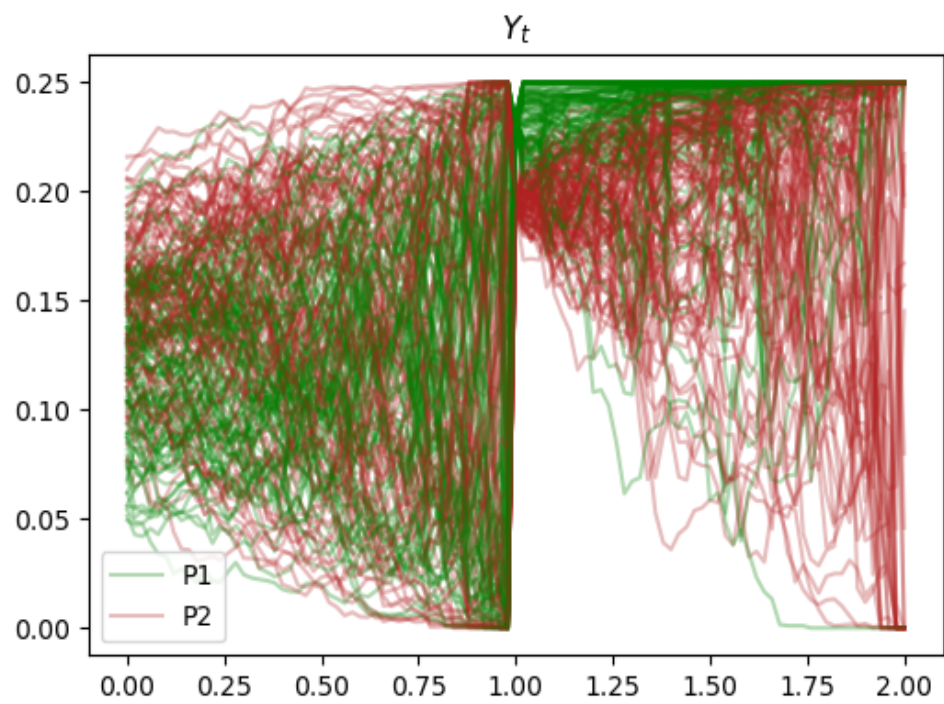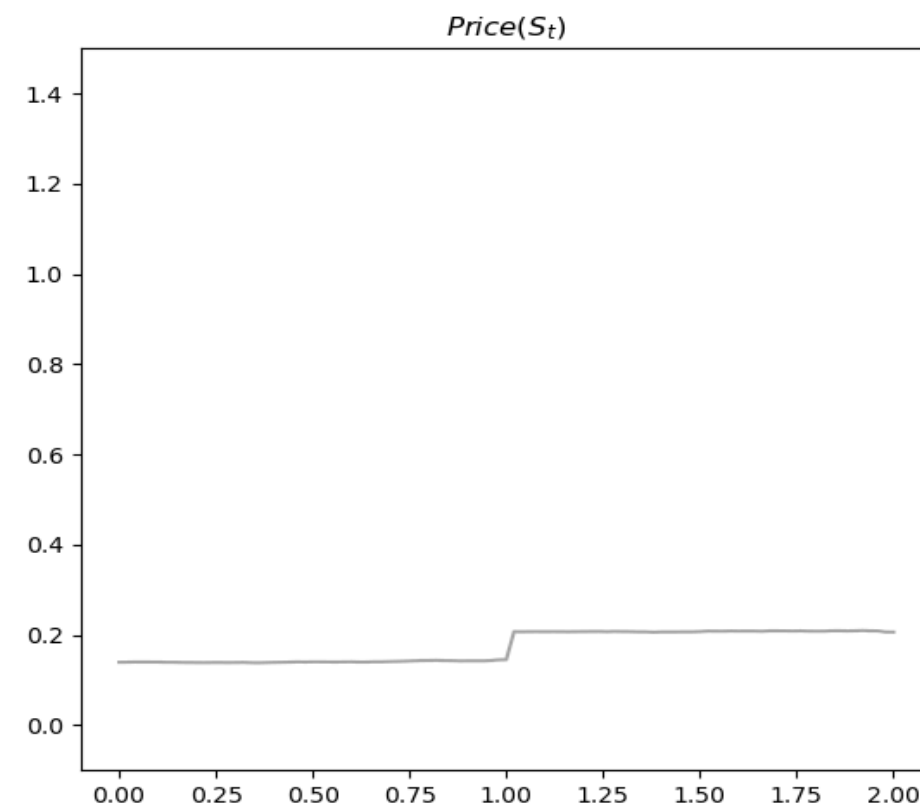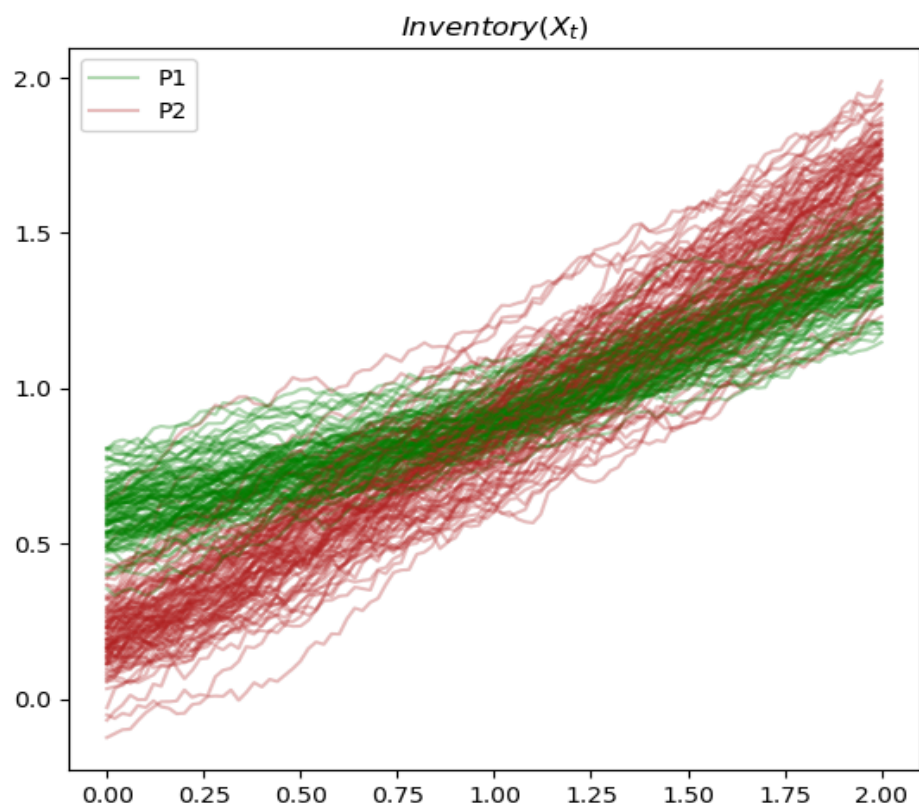| Expansion Rate | Generation Rate | Trading Rate |
| Accumulated Expansion | Accumulated Generation | Accumulated Trading |

```
,target_type='indicator',trick='clamp',loss_type='MSELoss',delta=0.01,w=0.25,lr=0.001)
```
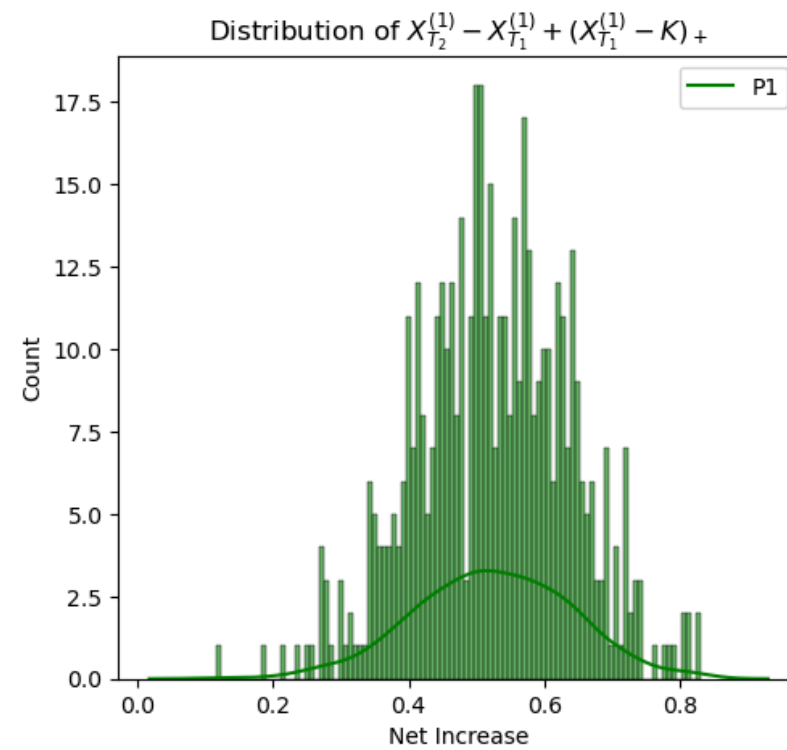
500 steps
