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='clamp',loss_type='MSELoss',delta=0.01,w=0.75,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=0.75,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

#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}
```

In [ ]:

```
#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 ,
                '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}
end_time=datetime.now().strftime('%B %d - %H:%M:%S')
```

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()
```

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',
                      'sigmoid_ind_0.001lr_500steps_MSE_1w') ## 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)
```

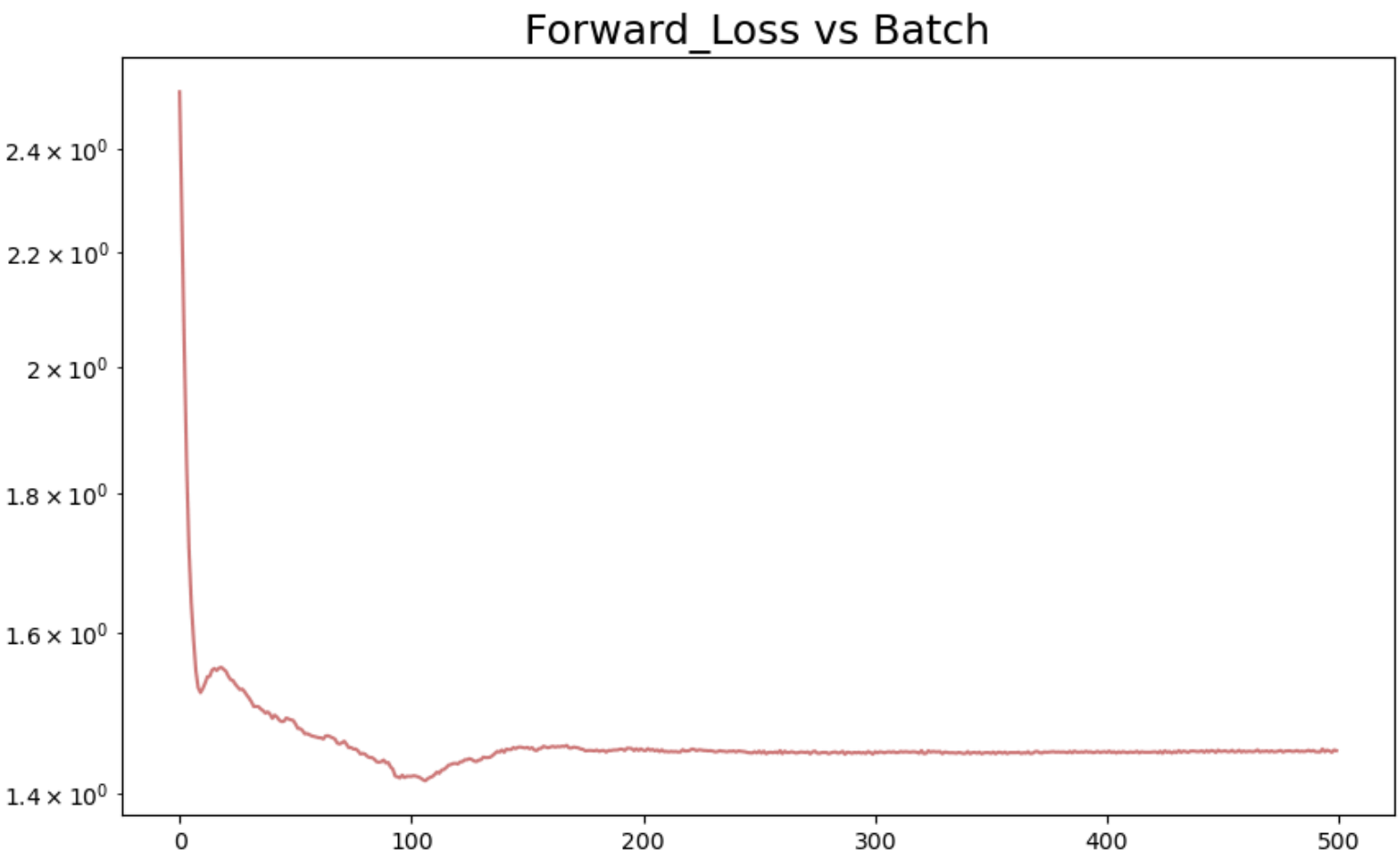
```
0 steps
Started @ September 09 - 17:59:06
Saved @ September 09 - 17:59:52
```

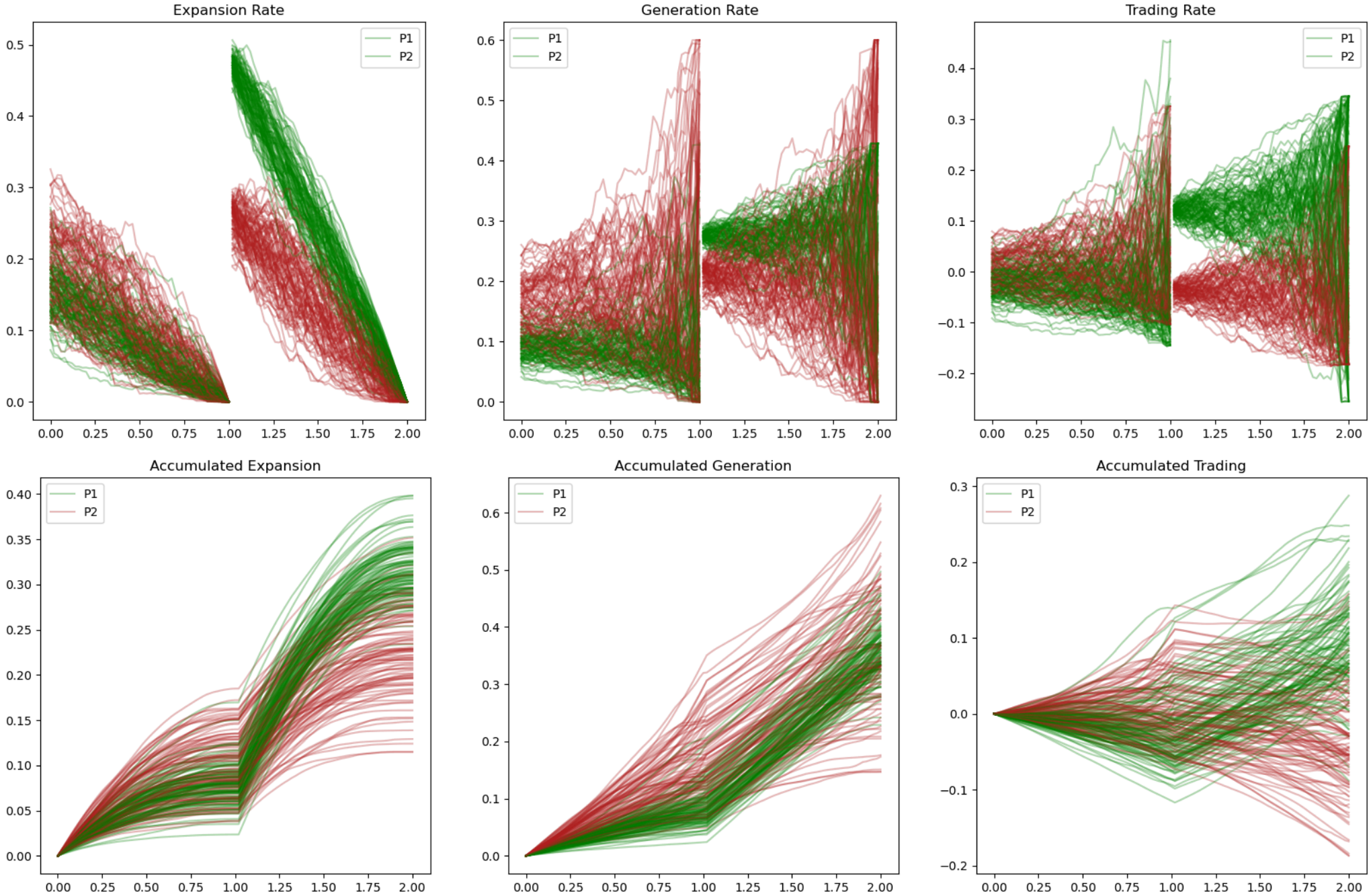
```
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
Cell In[10], line 6
      1 print(f"{len(main_models1.loss)} steps\nStarted @ {start_time}\nSaved @ {end_time}")   ## to examine whether the loss attribute is updat
ed in the module insteance
      2 dir_path=pathlib.Path(os.getcwd(),
      3                       'Results',
      4                       'Best Models Saved',
      5                       'sigmoid_ind_0.001lr_500steps_MSE_1w') ## 0.75, 0.5, 0.25
----> 6 dir_path.mkdir()
      7 path1=pathlib.Path(dir_path,'pop1.pt')
      8 path2=pathlib.Path(dir_path,'pop2.pt')

File ~/Downloads/Anaconda/anaconda3/envs/env_py_311/lib/python3.11/pathlib.py:1116, in Path.mkdir(self, mode, parents, exist_ok)
   1112 """
   1113 Create a new directory at this given path.
   1114 """
   1115 try:
-> 1116     os.mkdir(self, mode)
   1117 except FileNotFoundError:
   1118     if not parents or self.parent == self:

FileExistsError: [Errno 17] File exists: '/Users/orangeao/Orange Ao/Research And Projects/2024.5 Columbia_Steven/2Period/RunTwice_1Prd/Results/B
est Models Saved/sigmoid_ind_0.001lr_500steps_MSE_1w'
```

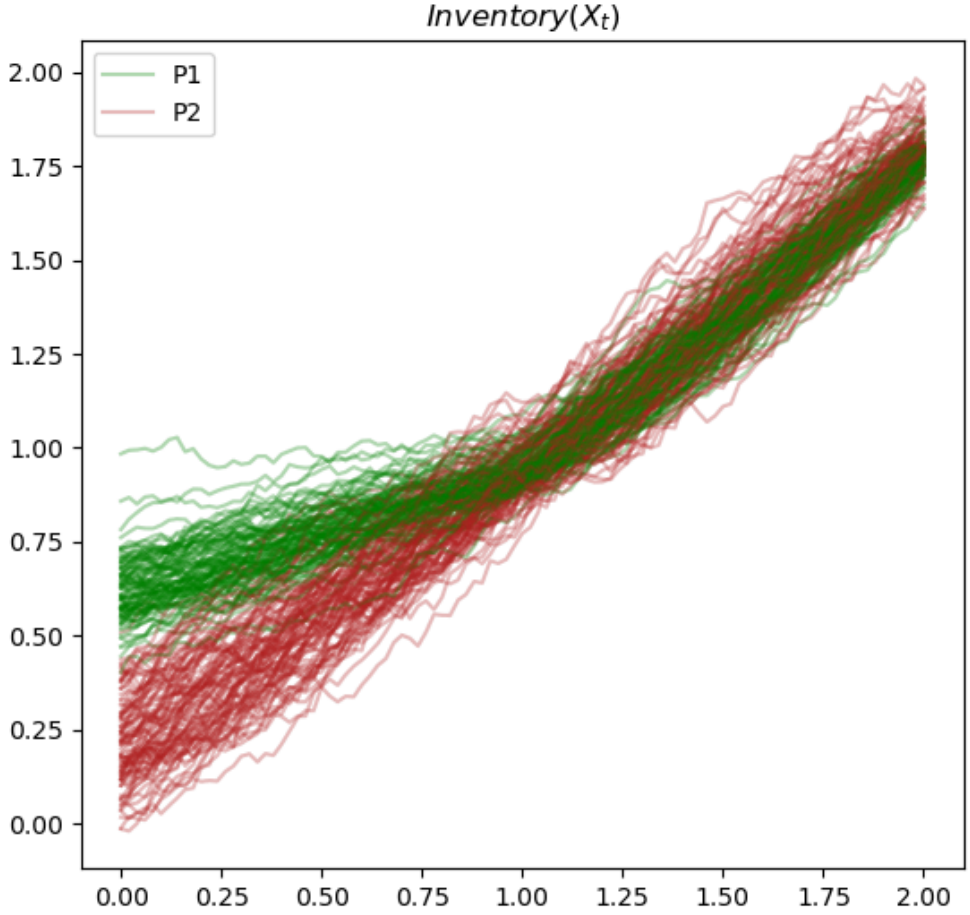
Load The Model

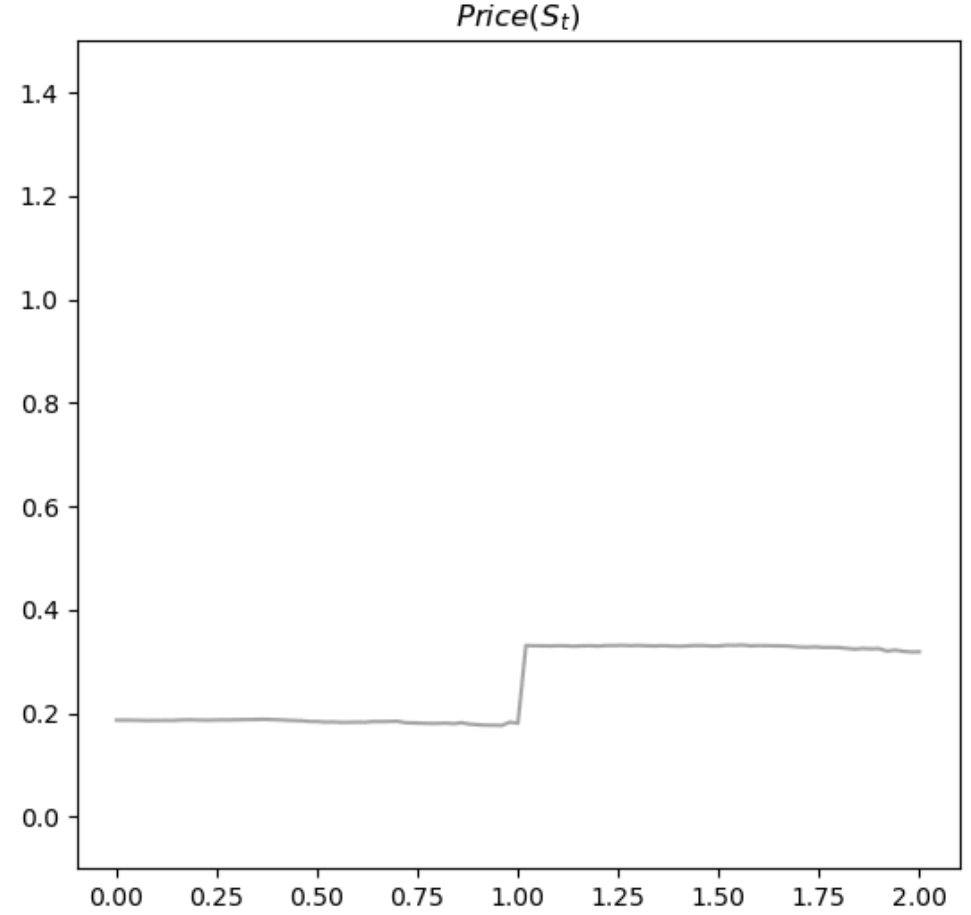
In [ ]:
```
# GlobalParams1=Params(param_type='k1',target_type='indicator',trick='clamp',loss_type='MSELoss',delta=0.03,K=0.9,lr=0.01)
# GlobalParams2=Params(param_type='k2',target_type='indicator',trick='clamp',loss_type='MSELoss',delta=0.03,K=0.9,lr=0.01)
models1=Main_Models(GlobalParams=GlobalParams1)
models2=Main_Models(GlobalParams=GlobalParams2)
path_dir=pathlib.Path(os.getcwd(),
```

```
                    "Results",
                    "Best Models Saved",
                    'sigmoid_ind_0.001lr_500steps_MSE_0.75w')
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.Decomposition_Inventory()
plot.Inventory_And_Price()
plot.Terminal_Convergence()
plot.Key_Processes()
```

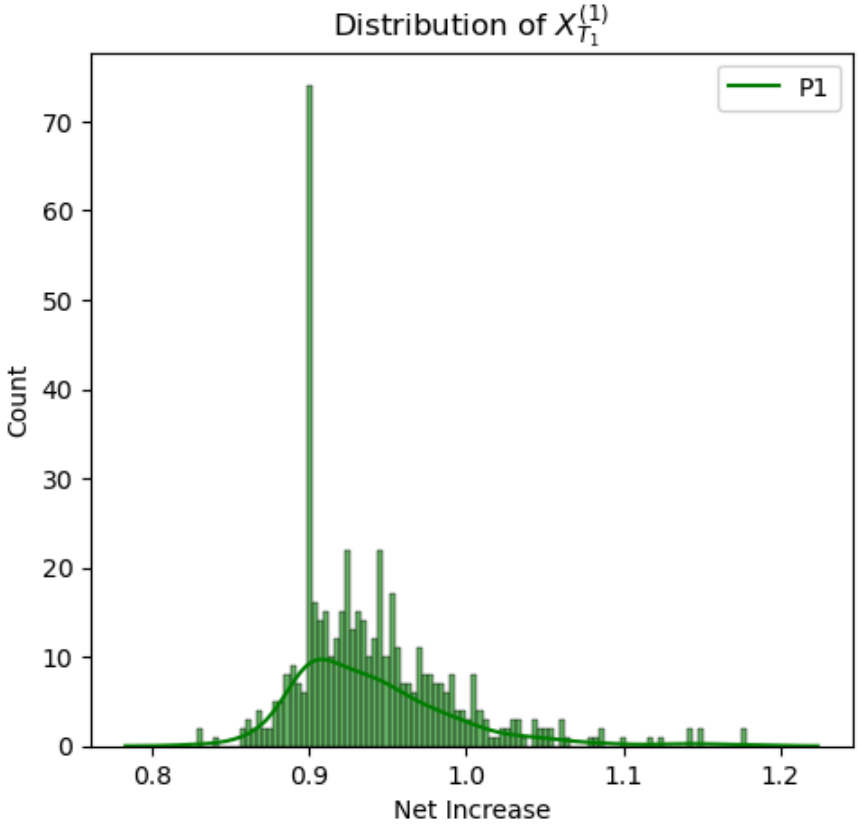

Forward_Loss vs Batch
2.4 × 10^0
2.2 × 10^0
2 × 10^0
1.8 × 10^0
1.6 × 10^0
1.4 × 10^0
0
100
200
300
400
500

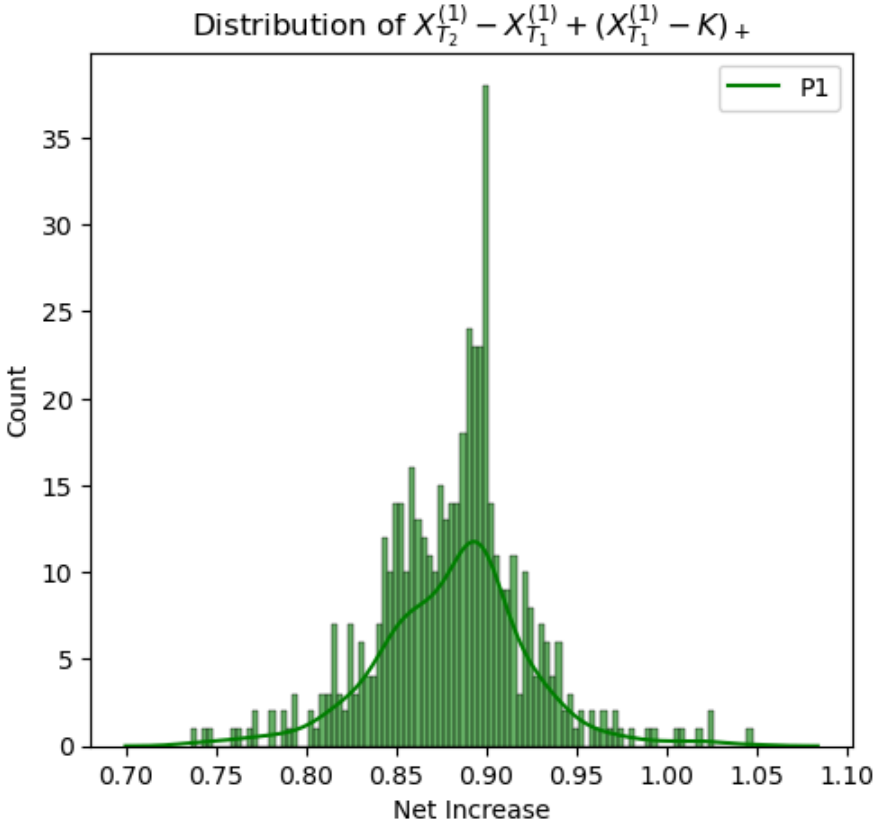

Expansion Rate
Generation Rate
Trading Rate
Accumulated Expansion
Accumulated Generation
Accumulated Trading
P1
P2


Inventory(X_t)
P1
P2
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

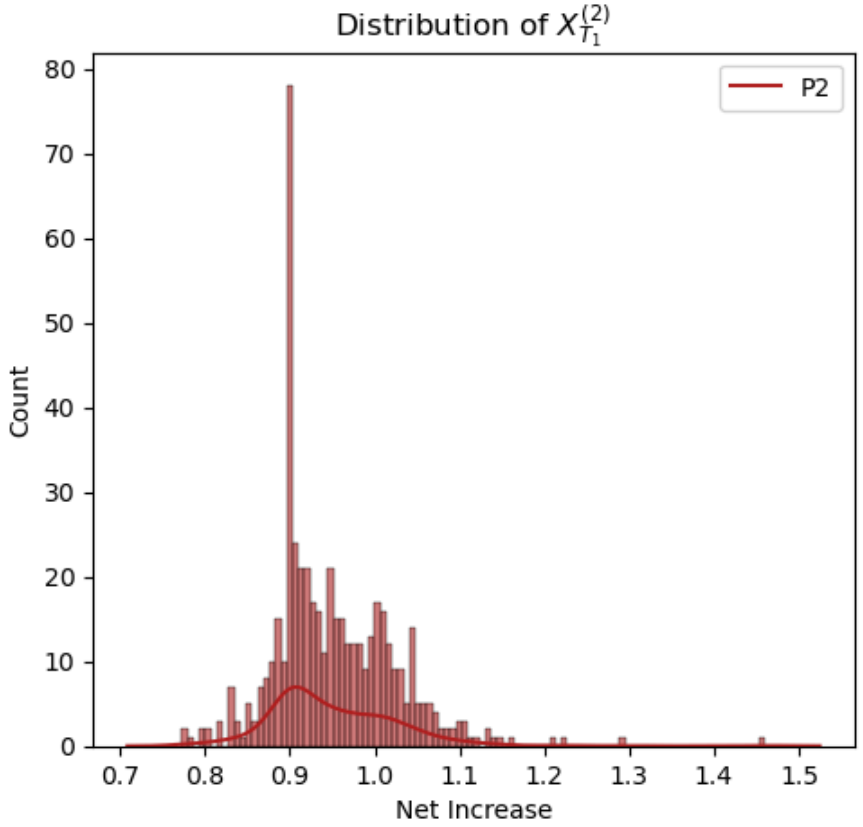

Price(S_t)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

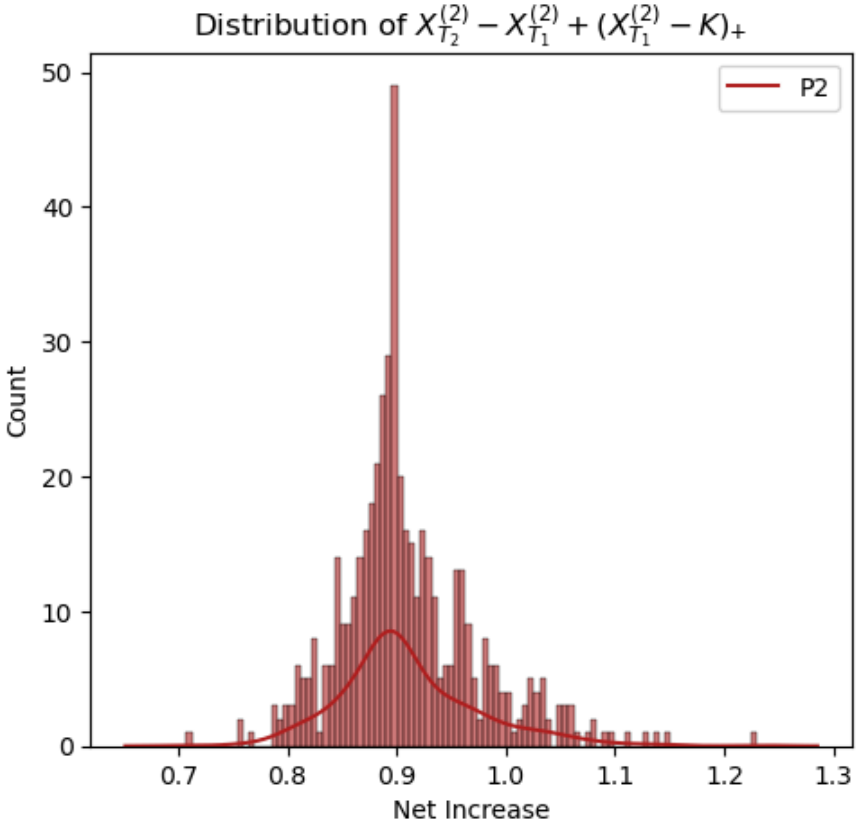
Periodic Inventory Increase - P1



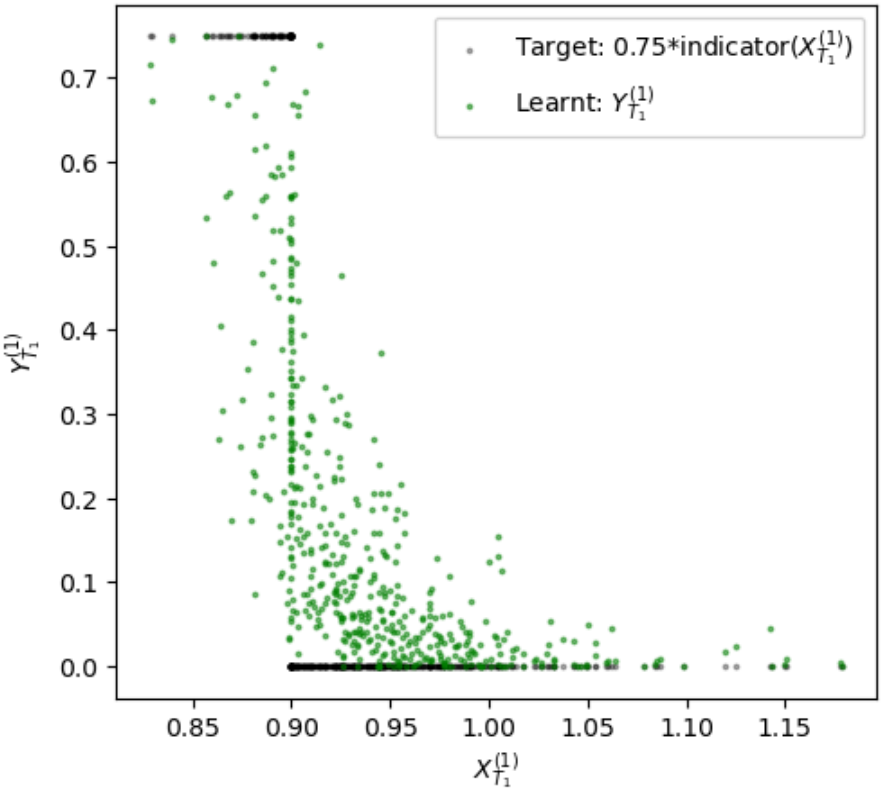


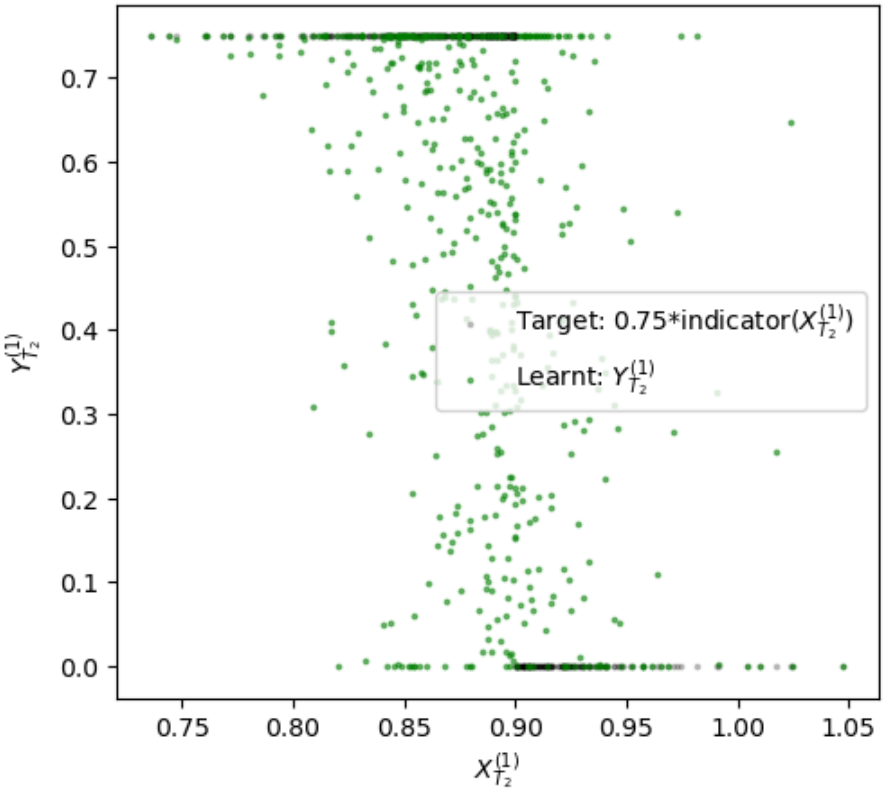
Periodic Inventory Increace - P2



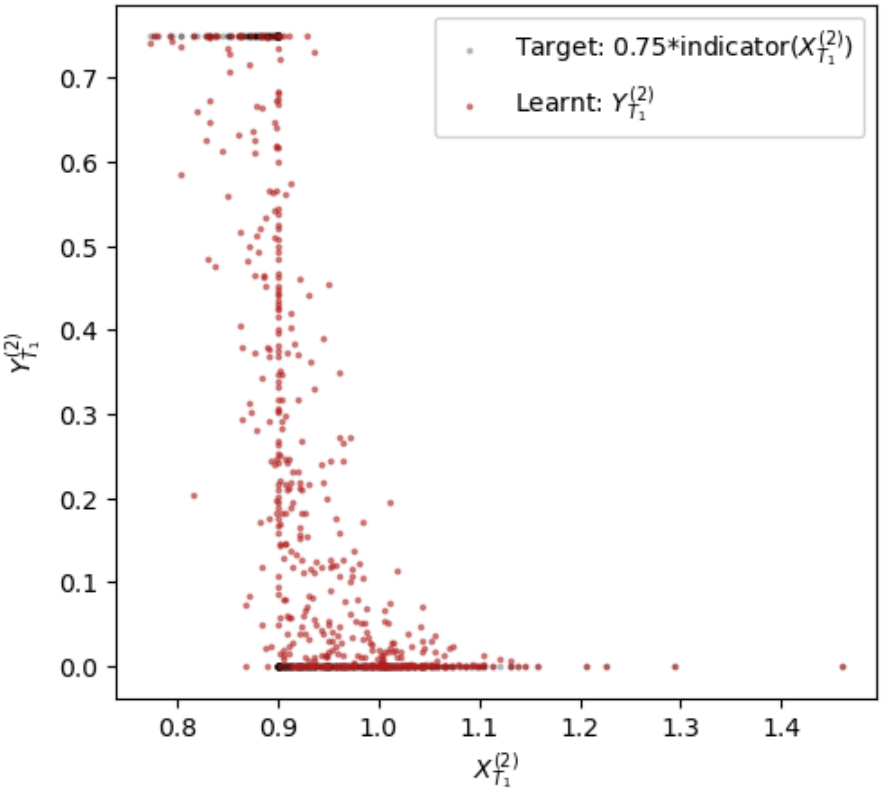


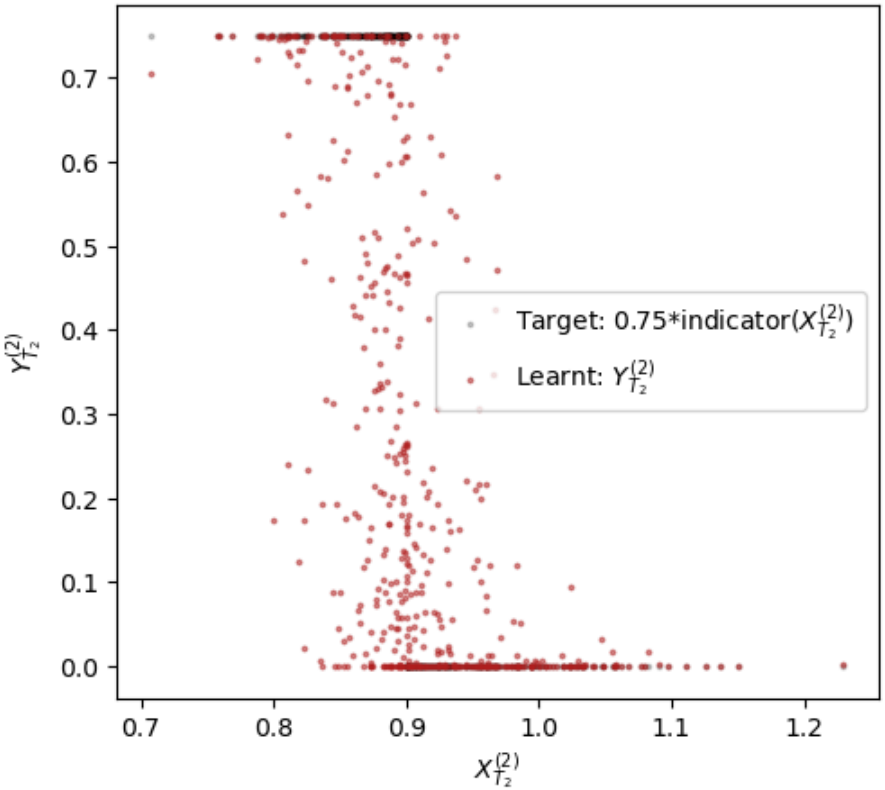
Population 1

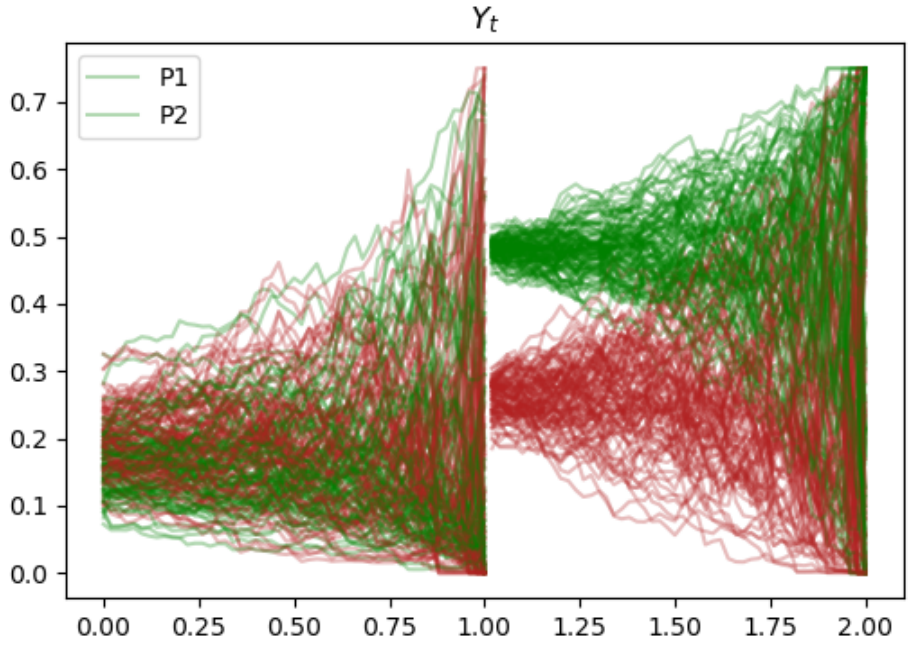




Population 2






Y_t
P1
P2
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00