Forex

January 31, 2020

[1]:

```python
from matplotlib import pyplot as plt
import importlib
import logging

from collections import Counter

import numpy as np
import pandas as pd

from src.data_import import Importing as importing
from src.fqi.FQI import FQI
from src.policy.QPolicy import QPolicy
from src.rewards.Position import Position
from src.rewards.UnrealizedReward import UnrealizedReward
from src.utils import set_position, create_all_combination, is_parallelizable
```

[2]:

```python
def random_action(state):
    the_random_action = pd.Series(np.random.randint(-1, 2, state.shape[0]),␣
 ↪dtype='category') #values: -1, 0, 1
    return the_random_action.rename('action')
def possible_actions(integer_positions):
    return [Position(integer) for integer in integer_positions]
```

[3]:

```python
start_date = "01/01"
end_date = "12/31"
historical_observations =2
columns_with_historical = [ "log_close"      ]
starting_time = 60
columns_with_current = ["open", "close"]
fee =1e-05
position_size = 10000
action_set = [1,-1,0]
model_name = "RandomForest"
model_parameters= {
            "n_estimators":4 ,
            "criterion": "mse",
            "n_jobs": -1,
            "min_impurity_decrease": 0.0001
        }
max_iteration =5
discount =0.99
run_parameters = {}
```

[4]:
```
print('Reading training set from file  ...')
file_data = importing.import_df("2016-EURUSD-1m.csv")
print('...Done')
```

```
Reading training set from file  …
…Done
```

[5]:
```
file_data
```

[5]:
```
         date time_min   close    open    high     low  minute  log_open  \
0       01/04    01:00  1.0847  1.0846  1.0847  1.0845    60.0  0.000000
1       01/04    01:01  1.0846  1.0847  1.0847  1.0846    61.0  0.000092
2       01/04    01:02  1.0845  1.0846  1.0846  1.0845    62.0  0.000000
3       01/04    01:03  1.0843  1.0845  1.0845  1.0842    63.0 -0.000092
4       01/04    01:04  1.0843  1.0843  1.0843  1.0843    64.0 -0.000277
…         …      …     …       …      …     …      …       …
319772  12/30    21:25  1.0518  1.0520  1.0520  1.0518  1285.0 -0.003037
319773  12/30    21:26  1.0518  1.0518  1.0518  1.0518  1286.0 -0.003227
319774  12/30    21:27  1.0520  1.0518  1.0520  1.0518  1287.0 -0.003227
319775  12/30    21:28  1.0520  1.0520  1.0520  1.0519  1288.0 -0.003037
319776  12/30    21:29  1.0519  1.0520  1.0520  1.0519  1289.0 -0.003037

        log_close  log_high   log_low    volume            datetime
0        0.000092  0.000092 -0.000092  1.000000 2016-01-04 01:00:00
1        0.000000  0.000092  0.000000  1.038156 2016-01-04 01:01:00
2       -0.000092  0.000000 -0.000092  1.322734 2016-01-04 01:02:00
3       -0.000277 -0.000092 -0.000369  1.701113 2016-01-04 01:03:00
4       -0.000277 -0.000277 -0.000277  1.915739 2016-01-04 01:04:00
…           …       …       …       …                 …
319772  -0.003227 -0.003037 -0.003227  0.227612 2016-12-30 21:25:00
319773  -0.003227 -0.003227 -0.003227  0.199005 2016-12-30 21:26:00
319774  -0.003037 -0.003037 -0.003227  0.269071 2016-12-30 21:27:00
319775  -0.003037 -0.003037 -0.003132  0.225124 2016-12-30 21:28:00
319776  -0.003132 -0.003037 -0.003132  0.200249 2016-12-30 21:29:00

[319777 rows x 13 columns]
```

[6]:
```
# Volatility Indicator Function: ATR - Average True Range
talib_fun = [ {"name":"ATR", "parameters": {"timeperiod": 10}}]
print('Applying talib functions to training set: {talib} ...'.
 ↪format(talib=talib_fun))
```

```
file_data_talib, talib_names = importing.apply_talib(file_data,    talib_fun )
print('...Done')
```

```
Applying talib functions to training set: [{'name': 'ATR', 'parameters':
{'timeperiod': 10}}] …
…Done
```

