Hwk2

March 2, 2023

1 Math 584 - Homework 2

Diane PERES

1.0.1 Importation

[ ]:
```
!pip install yfinance
```

[25]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
from pandas_datareader import data
import scipy.optimize
from scipy.interpolate import InterpolatedUnivariateSpline
from scipy.interpolate import griddata
import time
import yfinance as yf
```

1.1 1. Back-test

In this question, you back-test dynamic trading strategies that maximize the expected power utility. (Do not forget that the riskless return is a part of the universe of available assets.) Use the power utility with $= -3$, the size $T = 100$ of each trading window (measured in days), the size $N = 1000$ of each estimation window, and the number $d = 2$ of basic assets (including the riskless asset).

[3]:
```
  = -3 #  0 & <1 => U(w) is concave
T = 100 #size of each trading window (days) T>>1
N = 1000 #size of each estimation window (days) N>T
M = 2000 #size of the sample of data (days) M > N+T
d = 2 # the number of basic assets (including the riskless asset)
R = 0.01 #riskless return annualized
  = 0.02 # Transaction costs

# Tools funciton
sign = lambda x: math.copysign(1, x)
```

[27]:
```
# Downoad the prices
start_date = '2014-01-01'
end_date = '2022-01-01'
stock_data = yf.download('^GSPC', start_date, end_date)
price = stock_data['Adj Close'].values
ret = (price[1:]-price[:-1])/price[0:-1]
days = price.shape[0]
```

```
[**********************100%**********************]  1 of 1 completed
```

1.1.1 Function

[82]:
```
def myBackTest ( , T, N, M, R,  =0, optimal_strategy = False):

  """
  Back-test function without trading cost

  Arguments:
    determine the U(w) function
  T size of each trading window (days) T>>1
  N size of each estimation window (days) N>T
  M size of the sample of data (days) M > N+T
  R riskless return annualized
    Transaction cost
  optimal_strategy : Do we compute the optimal strategy taking the transaction␣
↪cost in account ?

  Return:
  PnL List
    annualized mean return
    annualized std of return
  S annualized sharpe ratio
    optimal weight
  """

  iter = math.floor((days-N-2)/T) # = (days - N)//T # Number of iterations␣
↪necessar
  R_daily = R/250 # Daily riskless return
    = [] #Optimal weight

  # ESTIMATION
  for i in range(iter):
    estimated_ret = ret[i*T:i*T+N]
      = np.mean(estimated_ret)
      = np.var(estimated_ret)
    mu = np.array([R_daily, ])
    sigma = np.array([0,np.sqrt( )])
```

```
  obj = lambda x : - sign( ) * 0.5 * ((1+ x @ (mu+sigma))** +(1+ x @ (mu-sigma))** )
   0 = [0.5,0.5]
  constr = {'type': 'eq', 'fun': lambda x:  np.sum(x) - 1}
  opt = scipy.optimize.minimize(obj,  0, tol=1e-20, constraints=constr, options={'maxiter':1e6})
   .append(np.array(opt.x))


# TRADING
PnL = []
daily_ret = []
cum_ret = 1
A, A_riskless = 0, 1

if ( ==0):
  print('Without trading cost')
  for i in range(iter):

    alpha =  [i]
    for j in ret[i*T+N:i*T+N+T]:
        cum_ret = cum_ret*(1+alpha@np.array([R_daily,j]))
        PnL.append(cum_ret)
        daily_ret.append(alpha@np.array([R_daily,j]))

else:
  print(f'With a trading cost of  ={ }')
  for i in range(iter):
      alpha =  [i]

      for j in ret[i*T+N:i*T+N+T]:

          cum_ret = cum_ret*(1+alpha@np.array([R_daily,j]) -   * abs(alpha[0] - A_riskless ) -   * abs( alpha[1] - A ))
          PnL.append(cum_ret)
          daily_ret.append(alpha@np.array([R_daily,j]) -   * abs( alpha[0] - A_riskless ) -   * abs( alpha[1] - A ))

          #Weight befor rebalancing
          A = ( alpha[1]*(1+j)) / (1+alpha@np.array([R_daily,j]))
          A_riskless = ( alpha[0]*(1+j)) / (1+alpha@np.array([R_daily,j]))

  = np.mean(daily_ret)*250
  = np.std(daily_ret)*np.sqrt(250)
S = ( -R)/
  = [i[1] for i in  ]
```

