In [ ]:
```
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')
```

In [ ]:
```
#Global parameters
GlobalParams1=Params(param_type='k1',target_type='indicator',trick='logit',loss_type='BCEWithLogitsLoss',delta=0.03,w=0.5,lr=0.0005,NT1=0)
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='logit',loss_type='BCEWithLogitsLoss',delta=0.03,w=0.5,lr=0.0005,NT1=0)
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

#How many optimization steps per batch
OptimSteps= 25

#Train on a single batch?
single_batch = True
```

```
#Set up main models for y0 and z (z will be list of models)
v0_model_main1 = Network(scaler_type='sigmoid')
u0_model_main1 = Network(scaler_type='sigmoid')
y0_model_main1 = Network(scaler_type='sigmoid')

zv_models_main1 = [Network() for i in range(NT1)]
zu_models_main1 = [Network() for i in range(NT1)]
zy_models_main1 = [Network() for i in range(NT2)]
main_models1=Main_Models(GlobalParams=GlobalParams1)
main_models1.create(v0_model=v0_model_main1,
                    u0_model=u0_model_main1,
                    y0_model=y0_model_main1,
                    zv_models=zv_models_main1,
                    zu_models=zu_models_main1,
                    zy_models=zy_models_main1,
                    forward_loss=forward_losses,
                    dB=dB1,
                    init_x=init_x1,
                    init_c=init_c1)

v0_model_main2 = Network(scaler_type='sigmoid')
u0_model_main2 = Network(scaler_type='sigmoid')
y0_model_main2 = Network(scaler_type='sigmoid')

zv_models_main2 = [Network() for i in range(NT1)]
zu_models_main2 = [Network() for i in range(NT1)]
zy_models_main2 = [Network() for i in range(NT2)]
main_models2=Main_Models(GlobalParams=GlobalParams2)
main_models2.create(v0_model=v0_model_main2,
                    u0_model=u0_model_main2,
                    y0_model=y0_model_main2,
                    zv_models=zv_models_main2,
                    zu_models=zu_models_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}
```

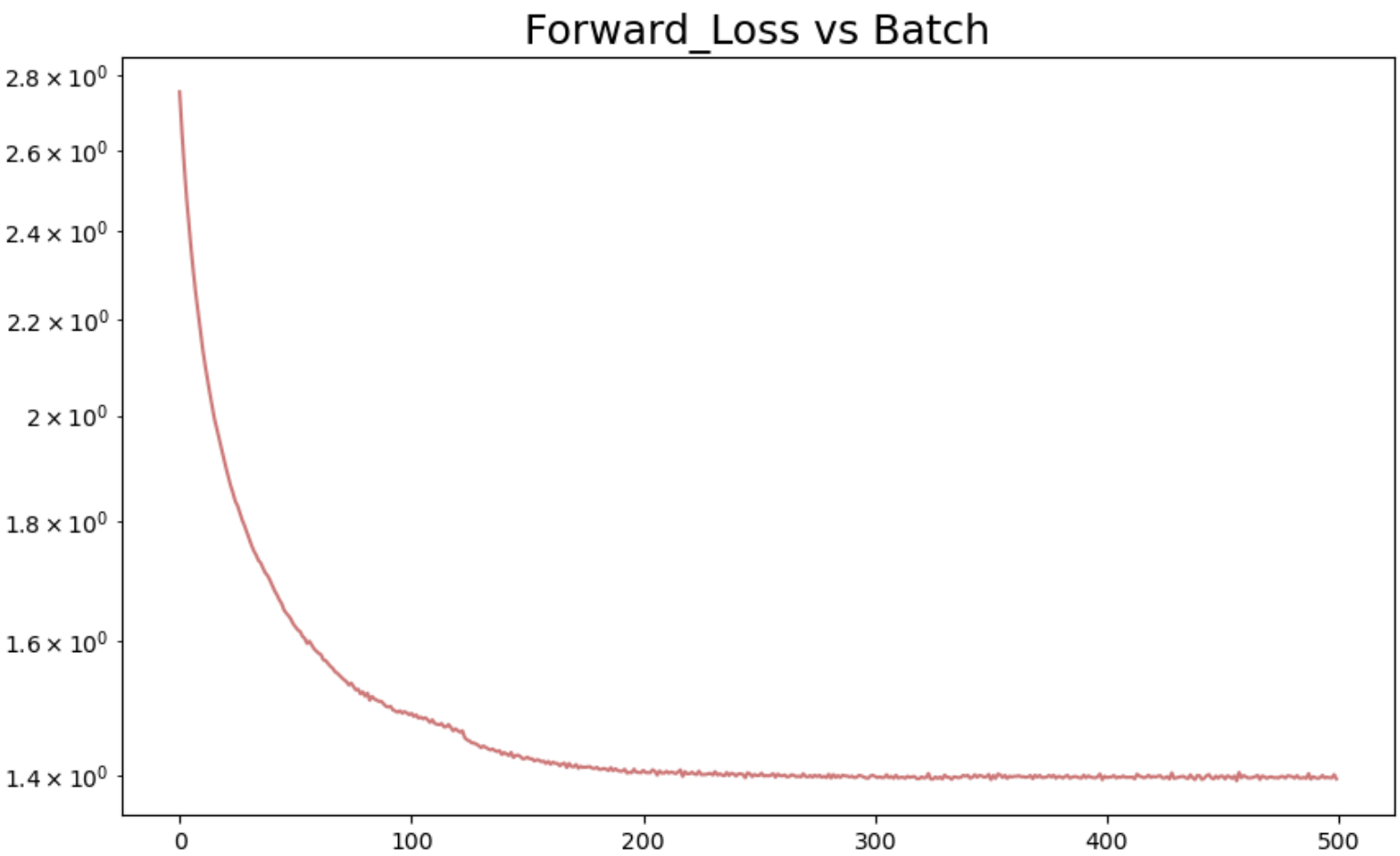
In [ ]:

