PSTAT 222C Homework 3

Alex Bernstein

```
{r, setup, include=FALSE} knitr::opts_chunk$set(   cache=TRUE )
```

```
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm,trange
from numba import jit,njit
import copy
import pdb
import time
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
import sklearn.gaussian_process as gp
from sklearn.gaussian_process.kernels import Matern
```

Problem 1

```
@njit
def explicit_FD_put(dt,ds,dv,K,freq=None):
    if(freq==None):
        freq = dt
    # Heston Model
    r = 0.05
    kappa = 1
    theta = 0.2
    eta = 0.5
    rho = -0.4

    # Option
    T=1
    S_max = 2.5*K
    V_min = 0.05
    V_max = 0.6
    # Stop set defintion.  time between allowed stops is freq/dt; can stop at T/freq indicies
```

```
    # Default is freq = dt; this is American option
    stop_set = np.arange((round(T/freq)))*(freq/dt)

    # Discretization
    time_n = round(T/dt)
    asset_n = round(S_max/ds)+1
    vol_n = round((V_max-V_min)/dv)
    vetS = ds*np.arange(asset_n)

    payoff_exit = np.maximum(K-vetS,0).repeat(vol_n).reshape((-1,vol_n))
    f = np.copy(payoff_exit)*1.0
    for i in range(time_n):
        newf = np.zeros((asset_n,vol_n))
        for j in range(1,asset_n-1):
            for k in range(1,vol_n-1):
                A = np.zeros((3,3))
                A[2,2] = A[0,0] = rho*eta*j*(V_min/dv+k)*dt/4
                A[0,2] = A[2,0] = -rho*eta*j*(V_min/dv+k)*dt/4
                A[1,1] = 1-r*dt - j**2*(V_min/dv+k)*dv*dt - eta**2*(V_min/dv+k)*dt/dv
                A[0,1] = -(r*j*dt/2 - j**2*(V_min/dv + k)*dv*dt/2)
                A[2,1] = r*j*dt/2 + j**2*(V_min/dv + k)*dv*dt/2
                A[1,0] = -(dt*kappa*(theta - (V_min/dv + k)*dv)/ (2*dv) - eta**2*(V_min/dv + k)*dt/(2*dv
                A[1,2] = dt*kappa*(theta - (V_min/dv + k)*dv)/(2*dv) + eta**2*(V_min/dv + k)*dt/(2*dv)
                newf[j,k] = (
                    A[0,0]*f[j-1,k-1] + A[0,1]*f[j-1,k] + A[0,2]*f[j-1,k+1] +
                    A[1,0]*f[j,k-1] + A[1,1]*f[j,k] + A[1,2]*f[j,k+1] +
                    A[2,0]*f[j+1,k-1]+A[2,1]*f[j+1,k]+A[2,2]*f[j+1,k+1])

        # Lower V Boundary
        newf[1:asset_n-1,0] = 2*newf[1:asset_n-1,1]-newf[1:asset_n-1,2]

        # Upper V Boundary
        newf[1:asset_n-1,vol_n-1] = 2*newf[1:asset_n-1,vol_n-2]-newf[1:asset_n-1,vol_n-3]

        # Lower S Boundary
        newf[0,:] = K*np.exp(-r*(i+1)*dt)

        if(i in stop_set):
            f = np.maximum(payoff_exit,newf)
        else:
            f = newf
    return f


S0, V0, dt, ds, dv, V_min, K = 100, 0.25, 10**(-5), 5, 0.05, 0.05, 100
```

```
# Obtaining American Put prices
pval = explicit_FD_put(dt, ds, dv, K)
# Option value at S0, V0
pval[round(S0/ds),round((V0 - V_min)/ dv)]
```

```
16.181423085622114
```

Problem 1a

Problem 1b

Problem 1c

We examine the two dimensional GBM model:

$$dS_t^1 = (r-d)S_t^1 dt + \sigma_1 S_t^1 dW_t^1$$
$$dS_t^2 = (r-d)S_t^2 dt + \sigma_2 S_t^2 dW_t^2$$