[7]:
```
print('Creating training dataframes for dates {start} to {end} ...'.
 ↪format(start=start_date,end=end_date))
current_state_no_position, next_state_no_position, price_info, minutes = ␣
 ↪importing.create_tuples(  start_date,end_date,file_data_talib,␣
 ↪historical_observations,columns_with_historical, columns_with_current+␣
 ↪talib_names, starting_time)
print('...Done')
```

```
Creating training dataframes for dates 01/01 to 12/31 …
…Done
```

[8]:
```
current_state_no_position
```

[8]:
```
        log_close_-2  log_close_-1  log_close_0    open   close       ATR
0           0.001382      0.001290     0.001750  1.0860  1.0865  0.000271
1           0.001290      0.001750     0.001842  1.0865  1.0866  0.000294
2           0.001750      0.001842     0.001658  1.0866  1.0864  0.000295
3           0.001842      0.001658     0.001750  1.0864  1.0865  0.000295
4           0.001658      0.001750     0.001474  1.0865  1.0862  0.000296
…                …            …           …       …      …       …
303912     -0.003513     -0.003513    -0.003037  1.0515  1.0520  0.000143
303913     -0.003513     -0.003037    -0.003227  1.0520  1.0518  0.000149
303914     -0.003037     -0.003227    -0.003227  1.0518  1.0518  0.000134
303915     -0.003227     -0.003227    -0.003037  1.0518  1.0520  0.000141
303916     -0.003227     -0.003037    -0.003037  1.0520  1.0520  0.000137

[303917 rows x 6 columns]
```

[9]:
```
next_state_no_position #1 day after wrt current_state_no_position
```

[9]:
```
        log_close_-2  log_close_-1  log_close_0    open   close       ATR
0           0.001290      0.001750     0.001842  1.0865  1.0866  0.000294
1           0.001750      0.001842     0.001658  1.0866  1.0864  0.000295
2           0.001842      0.001658     0.001750  1.0864  1.0865  0.000295
3           0.001658      0.001750     0.001474  1.0865  1.0862  0.000296
4           0.001750      0.001474     0.001842  1.0862  1.0866  0.000316
…                …            …           …       …      …       …
303912     -0.003513     -0.003037    -0.003227  1.0520  1.0518  0.000149
303913     -0.003037     -0.003227    -0.003227  1.0518  1.0518  0.000134
303914     -0.003227     -0.003227    -0.003037  1.0518  1.0520  0.000141
```

```
303915     -0.003227     -0.003037     -0.003037  1.0520  1.0520  0.000137
303916     -0.003037     -0.003037     -0.003132  1.0520  1.0519  0.000133

[303917 rows x 6 columns]
```

[10]:

```
action = random_action(current_state_no_position)
current_state, next_state = set_position(current_state_no_position,␣
 ↪next_state_no_position, action, minutes)

    # reward
reward = UnrealizedReward(fee, position_size).calculate(current_state, action,␣
 ↪price_info, minutes)
```

[11]:

```
action
```

[11]:

```
0          0
1          1
2          1
3          1
4          0
          ..
303912    -1
303913    -1
303914     1
303915    -1
303916     0
Name: action, Length: 303917, dtype: category
Categories (3, int64): [-1, 0, 1]
```

[12]:

```
reward
```

[12]:

```
0          0.000000
1         -0.100000
2         -1.840943
3          0.920387
4         -2.861922
             …
303912     4.552852
303913     1.901502
303914    -0.200000
303915     1.701141
303916    -0.100000
Name: position, Length: 303917, dtype: float64
```

