Appendix [Code]

```
Black-Scholes Hedging
Asset price model Underlying asset prices S are modeled by a GBM process.

dS(t) = μS(t)dt + σS(t)dWt (1)

The risk free interest rate is r.

Consider a single asset following the GBM dynamics (see Eq 1). Simulate the
option values and the value of a delta hedge set up to replicate the option.
Plot the error in the PnL for the option trader when the option is not hedged
(absence of replicating portfolio), and when the option is hedged. Show how
the hedge error would depend upon the time step used in the simulation. The
paramter values for the GBM process are

r = 0.06, T = 1, σ = 0.2, payoff is g(ST ) = max(K − ST , 0), K = 40, S0 = 36.
```

1. **Asset Pricing Model**

The underlying asset prices are modelled using the Geometric Brownian Motion. It is given by the equation- dS(t) = μS(t)dt + σS(t)dWt (1) Solving the above equation by Ito's calculas, we get- ST=Ste(r−12σ2)(dt)+σdt√Z Where z ~ N(0,1)

We can know generate paths using the equation of St.

2. **Monte Carlo Simulation**

   1. **Black Scholes Formula**

$S_T = S_t e^{(r-\frac{1}{2}\sigma^2)(dt)+\sigma\sqrt{dt}Z}$
Where
z ~ N(0,1)

In [367…
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed = 2020
```

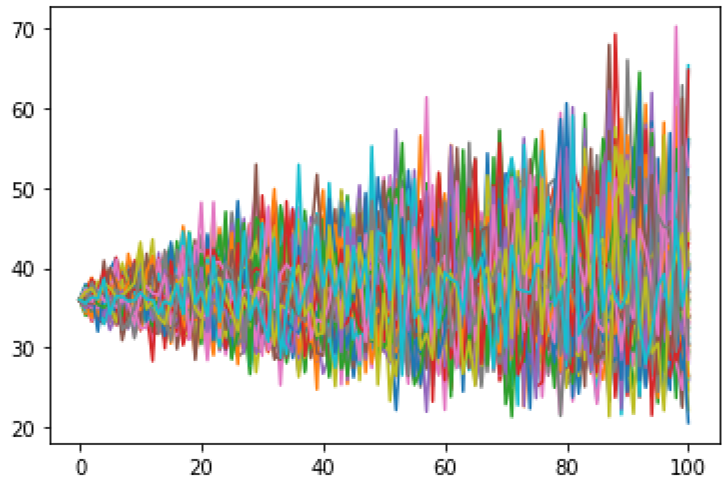
Monte Carlo simulation

In [368…
```
r = 0.06; T =1; sigma = 0.2; K = 40; S0 = 36; Nsteps = 100; Nrepl = 100
```

In [369…
```
def generate_asset_price(S,v,r,dt):
    return S * np.exp((r - 0.5 * v**2) * dt + np.multiply(v * np.sqrt(dt) , np.random.normal(0,1.0
```

In [370…
```
dt = np.array([np.array([i/Nsteps for i in range(Nsteps+1)]) for i in range(Nrepl)]) #Time to matu
S_T = generate_asset_price(36,sigma,r,dt)
```

In [371…
```
for i in range(len(S_T)):
  plt.plot(S_T[i])
plt.show()
```



In [372…
```
put_option_price_est = np.mean([np.exp(-r*T)*max(0,i) for i in np.array(K-S_T[:,-1])])
std_err = np.std([np.exp(-r*T)*max(0,i) for i in np.array(K-S_T[:,-1])])/Nrepl

put_option_value_MC = [[np.exp(-r*t_)*max(0,K-j) for j,t_ in zip(i,t)] for i,t in zip(S_T,dt)] #wr
print("""
The Put Option price estimate is : {} with standard error : {} """.format(put_option_price_est,std
```

```
The Put Option price estimate is : 4.570739859708868 with standard error : 0.04648218986365398
```

In [373…
```
call_option_price_est = np.mean([np.exp(-r*T)*max(0,i) for i in np.array(S_T[:,-1] -K)])
std_err = np.std([np.exp(-r*T)*max(0,i) for i in np.array(S_T[:,-1]-K)])/Nrepl

call_option_value_MC = [[np.exp(-r*t_)*max(0,j-K) for j,t_ in zip(i,t)] for i,t in zip(S_T,dt)]

print("""
The Call Option price estimate is : {} with standard error : {} """.format(call_option_price_est,s
```

```
The Call Option price estimate is : 1.7857003855382172 with standard error : 0.04578562289385697
```

Black Scholes formula based

$$C = S_t N(d_1) - Ke^{-rt} N(d_2)$$

where:

$$d_1 = \frac{ln\frac{S_t}{K} + (r + \frac{\sigma_v^2}{2})\,t}{\sigma_s\,\sqrt{t}}$$

and

$$d_2 = d_1 - \sigma_s\,\sqrt{t}$$

-

In [374…
```
dt = np.array([np.array([i/Nsteps for i in reversed(range(Nsteps+1))]) for i in range(Nrepl)]) #Ti
```

In [375…
```
from scipy.stats import norm
def d1(S,K,t,r,sigma):
  return ((np.log(S/K) + (r+(sigma**2/2))*t) / (sigma*np.sqrt(t)))
def d2(d1,sigma,t):
  return (d1 - sigma*np.sqrt(t))
def call(S,d1,K,r,t,d2):
  return (S*norm.cdf(d1) - K*np.exp(-r*t)*norm.cdf(d2))
```

```
def put(S,d1,K,r,t,d2):
  return (K*np.exp(-r*t)-S + call(S,d1,K,r,t,d2))
```

In [376…
```
r = 0.06; t =1; sigma = 0.2; K = 40; S = 36;
```

In [377…
```
d1_ = d1(S,K,t,r,sigma)
d2_ = d2(d1_,sigma,t)
put_option_price_est_BS = put(S,d1_,K,r,t,d2_)
call_option_price_est_BS = call(S,d1_,K,r,t,d2_)
print('Option price for put option is :',put_option_price_est_BS,'\nOption price for call option i
```

```
Option price for put option is : 3.8443077915968384
Option price for call option is : 2.1737264482268923
```

Option price for every time step and simulated delta

In [378…
```
import pandas as pd
```

In [379…
```
def diff_1(X):
  prev = 0
  a = []
  for i in X:
    a.append(i-prev)
    prev = i
  return a
```

In [380…
```
d1_ = d1(S_T,K,dt,r,sigma)
d2_ = d2(d1_,sigma,dt)
put_option_value_BS = put(S_T,d1_,K,r,dt,d2_)
call_option_value_BS = call(S_T,d1_,K,r,dt,d2_)
call_delta = norm.cdf(d1_)
put_delta = call_delta - 1
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: divide by zero enc
ountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [381…
```
call_option_value_BS[0]
```

Out[381…
```
array([2.17372645e+00, 1.92538009e+00, 2.08007675e+00, 1.97087731e+00,
       1.76738089e+00, 2.06176907e+00, 1.50336254e+00, 2.75938525e+00,
       9.67598822e-01, 2.42158114e+00, 1.37543574e+00, 2.48000220e+00,
       5.04807293e+00, 3.55584429e+00, 2.73310444e+00, 3.11655593e+00,
       5.93656623e-01, 1.50268003e+00, 1.21281670e+00, 6.95144618e-01,
       1.69670569e+00, 9.59905080e-01, 1.14748055e+00, 1.36646544e+00,
       2.93249305e+00, 7.93899116e-01, 1.12997335e+00, 1.18510798e+00,
       9.16313800e-01, 8.13375459e-01, 4.73296699e+00, 2.67483396e+00,
       1.56638137e+00, 2.37004311e+00, 2.61988374e+00, 2.40549760e+00,
       1.79291942e-01, 4.01778016e+00, 2.70369051e+00, 2.78336794e-01,
       1.82698282e+00, 1.38714366e+00, 2.24179569e+00, 1.13031740e+00,
       1.13038274e+00, 1.94518995e+00, 4.94836224e+00, 3.39056036e+00,
       2.20802794e+00, 4.60269232e+00, 1.58583621e+00, 2.60476522e-01,
       4.98097108e-02, 1.47958873e+00, 4.90598372e+00, 2.47669938e+00,
       1.57973364e+00, 9.34649656e-01, 4.27010912e+00, 3.02632994e+00,
       1.52711269e-01, 4.01433800e-01, 2.87696959e-03, 4.83868008e-02,
       2.95593734e-01, 2.40605857e-01, 2.88769884e+00, 7.52111751e-02,
       1.71110679e+00, 5.01972342e-03, 1.40014848e+00, 3.37219716e-04,
       6.07483682e-01, 1.94309395e+00, 4.54964401e-02, 1.45381158e+00,
       4.21549353e-01, 4.28527526e-02, 1.48466428e+00, 4.48968205e-01,
       6.14065857e+00, 2.97799341e-01, 2.67760678e-02, 9.45287120e-04,
       2.15171896e+00, 9.08667093e-03, 1.78024312e+00, 4.70614146e-02,
       2.17498419e+00, 1.19852919e+01, 6.54433039e-02, 1.07152187e-06,
```

```
        2.51201790e-04, 1.02199609e+01, 1.26700006e+00, 5.45632516e-04,
        2.85920201e-07, 2.72774763e-12, 1.18011478e-04, 3.25003652e-09,
        0.00000000e+00])
```

In [382…

```
pd.DataFrame(np.array([dt[0],S_T[1],call_option_value_BS[1],call_delta[1]]))
```

Out[382…

	0	1	2	3	4	5	6	7	8	9	
0	1.000000	0.990000	0.980000	0.970000	0.960000	0.950000	0.940000	0.930000	0.920000	0.910000	0.
1	36.000000	37.155778	38.759885	36.395048	34.678719	36.659394	35.737266	36.888387	37.037431	39.540395	34.
2	2.173726	2.705167	3.565923	2.285328	1.547582	2.362416	1.923958	2.424108	2.472377	3.848922	1.
3	0.449548	0.510904	0.593638	0.465921	0.368222	0.477072	0.423356	0.486387	0.493033	0.625896	0.

4 rows × 101 columns

In [383…

```
dt
```

Out[383…

```
array([[1.  , 0.99, 0.98, ..., 0.02, 0.01, 0.  ],
       [1.  , 0.99, 0.98, ..., 0.02, 0.01, 0.  ],
       [1.  , 0.99, 0.98, ..., 0.02, 0.01, 0.  ],
       ...,
       [1.  , 0.99, 0.98, ..., 0.02, 0.01, 0.  ],
       [1.  , 0.99, 0.98, ..., 0.02, 0.01, 0.  ],
       [1.  , 0.99, 0.98, ..., 0.02, 0.01, 0.  ]])
```

PnL Error

Plot the error in the PnL for the option trader when the option is not hedged (absence of replicating portfolio), and when the option is hedged. Show how the hedge error would depend upon the time step used in the simulation. The paramter values for the GBM process are

No hedge

In [384…

```
print(put_option_price_est,call_option_price_est)
```

```
4.570739859708868 1.7857003855382172
```

Put option

In [385…

```
interest_income = np.exp(r*1) * put_option_price_est_BS
interest_income
```

Out[385…

```
4.082026509286605
```

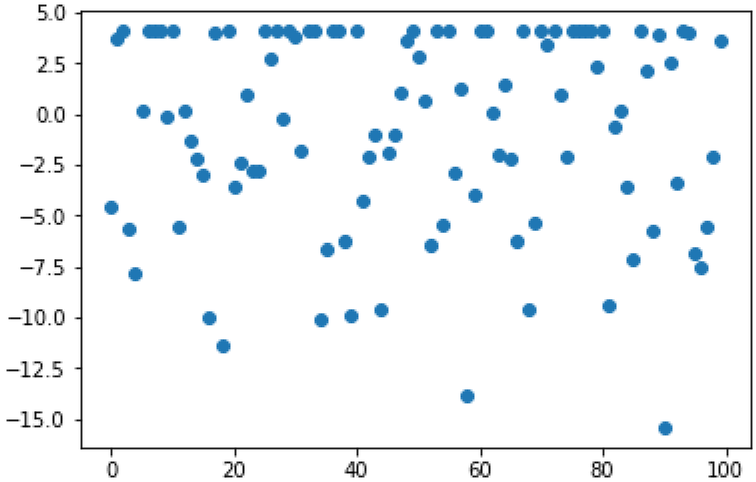
In [386…

```
Epnl_nohedge = np.mean([interest_income - max(0,K-i) for i in S_T[:,-1]])
Sd_pnl = np.std([interest_income - max(0,K-i) for i in S_T[:,-1]])
stderror = Sd_pnl/np.sqrt(Nrepl)

print("Expected Pnl with no hedge :{} \nStandard error :{}".format(Epnl_nohedge,stderror))
y = [interest_income - max(0,K-i) for i in S_T[:,-1]]
plt.scatter(list(range(len(y))),y)
plt.show()
```

```
Expected Pnl with no hedge :-0.7713521185038801
```

