Authors: Justin Beer, Anthony Dalton, Harsha Kommanapalli

Date: 2/05/2018

**Financial Applications of a Time Series**

**Future Contract Rolling for E-mini Russell 2000 futures**

**Abstract**

The purpose of this case study is to simulate future contract rolling on E-mini Russell 2000 futures with a continuous rolling index. Using Python and Jupyter Notebook, we will use pandas and pandas\_datareader to read in daily pricing information from Yahoo, add expiry dates and noise, and generate a weighting matrix to create future rolled returns that users of the dataset can use to properly estimate the proper time to trade out of a contract.

**Introduction**

In the trading of futures, a future refers to the process where a trader will close out open positions of contracts with nearer expiration dates as opposed to those contracts with later expirations dates. The transition period from the nearer expiring contract, the most liquid, to the next contract is where the term future contract rolling comes from. Using Python and Pandas, we will read in E-mini Russell 2000 futures data from 2010 to 2018 and simulate two contract expiration dates. With the data and expiration dates we can describe with analytics how quickly and how much the user of the tool should trade out of the nearer expiring contract to the next contract.

**Methods**

After importing pandas and pandas\_datareader the first step in the process was to find the correct ticker item for E-mini Russell 2000 futures. The supporting stock price information was read in from Yahoo with the following code:

px = web.get\_data\_yahoo('TF')['Adj Close'] \* 10

Data from January 4th, 2010 to January 30th, 2018 are imported into the dataset. After reading in the 2,033 values, the two contract dates were added into a series:

from pandas import Series, DataFrame

from datetime import datetime

expiry = {'TFH2': datetime(2018, 3, 30),

'TFM2': datetime(2018, 6, 29)}

expiry = Series(expiry).sort\_values()

We proceeded to add some random data and noise along with the prices for E-mini Russell 2000 from Yahoo to simulate the two contracts:

import numpy as np

np.random.seed(12347)

N = 200

walk = (np.random.randint(0, 200, size=N) - 100) \* 0.25

perturb = (np.random.randint(0, 20, size=N) - 10) \* 0.25

walk = walk.cumsum()

rng = pd.date\_range(px.index[0], periods=len(px) + N, freq='B')

near = np.concatenate([px.values, px.values[-1] + walk])

far = np.concatenate([px.values, px.values[-1] + walk + perturb])

prices = DataFrame({'TFH2': near, 'TFM2': far}, index=rng)

After reading in the new values, running a prices.tail() displays the new randomized column in addition to the previous column that is based off the actual amount of our stock in question.

|  |  |  |
| --- | --- | --- |
|  | TFH2 | TFM2 |
| 7/23/18 | 122.25 | 124 |
| 7/24/18 | 108.5 | 110.75 |
| 7/25/18 | 116.5 | 118.25 |
| 7/26/18 | 133 | 132.25 |
| 7/27/18 | 113 | 110.75 |

The next step is to splice the time series together using a weighting matrix. A function that was referenced in Python for Data Analysis by Wes McKinney allows passing in of three variables of the start date, the expiration date, and the sequence of contract names:

def get\_roll\_weights(start, expiry, items, roll\_periods=10):

# start : first date to compute weighting DataFrame

# expiry : Series of ticker -> expiration dates

# items : sequence of contract names

dates = pd.date\_range(start, expiry[-1], freq='B')

weights = DataFrame(np.zeros((len(dates), len(items))),

index=dates, columns=items)

prev\_date = weights.index[0]

for i, (item, ex\_date) in enumerate(expiry.iteritems()):

if i < len(expiry) - 1:

weights.ix[prev\_date:ex\_date - pd.offsets.BDay(), item] = 1

roll\_rng = pd.date\_range(end=ex\_date - pd.offsets.BDay(),

periods=roll\_periods + 1, freq='B')

decay\_weights = np.linspace(0, 1, roll\_periods + 1)

weights.ix[roll\_rng, item] = 1 - decay\_weights

weights.ix[roll\_rng, expiry.index[i + 1]] = decay\_weights

else:

weights.ix[prev\_date:, item] = 1

prev\_date = ex\_date

return weights

#The weights look like this around the TFM2 expiry:

#weights = get\_roll\_weights('3/1/2018', expiry, prices.columns)

with pd.option\_context('display.max\_rows', None, 'display.max\_columns', 3):

print(weights.loc['2018-03-01':'2018-04-01'])

**Results**

After running the function and passing the start date of March 1st, 2018, the expiration date of April 1st, 2018, and the prices.column row the following weights were returned:

|  |  |  |
| --- | --- | --- |
|  | **TFH2** | **TFM2** |
| 3/1/18 | 1.00 | 0.00 |
| 3/2/18 | 1.00 | 0.00 |
| 3/5/18 | 1.00 | 0.00 |
| 3/6/18 | 1.00 | 0.00 |
| 3/7/18 | 0.90 | 0.10 |
| 3/8/18 | 0.80 | 0.20 |
| 3/9/18 | 0.70 | 0.30 |
| 3/12/18 | 0.60 | 0.40 |
| 3/13/18 | 0.50 | 0.50 |
| 3/14/18 | 0.40 | 0.60 |
| 3/15/18 | 0.30 | 0.70 |
| 3/16/18 | 0.20 | 0.80 |
| 3/19/18 | 0.10 | 0.90 |
| 3/20/18 | 0.00 | 1.00 |
| 3/21/18 | 0.00 | 1.00 |
| 3/22/18 | 0.00 | 1.00 |
| 3/23/18 | 0.00 | 1.00 |
| 3/26/18 | 0.00 | 1.00 |
| 3/27/18 | 0.00 | 1.00 |
| 3/28/18 | 0.00 | 1.00 |
| 3/29/18 | 0.00 | 1.00 |
| 3/30/18 | 0.00 | 1.00 |

From March 1st to March 6th, the simulation shows that 100% of our position would stay in TFH2. Every day after, 10% of our position would be migrated over to TFM2 before our entire position would be migrated on March 20th, 2018.

**Conclusion**

Rolling over contracts is very nuanced and specific to the type of product that is being invested in. With the amount of domain expertise needed to stay ahead of the curve in volatile markets, it’s imperative to use code and functionality to simplify any aspect of a trader’s job that is possible. With future contract rolling, it is very simple to read in stock values, specify contract terms and dates, and know when to start moving positions from near term to farther out contracts. With our example of E-mini Russell 2000 and less than 100 lines of code, we were able to read in 2,033 unique days stock values, specify a start date of March 1st, 2018 and an expiry dates of April 1st, 2018, and explain to the most novice of traders how much of a position to move along with the timing for transition.

In future work, we’d fine tune the simulation practice of the second data frame much more thoroughly. It’d also be interesting to create a new dataframe that displays the adjusted price of both contract one, the mix of contract one and two, and finally contract two on its own.