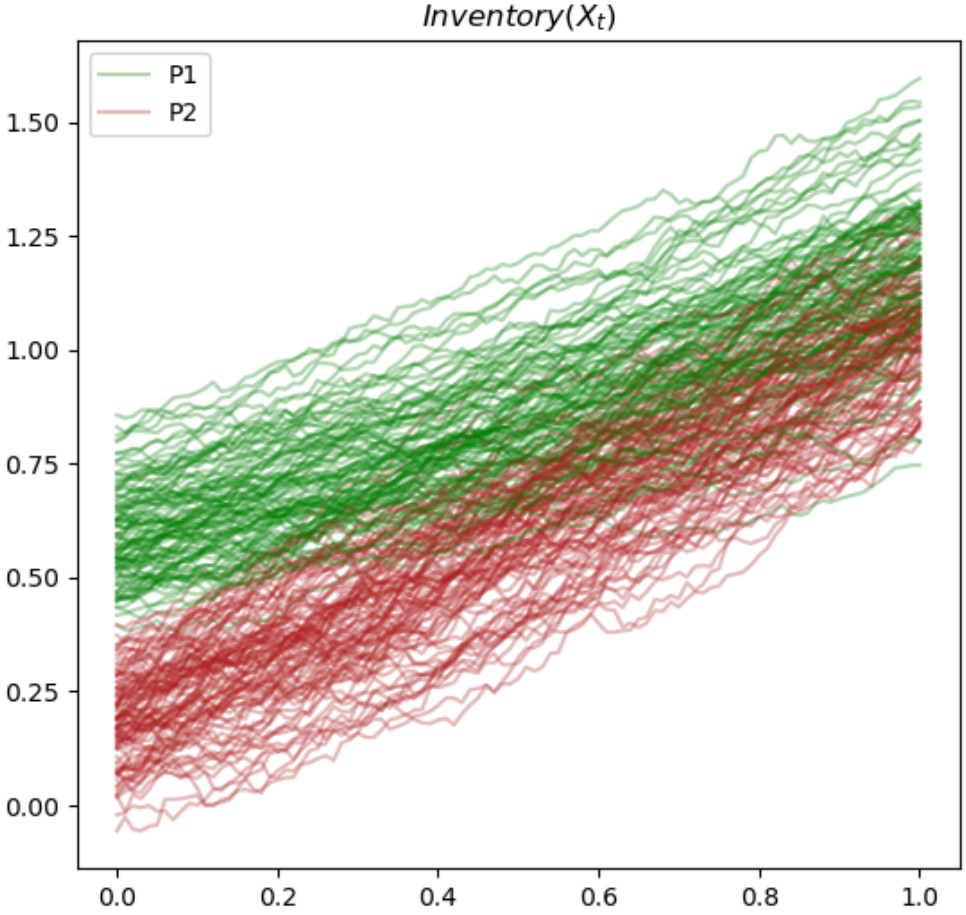
```
#Define optimization parameters
params=[]
params = list(main_models1.v0_model.parameters())+\
         list(main_models1.u0_model.parameters())+\
         list(main_models1.y0_model.parameters())+\
         list(main_models2.v0_model.parameters())+\
         list(main_models2.u0_model.parameters())+\
         list(main_models2.y0_model.parameters())
for i in range(NT1):
    params += list(main_models1.zv_models[i].parameters())
    params += list(main_models1.zu_models[i].parameters())
    params += list(main_models2.zv_models[i].parameters())
    params += list(main_models2.zu_models[i].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 ,
```

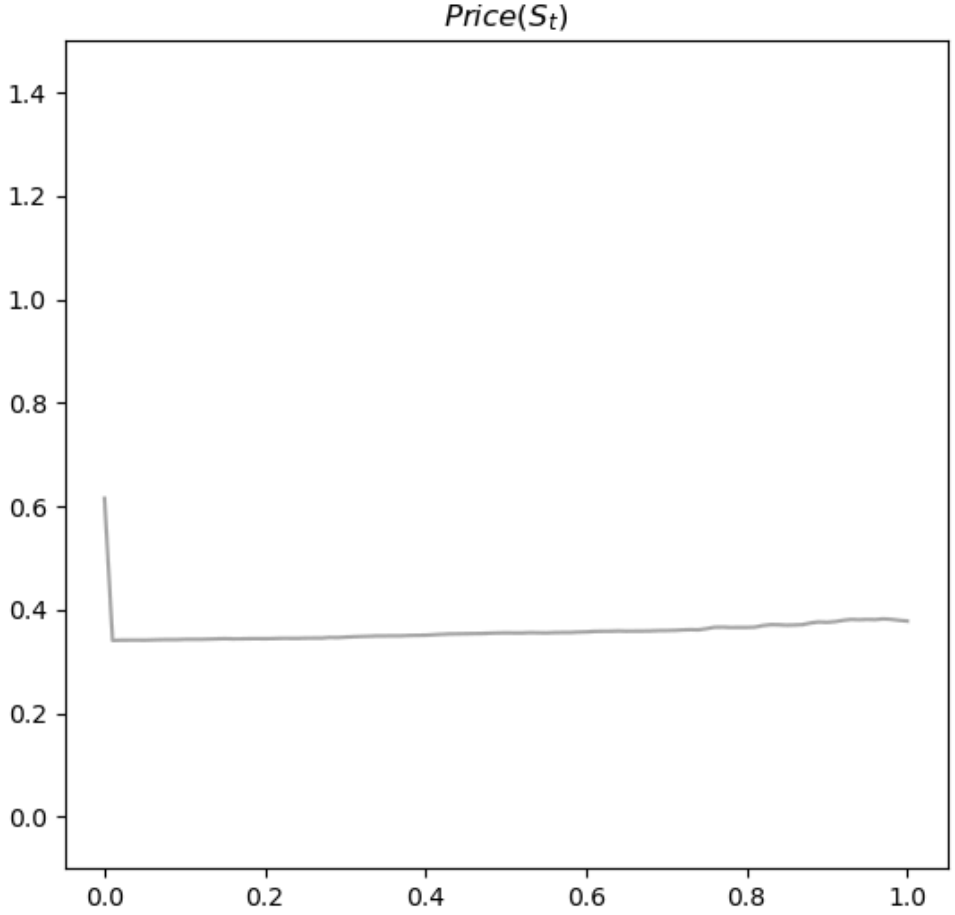
```
Average Error Est:  1.397084345817566
Batch Number:  486
Average Error Est:  1.3978942728042603
Batch Number:  487
Average Error Est:  1.3966359567642213
Batch Number:  488
Average Error Est:  1.4031148338317871
Batch Number:  489
Average Error Est:  1.3956307697296142
Batch Number:  490
Average Error Est:  1.3988628721237182
Batch Number:  491
Average Error Est:  1.3975618886947632
Batch Number:  492
Average Error Est:  1.3964572763442993
Batch Number:  493
Average Error Est:  1.3977624559402466
Batch Number:  494
Average Error Est:  1.4006269788742065
Batch Number:  495
Average Error Est:  1.3971956586837768
Batch Number:  496
Average Error Est:  1.398305344581604
Batch Number:  497
Average Error Est:  1.3975434303283691
Batch Number:  498
Average Error Est:  1.3972869396209717
Batch Number:  499
Average Error Est:  1.4017788648605347
Batch Number:  500
Average Error Est:  1.3960903835296632
```

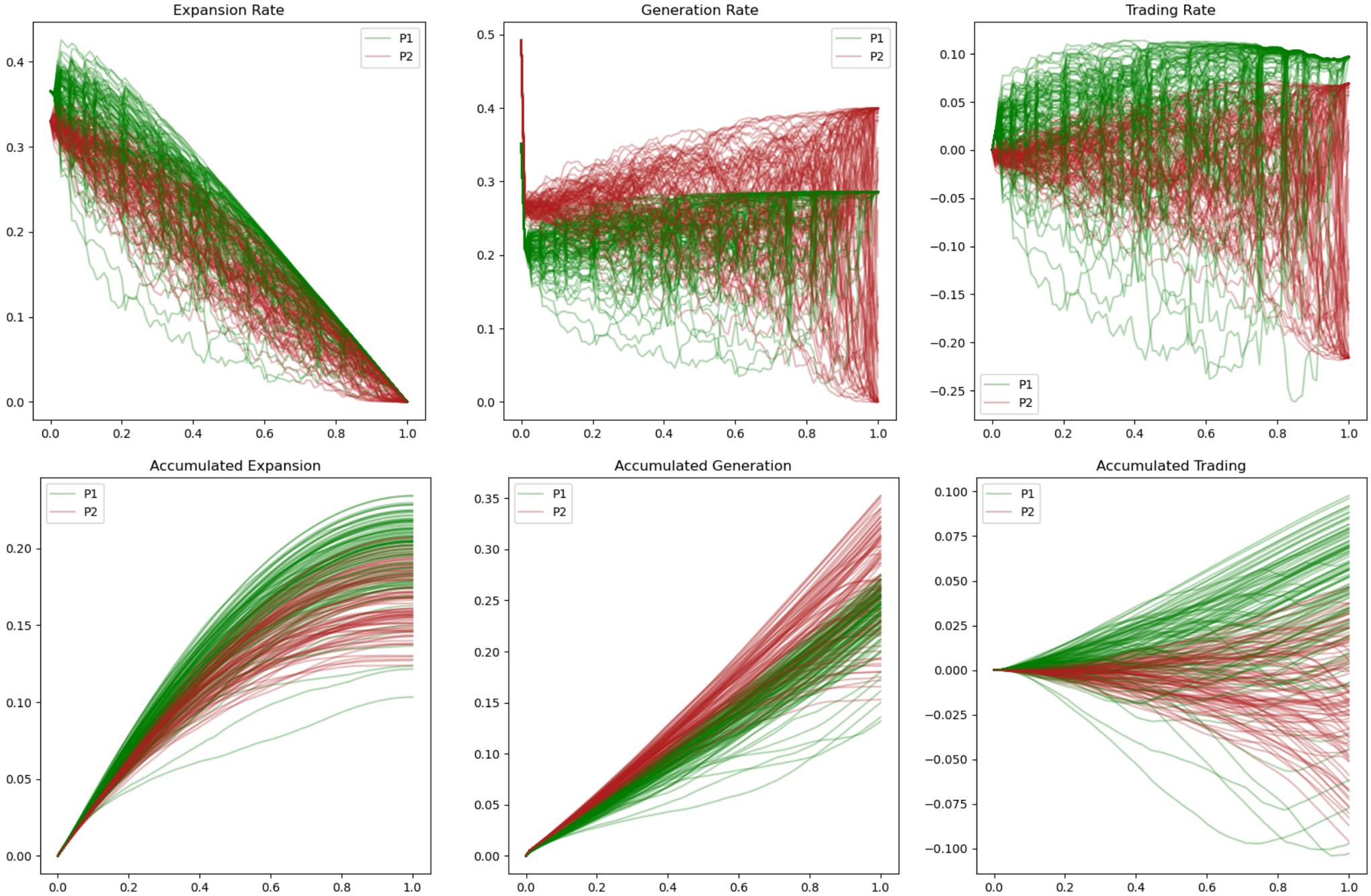
In [ ]:
```
plot=plot_results(pop1_dict=pop1_dict, pop2_dict=pop2_dict, loss=forward_losses)
plot.FwdLoss(log=True)
plot.Inventory_And_Price()
plot.Decomposition_Inventory()
plot.Key_Processes()
plot.Terminal_Convergence()
```

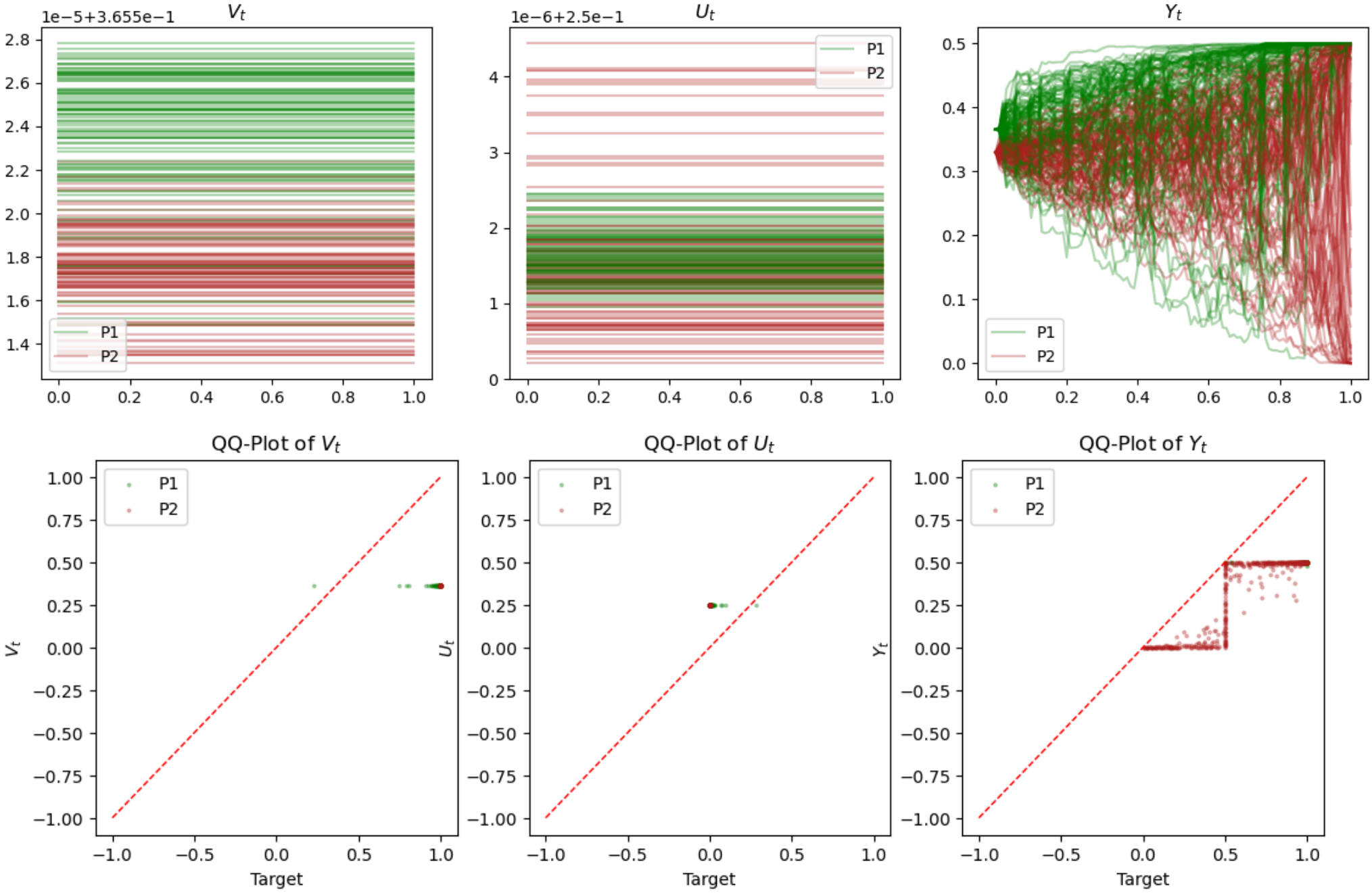
```
/Users/orangeao/Orange Ao/Research And Projects/2024.5 Columbia_Steven/2Period/utils.py:907: UserWarning: Dataset has 0 variance; skipping densi
ty estimate. Pass `warn_singular=False` to disable this warning.
  sns.kdeplot(self.pop1_path_dict['v'][idx0_v1,self.NT1], color="green",label=('$X_{T_1}^{(1)}$>'+f'{self.K}, '+'$V_{T_1}^{(1)}\\rightarrow$
0'))
```

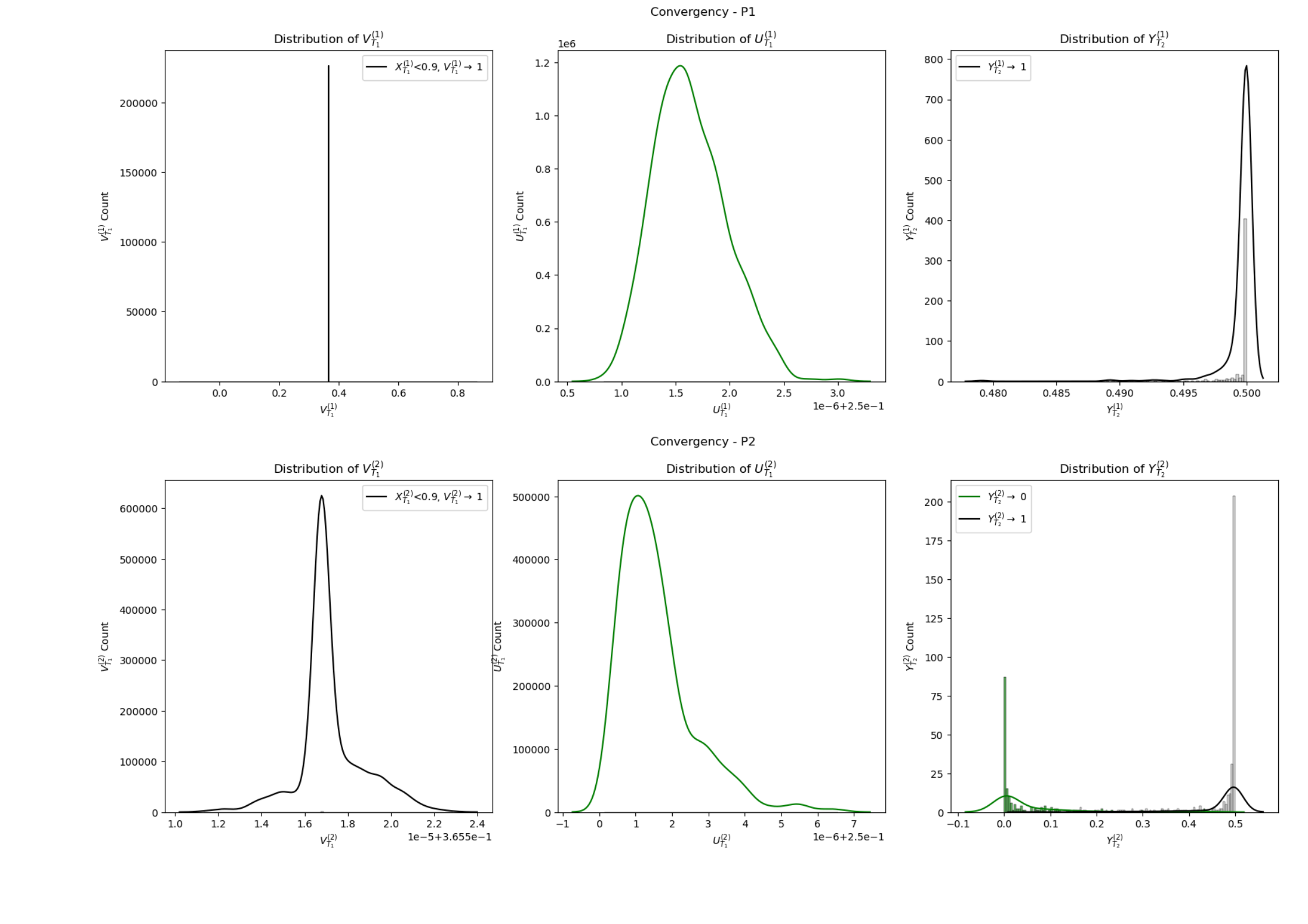

Forward_Loss vs Batch
2.8×10^0
2.6×10^0
2.4×10^0
2.2×10^0
2×10^0
1.8×10^0
1.6×10^0
1.4×10^0
0
100
200
300
400
500


Inventory(X_t)
P1
P2
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0


Price(S_t)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Expansion Rate
Generation Rate
Trading Rate
Accumulated Expansion
Accumulated Generation
Accumulated Trading
P1
P2


V_t
1e−5+3.655e−1
P1
P2
U_t
1e−6+2.5e−1
P1
P2
Y_t
P1
P2
QQ-Plot of V_t
P1
P2
V_t
U_t
Target
QQ-Plot of U_t
P1
P2
Y_t
Target
QQ-Plot of Y_t
P1
P2
Target


Convergency - P1
Distribution of $V_{T_1}^{(1)}$
$X_{T_1}^{(1)}<0.9,\ V_{T_1}^{(1)} \to 1$
$V_{T_1}^{(1)}$ Count
$V_{T_1}^{(1)}$
Distribution of $U_{T_1}^{(1)}$
1e6
$U_{T_1}^{(1)}$ Count
$U_{T_1}^{(1)}$
1e−6+2.5e−1
Distribution of $Y_{T_2}^{(1)}$
$Y_{T_2}^{(1)} \to 1$
$Y_{T_2}^{(1)}$ Count
$Y_{T_2}^{(1)}$
Convergency - P2
Distribution of $V_{T_1}^{(2)}$
$X_{T_1}^{(2)}<0.9,\ V_{T_1}^{(2)} \to 1$
$V_{T_1}^{(2)}$ Count
$V_{T_1}^{(2)}$
1e−5+3.655e−1
Distribution of $U_{T_1}^{(2)}$
$U_{T_1}^{(2)}$ Count
$U_{T_1}^{(2)}$
1e−6+2.5e−1
Distribution of $Y_{T_2}^{(2)}$
$Y_{T_2}^{(2)} \to 0$
$Y_{T_2}^{(2)} \to 1$
$Y_{T_2}^{(2)}$ Count
$Y_{T_2}^{(2)}$