Standard error :0.4935648796068808



Call option

In [387…

```
interest_income = np.exp(r*1) * call_option_price_est_BS
interest_income
```

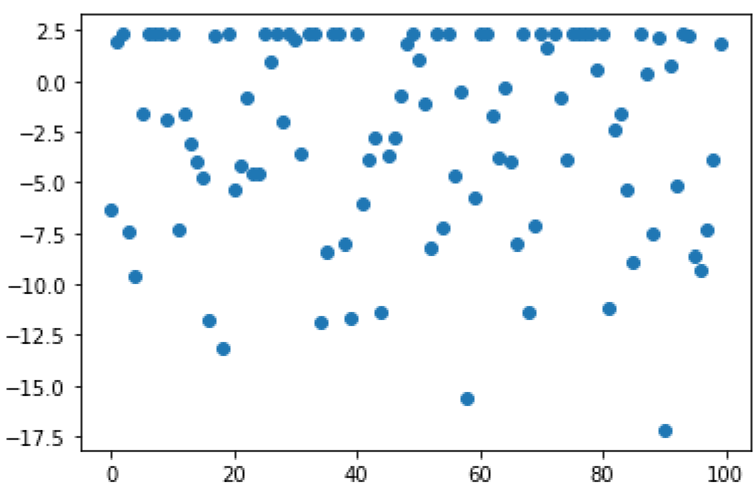
Out[387… 2.308142184919554

In [388…

```
Epnl_nohedge = np.mean([interest_income - max(0,i-K) for i in S_T[:,-1]])
Sd_pnl = np.std([interest_income - max(0,i-K) for i in S_T[:,-1]])
stderror = Sd_pnl/np.sqrt(Nrepl)

print("Expected Pnl with no hedge :{} \nStandard error :{}".format(Epnl_nohedge,stderror))
y = [interest_income - max(0,K-i) for i in S_T[:,-1]]
plt.scatter(list(range(len(y))),y)
plt.show()
```

Expected Pnl with no hedge :0.4120202543749361
Standard error :0.4861684769504125



With Delta Hedging PnL

Call option

In [389…

```
def portfolio_Value(stock, delta, option_value,rf_rate):
  dt = 1/(stock.shape[1]-1)

  stock_price = stock[:,1:]/stock[:,0:-1]

  dollars_in_stock = np.zeros(shape = stock.shape)
```

```
    dollars_in_stock[:,0] = stock[:,0]*delta[:,0]

    interest = np.zeros(shape = stock.shape)
    interest[:,0] = dollars_in_stock[:,0] - option_value[:,0]


    for i in range(1,dollars_in_stock.shape[1]):
      dollars_in_stock[:,i] = dollars_in_stock[:,i-1]*stock_price[:,i-1]
      interest[:,i] = interest[:,i-1]*np.exp(rf_rate*dt)

    portfolio_value = (dollars_in_stock - interest)

    return portfolio_value
```

In [390…

```
portfolio_value = portfolio_Value(S_T, call_delta, call_option_value_BS,r)
```

In [391…

```
#Showing for 1 path
d = {}
d['stock'] = S_T[0]
d['option_Value'] = call_option_value_BS[0]
d['delta'] = norm.cdf(d1(S_T[0],K,dt[0],r,sigma))
d= pd.DataFrame(d)
d['delta_s'] = np.multiply(d.delta,d.stock)
d['Portfolio_value'] = portfolio_value[0]
d['PnL_error'] = d.Portfolio_value - d.option_Value
d
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: divide by zero enc
ountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[391…

	stock	option_Value	delta	delta_s	Portfolio_value	PnL_error
0	36.000000	2.173726e+00	4.495483e-01	1.618374e+01	2.173726	0.000000
1	35.479185	1.925380e+00	4.189062e-01	1.486245e+01	1.931186	0.005806
2	35.890994	2.080077e+00	4.397955e-01	1.578470e+01	2.107901	0.027824
3	35.690530	1.970877e+00	4.267075e-01	1.522942e+01	2.009364	0.038487
4	35.250896	1.767381e+00	4.001127e-01	1.410433e+01	1.803304	0.035923
...	...	...	...	...	...	...
96	32.997330	2.859202e-07	1.115320e-06	3.680259e-05	-0.006789	-0.006790
97	31.719954	2.727748e-12	1.723807e-11	5.467907e-10	-0.589939	-0.589939
98	36.351168	1.180115e-04	4.416483e-04	1.605443e-02	1.483103	1.482985
99	35.846001	3.250037e-09	2.628792e-08	9.423168e-07	1.247089	1.247089
100	31.352097	0.000000e+00	0.000000e+00	0.000000e+00	-0.782062	-0.782062

101 rows × 6 columns

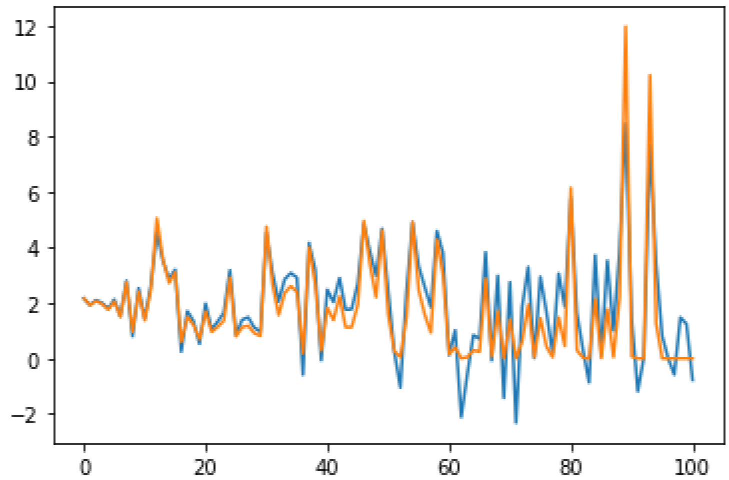
In [392…

```
plt.plot(d.Portfolio_value)
plt.plot(d.option_Value)
```

Out[392…

```
[<matplotlib.lines.Line2D at 0x7ff80f6672e8>]
```

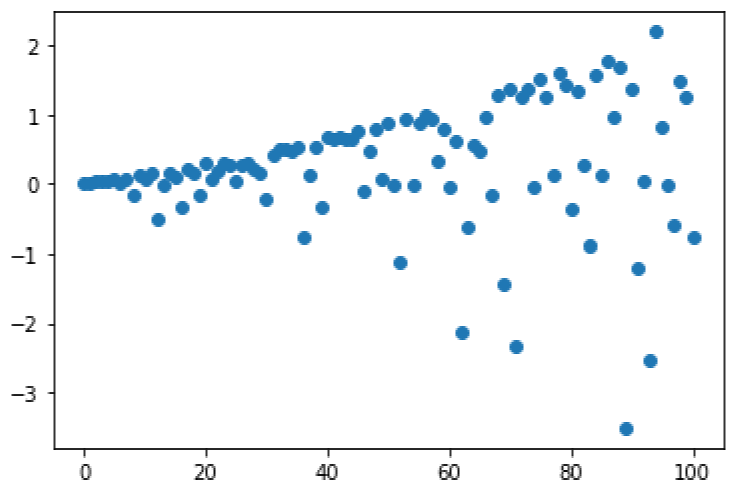
/



In [393…

```
plt.scatter(list(range(len(d.PnL_error))),d.PnL_error)
```

Out[393… <matplotlib.collections.PathCollection at 0x7ff80f5f4ef0>



In [400…

```
np.mean(d.PnL_error)
```

Out[400… 0.25299628575249744

Put option

In [395…

```
portfolio_value = portfolio_Value(S_T, put_delta, put_option_value_BS,r)
```

In [396…

```
#Showing for 1 path
d = {}
d['stock'] = S_T[0]
d['option_Value'] = put_option_value_BS[0]
d['delta'] = norm.cdf(d1(S_T[0],K,dt[0],r,sigma)) -1
d= pd.DataFrame(d)
d['delta_s'] = np.multiply(d.delta,d.stock)
d['Portfolio_value'] = portfolio_value[0]
d['PnL_error'] = d.Portfolio_value - d.option_Value
d
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: divide by zero enc
ountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[396…

	stock	option_Value	delta	delta_s	Portfolio_value	PnL_error
0	36.000000	3.844308	-0.550452	-19.816260	3.844308	-1.776357e-15

	stock	option_Value	delta	delta_s	Portfolio_value	PnL_error
1	35.479185	4.139386	-0.581094	-20.616733	4.145192	5.806271e-03
2	35.890994	3.904896	-0.560205	-20.106297	3.932720	2.782426e-02
3	35.690530	4.018797	-0.573292	-20.461113	4.057283	3.848675e-02
4	35.250896	4.277584	-0.599887	-21.146563	4.313507	3.592262e-02
...	...	...	...	...	...	...
96	32.997330	6.906786	-0.999999	-32.997293	6.899996	-6.789738e-03
97	31.719954	8.208111	-1.000000	-31.719954	7.618172	-5.899386e-01
98	36.351168	3.600979	-0.999558	-36.335113	5.083965	1.482985e+00
99	35.846001	4.130006	-1.000000	-35.846000	5.377095	1.247089e+00
100	31.352097	8.647903	-1.000000	-31.352097	7.865842	-7.820615e-01

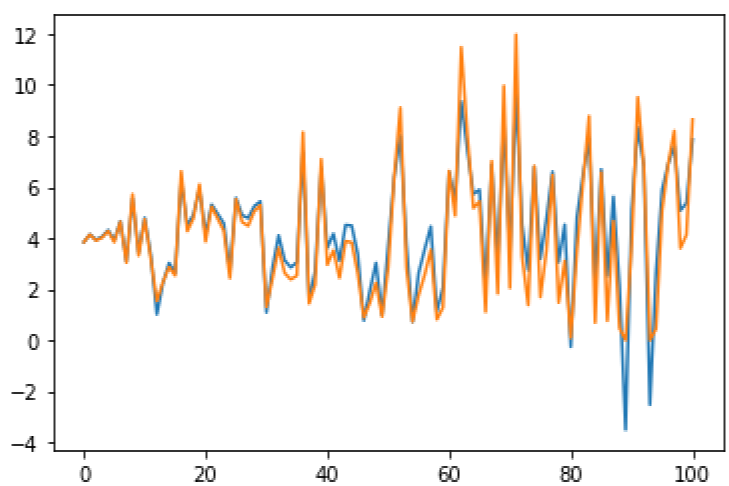
101 rows × 6 columns

In [397…

```
plt.plot(d.Portfolio_value)
plt.plot(d.option_Value)
```

Out[397…

```
[<matplotlib.lines.Line2D at 0x7ff80f667630>]
```

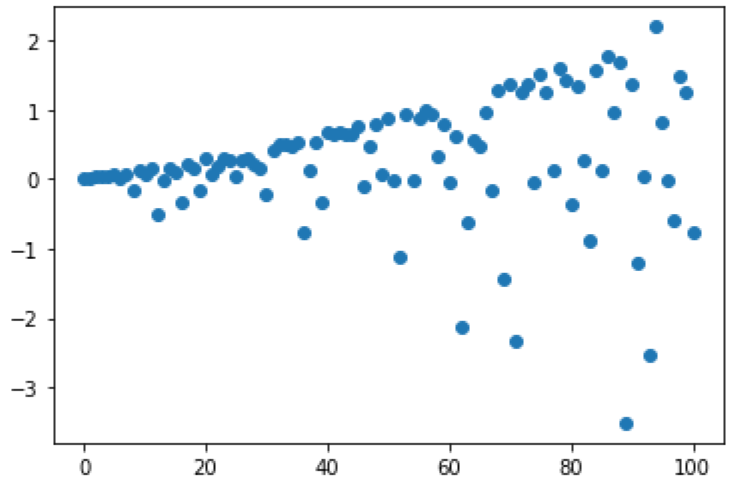


In [398…

```
plt.scatter(list(range(len(d.PnL_error))),d.PnL_error)
```

Out[398…

```
<matplotlib.collections.PathCollection at 0x7ff80f53eef0>
```



2
1
0
−1
−2
−3
0
20
40
60
80
100