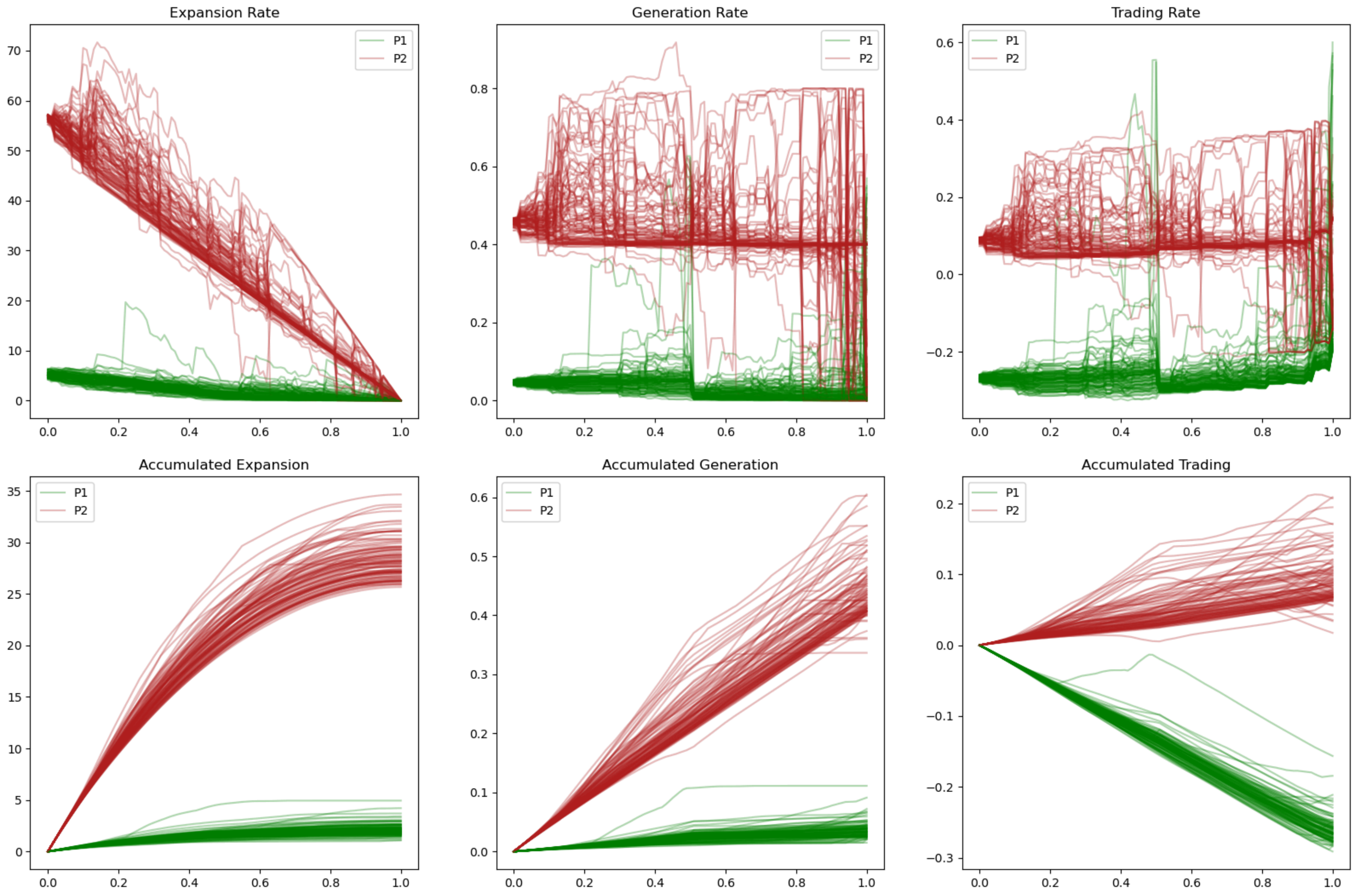


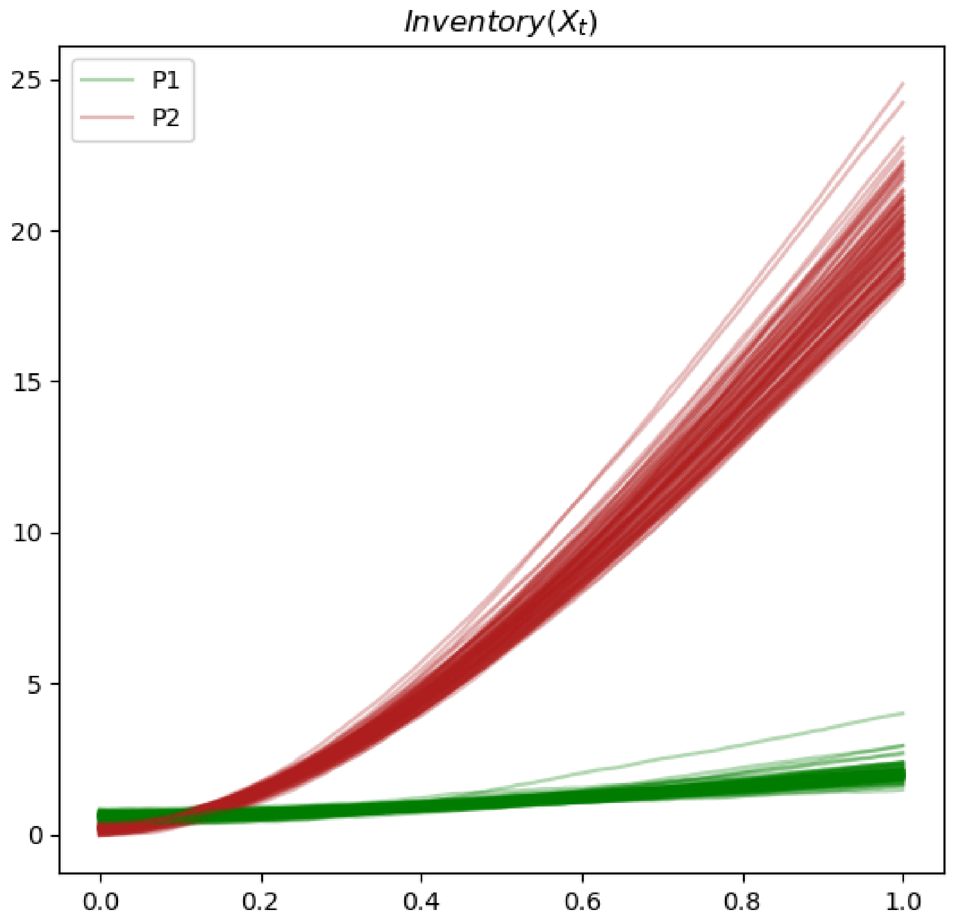
minmax: *none*

target: ind

