2021_04_04

April 5, 2021

0.1 CVXPY

[5]:
```
import cvxpy as cvx
import numpy as np
```

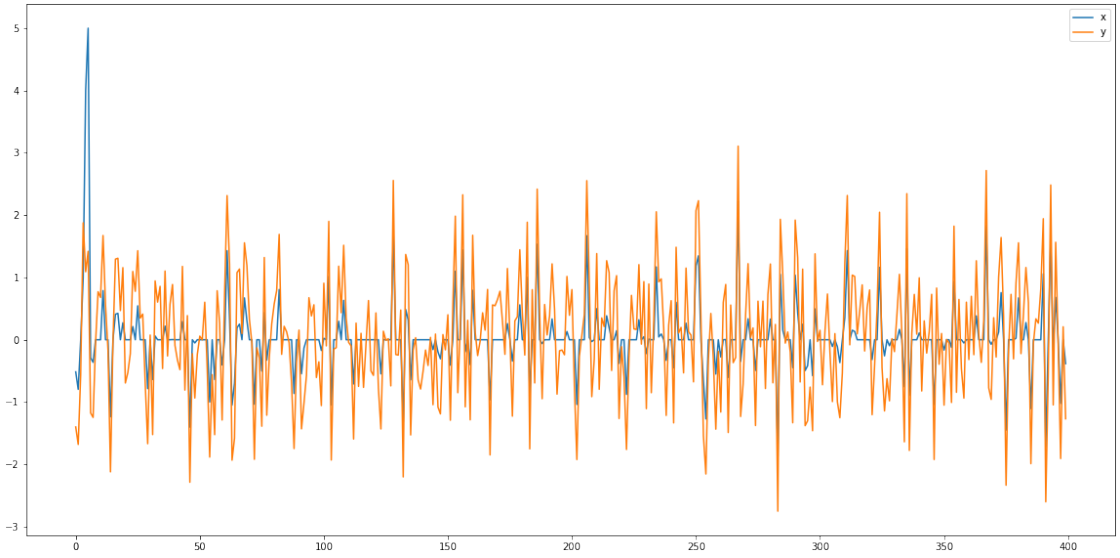
[8]:
```
# Create two scalar optimization variables.
x = cvx.Variable(400)
y = np.random.randn(400)

# Create two constraints.
constraints = [x[4]>= 4,
               x[5]>=5,
               cvx.norm1(x) <= 100]
obj = cvx.Minimize(cvx.norm(x-y))
```

[15]:
```
%%time

prob = cvx.Problem(obj, constraints)
prob.solve()  # Returns the optimal value.
print(f"""
    status:            {prob.status}
    optimal value:     {prob.value}
""")
```

```
    status:            optimal
    optimal value:     13.911054525821655

Wall time: 23 ms
```

[23]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
plt.plot(x.value); plt.plot(y)
plt.legend(['x', 'y'], loc='upper right')
```

[23]:
```
<matplotlib.legend.Legend at 0x1ec261d2430>
```

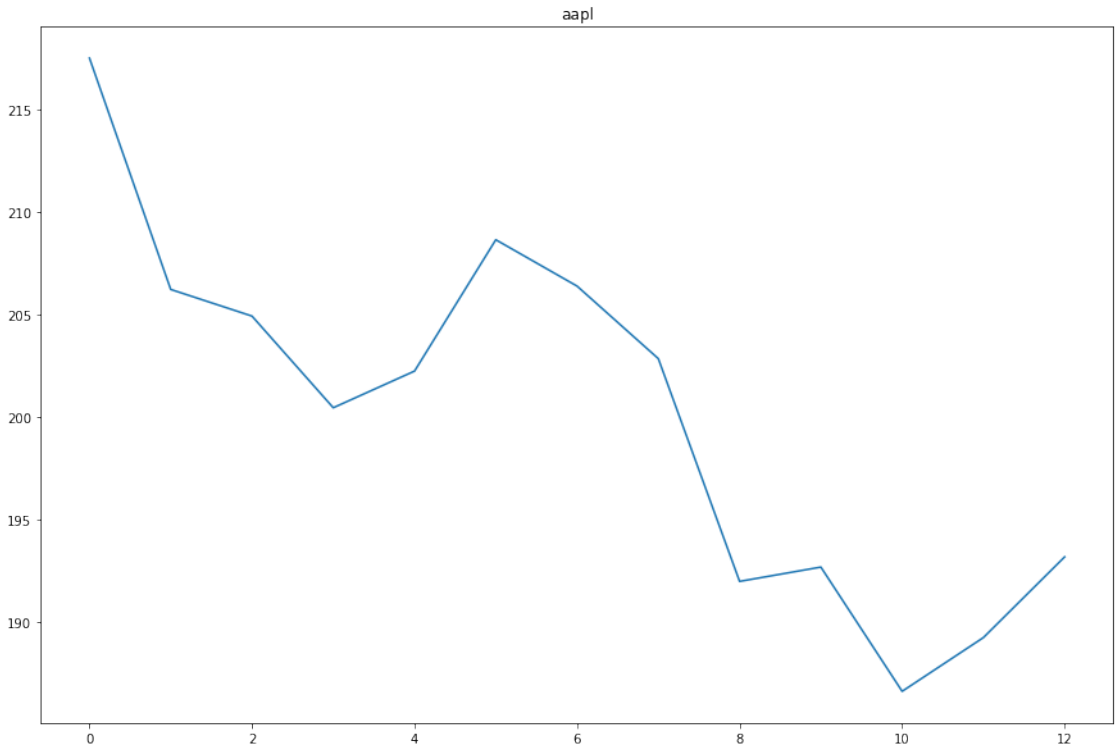


[ ]:

[26]:
```
price_info = np.load('data/algo_trading/data.npy', allow_pickle=True).item()
```

[28]:
```
with open('data/algo_trading/labels.txt', 'w') as f:
    for name in price_info.keys():
        f.write(name.upper()+'\n')
```

[32]:
```
def plot(name,data,s,start,end):
    plt.figure(figsize=(15,10))
    C=np.array(data[name][s])[start:end+1]
    plt.plot(C)
    plt.title(name)
plot('aapl',price_info,'Close',22,34)
```



[34]:
```
price_info['aapl'].head()
```

[34]:

	Close	Date	High	Low	Open	Volume	MACD	\
0	225.622	20181001	227.646	223.995	226.088	2.347549e+07	0.000000	
1	227.726	20181002	228.222	224.302	224.402	2.667174e+07	0.394886	
2	229.928	20181003	231.675	227.507	228.341	3.038102e+07	1.026799	
3	226.337	20181004	230.554	224.977	228.966	3.368405e+07	0.678610	
4	222.258	20181005	226.783	218.875	225.880	3.534031e+07	-0.343289	

	RSI	SO	OBV	EMA	SMI
0	0.000000	44.563133	2.347549e+07	225.622000	-10.873733
1	100.000000	88.265910	5.014722e+07	226.042800	-6.851314
2	37.815205	77.252604	8.052824e+07	226.819840	0.487604
3	36.088152	30.494792	4.684419e+07	226.723272	1.732747
4	45.368127	26.429687	1.150388e+07	225.830218	-3.007887

[ ]:
```
```

[38]:
```
from keras.models import Sequential
from keras.layers import Dense
```

[41]:
```
tmp
```

[41]:
```
array([225.622, 227.726, 229.928, 226.337, 222.258, 222.109, 225.255,
       212.534, 215.263, 221.077, 216.265, 221.425, 218.895, 214.826,
       217.307, 218.547, 220.581, 214.975, 217.307, 215.402, 210.629,
       213.238, 217.475, 206.203, 204.904, 200.438, 202.224, 208.624,
       206.365, 202.83 , 191.977, 192.674, 186.62 , 189.239, 193.172,
       184.111, 176.244, 176.224, 171.276, 170.2  , 174.94 , 181.054,
       177.788, 178.037, 181.283, 175.906, 174.492, 167.651, 168.388,
       168.995, 168.776, 169.593, 164.395, 163.997, 164.654, 160.552,
       156.828, 149.459, 146.124, 155.732, 155.334, 155.722, 157.485])
```

[45]:
```
data, X_trial = [], []
for name in price_info.keys():
    cur=price_info[name]
    C=np.array(cur['Close']); h = 10
    for i in range(h,len(C)-h):
        vec=[]
        for j in ['EMA','MACD','SMI','SO','RSI','Close']:
            vec+=list(np.array(cur[j])[i-h+1:i+1])
        vec.append((C[i+h]-C[i])/C[i])
        data.append(vec)
        if i==22: X_trial.append(vec[:-1])
```

[46]:
```
# from sklearn.preprocessing import StandardScaler, MinMaxScaler
# data_=scaler.fit_transform(data)

data = np.array(data)
np.random.shuffle(data)
size=int(0.85*len(data))
X_train, y_train = data[0:size,0:-1], data[0:size,-1]
X_test, y_test = data[size:,0:-1], data[size:,-1]
```

[47]:
```
model = Sequential()
model.add(Dense(100, input_dim=X_train.shape[1], activation='sigmoid'))
model.add(Dense(20,activation='sigmoid'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

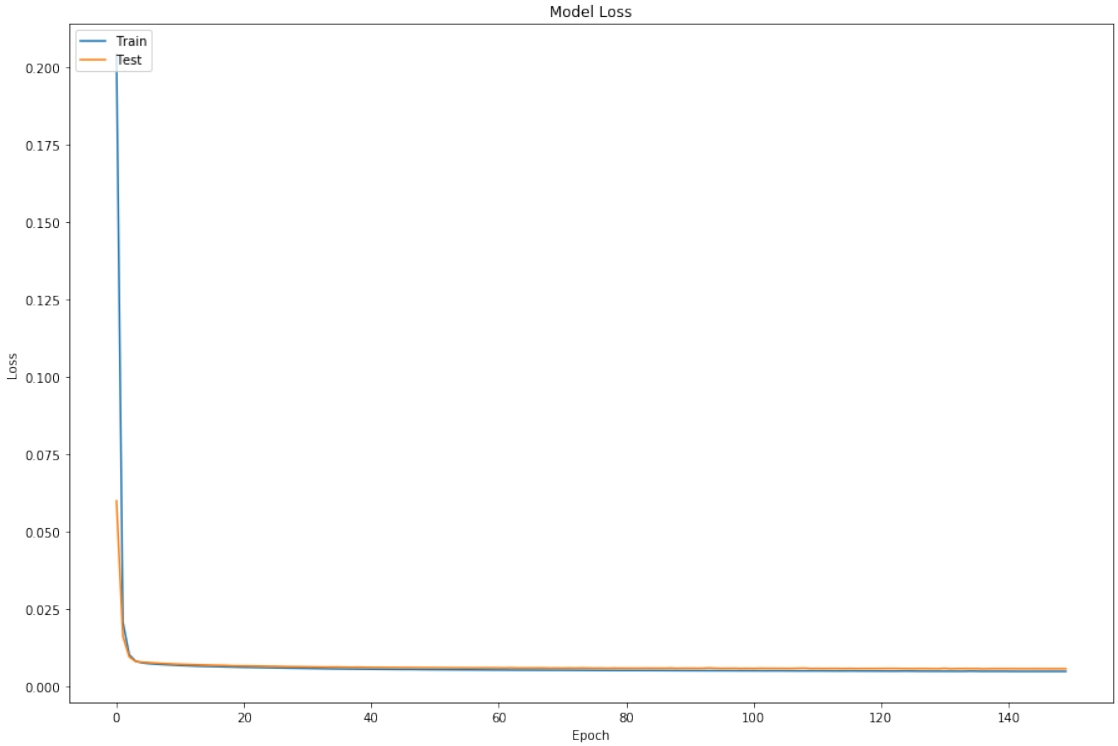
[ ]:
```
history=model.fit(X_train,y_train,validation_split=0.
↪2,epochs=150,batch_size=1000,verbose=1)
```

[50]:
```
model.evaluate(X_test,y_test)
```

```
122/122 [==============================] - 0s 3ms/step - loss: 0.0053
```

[50]:
```
0.005317376460880041
```

[52]:
```
def plot(data):
    plt.figure(figsize=(15,10))
    plt.plot(data['loss']); plt.plot(data['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss'); plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
plot(history.history)
```



0.1.1 Baseline Model

- $t-10 : t$ 600+ $30 \sim 60$ 0, MACD 0
- $t : t+10$, , performance ? 30

[57]:
```
testdates=[[price_info['a']['Date'][11],price_info['a']['Date'][22]]
           ,[price_info['a']['Date'][23],price_info['a']['Date'][33]]
           ,[price_info['a']['Date'][34],price_info['a']['Date'][43]]
           ,[price_info['a']['Date'][44],price_info['a']['Date'][52]]
           ,[price_info['a']['Date'][53],price_info['a']['Date'][62]]]
print(testdates,)
```

```
[['20181016', '20181031'], ['20181101', '20181115'], ['20181116', '20181130'],
['20181203', '20181214'], ['20181217', '20181231']]
```

[62]:
```
s=30; P, M = [{} for _ in range(6)], [{} for _ in range(6)]
for name in price_info.keys():
    cur=price_info[name]
    C=np.array(cur['Close']); O=np.array(cur['Open']); macd=np.
 ↪array(cur['MACD'])
    P[0][name] = (C[10]-O[0])/O[0]; M[0][name] = macd[10]
    P[1][name] = (C[22]-O[11])/O[11]; M[1][name] = macd[22]
    P[2][name] = (C[33]-O[23])/O[23]; M[2][name] = macd[33]
    P[3][name] = (C[43]-O[34])/O[34]; M[3][name] = macd[43]
    P[4][name] = (C[52]-O[44])/O[44]; M[4][name] = macd[52]
    P[5][name] = (C[62]-O[53])/O[53]; M[5][name] = macd[62]
for i in range(5):
    dd = sorted(P[i].items(), key=lambda x: x[1],reverse=True)
    pool1=[pair[0] for pair in dd[s:2*s] if pair[1]>0 and M[i][pair[0]]>0]
    performance=sum([P[i+1][n] for n in pool1])/float(s)
    benchmark=sum(P[i+1].values())/len(P[i+1].keys())
    print(benchmark,performance)
```

```
-0.02377470308569231 0.008416277793983176
0.00958486895368256 0.009895974212731584
0.008820059157016264 0.008668956915780204
-0.08420009378968411 -0.0790157009478927
-0.04260201275399443 -0.03327840588916
```

0.1.2

- , 10 ( ) ,

[73]:
```

```

[73]:
```
(25843, 61)
```

[ ]:
```

```

0.1.3 Zero Dirft Test

from AC model?

$$\Delta m_t = \mu + \lambda s_t + \epsilon_t; p_t = m_t + \gamma s_t$$

$$s_t^\star = S\left(\frac{1}{T} + \frac{T+1-2t}{4\gamma+2\lambda}\mu\right)$$

[86]:
```
import pandas as pd
from datetime import datetime, timedelta
```

[75]:
```
def size_optimal(T,S,mu,gamma,lbd):
    return S*(1.0/float(T)+(T+1.)*mu/float(4*gamma+2*lbd))-np.
 ↪arange(1,T+1)*S*mu/float(2*gamma+lbd)
```

```
print(size_optimal(10,10000,0.01,0.2,0.5),)
```

```
[1500.         1388.88888889 1277.77777778 1166.66666667 1055.55555556
  944.44444444  833.33333333  722.22222222  611.11111111  500.        ]
```

[78]:
```
df=pd.read_csv('data/algo_trading/A_test_quote.csv',header=0)
df.head(5)
```

[78]:

	DATE	TIME_M	EX	BID	BIDSIZ	ASK	ASKSIZ	QU_COND	\
0	20181217	9:30:00.318357475	Y	66.29	1	71.76	1	R	
1	20181217	9:30:00.760058312	X	68.61	1	69.51	3	R	
2	20181217	9:30:01.005804310	N	69.05	3	69.11	1	O	
3	20181217	9:30:01.006097451	P	68.62	1	69.23	1	R	
4	20181217	9:30:01.006124860	P	68.62	1	70.41	1	R	

	QU_SEQNUM	NATBBO_IND	QU_CANCEL	QU_SOURCE	SYM_ROOT	SYM_SUFFIX
0	15918201	A	NaN	C	A	NaN
1	15999901	A	NaN	C	A	NaN
2	16045207	G	NaN	C	A	NaN
3	16046601	A	NaN	C	A	NaN
4	16046802	A	NaN	C	A	NaN

10 , $bid > 0, ask < \inf$; market price

[98]:
```
Mt, T = [], []
for j in range(len(df['TIME_M'])):
    if np.array(df['TIME_M'])[j][:6]+'0' not in T and np.array(df['BID'])[j]>0␣
 ↪and np.array(df['ASK'])[j]<199999.99:
        T.append(np.array(df['TIME_M'])[j][:6]+'0')
        Mt.append(0.5*(np.array(df['BID'])[j]+np.array(df['ASK'])[j]))
    else:
        pass
```

[99]:
```
mt, tt=[], []; j=0
start=datetime.strptime("09:30:00", "%H:%M:%S")
for i in range(180):
    tt.append(start)
    if start==datetime.strptime(T[j],'%H:%M:%S'):
        mt.append(Mt[j]); j+=1
    else:
        mt.append(Mt[j])
    start+= timedelta(0,10)
mt=np.array(mt)
```

[100]:
```
df_t=pd.read_csv('data/algo_trading/A_test_trade.csv',header=0)
df_t.head(5)
```

[100]:
```
        DATE             TIME_M EX SYM_ROOT  SYM_SUFFIX TR_SCOND   SIZE  PRICE  \
0   20181217  9:30:01.005852000  N        A         NaN        0  19875  69.11
1   20181217  9:30:01.011290000  N        A         NaN      NaN    100  69.11
2   20181217  9:30:01.019645000  D        A         NaN      NaN    100  69.06
3   20181217  9:30:03.195176000  P        A         NaN      NaN    100  69.05
4   20181217  9:30:03.195210000  P        A         NaN        Q    100  69.05

   TR_CORR  TR_SEQNUM TR_SOURCE TR_RF
0        0     279001         C   NaN
1        0     283401         C   NaN
2        0     284301         C     T
3        0     311701         C   NaN
4        0     311801         C   NaN
```

[102]:
```
Pt, St, T = [], [], []; i=-1
for j in range(len(df_t['TIME_M'])):
    if np.array(df_t['TIME_M'])[j][:6]+'0' not in T:
        i+=1
        T.append(np.array(df_t['TIME_M'])[j][:6]+'0')
        Pt.append(np.array(df_t['PRICE'])[j])
        St.append(np.array(df_t['SIZE'])[j])
    else:
        St[i]+=np.array(df_t['SIZE'])[j]
```

[103]:
```
pt, st, ttt=[], [], []; j=0
start=datetime.strptime("09:30:00", "%H:%M:%S")
for i in range(180):
    ttt.append(start)
    if j<len(T):
        if start==datetime.strptime(T[j],'%H:%M:%S'):
            pt.append(Pt[j])
            st.append(St[j])
            j+=1
        else:
            pt.append(pt[-1])
            st.append(0)
    else:
        pt.append(pt[-1])
        st.append(0)
    start+=timedelta(0,10)
pt=np.array(pt); st=np.array(st)
```

[108]:
```
common=sorted(set(tt).intersection(set(ttt)))
```

0.1.4 Perform the first regression: $M_t - M_{t-1}$ vs. S_t

- : $\mu = 0$, Δm_t s_t

[109]:
```
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm
regr = linear_model.LinearRegression()
X=st[1:].reshape(-1,1)
y=np.diff(mt)
regr.fit(X,y)
mu=regr.intercept_
lbd=regr.coef_[0]
X2 = sm.add_constant(X)
est = sm.OLS(y, X2)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                 -0.005
Method:                 Least Squares   F-statistic:                    0.1384
Date:                Mon, 05 Apr 2021   Prob (F-statistic):              0.710
Time:                        15:44:58   Log-Likelihood:                -609.65
No. Observations:                 179   AIC:                             1223.
Df Residuals:                     177   BIC:                             1230.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1340      0.658      0.204      0.839      -1.164       1.432
x1            -0.0003      0.001     -0.372      0.710      -0.002       0.001
==============================================================================
Omnibus:                       60.517   Durbin-Watson:                   3.007
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2722.594
Skew:                          -0.056   Prob(JB):                         0.00
Kurtosis:                      22.106   Cond. No.                     1.07e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.07e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[110]:
```
from matplotlib import pyplot as plt
plt.plot(st[1:].reshape(-1,1),np.diff(mt),'ko')
plt.xlabel('S_t')
plt.ylabel('1st Diff M_t')
```

[110]: Text(0, 0.5, '1st Diff M_t')