Save The Models

In [ ]:
```
print(f"{len(main_models1.loss)} steps\nStarted @ {start_time}\nSaved @ {end_time}")   ## to examine whether the loss attribute is updated in t
dir_path=pathlib.Path(os.getcwd(),
                      'Results',
                      'Best Models Saved',
                      '1Prd_sigmoid_logit_0.0005lr_500steps_BCELogits_0.5w')
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)
```

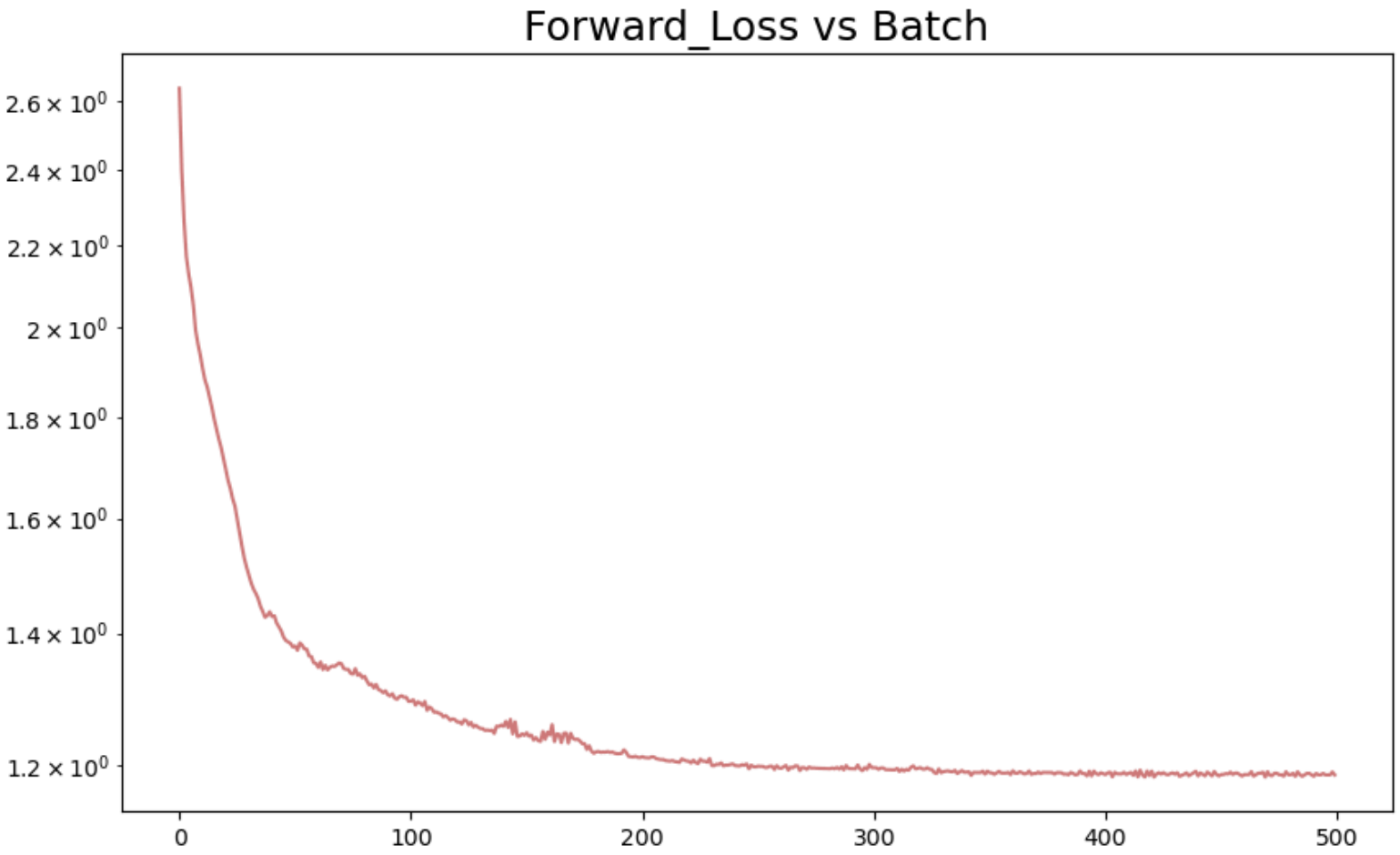
```
500 steps
Started @ August 20 - 22:28:08
Saved @ August 21 - 02:35:55
```