Started @ August 30 - 21:09:37
Saved @ August 31 - 01:14:33
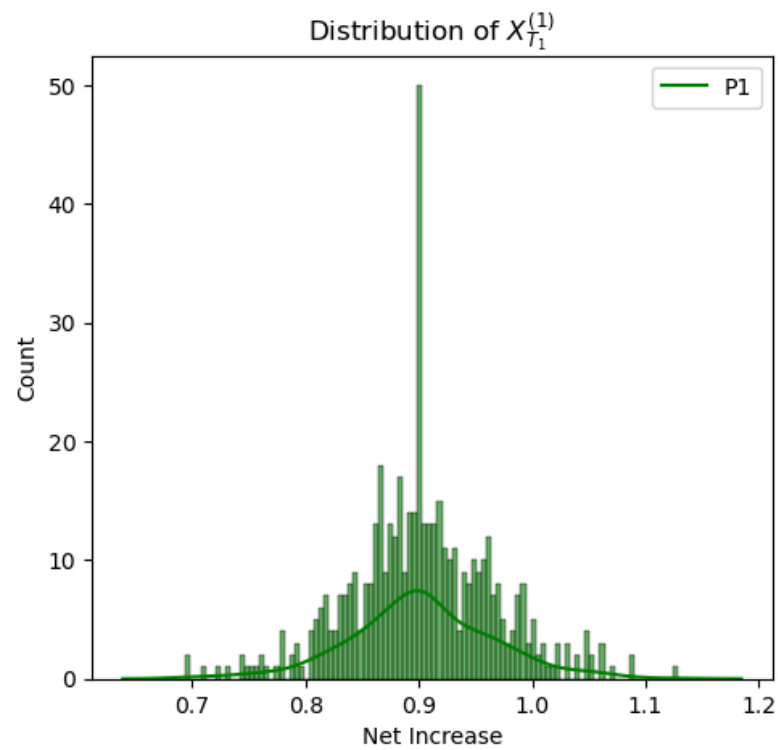
$Inventory(X_t)$

$Price(S_t)$

Y_t

Forward_Loss vs Batch

Periodic Inventory Increase - P1

Distribution of $X_{T_1}^{(1)}$

Distribution of $X_{T_2}^{(1)} - X_{T_1}^{(1)} + (X_{T_1}^{(1)} - K)_+$

Periodic Inventory Increace - P2

Distribution of $X_{T_1}^{(2)}$