[13]:

```
    # samples creation
```

```
samples = { 'current_state': current_state.copy(),'next_state': next_state.
 ↪copy(), 'reward': reward, 'action': action,  'minute': minutes,  'fee': fee,␣
 ↪'position_size': position_size,    'price_info': price_info }
samples = create_all_combination(samples, possible_actions(action_set))
print('Initializing model {model} with parameters {parameters} ...' .
 ↪format(model=model_name,  parameters=model_parameters))
model_module = importlib.import_module( '.' + model_name,'src.models')\

model = model_module.get_model(model_parameters, samples['current_state'].
 ↪copy(),   samples['reward'], samples['action'])
print('...Done')
```

```
Initializing model RandomForest with parameters {'n_estimators': 4, 'criterion':
'mse', 'n_jobs': -1, 'min_impurity_decrease': 0.0001} …
…Done
```

```
[14]: samples
```

```
[14]: {'current_state':         log_close_-2  log_close_-1  log_close_0    open
close       ATR  \
 0           0.001382      0.001290     0.001750  1.0860  1.0865  0.000271
 1           0.001290      0.001750     0.001842  1.0865  1.0866  0.000294
 2           0.001750      0.001842     0.001658  1.0866  1.0864  0.000295
 3           0.001842      0.001658     0.001750  1.0864  1.0865  0.000295
 4           0.001658      0.001750     0.001474  1.0865  1.0862  0.000296
 …                …           …           …      …      …      …
 303912     -0.003513     -0.003513    -0.003037  1.0515  1.0520  0.000143
 303913     -0.003513     -0.003037    -0.003227  1.0520  1.0518  0.000149
 303914     -0.003037     -0.003227    -0.003227  1.0518  1.0518  0.000134
 303915     -0.003227     -0.003227    -0.003037  1.0518  1.0520  0.000141
 303916     -0.003227     -0.003037    -0.003037  1.0520  1.0520  0.000137

          position
 0               1
 1               1
 2               1
 3               1
 4               1
 …            …
 303912          0
 303913          0
 303914          0
 303915          0
 303916          0

 [2735253 rows x 7 columns],
 'next_state':         log_close_-2  log_close_-1  log_close_0    open    close
```

```
ATR  \
 0           0.001290      0.001750     0.001842  1.0865  1.0866  0.000294
 1           0.001750      0.001842     0.001658  1.0866  1.0864  0.000295
 2           0.001842      0.001658     0.001750  1.0864  1.0865  0.000295
 3           0.001658      0.001750     0.001474  1.0865  1.0862  0.000296
 4           0.001750      0.001474     0.001842  1.0862  1.0866  0.000316
 …              …           …          …     …     …       …
 303912     -0.003513     -0.003037    -0.003227  1.0520  1.0518  0.000149
 303913     -0.003037     -0.003227    -0.003227  1.0518  1.0518  0.000134
 303914     -0.003227     -0.003227    -0.003037  1.0518  1.0520  0.000141
 303915     -0.003227     -0.003037    -0.003037  1.0520  1.0520  0.000137
 303916     -0.003037     -0.003037    -0.003132  1.0520  1.0519  0.000133

         position
 0              1
 1              1
 2              1
 3              1
 4              1
 …          …
 303912         0
 303913         0
 303914         0
 303915         0
 303916         0

 [2735253 rows x 7 columns],
 'reward': 0         4.501933
 1         0.820302
 2        -1.940943
 3         0.820387
 4        -2.861922
             …
 303912    0.000000
 303913   -0.000000
 303914    0.000000
 303915    0.000000
 303916    0.000000
 Length: 2735253, dtype: float64,
 'action':           0
 0       1.0
 1       1.0
 2       1.0
 3       1.0
 4       1.0
 …     …
 303912  0.0
```

```
303913  0.0
303914  0.0
303915  0.0
303916  0.0

[2735253 rows x 1 columns],
'minute': 0          120.0
1           121.0
2           122.0
3           123.0
4           124.0
            …
303912     1284.0
303913     1285.0
303914     1286.0
303915     1287.0
303916     1288.0
Length: 303917, dtype: float64,
'fee': 1e-05,
'position_size': 10000,
'price_info':           open   close
0       1.0860  1.0865
1       1.0865  1.0866
2       1.0866  1.0864
3       1.0864  1.0865
4       1.0865  1.0862
…          …      …
303912  1.0515  1.0520
303913  1.0520  1.0518
303914  1.0518  1.0518
303915  1.0518  1.0520
303916  1.0520  1.0520

[303917 rows x 2 columns]}
```