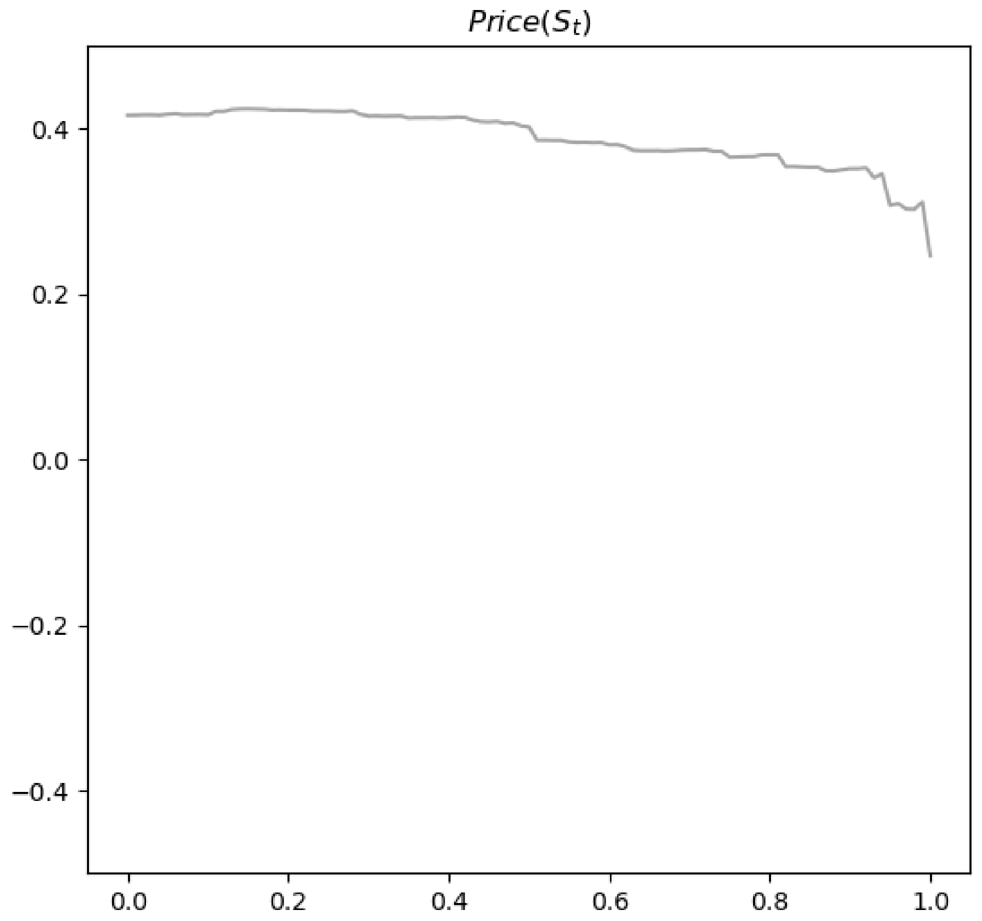
loss: BCELogits: 1*loss

trick: v_tilde,u_tilde,y_tilde = logit(v,u,y)

```
300 steps
Saved @ July 31 - 07:29:19
```



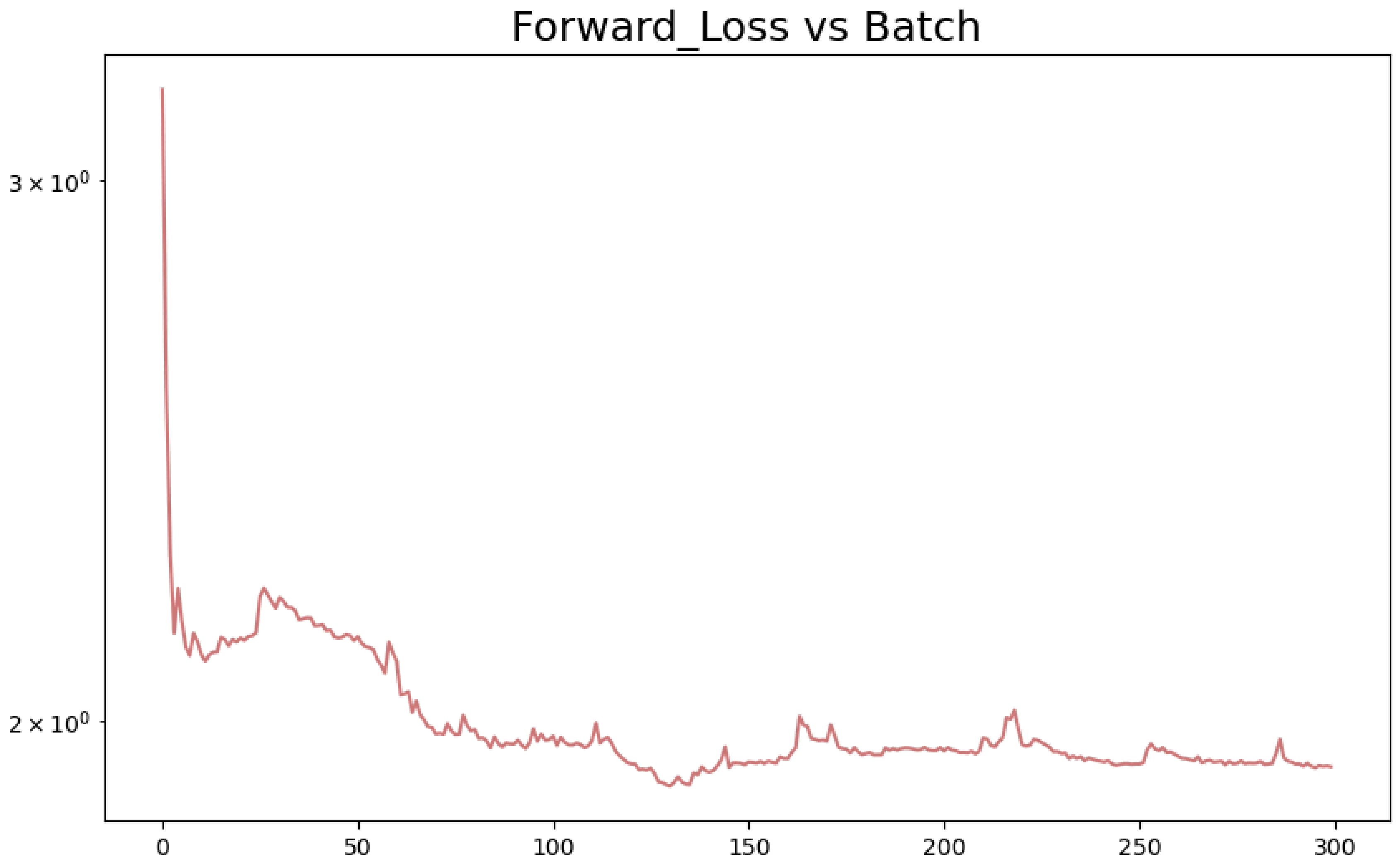
$Inventory(X_t)$
P1
P2
25
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0

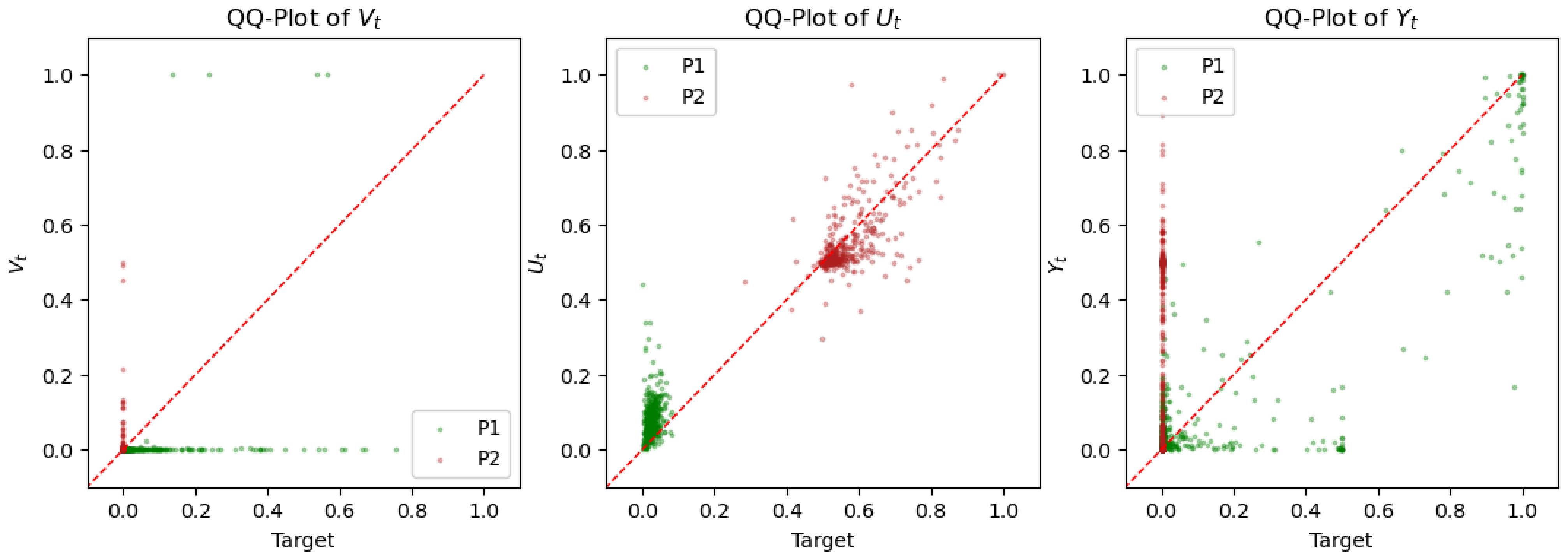


$Price(S_t)$
0.4
0.2
0.0
−0.2
−0.4
0.0
0.2
0.4
0.6
0.8
1.0

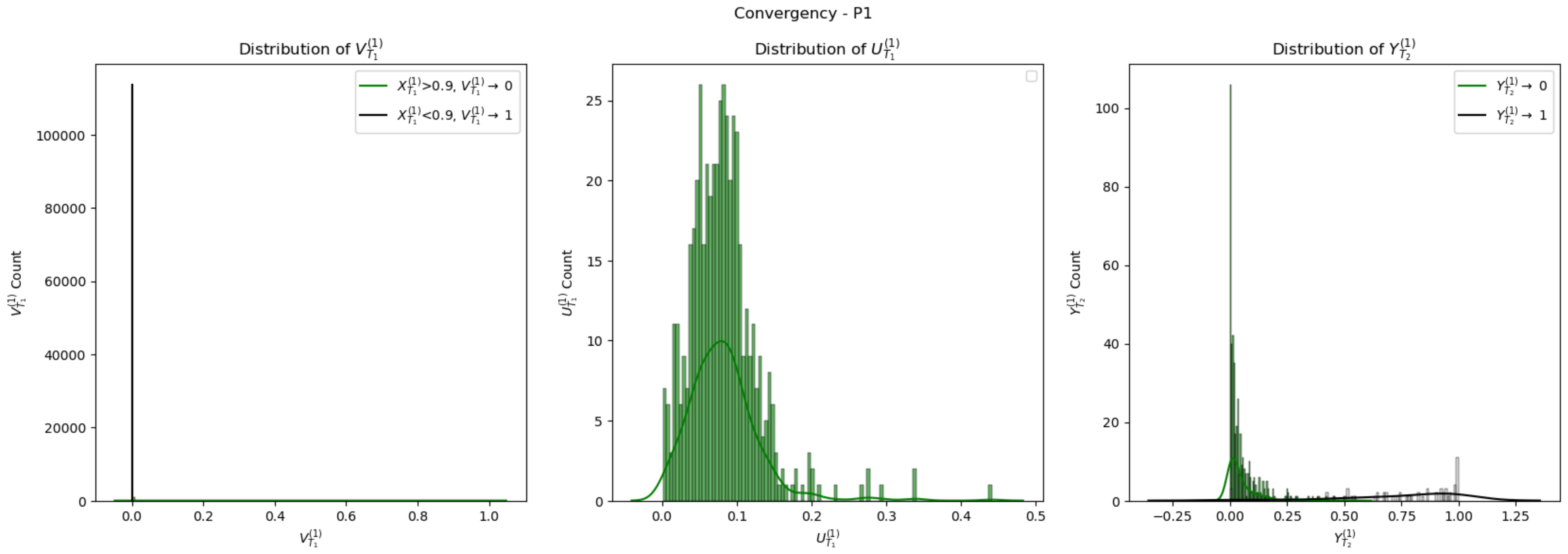
```
In [ ]: plot=plot_results(pop1_dict=pop1_dict, pop2_dict=pop2_dict, loss=forward_losses)
        plot.FwdLoss(log=True)
        plot.Inventory_And_Price()
        plot.Decomposition_Inventory()
        plot.Terminal_Convergence()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend()
is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend()
is called with no argument.
```

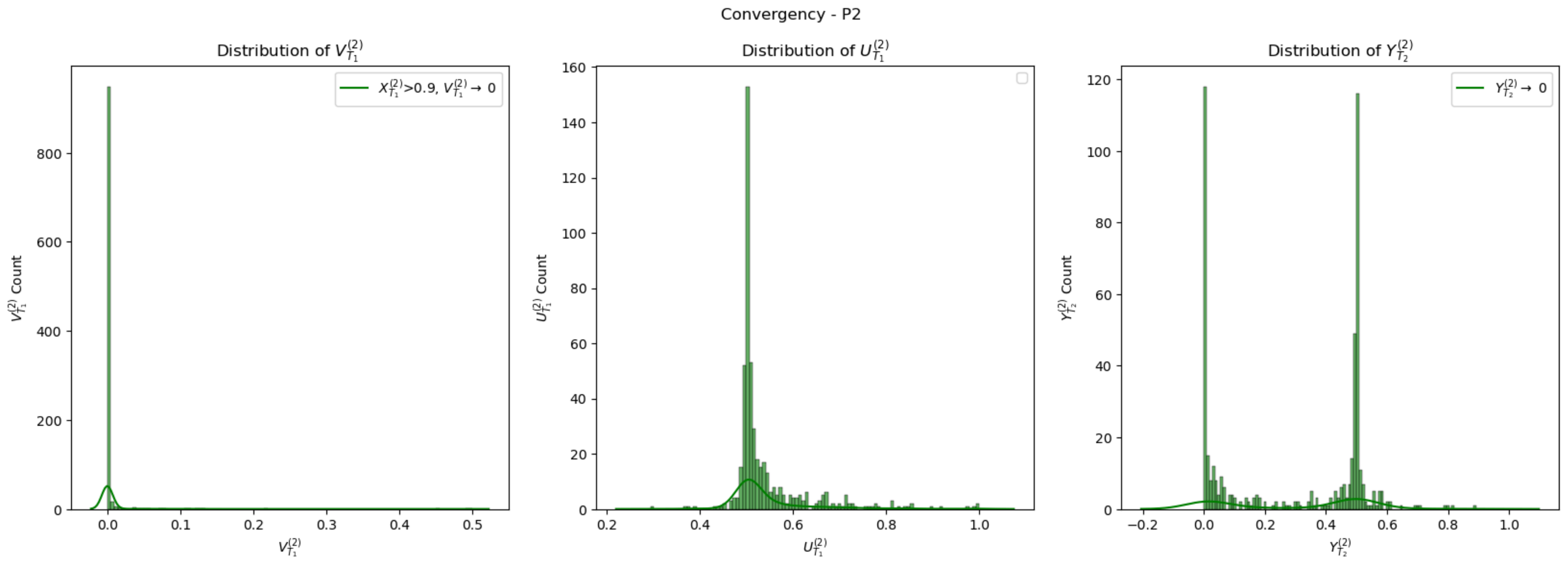




QQ-Plot of V_t
QQ-Plot of U_t
QQ-Plot of Y_t
V_t
U_t
Y_t
Target
P1
P2



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



Save The Models

```
In [ ]: print(f"{len(main_models1.loss)} steps\nSaved @ {datetime.now().strftime('%B %d - %H:%M:%S')}") ## to examine whether the loss
        dir_path=pathlib.Path(os.getcwd(),
                              'Results',
                              'Best Models Saved',
                              'Adamax_logit_0.01lr_300steps_BCELogits_1')
        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)
```

```
300 steps
Saved @ July 31 - 07:29:19
```

Load The Models

```
In [ ]: # dir_path=pathlib.Path(os.getcwd(),'Results','Best Models Saved','Adamax_logit_0.05delta_0.01lr_500steps_BCE')
        # path1=pathlib.Path(dir_path,'pop1.pt')
        # path2=pathlib.Path(dir_path,'pop2.pt')
```

```
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)

Out[ ]: <torch.autograd.anomaly_mode.set_detect_anomaly at 0x169c7eb50>
```

```
In [ ]: #Global parameters
        GlobalParams1=Params(param_type='k1',target_type='indicator',trick='logit',loss_type='BCEWithLogitsLoss',delta=0.03,K=0.9,lr=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,K=0.9,lr=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= 300
```

```python
#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()
u0_model_main1 = Network()
y0_model_main1 = Network()

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()
u0_model_main2 = Network()
y0_model_main2 = Network()

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 ,
                'GlobalParams':GlobalParams1,
                'main_models':main_models1}

        dB2 = SampleBMIncr(GlobalParams=GlobalParams2)   ## TODO: same dB?????
        init_x2 =  Sample_Init(GlobalParams=GlobalParams2)
        init_c2= torch.zeros_like(init_x2)
        pop2_dict={'dB':dB2,
                'init_x':init_x2 ,
                'init_c':init_c2 ,
                'GlobalParams':GlobalParams2,
                'main_models':main_models2}
```

```
# model_dict1=main_models1.load_entire_models(path=path1,overwrite=True)
# model_dict2=main_models2.load_entire_models(path=path2,overwrite=True)

# pop1_dict={'dB':main_models1.dB,
#             'init_x':main_models1.init_x ,
#             'init_c':main_models1.init_c ,
#             'GlobalParams':main_models1.GlobalParams,
#             'main_models':main_models1}

# pop2_dict={'dB':main_models2.dB,
#         'init_x':main_models2.init_x ,
#         'init_c':main_models2.init_c ,
#         'GlobalParams':main_models2.GlobalParams,
#         'main_models':main_models2}
```

In [ ]:

```
# plot=plot_results(pop1_dict=pop1_dict,pop2_dict=pop2_dict,loss=main_models1.loss)
# plot.FwdLoss(log=True)
# plot.Integrate_Inventory()
# plot.Decomposition_Inventory(base_rate=True,market_price=True)
# plot.Terminal_Convergence()
# print(datetime.now().strftime("%B %d - %H:%M:%S"))
```

Loss Scale

In [ ]:

```
pop1_path_dict, pop2_path_dict=get_target_path(pop1_dict=pop1_dict, pop2_dict=pop2_dict)
yx1=pop1_path_dict['yx'][:,-1]
yc1=pop1_path_dict['yc'][:,-1]
x1=pop1_path_dict['x'][:,-1]
yx2=pop2_path_dict['yx'][:,-1]
yc2=pop2_path_dict['yc'][:,-1]
x2=pop2_path_dict['x'][:,-1]

# ## MSE
# loss_yx1=torch.mean((yx1-target(x1,GlobalParams=pop1_dict['GlobalParams']))**2)
# loss_yc1=torch.mean((yc1)**2)
# loss_yx2=torch.mean((yx2-target(x2,GlobalParams=pop2_dict['GlobalParams']))**2)
# loss_yc2=torch.mean((yc2)**2)
# print(f'''loss_yx1: {loss_yx1}\tloss_yx2: {loss_yx2}\nloss_yc1: {loss_yc1}\tloss_yc2: {loss_yc2}
# lossTime1: {loss_yx1/loss_yc1}\tlossTime2: {loss_yx2/loss_yc2}\t''')

## BCE
yx1_tilde=torch.logit(-yx1)
yx2_tilde=torch.logit(-yx2)
loss_yx1=nn.BCELoss()(-yx1,-target(x1,GlobalParams=pop1_dict['GlobalParams']))
```

```
loss_yc1=torch.mean((yc1)**2)
loss_yx2=nn.BCELoss()(-yx2,-target(x2,GlobalParams=pop2_dict['GlobalParams']))
loss_yc2=torch.mean((yc2)**2)
print(f'''loss_yx1: {loss_yx1}\tloss_yx2: {loss_yx2}\nloss_yc1: {loss_yc1}\tloss_yc2: {loss_yc2}
lossTime1: {loss_yx1/loss_yc1}\tlossTime2: {loss_yx2/loss_yc2}\t''')
```

```
loss_yx1: 0.26865965127944946   loss_yx2: 0.0905696228146553
loss_yc1: 0.007297591306269169  loss_yc2: 0.0010008163517341018
lossTime1: 36.81483840942383    lossTime2: 90.49574279785156
```