Applying Ito's Lemma, we have:

$$df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial S^1}dS_t^1 + \frac{\partial f}{\partial S^2}dS_t^2 + \frac{1}{2}\frac{\partial^2 f}{\partial (S^1)^2}(dS_t^1)^2 + \frac{1}{2}\frac{\partial^2 f}{\partial (S^2)^2}(dS_t^2)^2 + +\frac{1}{2}\frac{\partial^2 f}{\partial S^1 \partial S^2}dS_t^1 dS_t^2$$

where we have $dt^2 = 0$, $dW_1 dW_2 = \rho dt$ with $\rho = 0$ and so:

$$(dS_t^1)^2 = \sigma_1^2 (S_t^1)^2 dt \qquad (dS_t^2)^2 = \sigma_2^2 (S_t^2)^2 dt \qquad dS_t^1 dS_t^2 = \rho \sigma_1 \sigma_2 S_t^1 S_t^2 dt = 0$$

And so, we have:

$$df = \left(\frac{\partial f}{\partial t} + (r-d)S_t^1 \frac{\partial f}{\partial S^1} + (r-d)S_t^2 \frac{\partial f}{\partial S^2} + \frac{1}{2}\sigma_1^2 (S_t^1)^2 \frac{\partial^2 f}{\partial (S_1)^2} + \frac{1}{2}\sigma_2^2 (S_t^2)^2 \frac{\partial^2 f}{\partial (S_2)^2} \right) dt + \sigma_1 S_t^1 dW_t^1 + \sigma_2 S_t^2 dW_t^2$$

Taking the expectation under the risk neutral measure, we find $\mathbb{E}[df] = rfdt$. Because the martingale terms have 0 expectation, we generate the following PDE:

$$\frac{\partial f}{\partial t} + (r-d)S_t^1 \frac{\partial f}{\partial S^1} + (r-d)S_t^2 \frac{\partial f}{\partial S^2} + \frac{1}{2}\sigma_1^2 (S_t^1)^2 \frac{\partial^2 f}{\partial (S_1)^2} + \frac{1}{2}\sigma_2^2 (S_t^2)^2 \frac{\partial^2 f}{\partial (S_2)^2} - rf = 0$$

Similar to the previous problems, we can discretize this PDE. Define $f_{i,j,k} = f(i\Delta t, j\Delta S^1, k\Delta S^2)$ where i, j, k are integers. Our derivatives are approximated as follows:

$$\frac{\partial f}{\partial t} = \frac{f_{i,j,k} - f_{i-1,j,k}}{\Delta t}$$
$$\frac{\partial f}{\partial S^1} = \frac{f_{i,j+1,k} - f_{i,j-1,j}}{2\Delta S^1} \qquad \frac{\partial f}{\partial S^2} = \frac{f_{i,j,k+1} - f_{i,j,k-1}}{2\Delta S^1}$$
$$\frac{\partial^2 f}{\partial (S^1)^2} = \frac{f_{i,j+1,k} - 2f_{i,j,k} + f_{i,j-1,k}}{(\Delta S^1)^2} \quad \frac{\partial^2 f}{\partial (S^2)^2} = \frac{f_{i,j,k+1} - 2f_{i,j,k} + f_{i,j,k-1}}{(\Delta S^2)^2}$$

Substituting these values and defining $S_j^1 = j\Delta S^1$ and $S_k^2 = k\Delta S^2$ we have:

$$\frac{f_{i,j,k} - f_{i-1,j,k}}{\Delta t} + (r-d)\left(S_j^1 \frac{f_{i,j+1,k} - f_{i,j-1,j}}{2\Delta S^1} + S_k^2 \frac{f_{i,j,k+1} - f_{i,j,k-1}}{2\Delta S^1}\right)$$
$$+ \frac{1}{2}\left(\sigma_1^2 (S_j^1)^2 \frac{f_{i,j+1,k} - 2f_{i,j,k} + f_{i,j-1,k}}{(\Delta S^1)^2} + \sigma_2^2 (S_j^2)^2 \frac{f_{i,j,k+1} - 2f_{i,j,k} + f_{i,j,k-1}}{(\Delta S^2)^2}\right) - r f_{i,j,k} = 0$$

Grouping terms, we have:

$$\begin{aligned} f_{i-1,j,k} =& f_{i,j,k}\left(1 - r\Delta t - \sigma_1^2 j^2 \Delta t - \sigma_2^2 k^2 \Delta t\right) \\ &+ f_{i,j-1,k}\left(\frac{\sigma_1^2 j^2 \Delta t}{2} - \frac{(r-d)j\Delta t}{2}\right) + f_{i,j+1,k}\left(\frac{(r-d)j\Delta t}{2} + \frac{\sigma_1^2 j^2 \Delta t}{2}\right) \\ &+ f_{i,j,k-1}\left(\frac{\sigma_2^2 j^2 \Delta t}{2} - \frac{\sigma_2^2 k^2 \Delta t}{2}\right) + f_{i,j,k+1}\left(\frac{(r-d)k\Delta t}{2} + \frac{\sigma_1^2 k^2 \Delta t}{2}\right) \end{aligned}$$

Problem 2

Problem 2a

```
sqrt = np.sqrt
@jit(nopython=True)
def mc_samples(M,Z,s0,T,dt):
    b0 = 0.08; b1 = -0.08; b2 = 0.5 ; K = 10
    l1 = 62; l2 = 40
    r1 = -0.044*np.ones(M); r2 = 0.007*np.ones(M)
    nsteps = int(T/dt)
    r_risk_free = 0.05
    S = np.ones((M,nsteps))*s0
    for j in range(1,nsteps):
        dW = sqrt(dt)*Z[:,j-1]
        s12 = np.maximum(b0 +r1*b1+sqrt(r2)*b2,0)
        S[:,j] = np.maximum(S[:,j-1]+S[:,j-1]*s12*dW,0)
        r1 = r1 + l1*(s12*dW - r1)*dt
        r2 = r2 + l2*( s12**2 -r2)*dt
        r2 = np.maximum(r2,0)
```

```
    return( np.maximum( np.exp(-r_risk_free*T)*(K-S[:,-1]),0))
M = 1000
T = 1/4
dt = 1/(2*365)
Z = np.random.normal(size=(M,int(1/dt)))
print(np.mean(mc_samples(M,Z,s0=10,T=T,dt=dt)))
```

```
0.3124128164576378
```

Problem 2b

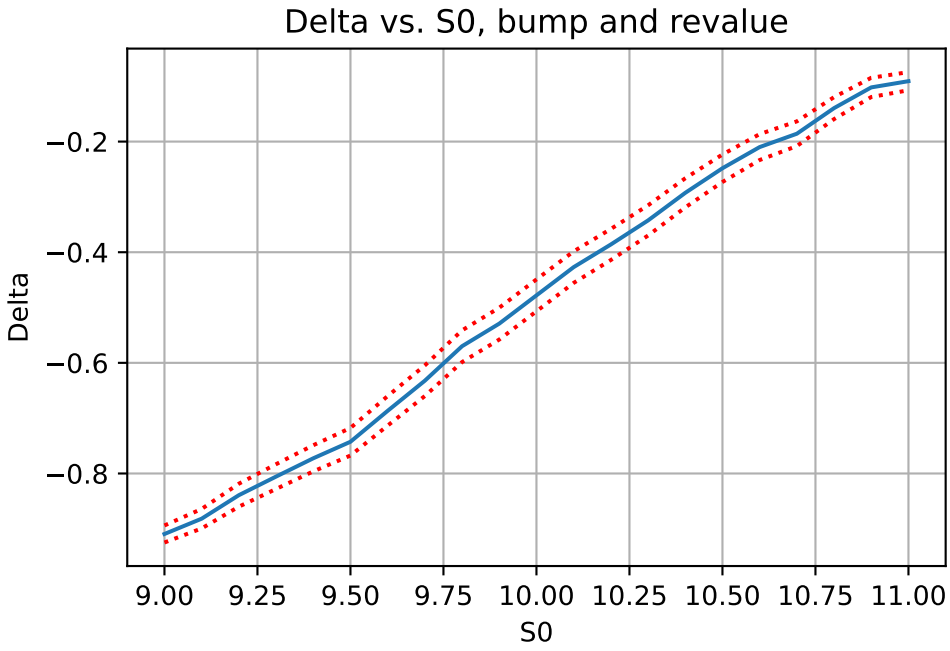
```
@jit(nopython=True)
def bump_eval_sim(npaths,s0,K,eps,T=1/4):
    dt = 1/(2*365)
    n_steps = int(T/dt)
    dW = np.random.normal(0,1,size=(npaths,n_steps))
    p_up = mc_samples(npaths,dW,s0+eps,T,dt)
    p_down = mc_samples(npaths,dW,s0-eps,T,dt)
    prices = (p_up+p_down)/2
    deltas = (p_up-p_down)/(2*eps)
    r ={}
    r['price'] = np.mean(prices)
    r['price_std']=np.std(prices)
    r['delta'] = np.mean(deltas)
    r['delta_std']=np.std(deltas)
    return r

K = 10; npaths = 1000
s0_vals =np.arange(9,11.1,0.1)
T = 1/4
dt = 1/(2*365)
eps = 0.005
price = []
price_ci = []
delta = []
delta_ci = []
for m in trange(len(s0_vals)):
    out = bump_eval_sim(npaths,s0_vals[m],K,eps)
    price.append(out['price'])
    delta.append(out['delta'])
    price_ci.append(1.96*(out['price_std']/sqrt(npaths)))
    delta_ci.append(1.96*(out['delta_std']/sqrt(npaths)))
```

```
data_all = pd.DataFrame(data = [price, price_ci,delta,delta_ci]).T
data_all.index = s0_vals
data_all.index.name = "S0"
data_all.columns = ["price","price ci","delta","delta ci"]

data_all
plt.plot(s0_vals,data_all["delta"])
plt.plot(s0_vals,data_all["delta"]-data_all["delta ci"],linestyle="dotted",color="red")
plt.plot(s0_vals,data_all["delta"]+data_all["delta ci"],linestyle="dotted",color="red")
plt.title("Delta vs. S0, bump and revalue")
plt.xlabel("S0")
plt.ylabel("Delta")
plt.grid(True)
plt.show()
```

```
0%|          | 0/21 [00:00<?, ?it/s]  5%|         | 1/21 [00:01<00:36,  1.84s/it] 90%|      | 19/21 [00:
```



Problem 2c

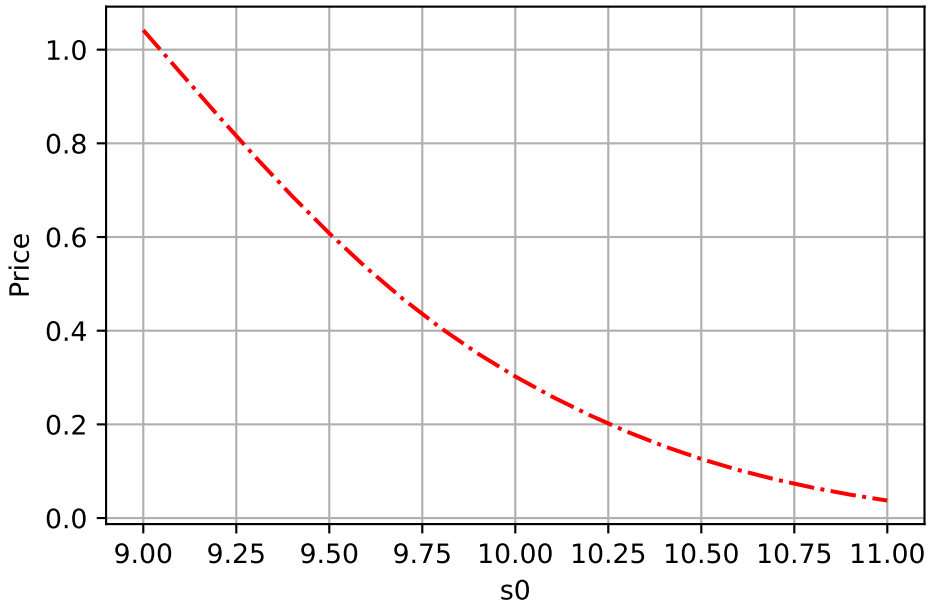
Let m_x be the posterior value at point x. The output for a gaussian process regression with N training points and kernel function $\kappa(\cdot,\cdot)$ is given by:

$$m_x = \mathbb{E}\{m|X,\mathbf{y}\} = \sum_{i=1}^{N} \alpha_i \kappa(x_i, x)$$

where $\alpha = (\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}$, $K_{i,j} = \kappa(x_i, x_j)$, and σ_n^2 is the uncertainty of each observation. Differentiating this expression with respect to S_0 is straightforward:

$$\partial_{S_0}\mathbb{E}\{m_x|X,\mathbf{y}\} = \sum_{i=1}^{n} \alpha_i \partial_{S_0}\kappa(x_i, x)$$

```
npaths, T , dt, K = 1000, 1/4, 1/(2*365), 100
## S0 values between 9 and 11 varying by 0.1
s0_vals = np.arange(9,11.01,0.1)
Y=[]
## Generate 1000 paths for each S0 value
sd_Y = []
for s in s0_vals:
    Z = np.random.normal(size=(npaths,int(1/dt)))
    sample_out = mc_samples(npaths,Z,s,T,dt)
    Y.append(np.mean(sample_out)*1.0)
    sd_Y.append(np.std(sample_out)*1.0)
# Y.reshape(-1,1)
obs_var=np.diag(np.maximum(1e-4,sd_Y))
kernel = gp.kernels.Matern(nu=2.5,length_scale_bounds=[1E-5,10000])+gp.kernels.WhiteKernel(noise_level=1
model = gp.GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10,alpha=1e-4)
#Fit Model
model.fit(s0_vals.reshape(-1,1),Y)
# Predict on S0 and get standard deviations
fit_out,sigma = model.predict(s0_vals.reshape(-1,1),return_std=True)
# data_all["price gp"] = model.predict(s0_vals.reshape(-1,1))
data_all["price gp"] = fit_out
plt.plot(s0_vals,data_all["price gp"],linestyle="dashdot",color="red")
# plt.plot(s0_vals,data_all["price"])
plt.xlabel("s0")
plt.ylabel("Price")
plt.grid(True)
plt.show()
```




```
# Formula from Paper

def matern_deriv_mult(x,x_train,l):
    num = -5/(3*l**2)*(x-x_train)-(5**(1.5))/(3*l**3)*(x-x_train)*np.abs(x-x_train)
    denom = 1 + sqrt(5)/l*np.abs(x-x_train)+5/(3*l**2)*(x-x_train)**2
    return (num/denom).ravel()


def matern_deriv_var(y,l):
    covmat = model.kernel_(s0_vals,s0_vals) + obs_var
    mult_term = np.linalg.inv(covmat)
    d52 = matern_deriv_mult(x,y,l)
    return -5/(3*l**2)- (d52 @ mult_term) @ d52


ell =model.kernel_.get_params()['k1'].get_params()['length_scale']
# ell =model.kernel_.get_params()['length_scale']
# Initialize 0 vector
delta_gp = np.zeros(len(s0_vals))*0.0

# x_a is point at which deriv is taken- in this case, we sweep across s0 values
for key,x in enumerate(s0_vals):
    kern_val = model.kernel_(x.reshape(-1,1),s0_vals.reshape(-1,1),eval_gradient=False).ravel() # Vecto
```

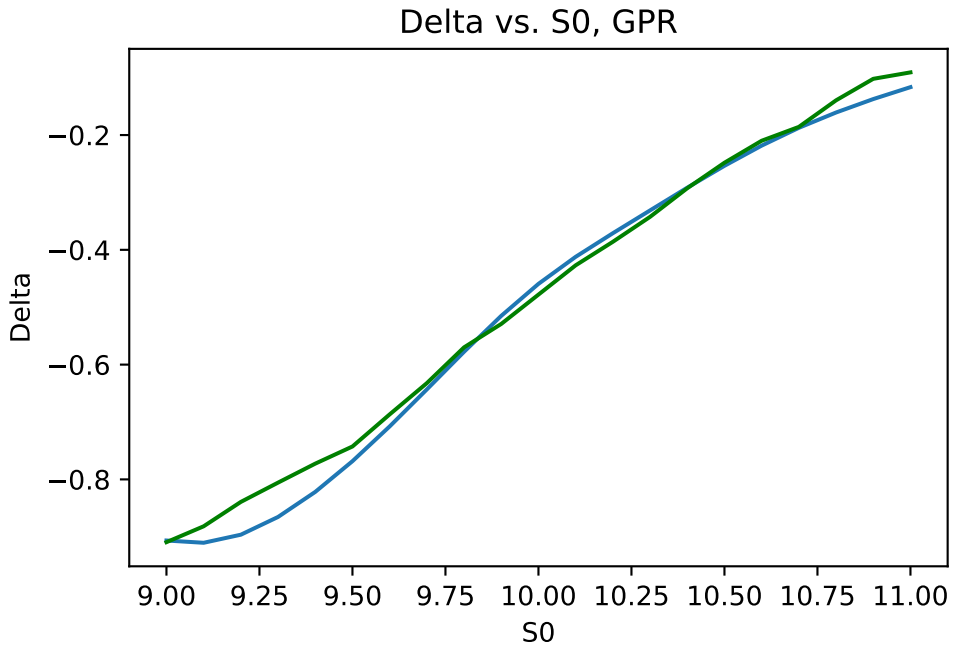
```
    # pdb.set_trace()
    delta_gp[key] = sum(matern_deriv_mult(x,s0_vals,ell)* kern_val * model.alpha_)


# for key,x in enumerate(s0_vals):
#     kern_val = model.kernel_(x,s0_vals.reshape(-1,1),eval_gradient=False).ravel() # Vector of kernel V
#     kern_coef = matern_deriv_mult(x,s0_vals,ell).ravel() #Vector ofderivative coefficents
#     d_52_k = kern_coef * kern_val #multiplied elementwise from above
#     delta_gp[key] = d_52_k @ model.alpha_ #inner product with regression coefficients


print(delta_gp)
plt.plot(s0_vals,delta_gp)
plt.plot(s0_vals,data_all["delta"],color="green")
plt.title("Delta vs. S0, GPR")
plt.xlabel("S0")
plt.ylabel("Delta")
```

```
[-0.90649153 -0.91043131 -0.89635422 -0.86543576 -0.82178816 -0.76826583
 -0.70770619 -0.64321449 -0.57764197 -0.51501676 -0.45930966 -0.41217734
 -0.3710518  -0.3313049  -0.29171082 -0.25360234 -0.21844081 -0.18741697
 -0.16071468 -0.13730817 -0.11654364]
```

```
Text(0, 0.5, 'Delta')
```



Problem 2d

```
# @jit(nopython=True)
@njit
def make_data(n):
    T,dt,M = 1/4,1/(2*365), 1000
    s0_vals = (np.random.rand(n)*2+9)*1.0#np.random.uniform(9,11,size=n)

    price = np.zeros(n)


    for i in range(n):
        Z = np.random.normal(0,1,size=(npaths,round(T/dt)))
        price[i] = np.mean(mc_samples(M,Z,s0_vals[i],T,dt))*1.0

    return(s0_vals,price)
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
epsilon = 0.005
class NN(nn.Module):
    def __init__(self, activation):
        super().__init__()
        self.layer1 = nn.Linear(1, 30)
        self.layer2 = nn.Linear(30, 1)
        self.activation = activation
    def forward(self, x):
        x = self.activation(self.layer1(x))
        x = self.layer2(x)
        return x
```

```
def train(net,x,y,epoch_num):
    mse = nn.MSELoss()
    optimizer = optim.Adam(net.parameters(),lr = 0.01)
    print("Starting Training")
    for epoch in range(epoch_num):
        optimizer.zero_grad()
        predictions = net.forward(x)
        loss = mse(predictions,y)
        loss.backward()
        optimizer.step()
        if(epoch + 1 )%(round(epoch_num/4)) == 0:
```

```
            print(f'Epoch [{epoch +1}/{epoch_num}], Loss: {loss.item()}')
```

```
torch.device('cuda')

N = 10000
epochs=10000
(x,y) = make_data(N)
x = torch.tensor(x, dtype = torch.float32).unsqueeze(1).requires_grad_()
y = torch.tensor(y, dtype = torch.float32).unsqueeze(1).requires_grad_()
```

```
net_relu = NN(nn.ReLU())
train(net_relu,x,y,epochs )
```

```
Starting Training
Epoch [2500/10000], Loss: 0.004983896389603615
Epoch [5000/10000], Loss: 0.00498389033600688
Epoch [7500/10000], Loss: 0.004983894526958466
Epoch [10000/10000], Loss: 0.00535797281190753
```

Using `tanh` instead of `relu`:

```
net_tanh = NN(nn.Tanh())
train(net_tanh,x,y,epochs)
```

```
Starting Training
Epoch [2500/10000], Loss: 0.0005425007548183203
Epoch [5000/10000], Loss: 0.0002687697997316718
Epoch [7500/10000], Loss: 0.0026879969518631697
Epoch [10000/10000], Loss: 0.0002374121395405382
```

Using `ELU`

```
net_elu = NN(nn.ELU())
train(net_elu,x,y,epochs)
```

```
Starting Training
Epoch [2500/10000], Loss: 0.0010804698104038835
Epoch [5000/10000], Loss: 0.0002780384675133973 4
Epoch [7500/10000], Loss: 0.0020050639286637306
Epoch [10000/10000], Loss: 0.0003496349963825196
```

```
s0 = torch.arange(9,11.1,0.1)
plt.plot(s0,net_relu(s0.unsqueeze(1)).detach(),color='red',label="ReLU",linewidth = 0.5)
plt.plot(s0,net_tanh(s0.unsqueeze(1)).detach(),color='blue',label="Tanh",linewidth = 0.5)
```

```
plt.plot(s0,net_elu(s0.unsqueeze(1)).detach(),color='green',label="ELU",linewidth = 0.5)
plt.legend()
plt.grid(True)
```