Distribution of $X_{T_2}^{(2)} - X_{T_1}^{(2)} + (X_{T_1}^{(2)} - K)_+$

Convergency - P1

Distribution of $Y_{T_1}^{(1)}$ — Distribution of $Y_{T_2}^{(1)}$

Convergency - P2

Distribution of $Y_{T_1}^{(2)}$ — Distribution of $Y_{T_2}^{(2)}$

QQ-Plot of Y_{T_1} — QQ-Plot of Y_{T_2}
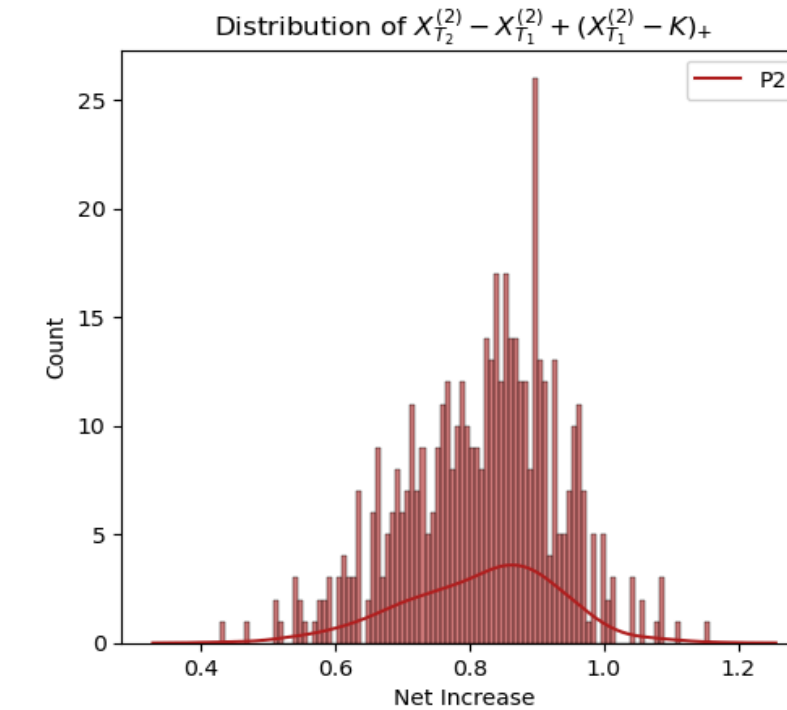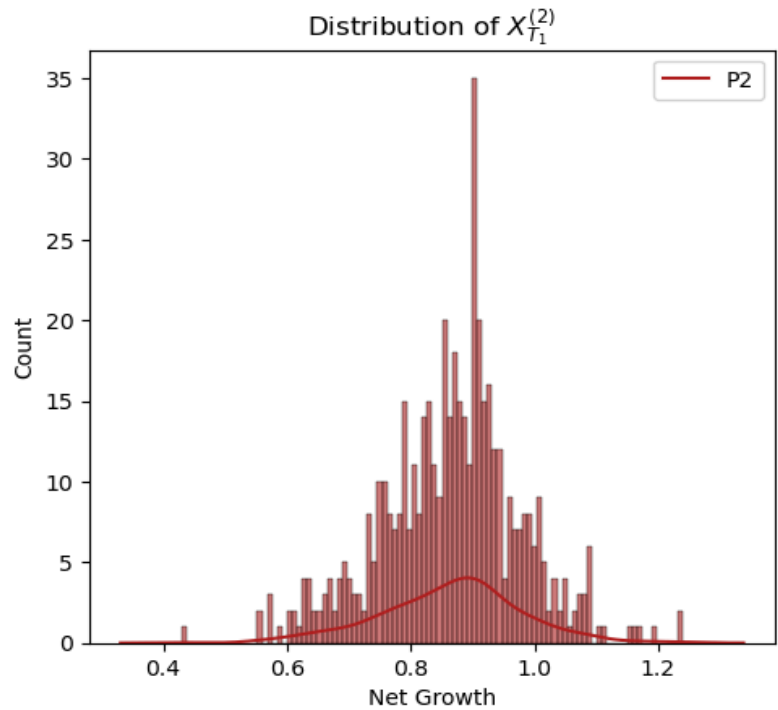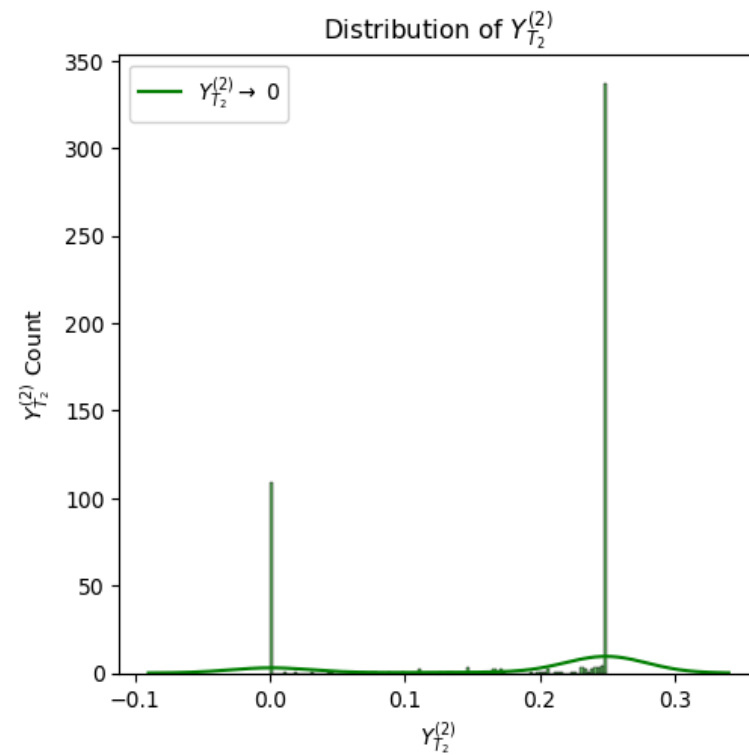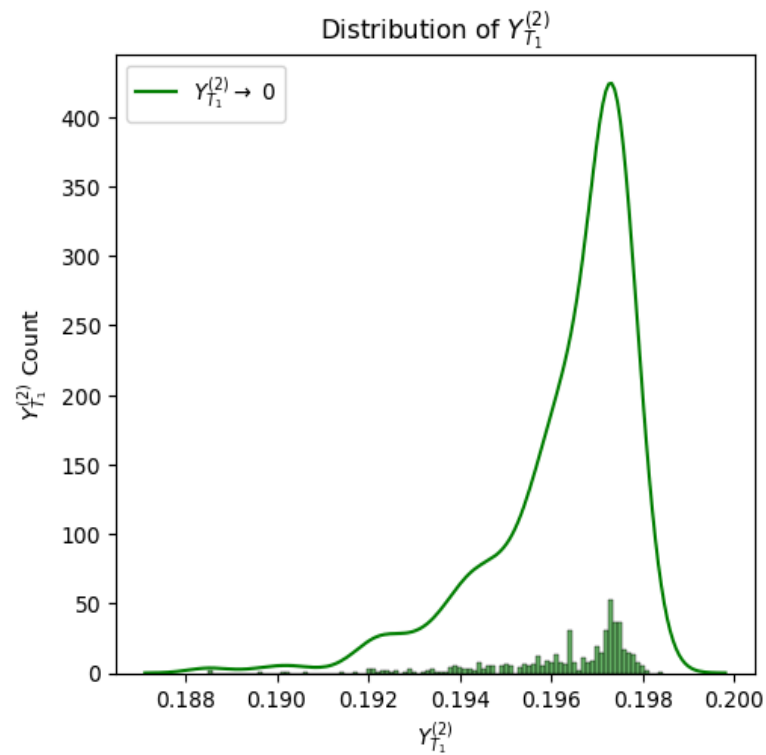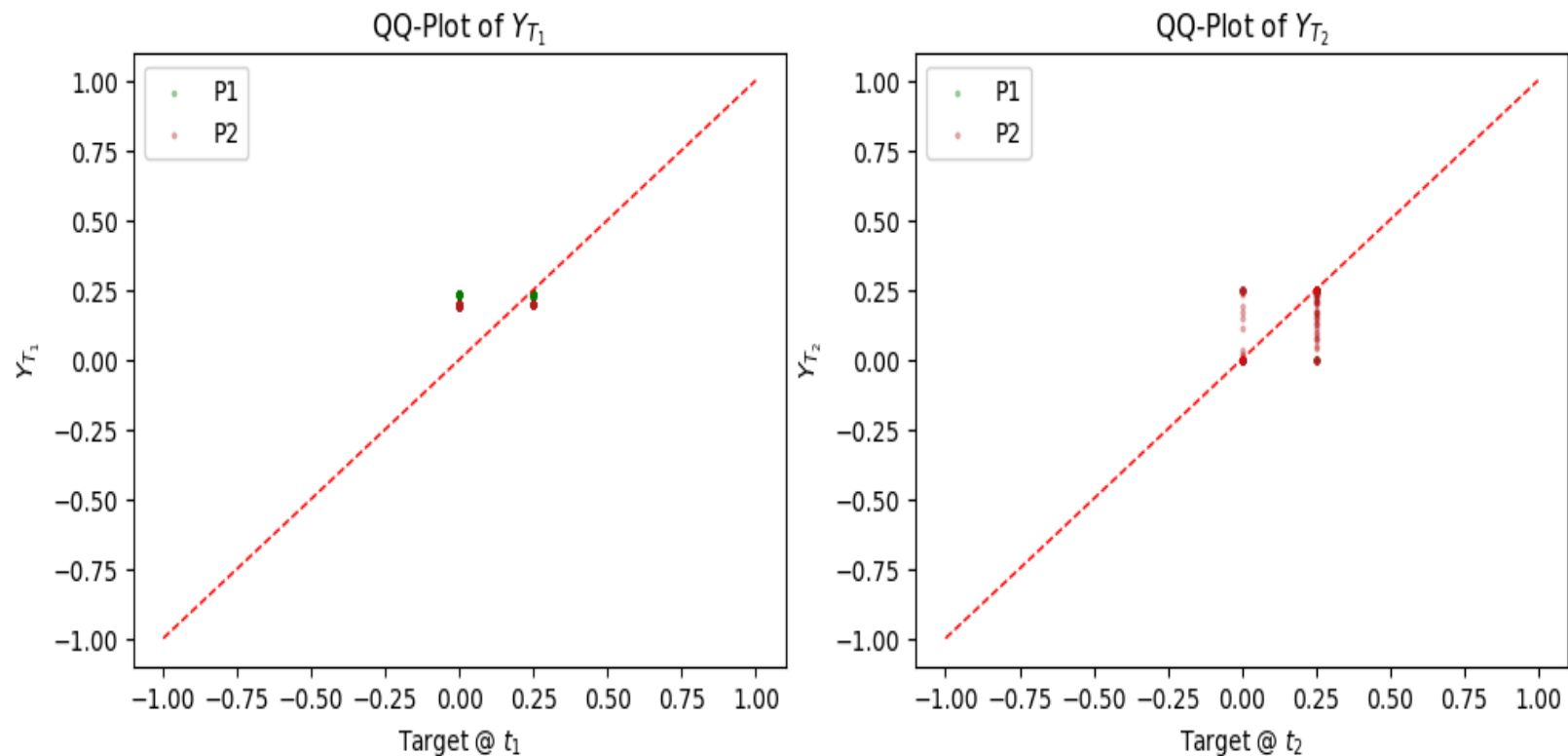
Save The Models

```
print(f"{len(main_models1.loss)} steps\nStarted @ {start_time}\nSaved @ {end_time}")    ## to examine whether the loss attribute is
dir_path=pathlib.Path(os.getcwd(),
                      'Results',
                      'Best Models Saved',
                      'sigmoid_ind_0.001lr_500steps_MSE_0.25w') ## 0.75, 0.5, 0.25
dir_path.mkdir()
path1=pathlib.Path(dir_path,'pop1.pt')
path2=pathlib.Path(dir_path,'pop2.pt')
main_models1.save_entire_models(path=path1)
main_models2.save_entire_models(path=path2)
```

```
500 steps
Started @ August 30 - 21:09:37
Saved @ August 31 - 01:14:33
```

```python
import numpy as np
import torch as torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import random
from scipy.stats import norm

import os
import pathlib

from Model import *
from utils import *

torch.autograd.set_detect_anomaly(True)
start_time=datetime.now().strftime('%B %d - %H:%M:%S')
```
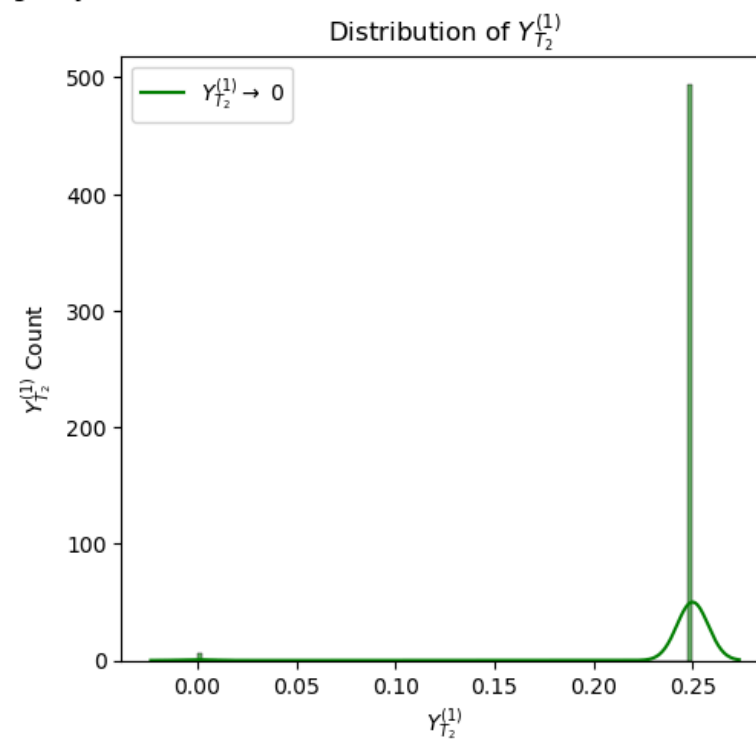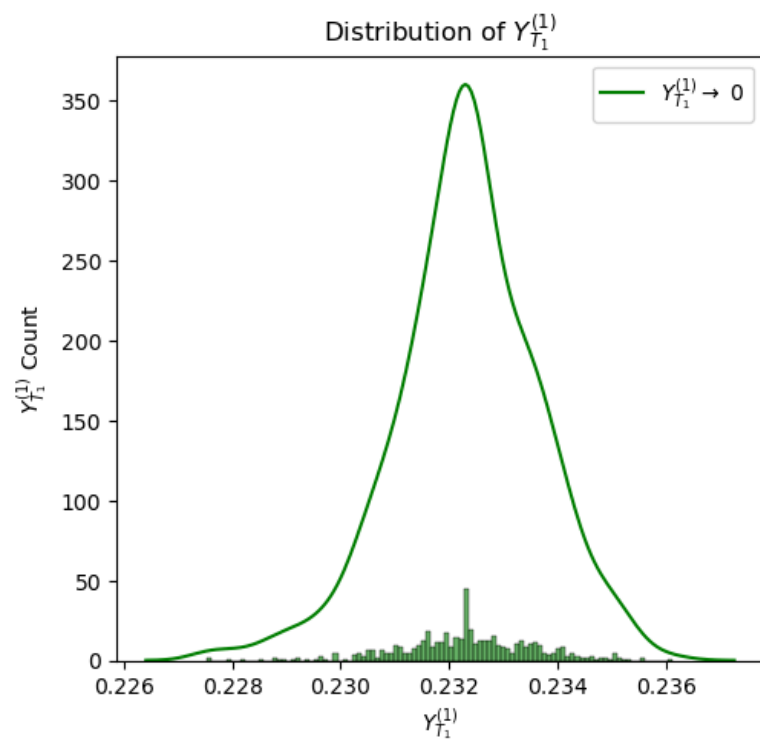
```python
#Global parameters
GlobalParams1=Params(param_type='k1',target_type='indicator',trick='clamp',loss_type='MSELoss',delta=0.01,w=0.25,lr=0.001)
dB1 = SampleBMIncr(GlobalParams=GlobalParams1)
init_x1 =  Sample_Init(GlobalParams=GlobalParams1)
init_c1= torch.zeros_like(init_x1)

GlobalParams2=Params(param_type='k2',target_type='indicator',trick='clamp',loss_type='MSELoss',delta=0.01,w=0.25,lr=0.001)
dB2 = SampleBMIncr(GlobalParams=GlobalParams2)   ## TODO: same dB?????