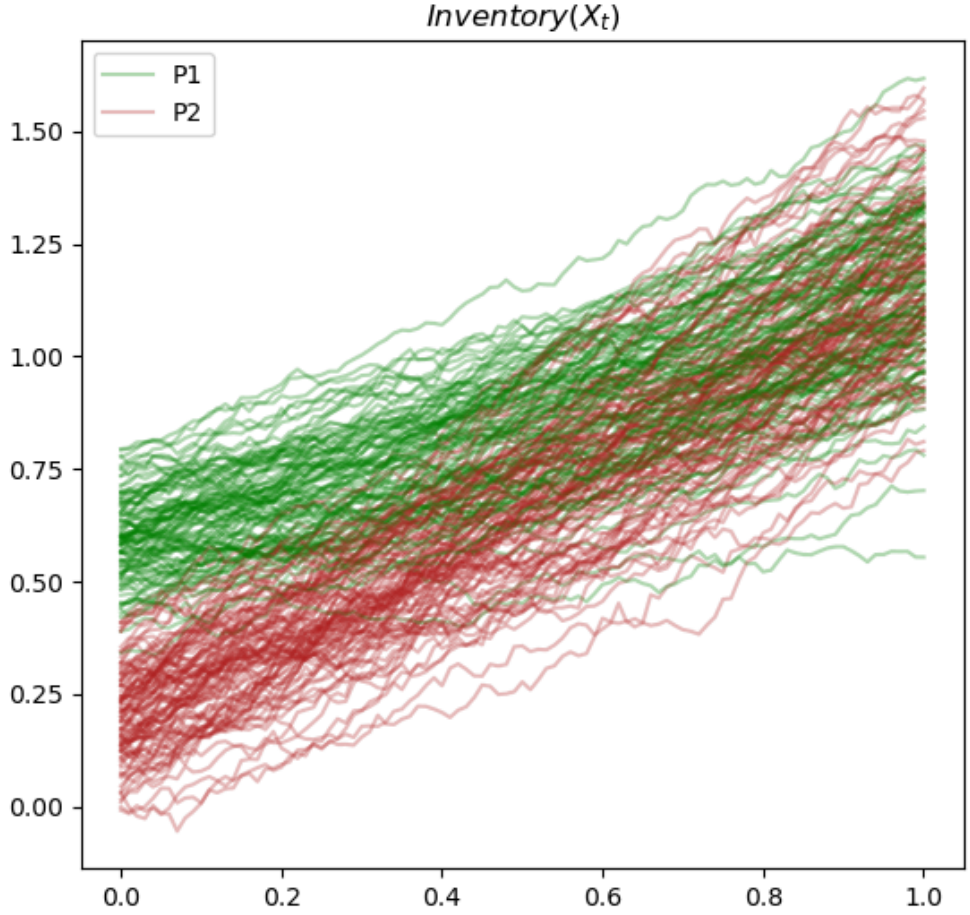
Load The Models

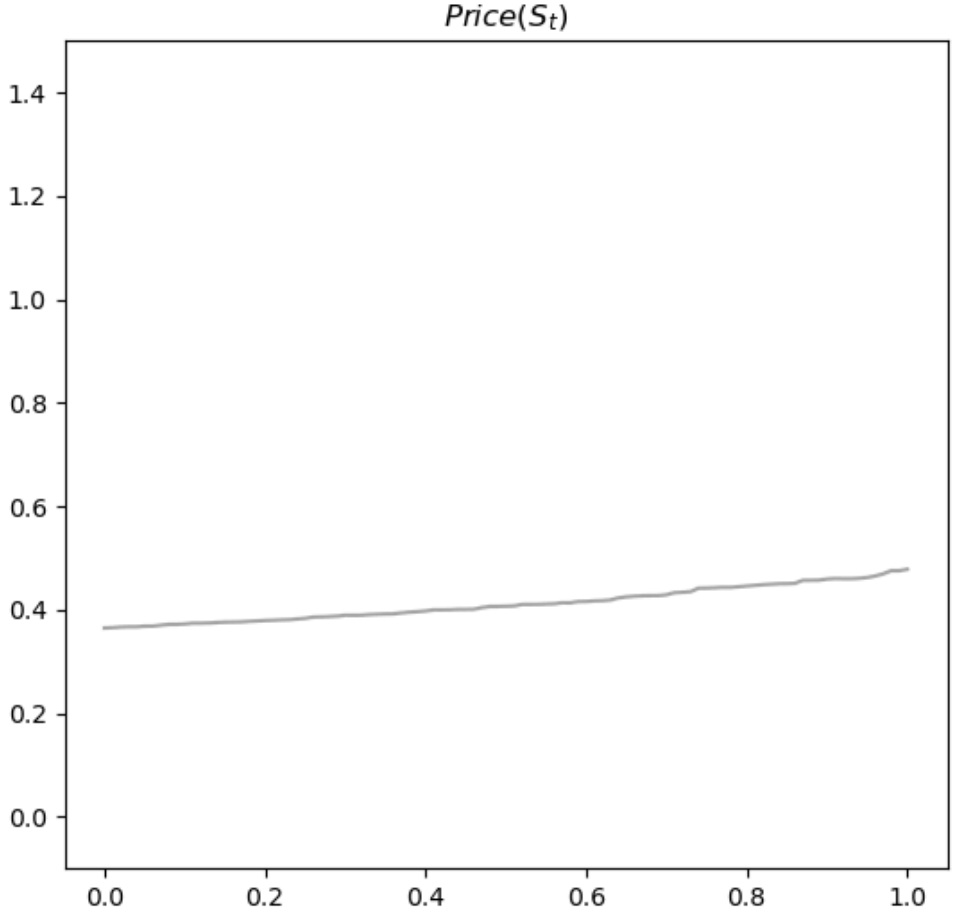
In [ ]:
```
# GlobalParams1=Params(param_type='k1',target_type='indicator',trick='clamp',loss_type='MSELoss',delta=0.03,K=0.9,lr=0.01)
# GlobalParams2=Params(param_type='k2',target_type='indicator',trick='clamp',loss_type='MSELoss',delta=0.03,K=0.9,lr=0.01)
models1=Main_Models(GlobalParams=GlobalParams1)
models2=Main_Models(GlobalParams=GlobalParams2)
path_dir=pathlib.Path(os.getcwd(),
                      "Results",
                      "Best Models Saved",
                      'minmax_logit_0.001lr_500steps_BCELogits_0.5w')
path1=pathlib.Path(path_dir,'pop1.pt')
path2=pathlib.Path(path_dir,'pop2.pt')
model_dict1=models1.load_entire_models(path=path1,overwrite=True)
model_dict2=models2.load_entire_models(path=path2,overwrite=True)

dB1=model_dict1['dB']
init_x1=model_dict1['init_x']
init_c1=model_dict1['init_c']
pop1_dict= {'dB':dB1,
            'init_x':init_x1,
            'init_c':init_c1,
            'GlobalParams':GlobalParams1,
            'main_models':models1}

dB2=model_dict2['dB']
init_x2=model_dict2['init_x']
init_c2=model_dict2['init_c']

pop2_dict= {'dB':dB2,
            'init_x':init_x2 ,
            'init_c':init_c2 ,
```

