In [6]:

```python
import cvxpy as cp
import pandas as pd
import numpy as np
import yfinance
import calendar as cd
import matplotlib.pyplot as plt
import scipy
import tensorflow as tf
from sklearn import preprocessing
import keras
from keras.models import Model
from keras.layers import Dense, Dropout, LSTM, Input, Activation, concatenate
from keras import optimizers
from keras.utils.vis_utils import plot_model

%matplotlib inline
```

Importation des données

In [7]:

```python
data_history = pd.read_csv('data.csv')

#Replace NaN with stock prices if needed (no replacement in some cases)
for i in range(len(data_history)-1):
  for column in data_history.columns:
    if str(data_history.at[i,column]) == 'nan' and str(data_history.at[i+1,column]) != '
nan':
      try:
        data_history.at[i,column] = data_history.at[i-1,column]
      except:
        pass

pd.isnull(data_history).sum()
data_history.set_index("Date", inplace = True)
```

In [8]:

```python
dh_bool = data_history.isnull().any()
data_441 = data_history[dh_bool[dh_bool == False].index] # Remove columns with at least
one NaN
var_data = (data_441.diff().iloc[1:])/data_441.shift(1).iloc[1:]

var_data23 = (data_441.diff(23).iloc[23:])/data_441.shift(23).iloc[23:]
var_data23
```

Out[8]:

Date	AMZN	AES	IBM	AMD	ADBE	APD	BXP	ALL	HON	AA	AMGN	HES
2003-02-05	0.132856	-0.058642	-0.041082	-0.282454	0.036427	-0.028673	-0.025815	-0.094046	-0.037630	-0.150945	0.039538	-0.169865
2003-02-06	0.076511	-0.055072	-0.048858	-0.269452	0.008842	-0.029419	-0.029500	-0.158033	-0.030303	-0.175955	0.042692	-0.17700
2003-02-07	0.035749	-0.133333	-0.075847	-0.296089	-0.049351	-0.070016	-0.034250	-0.183684	-0.070731	-0.190185	0.036204	-0.19702
2003-02-10	-0.006961	-0.192879	-0.092308	-0.278940	-0.065942	-0.065906	-0.028340	-0.158604	-0.060437	-0.181504	0.054538	-0.180902
2003-02-11	-0.011418	-0.116418	-0.078981	-0.240658	-0.015850	-0.049142	-0.023186	-0.162011	-0.051021	-0.086269	0.093494	-0.17927

...	...	...	...	...	...	...	...	...	...	...	...	..
Date	AMZN	AES	IBM	AMD	ADBE	APD	BXP	ALL	HON	AA	AMGN	HES
2021-02-23	-0.021107	-0.022120	-0.059632	-0.045183	-0.004109	-0.081065	0.081018	-0.026937	-0.014610	0.090193	-0.072006	0.096370
2021-02-24	-0.044590	0.033188	-0.051603	-0.050147	0.009745	-0.068469	0.138182	-0.021315	0.028261	0.359320	-0.080665	0.133257
2021-02-25	-0.071401	-0.021238	0.046341	-0.111758	-0.028109	-0.081876	0.125082	-0.014489	0.017041	0.317526	-0.095812	0.117303
2021-02-26	-0.061041	0.019285	0.016354	-0.102199	-0.029085	-0.083865	0.085999	-0.039207	0.006815	0.269390	-0.118660	0.128854
2021-03-01	-0.063408	0.099659	0.001739	-0.109533	-0.023747	-0.061369	0.086173	-0.004839	0.031603	0.382543	-0.115981	0.161217

4553 rows × 441 columns

In [9]:

```
MarketCaps = pd.read_excel("DataProjets.xlsx",sheet_name = "MarketCaps")
mapping = pd.read_excel("DataProjets.xlsx",sheet_name = "Mapping")[["Sedol","Tickers"]]
```

In [10]:

```
origin_index = list(var_data.columns)
dic_s_to_t = {}
for i in range(len(mapping)):
  l = mapping.iloc[i]
  dic_s_to_t[l['Sedol']] = l['Tickers']

dic_s_to_t['Unnamed: 0'] = 'Date' # No name for this column in the Excel file
MarketCaps = MarketCaps.rename(columns = dic_s_to_t)
MarketCaps441 = MarketCaps[['Date']+origin_index]

# Company went private on May 31 2010 but market caps still give a number
MarketCaps441.loc[MarketCaps441['Date'] == pd.Timestamp('2010-05-31 00:00:00'),'Date'] =
pd.Timestamp('2010-05-28 00:00:00')
MarketCaps441.loc[MarketCaps441['Date'] == pd.Timestamp('2013-03-29 00:00:00'),'Date'] =
pd.Timestamp('2013-03-28 00:00:00')
MarketCaps441.loc[MarketCaps441['Date'] == pd.Timestamp('2018-03-30 00:00:00'),'Date'] =
pd.Timestamp('2018-03-29 00:00:00')
MarketCaps441.set_index("Date", inplace = True)
list_dates = MarketCaps['Date']
MarketCaps441 = (MarketCaps441.T/MarketCaps441.T.sum()).T # Renormalize each line because
we remove columns with NaN so sum is no longer 1
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1763: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  isetter(loc, value)
```

In [11]:

```
last_day_month = [] # Indices of the last day of each month in vardata

current_month = "01"
date2 = 0
for i,date in enumerate(var_data23.index):
  date1,date2 = date2,date
  if date[5:7] != current_month:
    current_month = date[5:7]
    if i > 506:
      last_day_month.append((i-1,date1))

last_day_month = last_day_month[:-2] # Remove last indices because market caps end earlie
r
```

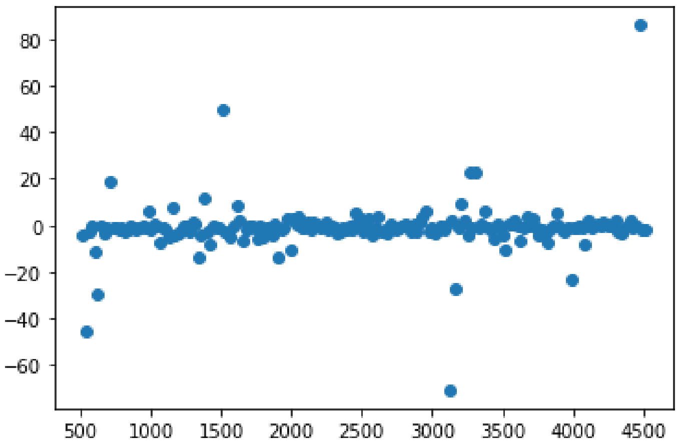
Méthodes d'optimisation

In [12]:

```
# Estimating parameters

def estimate_mu(t):
  return var_data23.iloc[t+1-46:t+1].mean()

def estimate_sigma(t):
  return (var_data23.iloc[t+1-2*23:t+1].cov())

def real_mu(t):
  return var_data23.iloc[t+23]
```

In [13]:

```
real_mus = [real_mu(t).mean() for t,a in last_day_month]
estimate_mus = [estimate_mu(t).mean() for t,a in last_day_month]
x = [t for t,a in last_day_month]
ecart = [((real_mu(t)-estimate_mu(t))/estimate_mu(t)).mean() for t,a in last_day_month]
plt.scatter(x,ecart)
plt.show()
```



In [15]:

```
# Estimating Omega (matrix of estimation errors on mu)
# We will assume that the errors are not correlated
# We could set Omega = diag(Sigma) but there is a (small) difference between the variance
of mu and the variance of the estimation errors on mu

def estimate_omega(t):
  errors_mu = pd.DataFrame(columns = origin_index)
  for i in range(24,2*260):
    errors_mu.loc[i] = (real_mu(t-i)-estimate_mu(t-i))
  Omega = errors_mu.cov()/25
  return Omega
```

Fonction générique d'optimisation de portefeuille (qu'il suffit de légèrement adapter en fonction des cas)

In [17]:

```
wfinal = pd.DataFrame(columns = origin_index) # Initializing DataFrame

for indice_day,date_day in last_day_month:

  # Choosing the data we want to work with:
  n_top = 441 # Number of companies with the highest capitalizations
  marketcap_sorted = MarketCaps441.loc[date_day].T.sort_values(ascending=False).iloc[:n_t
op]
  index_top = list(marketcap_sorted.index)
  index_top = [x for x in origin_index if x in index_top]
  index_last = [[x for x in origin_index if x not in index_top]]
```

```
    # Parameters
    w = cp.Variable(n_top)
    sig = 0.3
    k = 0.2
    lam = 10 # The greater lambda is, the lowest vol will be

    # Estimation of the other parameters
    sigma = np.array(estimate_sigma(indice_day).loc[index_top][index_top])
    mu = np.array(estimate_mu(indice_day)[index_top])

    # Defining the problem
    objective = cp.Maximize(cp.matmul(mu.T,w)) # (cp.quad_form(w,sigma))
    constraints = [w >= 0,cp.sum(w) == 1, cp.quad_form(w,sigma) <= sig**2]
    prob = cp.Problem(objective, constraints)

    # Solving the problem
    prob.solve()
    w_df = pd.DataFrame(list(w.value), index = index_top, columns = [date_day]).T

    # Exporting solution
    for new_index in index_last:
        w_df[new_index] = 0

    w_df = w_df[origin_index] # w_df columns' order changed
    wfinal = wfinal.append(round(w_df,5))
    #print(date_day)
```

Machine Learning

Modèle de base

In [18]:

```
df = data_441['VZ']
df2 = df.reset_index()
df2 = df2['VZ'].to_numpy().reshape(-1,1)

data_normaliser = preprocessing.MinMaxScaler()
data_normalised = data_normaliser.fit_transform(df2)

history_points = 3

ohlcv_histories_normalised = np.array([data_normalised[i  : i + history_points].copy() f
or i in range(len(data_normalised) - history_points)])
next_day_open_values_normalised = np.array([data_normalised[:,0][i + history_points].cop
y() for i in range(len(data_normalised) - history_points)])
next_day_open_values_normalised = np.expand_dims(next_day_open_values_normalised, -1)

next_day_open_values = np.array([df2[:,0][i + history_points].copy() for i in range(len(
df2) - history_points)])
next_day_open_values = np.expand_dims(next_day_open_values, -1)
y_normaliser = preprocessing.MinMaxScaler()
next_day_open_values = y_normaliser.fit_transform(next_day_open_values.reshape(next_day_o
pen_values.shape[0],1))

test_split = 0.9 # Percent of data to be used for testing
n = int(ohlcv_histories_normalised.shape[0] * test_split)

# Splitting the dataset up into train and test sets
ohlcv_train = ohlcv_histories_normalised[:n]
y_train = next_day_open_values[:n]

ohlcv_test = ohlcv_histories_normalised[n:]
y_test = next_day_open_values[n:]
```

In [28]:

```
lstm_input = Input(shape = (history_points, 1), name = 'lstm_input')
```

```
x = LSTM(50, name = 'lstm_0')(lstm_input)
x = Dropout(0.2, name = 'lstm_dropout_0')(x)
x = Dense(64, name = 'dense_0')(x)
x = Activation('sigmoid', name = 'sigmoid_0')(x)
x = Dense(1, name = 'dense_1')(x)
output = Activation('linear', name = 'linear_output')(x)
model = Model(inputs = lstm_input, outputs = output)

adam = optimizers.Adam(lr = 0.0005)

model.compile(optimizer = adam, loss='mse')
plot_model(model, show_shapes = True,show_layer_names = True)

model.fit(x = ohlcv_train, y = y_train, batch_size = 32, epochs = 30, shuffle = True, va
lidation_split = 0.1, verbose = 0)
evaluation = model.evaluate(ohlcv_test, y_test)
```

```
15/15 [==============================] - 0s 1ms/step - loss: 7.5306e-04
```

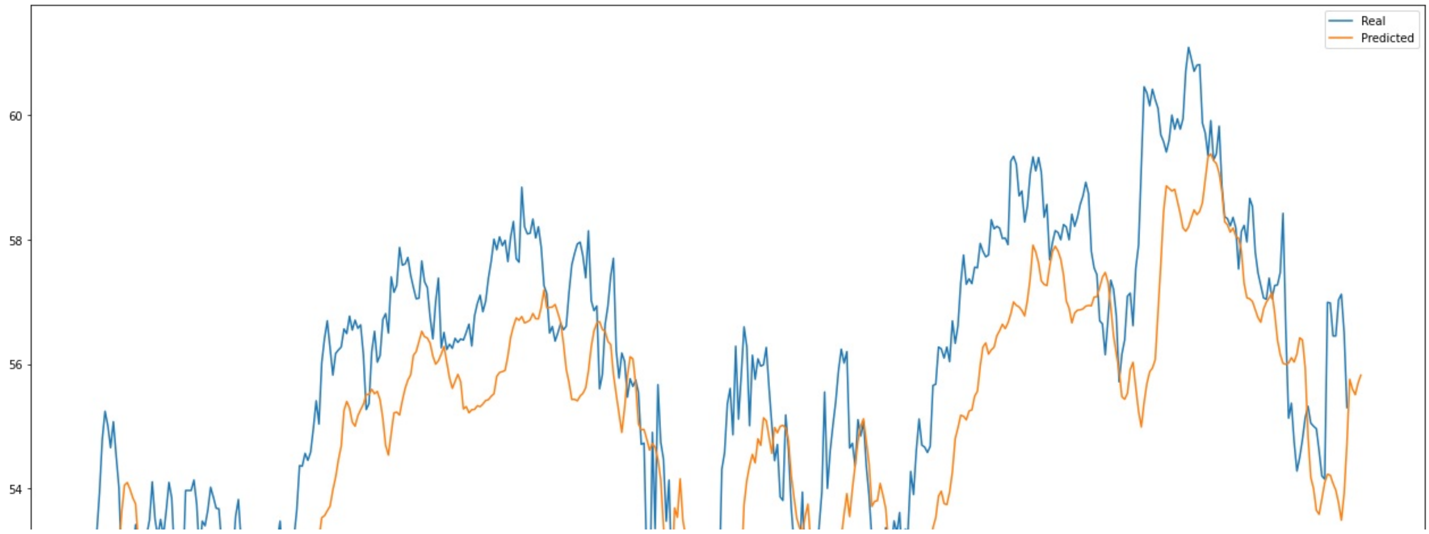
In [29]:

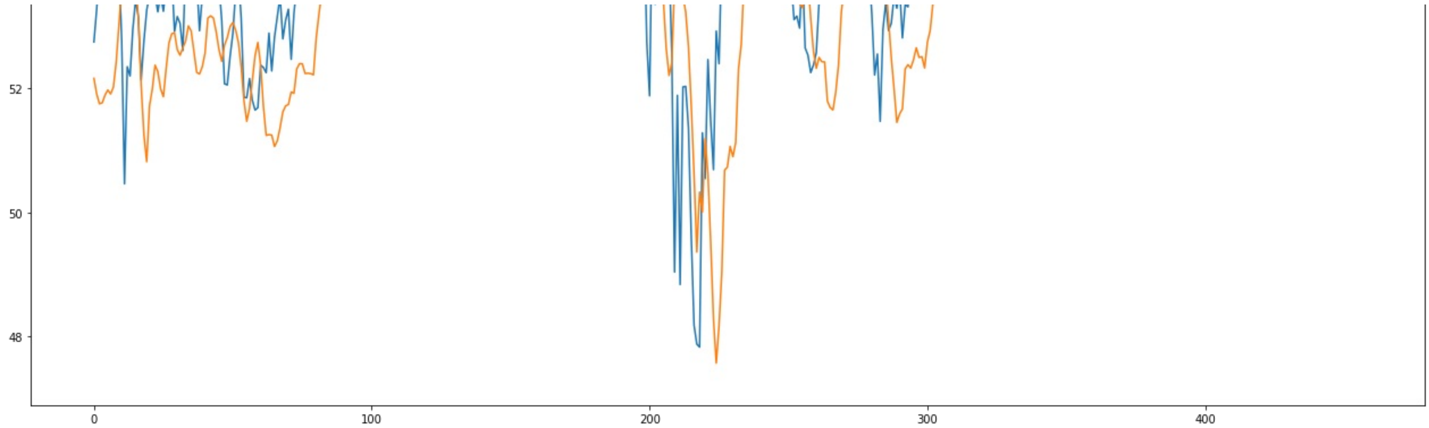
```
y_test_predicted = model.predict(ohlcv_test)
y_test_predicted = y_normaliser.inverse_transform(y_test_predicted)

y_predicted = model.predict(ohlcv_histories_normalised)
y_predicted = y_normaliser.inverse_transform(y_predicted)
```

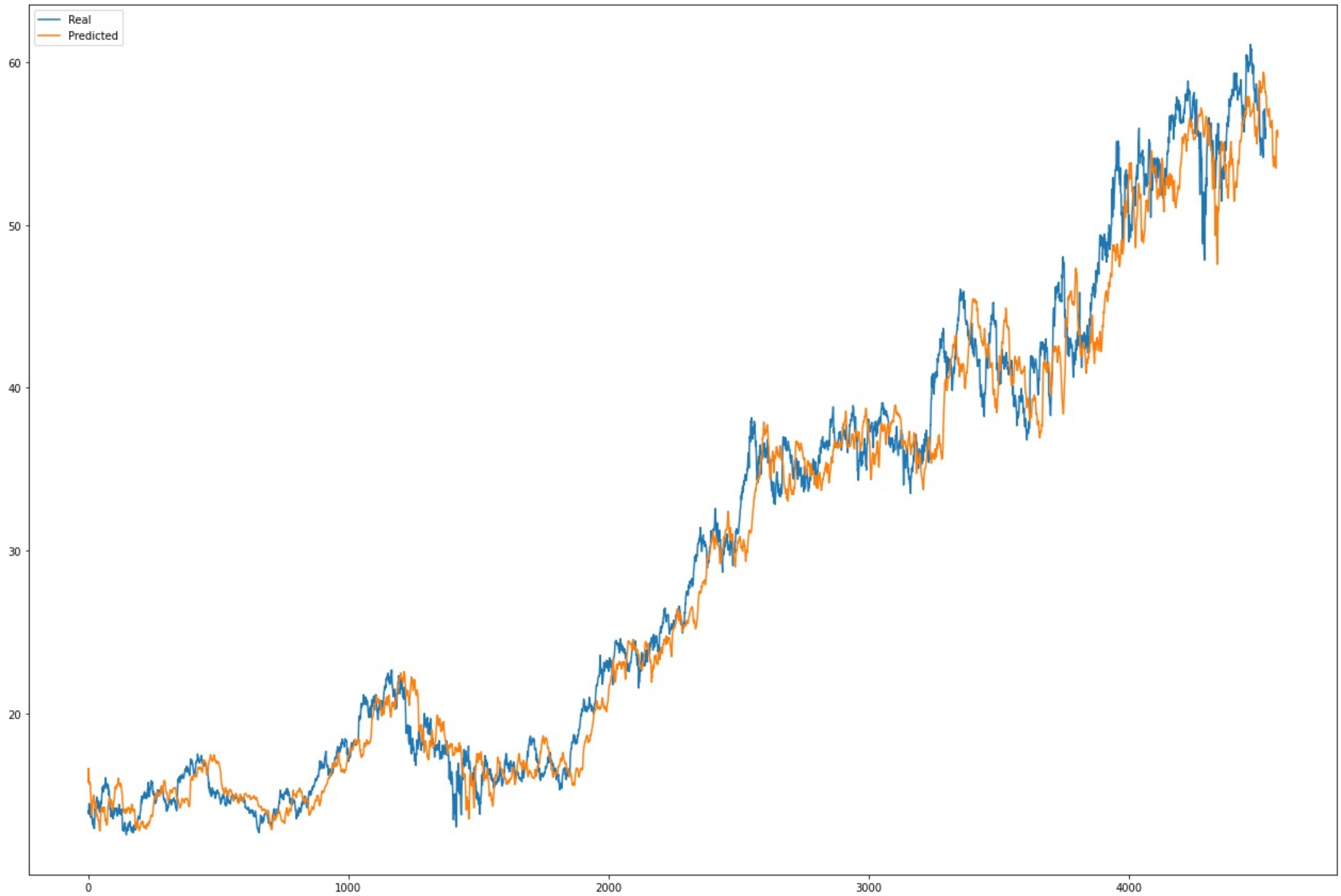
```
WARNING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_function.<
locals>.predict_function at 0x7f10d3dd2950> triggered tf.function retracing. Tracing is e
xpensive and the excessive number of tracings could be due to (1) creating @tf.function r
epeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python object
s instead of tensors. For (1), please define your @tf.function outside of the loop. For (
2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes t
hat can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/
guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/fu
nction for  more details.
```

In [30]:

```
plt.gcf().set_size_inches(22, 15, forward = True)

start = 0
end = -1
unscaled_y_test = df2[-453:]
real = plt.plot(unscaled_y_test[start:end], label='real')
pred = plt.plot(y_test_predicted[start:end], label='predicted')
plt.legend(['Real', 'Predicted'])
plt.show()

plt.gcf().set_size_inches(22, 15, forward=True)
real1 = plt.plot(df2[50:], label='real')
pred1 = plt.plot(y_predicted, label='predicted')
plt.legend(['Real', 'Predicted'])
plt.show()
```

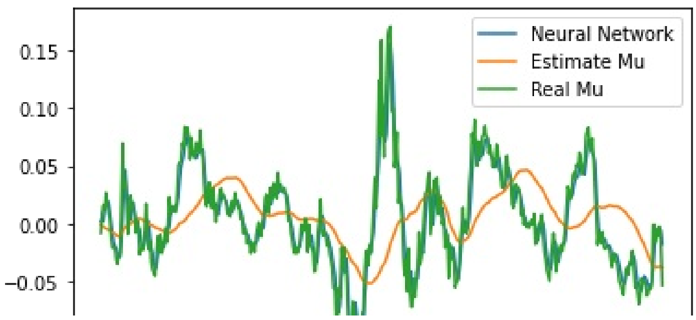


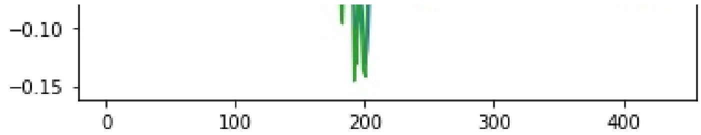




In [31]:

```
l = list(y_test_predicted[:,0])
rendements = [(l[i]-l[i-23])/l[i-23] for i in range(23,len(l))]
rendements_estimate_mu = [estimate_mu(t)['VZ'] for t in range(len(var_data23)-1-len(l)+2
3,len(var_data23)-1)]
rendements_reels = [real_mu(t)['VZ'] for t in range(len(var_data23)-1-len(l),len(var_dat
a23)-1-23)]

plt.plot(rendements, label = "Neural Network")
plt.plot(rendements_estimate_mu, label = "Estimate Mu")
plt.plot(rendements_reels, label = "Real Mu")
plt.legend()
plt.show()
```





Implémentation de l'estimateur

In [32]:

```
def rmse(l1,l2):
    return sum([(abs(l1[i] - l2[i])**2)/len(l1) for i in range(len(l1))])

last_day_month_2 = [] # Indices of the last day of each month in vardata

current_month = "01"
date2 = 0
for i,date in enumerate(var_data23.index):
  date1,date2 = date2,date
  if date[5:7] != current_month:
    current_month=date[5:7]
    if i>1:
      last_day_month_2.append((i-1,date1))

last_day_month_2 = last_day_month_2[:-2] # Remove last indices because marketcaps stops
earlier
```

In [25]:

```
def estimate_mu_nn_stock(t,stock):
  df = data_441[stock]
  df = df[[x for (_,x) in last_day_month_2]]
  df2 = df.reset_index()
  df2 = df2[stock].to_numpy().reshape(-1,1)

  i0 = 0
  for i,(t0,_) in enumerate(last_day_month_2):
    if t0 > t:
      break
    i0 += 1

  df2 = df2[:i0,:]
  data_normaliser = preprocessing.MinMaxScaler()
  data_normalised = data_normaliser.fit_transform(df2)

  history_points = 3

  ohlcv_histories_normalised = np.array([data_normalised[i  : i + history_points].copy()
for i in range(len(data_normalised) - history_points)])
  next_day_open_values_normalised = np.array([data_normalised[:,0][i + history_points].c
opy() for i in range(len(data_normalised) - history_points)])
  next_day_open_values_normalised = np.expand_dims(next_day_open_values_normalised, -1)

  next_day_open_values = np.array([df2[:,0][i + history_points].copy() for i in range(le
n(df2) - history_points)])
  next_day_open_values = np.expand_dims(next_day_open_values, -1)
  y_normaliser = preprocessing.MinMaxScaler()
  next_day_open_values = y_normaliser.fit_transform(next_day_open_values.reshape(next_day
_open_values.shape[0],1))

  test_split = 1 # Percent of data to be used for testing
  n = int(ohlcv_histories_normalised.shape[0] * test_split)

  # Splitting the dataset up into train and test sets
  ohlcv_train = ohlcv_histories_normalised[:n]
  y_train = next_day_open_values[:n]

  ohlcv_test = ohlcv_histories_normalised[n:]
  y_test = next_day_open_values[n:]

  lstm_input = Input(shape = (history_points, 1), name = 'lstm_input')
```

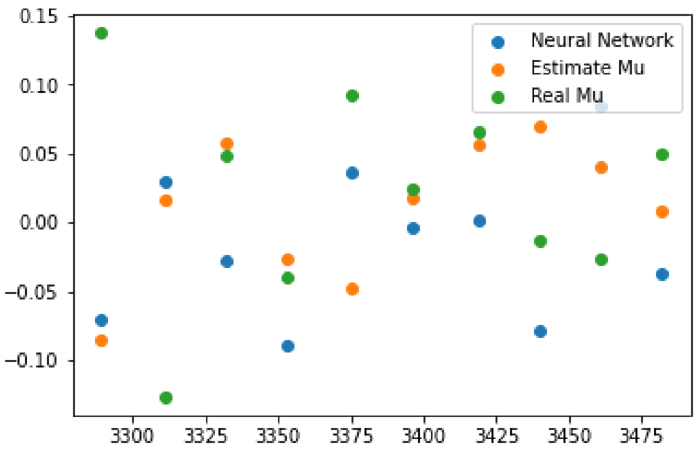
```
  x = LSTM(50, name = 'lstm_0')(lstm_input)
  x = Dropout(0.2, name = 'lstm_dropout_0')(x)
  x = Dense(64, name = 'dense_0')(x)
  x = Activation('sigmoid', name = 'sigmoid_0')(x)
  x = Dense(1, name = 'dense_1')(x)
  output = Activation('linear', name = 'linear_output')(x)
  model = Model(inputs = lstm_input, outputs = output)

  adam = optimizers.Adam(lr = 0.005)

  model.compile(optimizer = adam, loss = 'mse')
  model.fit(x = ohlcv_train, y = y_train, batch_size = 1, epochs = 20, shuffle = True, v
alidation_split = 0.1, verbose = 0)
  y = model.predict(np.expand_dims(data_normalised[-history_points:], axis=0))
  y = y_normaliser.inverse_transform(y)
  return (y[0][0]-df2[-1][0])/df2[-1][0]

def estimate_mu_nn(t,list_stocks):
  res = {}
  for stock in list_stocks:
    res[stock] = estimate_mu_nn_stock(t,stock)
  return res
```

In [33]:

```
# Estimating the evolution of the RMSE
t_interval = [t for t in list(range(3000,3500)) if t in [t0 for (t0,_) in last_day_month
]][-10:]

rendements = [estimate_mu_nn(t,['AAPL'])['AAPL'] for t in t_interval]
rendements_estimate_mu = [estimate_mu(t)['AAPL'] for t in t_interval]
rendements_reels = [real_mu(t)['AAPL'] for t in t_interval]

plt.scatter(t_interval,rendements, label = "Neural Network")
plt.scatter(t_interval,rendements_estimate_mu, label = "Estimate Mu")
plt.scatter(t_interval,rendements_reels, label = "Real Mu")
plt.legend()
plt.show()

print(rmse(rendements_reels,rendements))
print(rmse(rendements_estimate_mu,rendements_reels))
```

```
WARNING:tensorflow:5 out of the last 162 calls to <function Model.make_predict_function.<
locals>.predict_function at 0x7f10d6531170> triggered tf.function retracing. Tracing is e
xpensive and the excessive number of tracings could be due to (1) creating @tf.function r
epeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python object
s instead of tensors. For (1), please define your @tf.function outside of the loop. For (
2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes t
hat can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/
guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/fu
nction for  more details.
WARNING:tensorflow:6 out of the last 163 calls to <function Model.make_predict_function.<
locals>.predict_function at 0x7f10d2b1a830> triggered tf.function retracing. Tracing is e
xpensive and the excessive number of tracings could be due to (1) creating @tf.function r
epeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python object
s instead of tensors. For (1), please define your @tf.function outside of the loop. For (
2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes t
hat can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/
guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/fu
nction for  more details.
WARNING:tensorflow:7 out of the last 164 calls to <function Model.make_predict_function.<
locals>.predict_function at 0x7f10d65a7cb0> triggered tf.function retracing. Tracing is e
xpensive and the excessive number of tracings could be due to (1) creating @tf.function r
epeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python object
s instead of tensors. For (1), please define your @tf.function outside of the loop. For (
2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes t
hat can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/
guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/fu
nction for  more details.
WARNING:tensorflow:8 out of the last 165 calls to <function Model.make_predict_function.<
locals>.predict_function at 0x7f10d64dbb90> triggered tf.function retracing. Tracing is e
xpensive and the excessive number of tracings could be due to (1) creating @tf.function r
```

```
epeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python object
s instead of tensors. For (1), please define your @tf.function outside of the loop. For (
2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes t
hat can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/
guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/fu
nction for  more details.
WARNING:tensorflow:9 out of the last 166 calls to <function Model.make_predict_function.<
locals>.predict_function at 0x7f10d61adcb0> triggered tf.function retracing. Tracing is e
xpensive and the excessive number of tracings could be due to (1) creating @tf.function r
epeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python object
s instead of tensors. For (1), please define your @tf.function outside of the loop. For (
2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes t
hat can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/
guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/fu
nction for  more details.
WARNING:tensorflow:10 out of the last 167 calls to <function Model.make_predict_function.
<locals>.predict_function at 0x7f10d2b29830> triggered tf.function retracing. Tracing is
expensive and the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of the loop. For
(2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes
that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org
/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/f
unction for  more details.
WARNING:tensorflow:11 out of the last 168 calls to <function Model.make_predict_function.
<locals>.predict_function at 0x7f10d2adb290> triggered tf.function retracing. Tracing is
expensive and the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objec
ts instead of tensors. For (1), please define your @tf.function outside of the loop. For
(2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes
that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org
/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/f
unction for  more details.
```




```
0.010818311129232207
0.010379599106236806
```

In [40]:

```
wfinal_1_5 = pd.DataFrame(columns=origin_index) # Initializing DataFrame

for indice_day,date_day in last_day_month:
  if indice_day in list(range(3500,4000)):

    n_top = 2

    marketcap_sorted = MarketCaps441.loc[date_day].T.sort_values(ascending = False).iloc
[:n_top]
    index_top = list(marketcap_sorted.index)
    index_top = [x for x in origin_index if x in index_top]
    index_last = [[x for x in origin_index if x not in index_top]]

    # Parameters
    w = cp.Variable(n_top)
    sig = 0.128
    k = 0.2
```

```
    lam = 10

    #Estimations of the other parameters
    sigma=np.array(estimate_sigma(indice_day).loc[index_top][index_top])
    mu = np.array(list(estimate_mu_nn(indice_day,index_top).values()))
    Omega = np.zeros(sigma.shape)

    for i in range(sigma.shape[0]):
      Omega[i,i] = sigma[i,i]

    L,d,_ = scipy.linalg.ldl(sigma)
    d = np.diag(d).copy()
    inds = d >= d.max()*1e-8
    d = d[inds]
    d = np.sqrt(d)
    d.shape = (-1,1)
    Q = d * L.T[inds]

    # Defining the problem
    objective = cp.Maximize(cp.matmul(mu.T,w) - k*cp.norm(cp.matmul(Q, w)) - lam*cp.quad
_form(w,sigma))
    constraints = [w >= 0,cp.sum(w) ==1]
    prob = cp.Problem(objective, constraints)

    # Solving the problem
    prob.solve()
    w_df = pd.DataFrame(list(w.value),index = index_top, columns = [date_day]).T

    # Exporting solution
    for new_index in index_last:
        w_df[new_index] = 0

    w_df = w_df[origin_index]
    wfinal_1_5 = wfinal_1_5.append(round(w_df,5))
    #print(date_day)
```

In [37]:

```
def strat_autofin2(wfi):
  argent = [100]
  for i,(ind,date) in enumerate(last_day_month[:-1]):
    if ind in list(range(3500,4000)):
      actions_debut = argent[-1]*wfi.loc[date]/data_441.loc[date]
      argent.append((actions_debut*data_441.loc[last_day_month[i+1][1]]).sum())
  plt.plot(list(range(len(argent))), argent)
  plt.show()

#strat_autofin(wfinal_1_2)
```