init_x2 =  Sample_Init(GlobalParams=GlobalParams2)
init_c2= torch.zeros_like(init_x2)

NT1=GlobalParams1.NT1
NT2=GlobalParams1.NT2
dt=GlobalParams1.dt
device=GlobalParams1.device
learning_rate = GlobalParams1.lr

#Forward Loss
forward_losses = []

#How many batches
MaxBatch= 500
```

```python
#How many optimization steps per batch
OptimSteps= 25

#Train on a single batch?
single_batch = True

#Set up main models for y0, yt1, and z (z will be list of models)
y0_model_main1 = Network(scaler_type='sigmoid')
yt1_model_main1 = Network(scaler_type='sigmoid')
zy_models_main1 = [Network() for i in range(NT2)]
main_models1=Main_Models(GlobalParams=GlobalParams1)
main_models1.create(y0_model=y0_model_main1,
                    yt1_model=yt1_model_main1,
                    zy_models=zy_models_main1,
                    forward_loss=forward_losses,
                    dB=dB1,
                    init_x=init_x1,
                    init_c=init_c1)

y0_model_main2 = Network(scaler_type='sigmoid')
yt1_model_main2 = Network(scaler_type='sigmoid')
zy_models_main2 = [Network() for i in range(NT2)]
main_models2=Main_Models(GlobalParams=GlobalParams2)
main_models2.create(y0_model=y0_model_main2,
                    yt1_model=yt1_model_main2,
                    zy_models=zy_models_main2,
                    forward_loss=forward_losses,
                    dB=dB2,
                    init_x=init_x2,
                    init_c=init_c2)

pop1_dict={'dB':dB1,
           'init_x':init_x1 ,
           'init_c':init_c1 ,
           'GlobalParams':GlobalParams1,
           'main_models':main_models1}

pop2_dict={'dB':dB2,
           'init_x':init_x2 ,
           'init_c':init_c2 ,
           'GlobalParams':GlobalParams2,
           'main_models':main_models2}
```

```python
#Define optimization parameters
params=[]
params = list(main_models1.y0_model.parameters())+\
        list(main_models1.yt1_model.parameters())+\
        list(main_models2.y0_model.parameters())+\
        list(main_models2.yt1_model.parameters())

for i in range(NT2):
    params += list(main_models1.zy_models[i].parameters())
    params += list(main_models2.zy_models[i].parameters())

#Set up optimizer and scheduler
optimizer = optim.Adamax(params, lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=100, gamma=0.95)

for k in range(0,MaxBatch):

    print("Batch Number: ", k+1)
    sloss=0
    #optimize main network wrt the foward loss
    for l in range(0,OptimSteps):
        optimizer.zero_grad()
        loss = get_foward_loss(pop1_dict=pop1_dict, pop2_dict=pop2_dict)
        loss.backward()
        # torch.nn.utils.clip_grad_norm_(parameters=params,max_norm=0.7)
        optimizer.step()
        scheduler.step()
        nloss = loss.detach().numpy()
        sloss += nloss
        # print('OptimStep: '+ str(l+1))
        # print('forward_loss: ' + str(nloss))
    avgloss = sloss/OptimSteps
    print("Average Error Est: ", avgloss)
    forward_losses.append(avgloss)

    #Generate a new batch if using multiple batches
    if(not single_batch):

        dB1 = SampleBMIncr(GlobalParams=GlobalParams1)
        init_x1 =  Sample_Init(GlobalParams=GlobalParams1)
        init_c1= torch.zeros_like(init_x1)
        pop1_dict={'dB':dB1,
                'init_x':init_x1 ,
                'init_c':init_c1 ,
```