[15]:

```
logger = logging.getLogger("Calibration")
fqi_configuration = {
        'possible_actions': possible_actions(action_set),
        'max_iterations': max_iteration,
        'discount': discount,
        'sample_iterations': 1
}
print('Running FQI with parameters {parameters} ...'.
 ↪format(parameters=fqi_configuration))
if is_parallelizable(model):
        model.set_params(n_jobs=-1)
fqi = FQI(samples, model, fqi_configuration, logger)
```

```
fitted_model, q_norms, losses = fqi.run(**(run_parameters))
print('...Done')
```

```
Running FQI with parameters {'possible_actions': [<Position.L: 1>, <Position.S:
-1>, <Position.F: 0>], 'max_iterations': 5, 'discount': 0.99,
'sample_iterations': 1} …
…Done
```

[16]:

```
  # optimal policy applied to training
print('Applying optimal policy to training set...')
current_state_train = current_state_no_position.copy()
current_state_train['position'] = Position.F
if is_parallelizable(model):
    fitted_model.set_params(n_jobs=1)
policy = QPolicy(fitted_model)
optimal_state_train, optimal_actions_train = policy.apply(minutes,␣
 ↪current_state_train, possible_actions(action_set))
reward_train = UnrealizedReward(fee,position_size).
 ↪calculate(optimal_state_train,  optimal_actions_train,  price_info, minutes)
print('...Done')
```

```
Applying optimal policy to training set…
…Done
```

[17]:

```
print('Reading testing set from file  ...')
file_data_testing  = importing.import_df("2017-EURUSD-1m.csv")
print('...Done')
```

```
Reading testing set from file  …
…Done
```

[18]:

```
print('Applying talib functions to testing set: {talib} ...'.
 ↪format(talib=talib_fun))
file_data_testing_talib, talib_names  = importing.
 ↪apply_talib(file_data_testing,    talib_fun )
print('...Done')


print('Creating training dataframes for dates {start} to {end} ...'.
 ↪format(start=start_date,end=end_date))
current_state_testing,  next_state_testing, price_info_testing, ␣
 ↪minutes_testing   =  importing.create_tuples(start_date, end_date,␣
 ↪file_data_testing_talib, historical_observations,columns_with_historical,␣
 ↪columns_with_current+ talib_names, starting_time)

print('...Done')
```

```
Applying talib functions to testing set: [{'name': 'ATR', 'parameters':
{'timeperiod': 10}}] …
…Done
Creating training dataframes for dates 01/01 to 12/31 …
…Done
```

[19]:

```
print('Applying optimal policy to testing set...')
current_state_testing['position'] = Position.F
if is_parallelizable(model):
    fitted_model.set_params(n_jobs=1)
policy = QPolicy(fitted_model)
optimal_state, optimal_actions = policy.
 ↪apply(minutes_testing,current_state_testing,possible_actions(action_set))
print('...Done')

print('Calculating optimal reward achieved in testing set...')
reward_testing = UnrealizedReward(fee, position_size).
 ↪calculate(optimal_state,optimal_actions, price_info_testing, minutes_testing)
print('...Done')
```

```
Applying optimal policy to testing set…
…Done
Calculating optimal reward achieved in testing set…
…Done
```

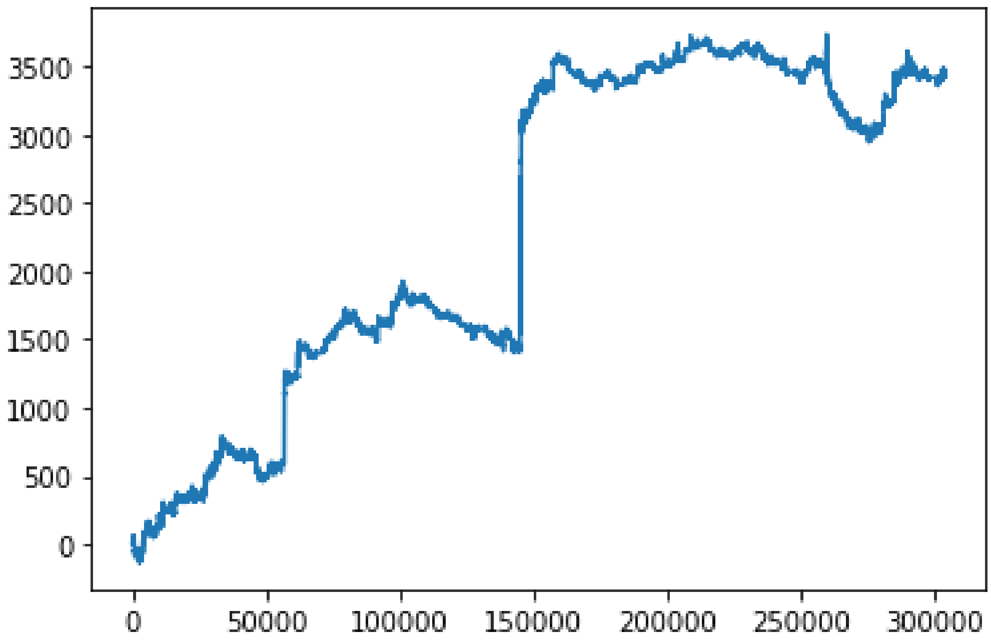
[22]:

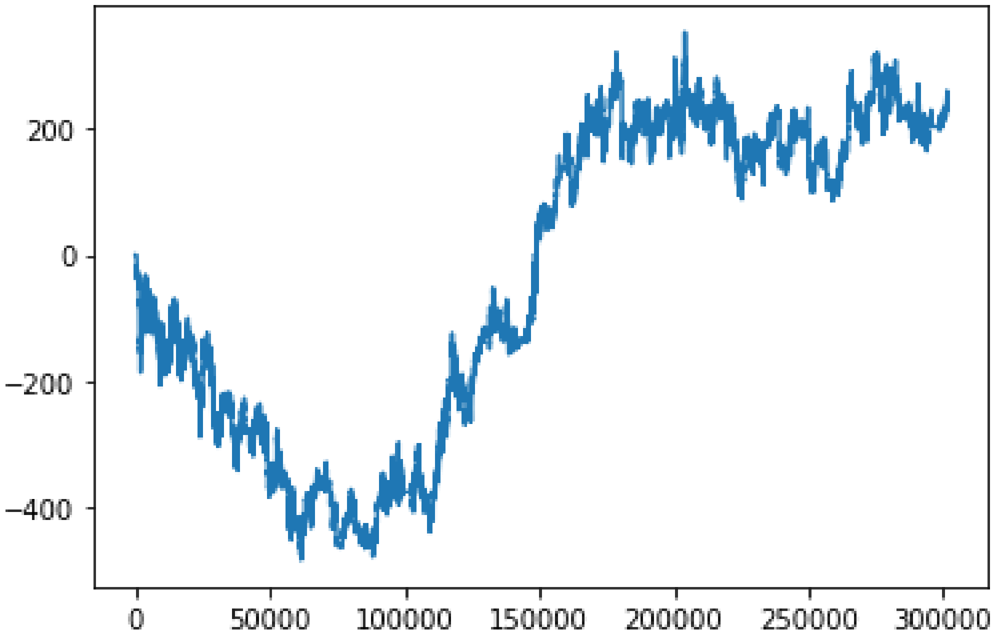
```
print('############################  Output  ############################')
print('Sum of optimal Reward(training set): {reward}'.
 ↪format(reward=sum(reward_train)))
print('Sum of optimal Reward(testing set): {reward}'.
 ↪format(reward=sum(reward_testing)))
print('Optimal action summary (training set): {summary}'.
 ↪format(summary=Counter(optimal_actions_train)))
print('Optimal action summary (testing set): {summary}'.
 ↪format(summary=Counter(optimal_actions)))
```

```
############################  Output  ############################
Sum of optimal Reward(training set): 3425.092967586491
Sum of optimal Reward(testing set): 236.15704161191402
Optimal action summary (training set): Counter({1.0: 192692, 0.0: 108988, -1.0:
2237})
Optimal action summary (testing set): Counter({1.0: 172632, 0.0: 127945, -1.0:
1025})
```

[23]:

```
plt.plot(np.cumsum(reward_train))
plt.show()
plt.plot(np.cumsum(reward_testing))
plt.show()
```


3500
3000
2500
2000
1500
1000
500
0
0
50000
100000
150000
200000
250000
300000


200
0
−200
−400
0
50000
100000
150000
200000
250000
300000

[ ]:

[ ]: