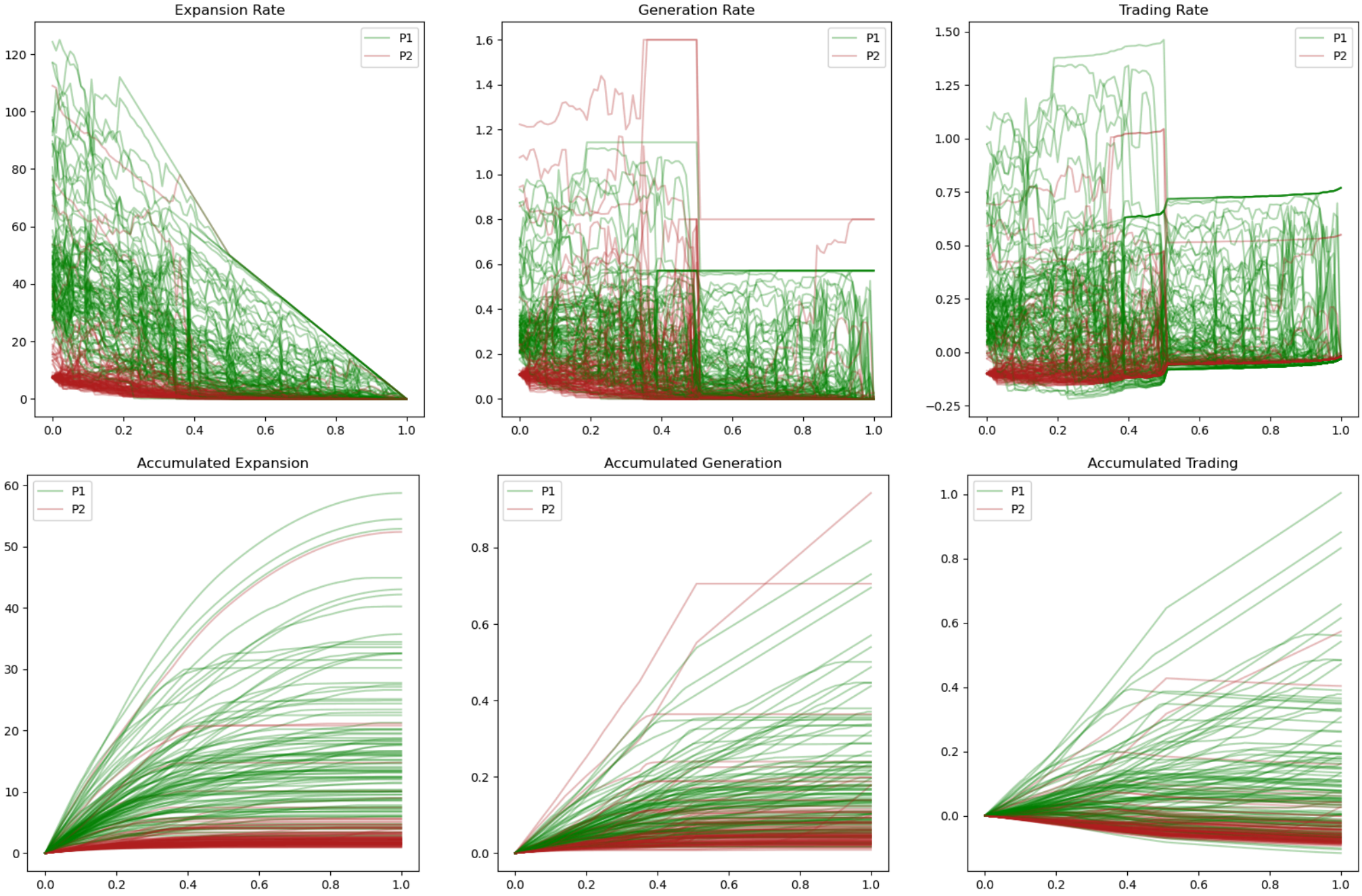
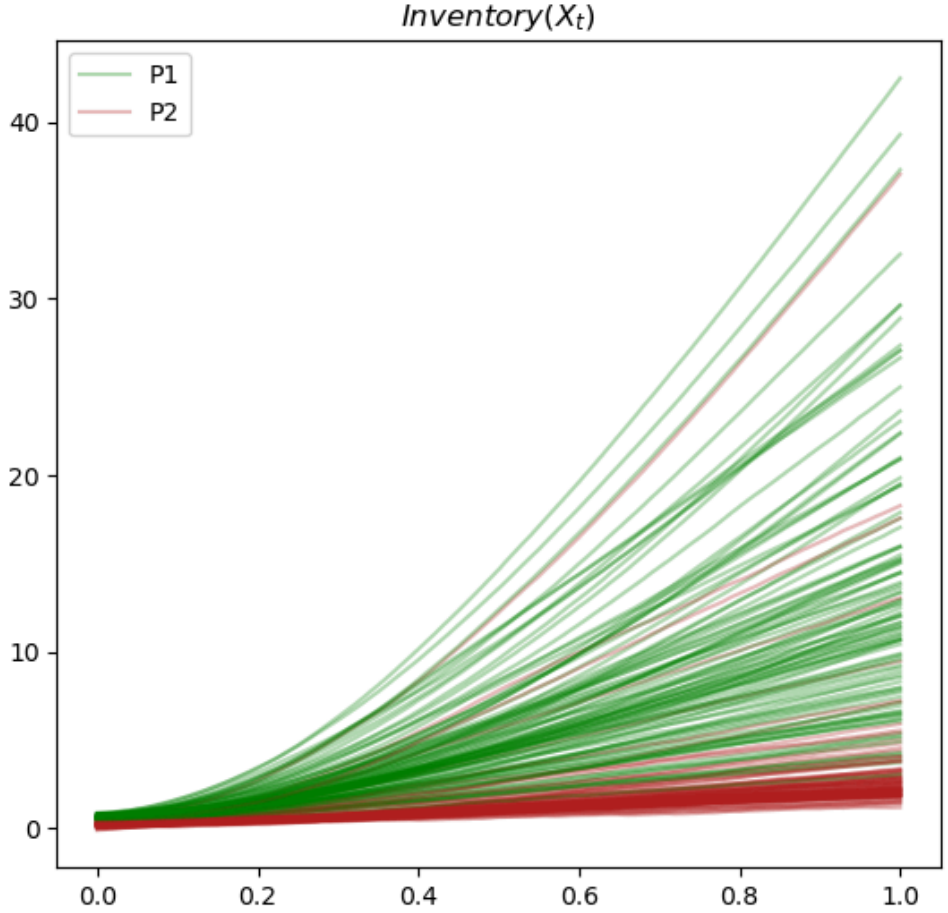
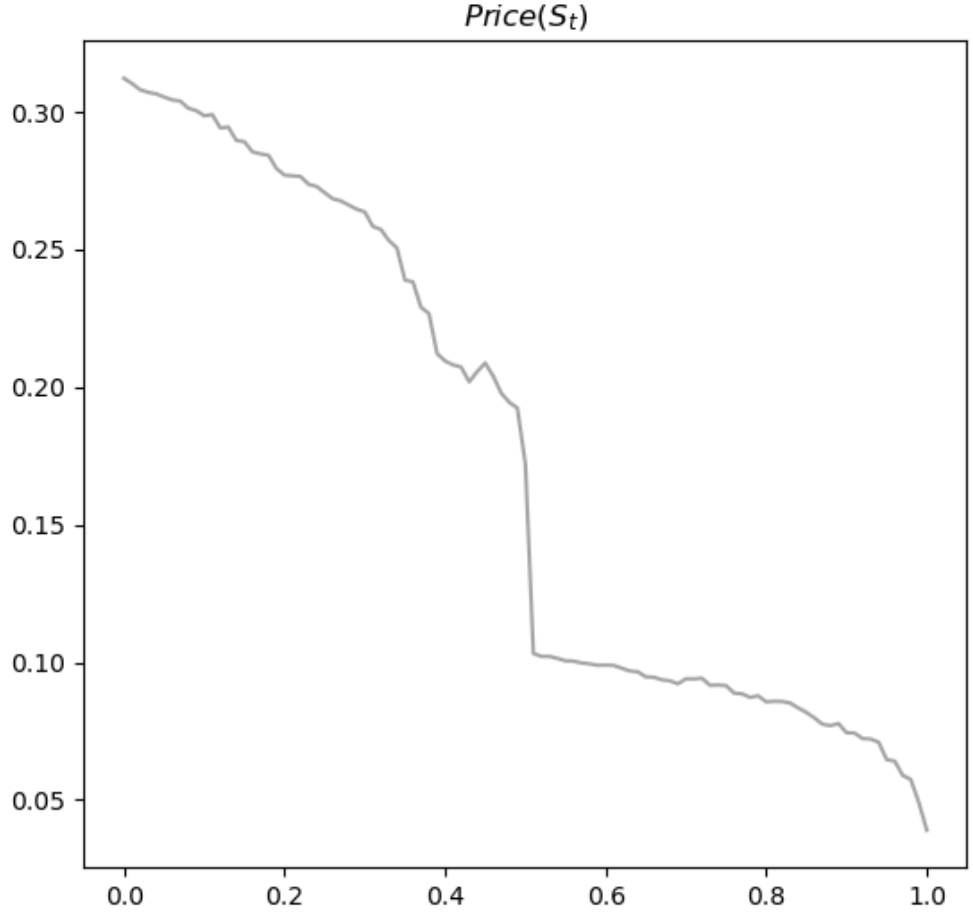
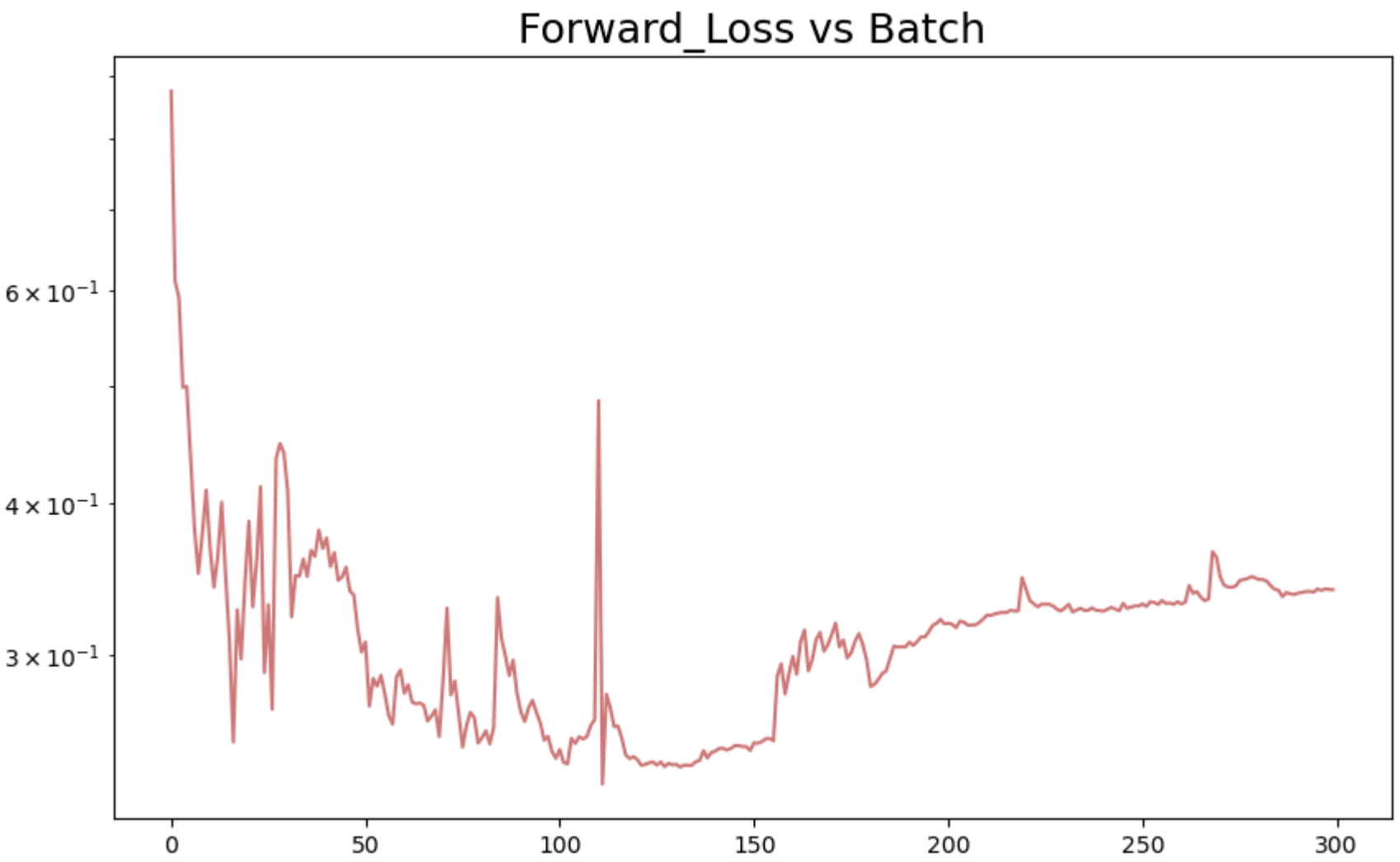


minmax: initial only (v0, u0, y0)
target: ind
loss: BCE: 1*loss
trick: *y*(1-y) + torch.clamp(0,1)

300 steps
Saved @ August 01 – 13:20:55

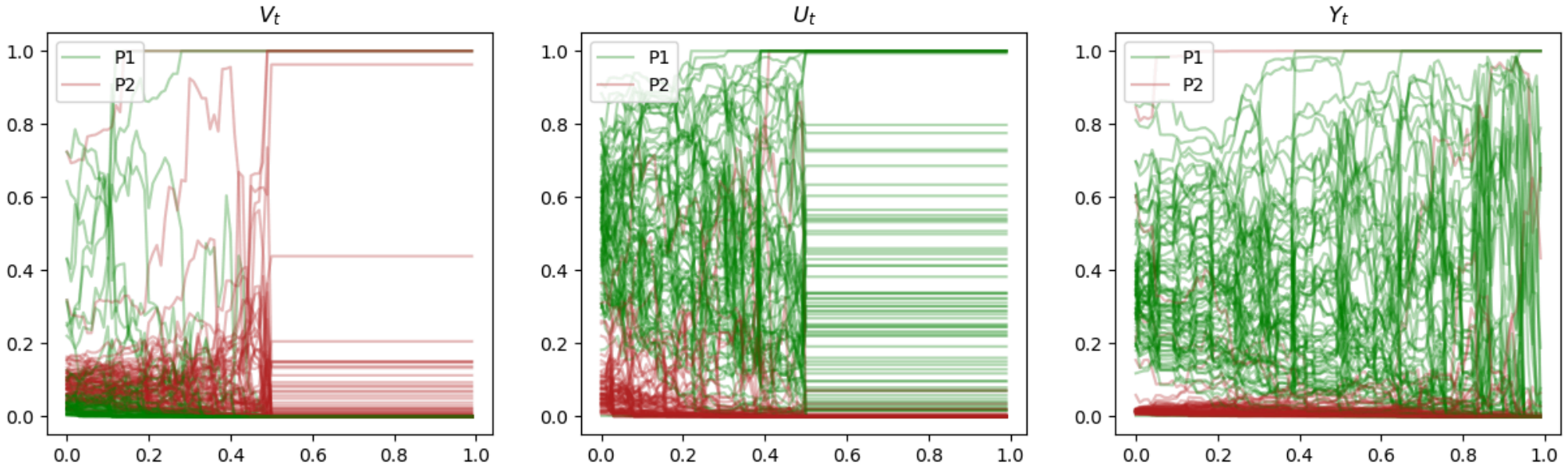

Inventory(X_t)
P1
P2
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0

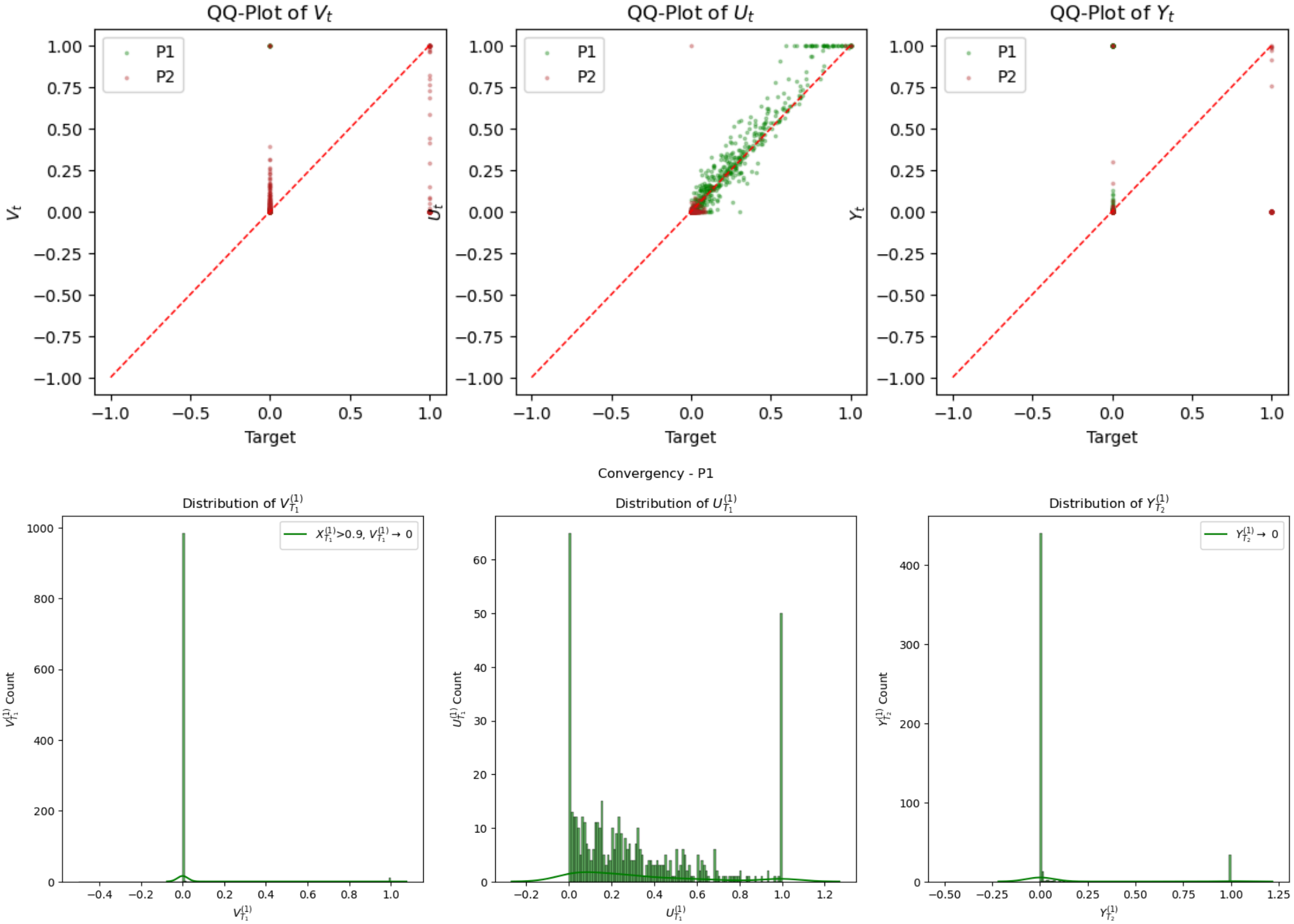

Price(S_t)
0.30
0.25
0.20
0.15
0.10
0.05
0.0
0.2
0.4
0.6
0.8
1.0

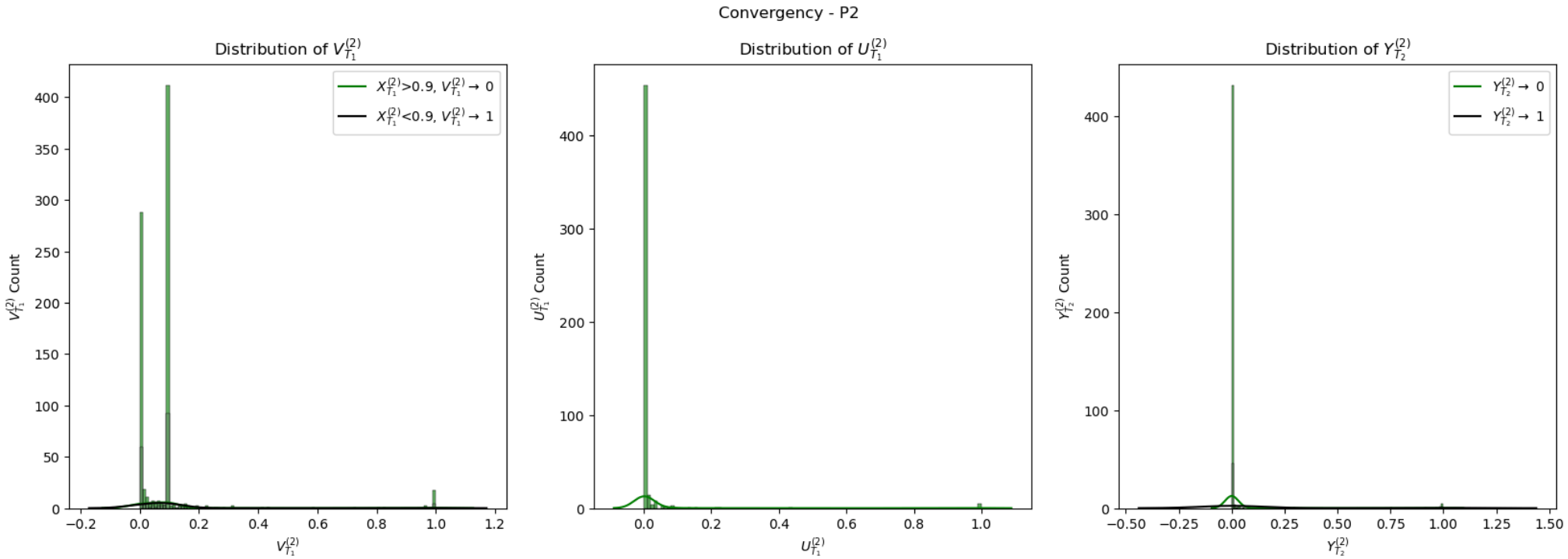

Forward_Loss vs Batch
6×10^{-1}
4×10^{-1}
3×10^{-1}
0
50
100
150
200
250
300

```
    ax6,=plt.plot( t, pop2_path_dict['y'][i,:NT2], color="firebrick", alpha=0.3)
plt.legend({'P1':ax5,'P2':ax6})
```

Out[ ]: <matplotlib.legend.Legend at 0x13a9a9690>




QQ-Plot of V_t
QQ-Plot of U_t
QQ-Plot of Y_t
P1
P2
Target
V_t
U_t
Y_t
Convergency - P1
Distribution of $V_{T_1}^{(1)}$
Distribution of $U_{T_1}^{(1)}$
Distribution of $Y_{T_2}^{(1)}$
$X_{T_1}^{(1)}>0.9$, $V_{T_1}^{(1)} \to 0$
$Y_{T_2}^{(1)} \to 0$
$V_{T_1}^{(1)}$ Count
$U_{T_1}^{(1)}$ Count
$Y_{T_2}^{(1)}$ Count



Save The Models

In [ ]:
```
print(f"{len(main_models1.loss)} steps\nSaved @ {datetime.now().strftime('%B %d - %H:%M:%S')}")   ## to examine whether the los
dir_path=pathlib.Path(os.getcwd(),
                      'Results',
                      'Best Models Saved',
                      'minmax_ind_0.01lr_300steps_MSE_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 @ August 01 - 13:20:55
```

Load The Models

In [ ]:
```
# dir_path=pathlib.Path(os.getcwd(),'Results','Best Models Saved','Adamax_clamp_-sig(0.9-x)_0.05delta_0.01lr_500steps_MSE')
# 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 0x30cc58d90>
```

In [ ]:
```
#Global parameters
GlobalParams1=Params(param_type='k1',target_type='indicator',trick='clamp',loss_type='MSELoss',delta=0.01,K=0.9,lr=0.01)
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,K=0.9,lr=0.01)
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
```

```
#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(minmax=True)
u0_model_main1 = Network(minmax=True)
y0_model_main1 = Network(minmax=True)

zv_models_main1 = [Network(minmax=False) for i in range(NT1)]
zu_models_main1 = [Network(minmax=False) for i in range(NT1)]
zy_models_main1 = [Network(minmax=False) 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(minmax=True)
u0_model_main2 = Network(minmax=True)
y0_model_main2 = Network(minmax=True)

zv_models_main2 = [Network(minmax=False) for i in range(NT1)]
zu_models_main2 = [Network(minmax=False) for i in range(NT1)]
zy_models_main2 = [Network(minmax=False) 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}
```

```
Batch Number:  300
Average Error Est:  0.33987167954444886
```

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()
```

```
/Users/orangeao/Orange Ao/Research And Projects/2024.5 Columbia_Steven/2Period/utils.py:881: UserWarning: Dataset has 0 varianc
e; skipping density estimate. Pass `warn_singular=False` to disable this warning.
  sns.kdeplot(self.pop1_path_dict['v'][idx1_v1,self.NT1], color="black",label=('$X_{T_1}^{(1)}$<'+f'{self.K}, '+'$V_{T_1}^{(1)}
\\rightarrow$ 1'))
/Users/orangeao/Orange Ao/Research And Projects/2024.5 Columbia_Steven/2Period/utils.py:898: UserWarning: Dataset has 0 varianc
e; skipping density estimate. Pass `warn_singular=False` to disable this warning.
  sns.kdeplot(self.pop1_path_dict['y'][idx1_y1,self.NT2], color="black",label=('$Y_{T_2}^{(1)}\\rightarrow$ 1'))
```

```
# 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"))
```

In [ ]:

```
pop1_path_dict,pop2_path_dict=get_target_path(pop1_dict=pop1_dict,pop2_dict=pop2_dict)
plt.figure(figsize=(15,4))
plt.subplot(131)
plt.title("$V_t$")
t=np.array([i for i in range(NT2)]) * dt
for i in range(80):
    ax1,=plt.plot(t, pop1_path_dict['v'][i,:NT2], color="green", alpha=0.3)
    ax2,=plt.plot( t, pop2_path_dict['v'][i,:NT2], color="firebrick", alpha=0.3)
plt.legend({'P1':ax1,'P2':ax2})

plt.subplot(132)
plt.title("$U_t$")
t=np.array([i for i in range(NT2)]) * dt
for i in range(80):
    ax3,=plt.plot(t, pop1_path_dict['u'][i,:NT2], color="green", alpha=0.3)
    ax4,=plt.plot( t, pop2_path_dict['u'][i,:NT2], color="firebrick", alpha=0.3)
plt.legend({'P1':ax3,'P2':ax4})

plt.subplot(133)
plt.title("$Y_t$")
t=np.array([i for i in range(NT2)]) * dt
for i in range(80):
    ax5,=plt.plot(t, pop1_path_dict['y'][i,:NT2], color="green", alpha=0.3)
```