```
return(PnL,  ,  , S,  )
```

1.1.2 (a)

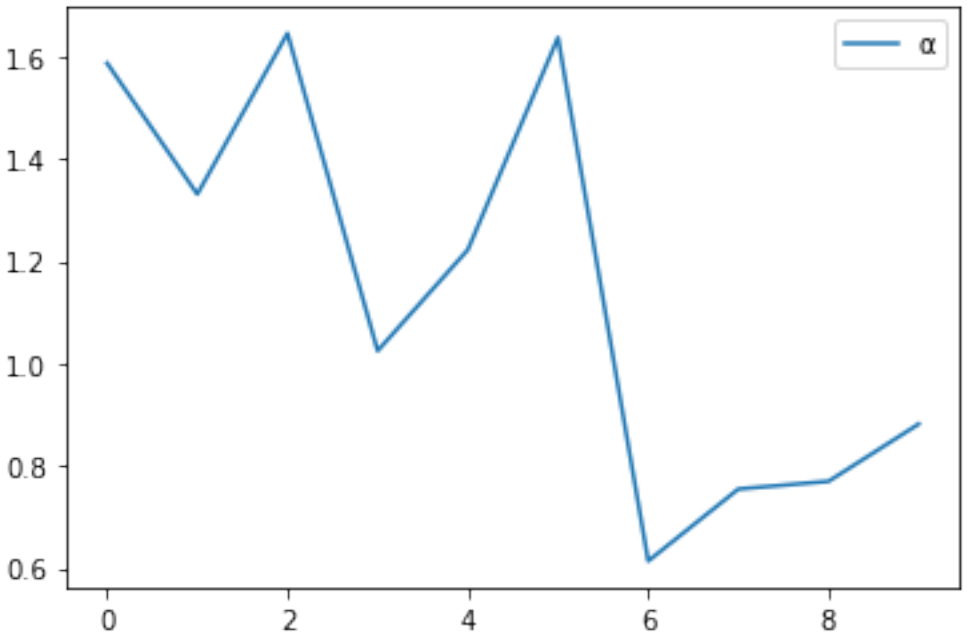
Construct and **back-test** the dynamic strategy - a portfolio - that **maximizes E(W_T( ))^ /)** over all strategies = (_0, . . . , _T−1).

The model for the risky return is: > **Rt = + _t**, > > with i.i.d. **{ _t}^T_t=1** > > s.t. **_t =** ± with the prob. 1/2,

where and should be approximated via the sample mean (^) and the sample standard deviation (^) of the returns of the risky asset (over each estimation window). Plot the **PnL** process of this strategy. Print the **annualized mean**, **standard deviation** and the **Sharpe ratio** of the returns of the generated wealth process.

[70]:
```
PnL,  ,  , S,  = myBackTest ( , T, N, M, R)

print(f'The annualized mean return is: {round( ,5)}')
print(f'The annualized std of return is: {round( ,5)}')
print(f'The annualized sharpe ratio is: {round(S,5)}')

plt.plot(PnL, label = 'PnL process of the dynamis strategy')
plt.legend()
plt.xlabel('Days')
plt.ylabel('PnL')
plt.show()

plt.plot( , label = ' ')
plt.legend()
plt.show()
```

```
Without trading cost
The annualized mean return is: 0.12654
The annualized std of return is: 0.30622
The annualized sharpe ratio is: 0.38057
```



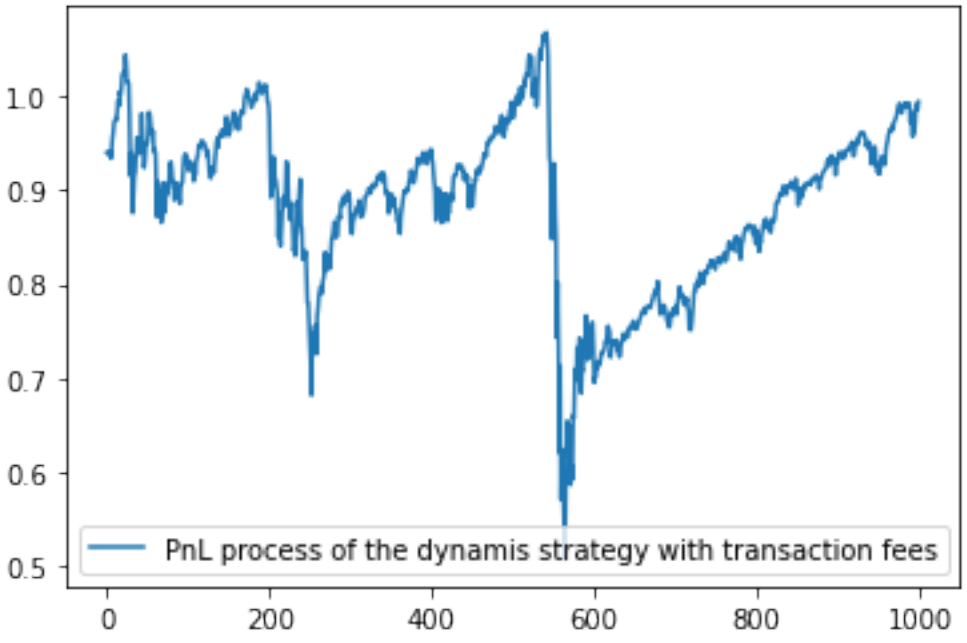
In parts b and c, it makes a difference what initial values of the "weights before rebalancing" you use at the beginning of each trading window. While other choices are possible, here in, you need to assume that the initial "weights before rebalancing" at the beginning of a trading window are given by the weights obtained at the end of the previous window.

1.1.3 (b)

Using the strategy computed in part (a), construct its PnL process in the presence of proportional transaction costs of size **= 0.02**. Plot the resulting PnL process. Print the **annualized mean**, **standard deviation** and the **Sharpe ratio** of the returns of the generated wealth process.

[83]:
```
PnL_fee,  _fee,  _fee, S_fee,  _fee = myBackTest ( , T, N, M, R,  )

print(f'The annualized mean return is: {round( _fee,5)}')
print(f'The annualized std of return is: {round( _fee,5)}')
print(f'The annualized sharpe ratio is: {round(S_fee,5)}')

plt.plot(PnL_fee, label = 'PnL process of the dynamis strategy with transaction␣
 ↪fees')
plt.legend()
plt.xlabel('Days')
plt.ylabel('PnL')
plt.show()
```

```
With a trading cost of  =0.02
The annualized mean return is: 0.04701
The annualized std of return is: 0.30934
The annualized sharpe ratio is: 0.11963
```



1.1.4 (c)

Repeat part (a) with proportional transaction costs of size = **0.02**. Note that, unlike part (b), here you need to find the **optimal strategy in the presence of transaction costs**, as opposed to re-using the strategy computed in part (a):

- To compute the value function and the feedback optimal strategy via DPP, use an **equidistant grid on [−1, 2.5]**, consisting of 100 points, for the possible values of the "weights before rebalancing".
- To compute the value function and the optimal strategy outside of the grid points, use **linear interpolation between** the grid points and **constant extrapolation outside** of [−1, 2.5].

On the very first trading day in your sample, i.e. on day N , your capital is fully invested in the riskless asset before rebalancing (i.e., right before you decide on the optimal portfolio weights to be used at that time). Plot the **PnL** process of this strategy. Print the **annualized mean**, **standard deviation** and **Sharpe ratio** of the returns of the generated wealth process.

[100]:
```
# INITIALISATION
m = 100 #grid size
grid = np.linspace(-1,2.5,m).tolist() #List of dimension mx1
iter = math.floor((days-N-2)/T) # = (days - N)//T # Number of iterations necessar
total_ =[]

for i in range(iter):
    = [] #  matrix of dimensions (T-1)xm
  v = [[(1/ ) for i in range(m)]] # v matrix of dimensions Txm

  # ESTIMATION
  estimated_ret = ret[i*T:i*T+N]
    = np.mean(estimated_ret)
    = np.var(estimated_ret)

  for t in range (T-1, 0, -1): # compute each row
    sub , subv = [], []

    # INTERPOLATION
    spl = scipy.interpolate.InterpolatedUnivariateSpline(grid,v[-1],k=1)

    # OPTIMISATION
    for i in range(m):
      a = grid[i]
      obj = lambda x : -0.5*( spl( x*(1+ + ) / (1+x*( + )) ) * (1+x*( + ) -   *abs(x-a))**  + spl( x*(1+ - ) / (1+x*( - )) ) * (1+x*( - ) -   * abs(x-a))** )
       0 = 0.5
      opt = scipy.optimize.minimize(obj,  0, tol=1e-20, bounds=((-1, 2.5),),options={'maxiter':1e6})
      sub .append([1-opt.x[0] ,opt.x[0]])
      subv.append(-opt.fun[0])
```

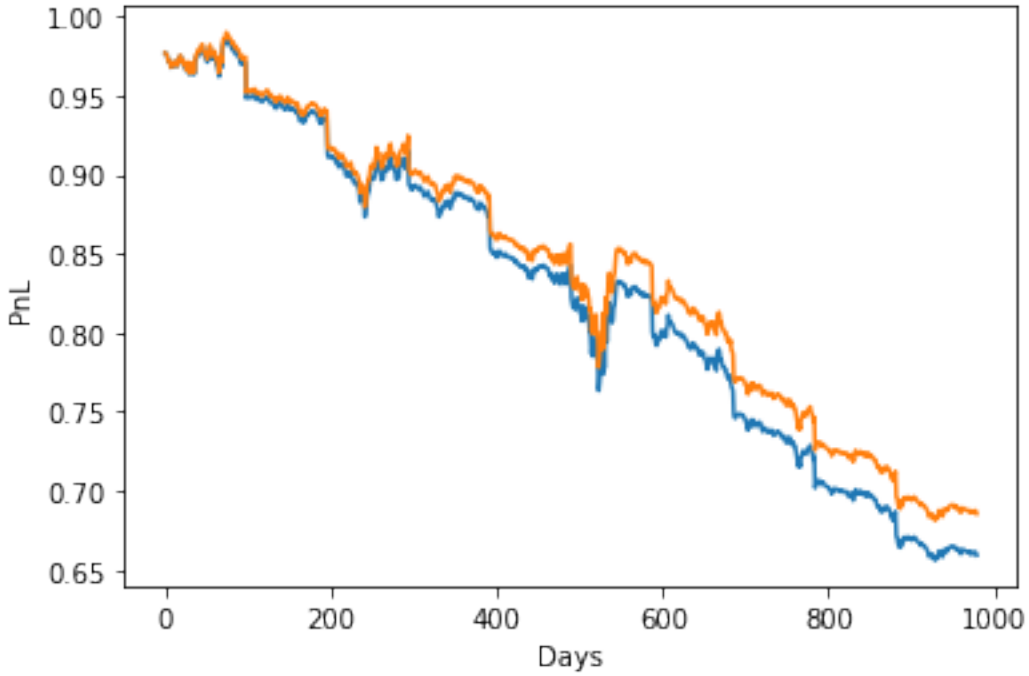
```
    .append(sub )
    v.append(subv)
  total_ .append( )
```

[101]:
```
# ALL WINDOWS
# TRADING with interpolation
R_daily = R/250 # Daily riskless return
new_A, A, cum_ret, PnL, daily_ret =[[0,1]], [[0,1]], 1, [], []

for i in range(iter):
  new_ , new_A, A = [], [[0,1]], [0]  #on day N , your capital is fully␣
 ↪invested in the riskless asset before rebalancing
  trading_ret = ret[i*T+N:i*T+N+T]
   ,  , S = [], [], []

  for t in range(T-2, 0, -1):

    sub_cum_ret, subPnL, sub_daily_ret = 1, [], []
    sub_ret = np.array([trading_ret[t]]*m)
    sub_ret_1 = np.array([1+trading_ret[t]]*m)

    #INTERPOLATION
    sub  = np.array([j[1] for j in total_ [i][t]])
    sub _riskless = np.array([j[0] for j in total_ [i][t]])

    #Weight befor interpolation & rebalancing: A_t
    A.append([(sub _riskless@sub_ret_1) /␣
 ↪(1+sub _riskless@sub_ret+sub @sub_ret), (sub @sub_ret_1) /␣
 ↪(1+sub _riskless@sub_ret+sub @sub_ret)])

    #After Ineterpolation
    new_ .append([1-spl(new_A[-1][1]),spl(new_A[-1][1])]) #Weight  2 =  * (t,A_t)

    cum_ret = cum_ret*(1+new_ [-1]@np.array([R_daily,trading_ret[t]]) -   * abs(␣
 ↪np.array(new_ [-1]) - np.array(new_A[-1]) ))
    PnL.append(cum_ret)
    daily_ret.append(new_ [-1]@np.array([R_daily,trading_ret[t]])-   * abs( np.
 ↪array(new_ [-1]) - np.array(new_A[-1]) ))

    #Add A_t+1
    new_A.append([(new_ [-1][0]*(1+trading_ret[t])) /␣
 ↪(1+new_ [-1][0]*R_daily+new_ [-1][0]*trading_ret[t]),(new_ [-1][1]*(1+trading_ret[t]))␣
 ↪/ (1+new_ [-1][0]*R_daily+new_ [-1][0]*trading_ret[t])]) #Weight b4␣
 ↪rebalancing A_t*

 = np.mean(daily_ret)*250
```

```
  = np.std(daily_ret)*np.sqrt(250)
S = ( -R)/
```

[106]:
```
print(f'The annualized mean return for each value of a are: { }')
print(f'The annualized std of return for each value of a are: { }')
print(f'The annualized sharpe ratio for each value of a are: {S}')

plt.plot(PnL)
plt.xlabel('Days')
plt.ylabel('PnL')
plt.show()
```

```
The annualized mean return for each value of a are: -0.09920478803188584
The annualized std of return for each value of a are: 0.06452898231213994
The annualized sharpe ratio for each value of a are: -1.6923370572255403
```



2 Download as PDF

[ ]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[ ]:
```
!jupyter nbconvert --to pdf /content/Hwk2.ipynb
```