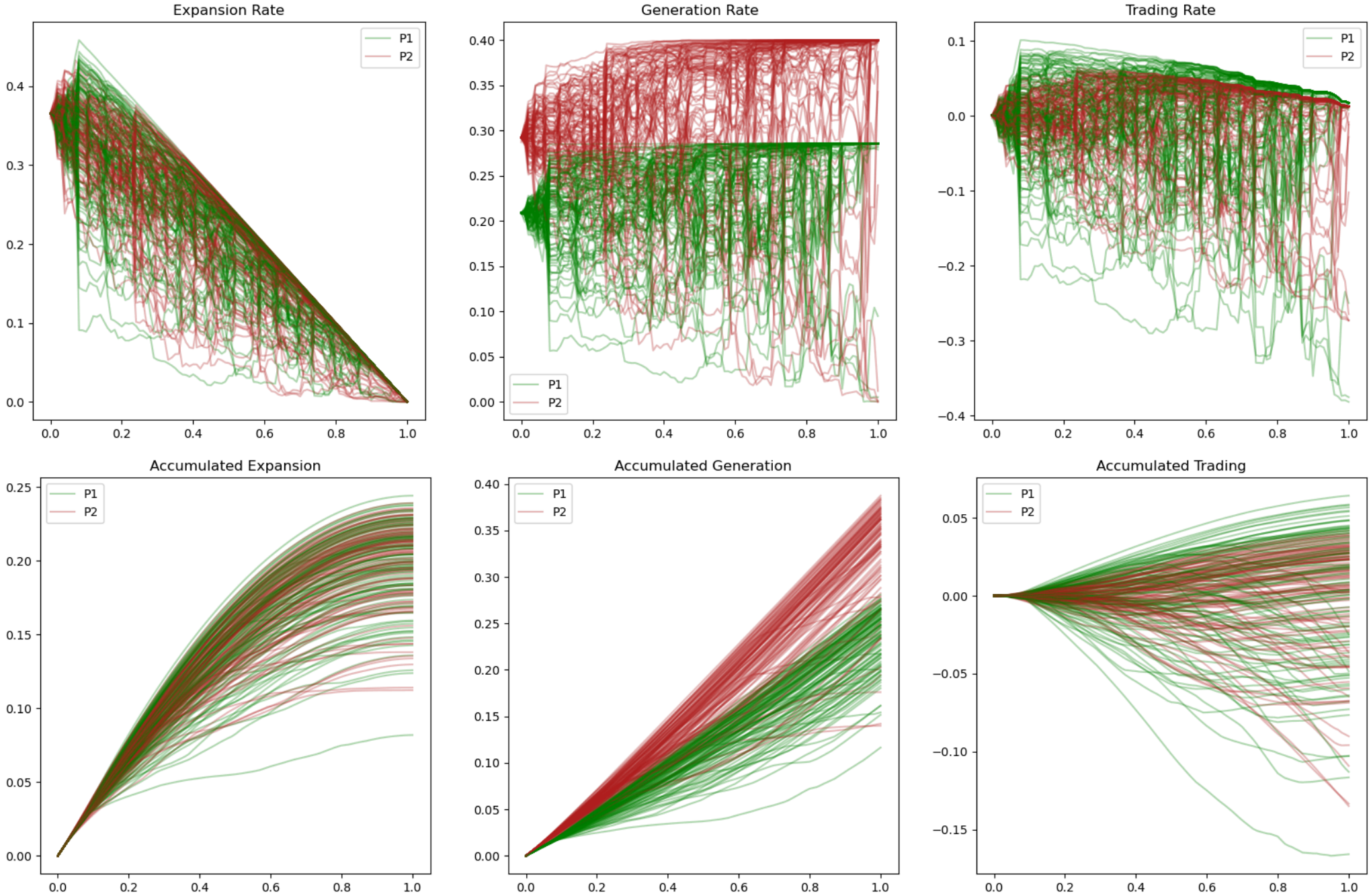
```
            'GlobalParams':GlobalParams2,
            'main_models':models2}
dt=GlobalParams1.dt
NT1=GlobalParams1.NT1
NT2=GlobalParams1.NT2
NumTrain=GlobalParams1.NumTrain
K=GlobalParams1.K
loss=models1.loss

plot=plot_results(pop1_dict=pop1_dict,pop2_dict=pop2_dict,loss=loss)
plot.FwdLoss(log=True)
plot.Inventory_And_Price()
plot.Decomposition_Inventory()
plot.Key_Processes()
# plot.Terminal_Convergence()
```

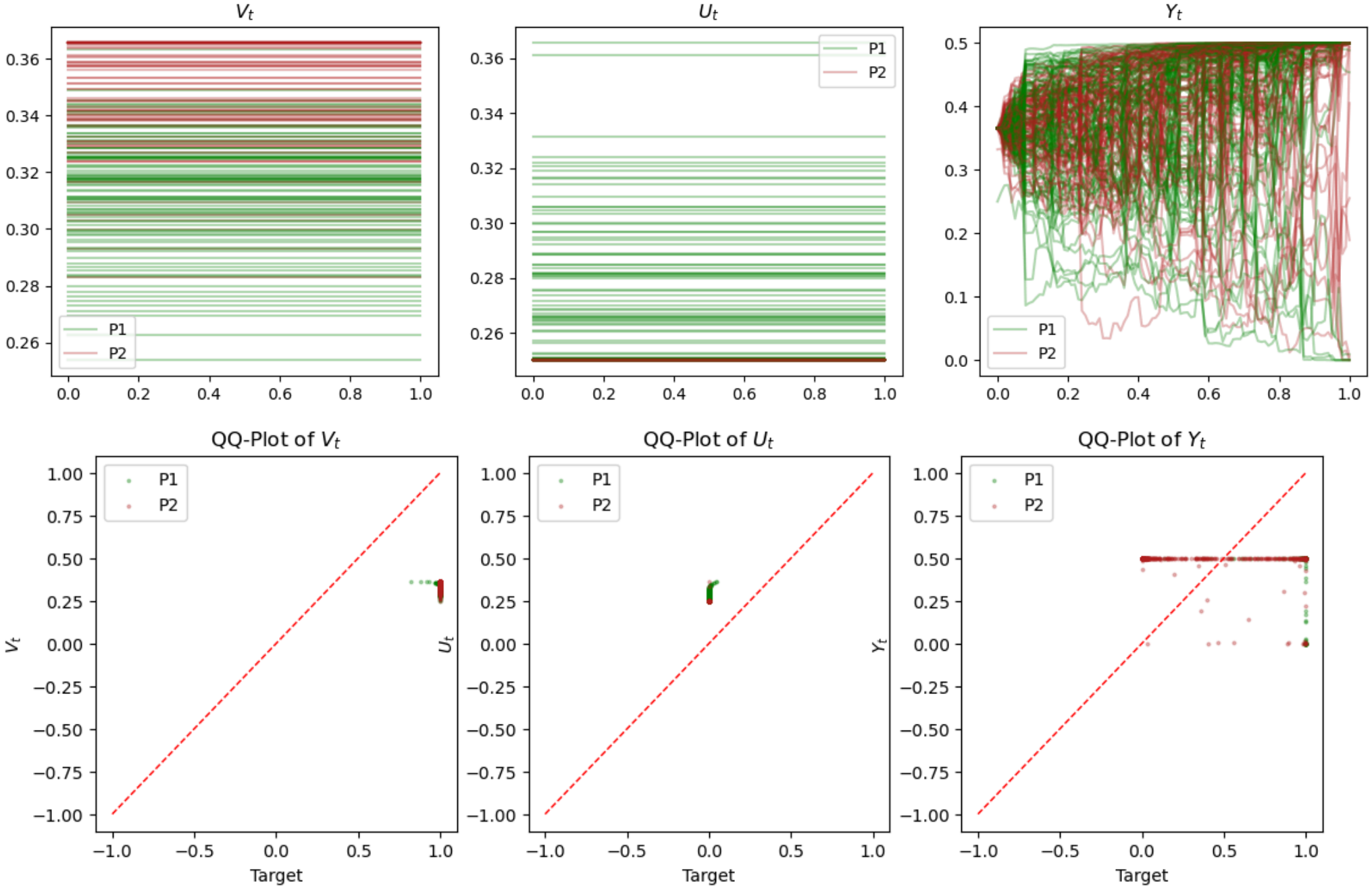
```
/Users/orangeao/Orange Ao/Research And Projects/2024.5 Columbia_Steven/2Period/utils.py:924: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=False` to disable this warning.
  sns.kdeplot(self.pop1_path_dict['y'][idx0_y1,self.NT2], color="green",label=('$Y_{T_2}^{(1)}\\rightarrow$ 0'))
```

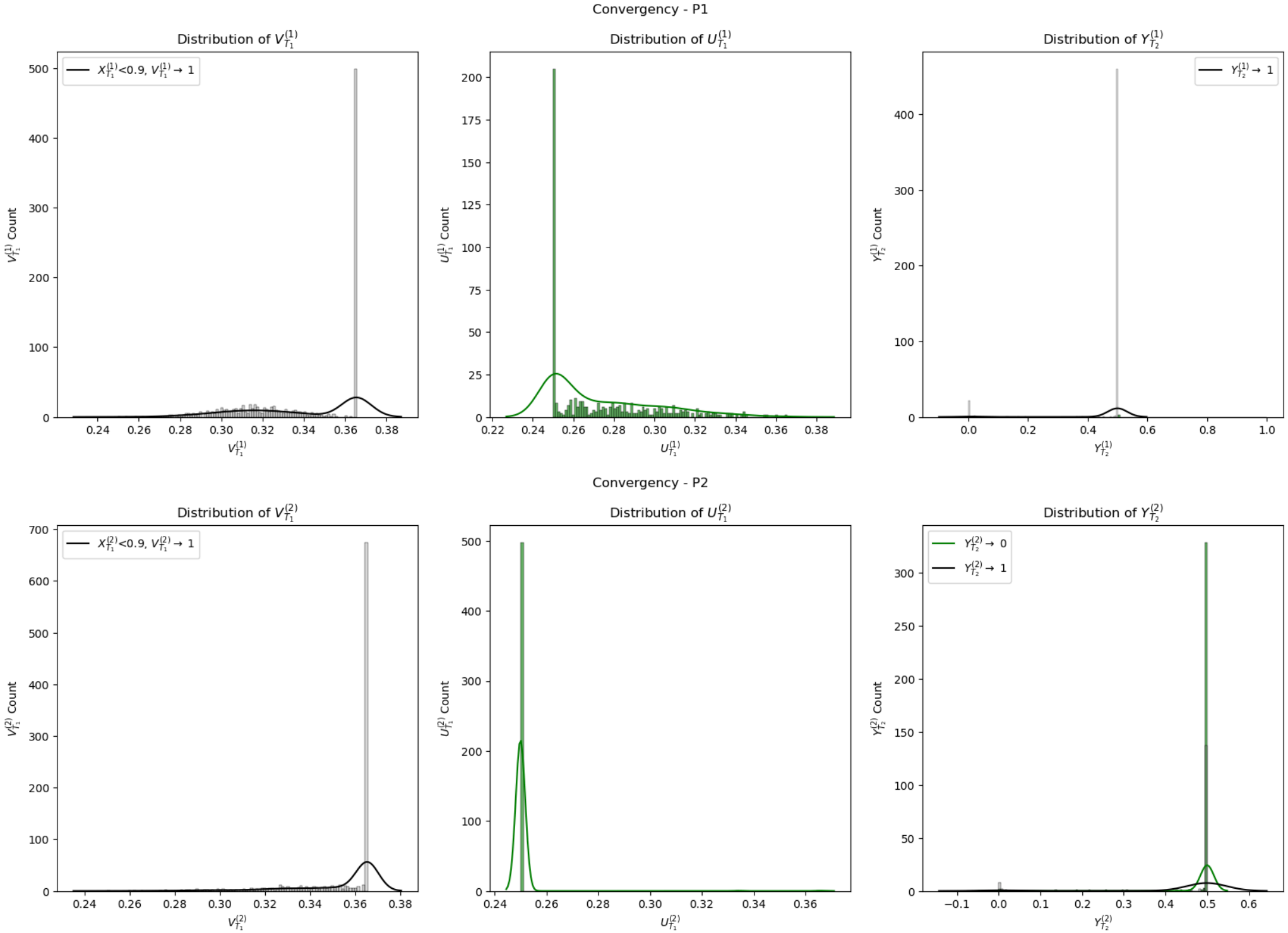



$Inventory(X_t)$
P1
P2
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0


$Price(S_t)$
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Expansion Rate
Generation Rate
Trading Rate
Accumulated Expansion
Accumulated Generation
Accumulated Trading
P1
P2


V_t
U_t
Y_t
P1
P2
QQ-Plot of V_t
QQ-Plot of U_t
QQ-Plot of Y_t
V_t
U_t
Y_t
Target
Target
Target


Convergency - P1
Distribution of $V_{T_1}^{(1)}$
$X_{T_1}^{(1)}<0.9$, $V_{T_1}^{(1)} \to 1$
$V_{T_1}^{(1)}$ Count
$V_{T_1}^{(1)}$
Distribution of $U_{T_1}^{(1)}$
$U_{T_1}^{(1)}$ Count
$U_{T_1}^{(1)}$
Distribution of $Y_{T_2}^{(1)}$
$Y_{T_2}^{(1)} \to 1$
$Y_{T_2}^{(1)}$ Count
$Y_{T_2}^{(1)}$
Convergency - P2
Distribution of $V_{T_1}^{(2)}$
$X_{T_1}^{(2)}<0.9$, $V_{T_1}^{(2)} \to 1$
$V_{T_1}^{(2)}$ Count
$V_{T_1}^{(2)}$
Distribution of $U_{T_1}^{(2)}$
$U_{T_1}^{(2)}$ Count
$U_{T_1}^{(2)}$
Distribution of $Y_{T_2}^{(2)}$
$Y_{T_2}^{(2)} \to 0$
$Y_{T_2}^{(2)} \to 1$
$Y_{T_2}^{(2)}$ Count
$Y_{T_2}^{(2)}$

In [ ]:

```
path1,path2=get_target_path(pop1_dict=pop1_dict,pop2_dict=pop2_dict)
path1['generation'][:,0]
path2['generation'][:,0]
```

```
Out[ ]: tensor([0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2905, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2658, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2871, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2194, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2876, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2891, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2907, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2839, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2623, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2914, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
        0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
```

```
       0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
       0.2924, 0.2924, 0.2924, 0.2924, 0.2000, 0.2924, 0.2924, 0.2924, 0.2924,
       0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924, 0.2924,
       0.2924, 0.2924, 0.2924, 0.2924, 0.2878, 0.2924, 0.2924, 0.2924, 0.2924,
       0.2924, 0.2924, 0.2924, 0.2924, 0.2924])
```

In [ ]: