

lr=0.002

target: sigma=0.08

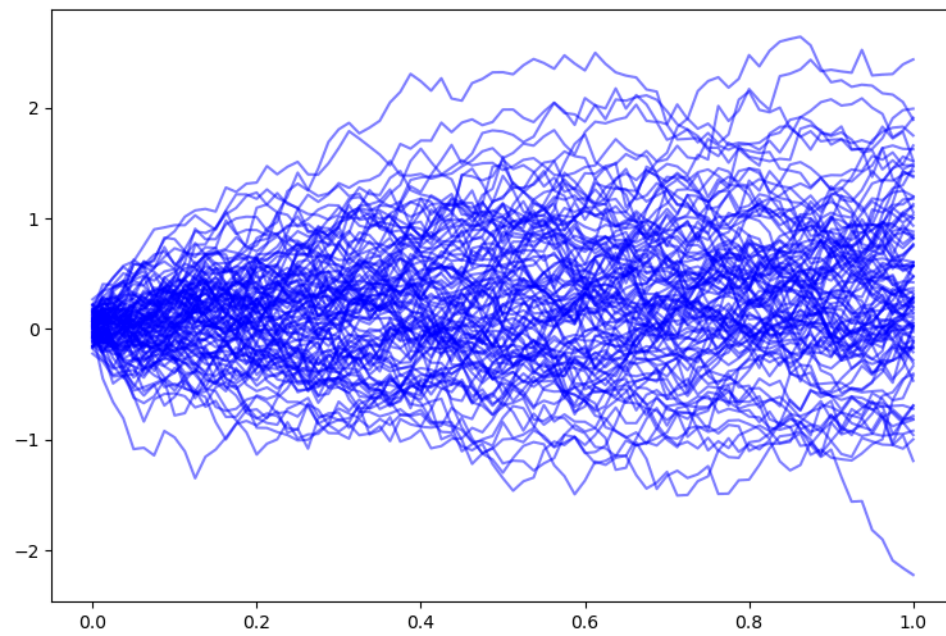
optimizer: AdamW(weight_decay=5e-4)

scheduler: StepLR(step=50, gamma=0.995)

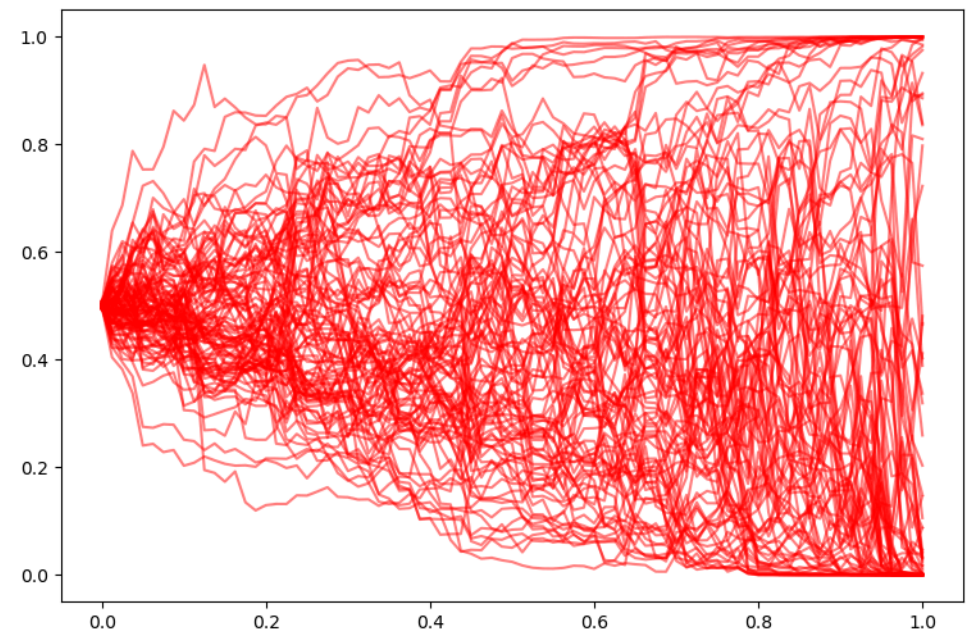
MaxBatch=500

OptimStep=20

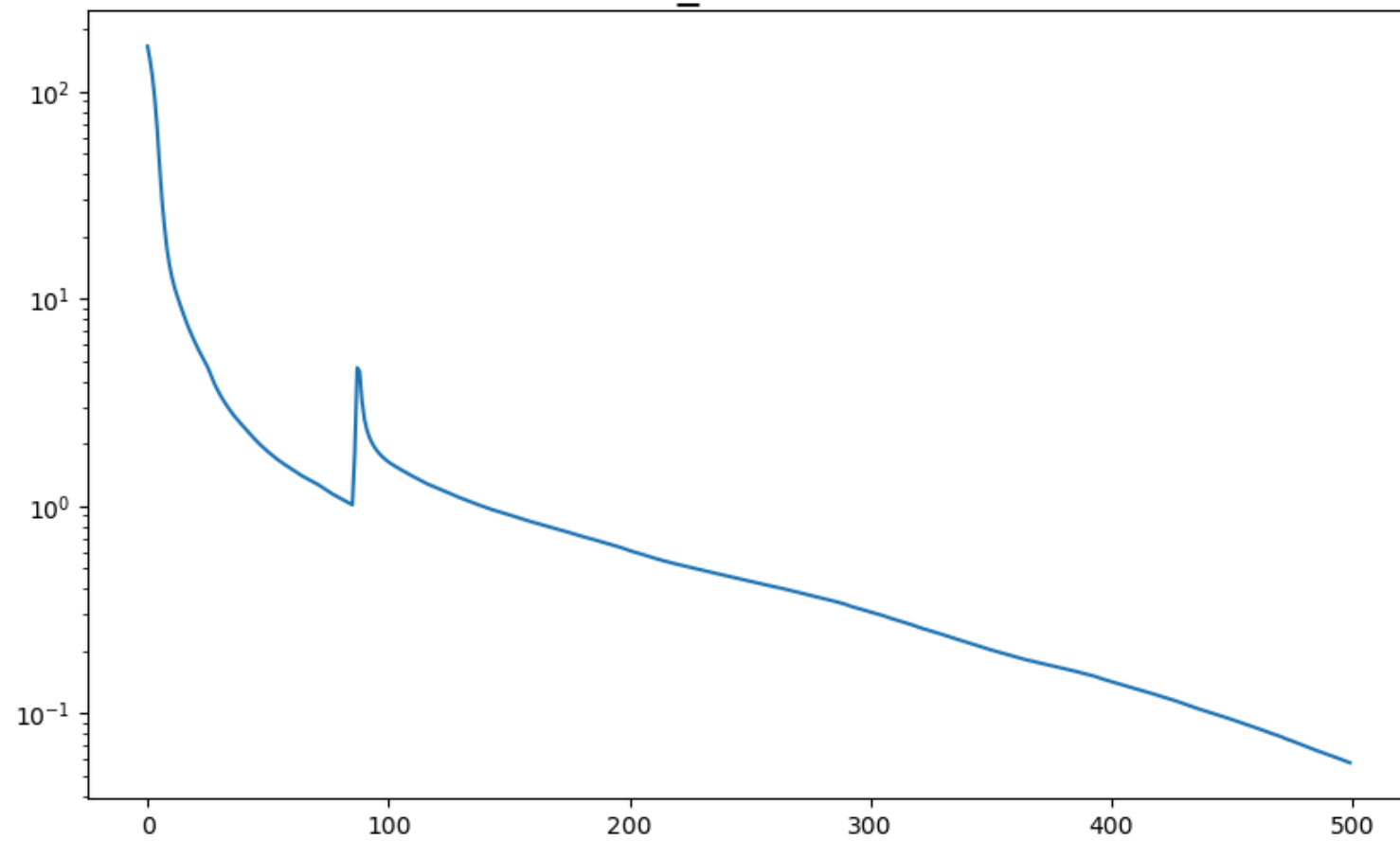
X



Y Values



Forward_Loss vs Batch



```
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
import time
import random
from scipy.stats import norm
```

```
In [ ]: #Model and Params
#Numbers
NumTrain=500
NT=80
dt=1/NT
sigma=0.08
#Forward Loss
forward_losses = []

#Forward Loss
forward_losses = []
#Network Class for FBSDE
class Network(nn.Module):
    def __init__(self, lr, input_dims, fc1_dims, fc2_dims, n_outputs):
        """
        lr: learning rate
        """
        super(Network, self).__init__()

        #Pass input parameters
        self.input_dims = input_dims
        self.fc1_dims = fc1_dims
        self.fc2_dims = fc2_dims
        self.n_out = n_outputs

        #Construct network
        self.fc1 = nn.Linear(*self.input_dims, self.fc1_dims)
        nn.init.xavier_uniform_(self.fc1.weight)
        self.fc2 = nn.Linear(self.fc1_dims, self.fc2_dims)
        nn.init.xavier_uniform_(self.fc2.weight)
        self.fc3 = nn.Linear(self.fc2_dims, self.n_out)
        nn.init.xavier_uniform_(self.fc3.weight)

        self.optimizer = optim.Adam(self.parameters(), lr=lr)
        self.device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
        self.to(self.device)
```

```

def forward(self, input):
    x = F.relu(self.fc1(input))
    x= F.relu(self.fc2(x))
    output = self.fc3(x)
    return output

## Functions
def Sample_Init(N,mean=0,sd=0.1):
    '''
    Generate N samples of x0
    '''

    xi = np.random.normal(mean,sd,size=N)

    return torch.FloatTensor(xi).view(-1,1)

def SampleBMIncr(T, Npaths, Nsteps):
    # Returns Matrix of Dimension Npaths x Nsteps With Sample Increments of of BM
    # Here an increment is of the form dB
    dt = T / Nsteps
    dB = np.sqrt(dt) * np.random.randn(Npaths, Nsteps)
    return torch.FloatTensor(dB)

def target(x,sigma=sigma):
    x=x.detach().numpy()
    return torch.FloatTensor(-x/sigma)

# Forward Loss
def get_foward_loss_coupled(dB, init_x,NT, target,y0_model, z_models):
    x = init_x
    # y = torch.rand_like(x)
    y_tilde=y0_model(x)
    y=torch.sigmoid(y_tilde)
    for j in range(1, NT+1):

        z = z_models[j-1](x)
        x =x+ y*dt+ dB[:,j].view(-1,1)
        y_tilde = (y_tilde +(z**2)*(1-2/(1+torch.exp(y_tilde)))/2*dt + z * dB[:,j].view(-1,1))#.clamp(min=-1,max=1)
        y=torch.sigmoid(y_tilde)

    loss=torch.mean((y_tilde-target(x))**2)
    return loss

def get_target_path_coupled(dB, init_x,NumBM, NT,y0_model, z_models):
    x_path = torch.ones(NumBM,NT+1)
    y_path = torch.ones(NumBM,NT+1)
    x = init_x
    # y = torch.rand_like(x)
    y_tilde=y0_model(x)
    y=torch.sigmoid(y_tilde)
    x_path[:,0] = x.squeeze()

```

```

y_path[:,0] = y.squeeze()
for j in range(1, NT+1):
    z = z_models[j-1](x)
    x += y*dt+ dB[:,j].view(-1,1)
    y_tilde = (y_tilde +(z**2)*(1-2/(1+torch.exp(y_tilde)))/2 *dt + z * dB[:,j].view(-1,1))#.clamp(min=-1,max=1)
    y=torch.sigmoid(y_tilde)
    x_path[:,j] = x.squeeze()
    y_path[:,j] = y.squeeze()
return x_path.detach(), y_path.detach()

class plot_results():
    def __init__(self, loss=forward_losses, sigma=sigma, Npaths=100, NumTrain=NumTrain, NT=NT):
        self.loss=loss
        self.x_path, self.y_path=get_target_path_coupled(dB, init_x, y0_model=y0_model_main, z_models=z_models_main, NumBM=NumTrain, NT=NT)
        self.number_of_paths=np.minimum(Npaths, NumTrain)
        self.sigma=sigma

    def FwdLoss(self, log=True):
        plt.figure(figsize=(10,6))
        plt.title("Forward_Loss vs Batch", fontsize=18)
        plt.plot(self.loss)

        if log==True:
            plt.yscale('log')

    def results(self, seed=0):
        random.seed(seed)
        idx_list = np.random.choice(NumTrain, self.number_of_paths, replace = False)
        x_plot = self.x_path.detach().numpy()[idx_list]
        y_plot = self.y_path.detach().numpy()[idx_list]
        t = np.array([i for i in range(NT+1)]) * 1/(NT)
        plt.figure(figsize=(20,6))
        plt.subplot(121)
        for i in range(self.number_of_paths):
            plt.plot(t, x_plot[i], color="blue", alpha=0.5)
        plt.title("X")

        plt.subplot(122)
        for i in range(self.number_of_paths):
            plt.plot(t, y_plot[i], color="red", alpha=0.5)
        plt.title("Y Values")

    ### Integrated Plots
    random.seed(seed)
    idx=random.randint(0, self.number_of_paths)
    plt.figure(figsize=(10,8))
    plt.subplot()
    plt.plot(t, x_plot[idx], color="blue", alpha=0.5, label='X')
    plt.plot(t, y_plot[idx], color="black", linestyle='--', alpha=0.5, label='Y Values')
    plt.hlines(y=[0,1], xmin=0, xmax=1, colors='firebrick', linestyle='-.')
    plt.title("Comparison of A Particular Path")

```

```
plt.legend()
```

```
def qq_plot(self, sigma=sigma):  
    plt.figure()  
    plt.title("QQ-Plot")  
    x_sigmoid=1/(1+np.exp(self.x_path[:,-1]/sigma))  
    plt.scatter(x_sigmoid, self.y_path[:,-1], s=3)  
    plt.plot(np.linspace(0,1,5), np.linspace(0,1,5), linestyle='--', linewidth=1, color='r')
```

```
In [ ]: ## Train
```

```
torch.autograd.set_detect_anomaly(True)
```

```
dB = SampleBMIncr(1, Npaths=NumTrain, Nsteps=NT+1)
```

```
init_x = Sample_Init(N=NumTrain)
```

```
#Forward Loss
```

```
forward_losses = []
```

```
#How many batches?
```

```
MaxBatch= 500
```

```
#How many optimization steps per batch
```

```
OptimSteps= 20
```

```
#Set Learning rate
```

```
learning_rate = 0.002
```

```
#Train on a single batch?
```

```
single_batch = True
```

```
#Set up main models for y0 and z (z will be list of models)
```

```
layer_dim = 10
```

```
y0_model_main = Network(lr=learning_rate, input_dims=[1], fc1_dims=layer_dim, fc2_dims=layer_dim,  
                        n_outputs=1)
```

```
z_models_main = [Network(lr=learning_rate, input_dims=[1], fc1_dims=layer_dim, fc2_dims=layer_dim,  
                        n_outputs=1) for i in range(NT)]
```

```
#Define optimization parameters
```

```
# params = list(y0_model_main.parameters())
```

```
params=[]
```

```
for i in range(NT):
```

```
    params += list(z_models_main[i].parameters())
```

```
#Set up optimizer and scheduler
```

```
optimizer = optim.AdamW(params, lr=learning_rate, weight_decay=5e-4)
```

```
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.995)
```

```
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_coupled(dB, init_x,NT=NT,target=target, y0_model=y0_model_main, z_models=z_models_main)
    # print(loss)

    loss.backward()
    # print(params)
    # torch.nn.utils.clip_grad_norm_(parameters=params,max_norm=5,norm_type=1)
    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):
    dB = SampleBMIncr(1, Npaths=NumTrain, Nsteps=NT+1)
    init_x = Sample_Init(N=NumTrain)

plot=plot_results(loss=forward_losses)
plot.FwdLoss()
plot.results()
plot.qq_plot()

```


Batch Number: 1
Average Error Est: 165.71779251098633
Batch Number: 2
Average Error Est: 142.69447708129883
Batch Number: 3
Average Error Est: 118.73455238342285
Batch Number: 4
Average Error Est: 92.51714744567872
Batch Number: 5
Average Error Est: 66.35962448120117
Batch Number: 6
Average Error Est: 43.473055839538574
Batch Number: 7
Average Error Est: 30.0159818649292
Batch Number: 8
Average Error Est: 22.41275224685669
Batch Number: 9
Average Error Est: 17.628997707366942
Batch Number: 10
Average Error Est: 14.778942346572876
Batch Number: 11
Average Error Est: 12.904810285568237
Batch Number: 12
Average Error Est: 11.591102647781373
Batch Number: 13
Average Error Est: 10.578693771362305
Batch Number: 14
Average Error Est: 9.755928230285644
Batch Number: 15
Average Error Est: 9.031307315826416
Batch Number: 16
Average Error Est: 8.374949264526368
Batch Number: 17
Average Error Est: 7.791522717475891
Batch Number: 18
Average Error Est: 7.270590758323669
Batch Number: 19
Average Error Est: 6.826401090621948
Batch Number: 20
Average Error Est: 6.423320984840393
Batch Number: 21
Average Error Est: 6.0537049770355225
Batch Number: 22
Average Error Est: 5.730281949043274
Batch Number: 23
Average Error Est: 5.436657333374024
Batch Number: 24
Average Error Est: 5.171663784980774
Batch Number: 25
Average Error Est: 4.923022437095642
Batch Number: 26

Average Error Est: 0.06681937500834464
Batch Number: 486
Average Error Est: 0.06615636795759201
Batch Number: 487
Average Error Est: 0.06553151309490204
Batch Number: 488
Average Error Est: 0.06494749151170254
Batch Number: 489
Average Error Est: 0.06435851640999317
Batch Number: 490
Average Error Est: 0.06382912024855614
Batch Number: 491
Average Error Est: 0.06315323561429978
Batch Number: 492
Average Error Est: 0.06257949974387884
Batch Number: 493
Average Error Est: 0.061932251788675785
Batch Number: 494
Average Error Est: 0.06125153936445713
Batch Number: 495
Average Error Est: 0.060682787001132964
Batch Number: 496
Average Error Est: 0.06002087127417326
Batch Number: 497
Average Error Est: 0.05940341539680958
Batch Number: 498
Average Error Est: 0.05883570425212383
Batch Number: 499
Average Error Est: 0.058239319175481793
Batch Number: 500
Average Error Est: 0.05762508977204561