Fundamental Analysis of Securities Trading
(IV) Pairs Trading F

Kuan-Lun WANG\textsuperscript{12}
National Taiwan University
Preliminary Draft: Monday 25\textsuperscript{th} May, 2020

\textsuperscript{1}E-mail: d06922002@csie.ntu.edu.tw
\textsuperscript{2}Home Page: https://www.csie.ntu.edu.tw/~d06922002/

Short Biodata

- Research interests:
  - time series models.
  - simulation modeling.
  - portfolio choice.
- Central themes of my application:
  - multivariate pairs trading in real time.
  - assets searching with a long-run equilibrium.
  - riskless portfolio building.
- Current work:
  - cointegration test.
  - structural change analysis.
  - the probability estimation of mean reversion.
We want to write an OLS function. But

Wiki: Ordinary Least Squares

In statistics, ordinary least squares (OLS) is a type of linear least squares method for estimating the unknown parameters in a linear regression model. OLS chooses the parameters of a linear function of a set of explanatory variables by the principle of least squares: minimizing the sum of the squares of the differences between the observed dependent variable (values of the variable being predicted) in the given dataset and those predicted by the linear function.

https://en.wikipedia.org/wiki/Ordinary_least_squares

That is, we know the following information

1. there are some models;
   - one of model is linear regression.
2. there are some estimators of linear regression.
   - one of estimator is OLS.
Listing 1: example

```python
def fun():
    return 1,2,3
```

Listing 2: main

```python
if __name__ == '__main__':
    a=fun()  # a=(1,2,3) not a=1
    a,b,c=fun()  # a=1, b=2, c=3
    a,b=fun()  # Error
```

Design Principle [1]

Classes should be open for extension, but closed for modification.

Listing 3: example.py

```python
def fun():
    returnclass=funReturn(1,2,3)
    return returnclass

class funReturn():
    def __init__(self,a,b,c):
        self.a=a
        self.b=b
        self.c=c

if __name__ == '__main__':
    returnclass=fun()
    a,b,c=returnclass.a,returnclass.b,returnclass.c
    print(a,b,c)
```
Listing 4: example.py

```python
def fun():
    returnclass=funReturn(1,2,3,4)
    return returnclass

class funReturn():
    def __init__(self,a,b,c,d):
        self.a=a
        self.b=b
        self.c=c
        self.d=d

if __name__ == '__main__':
    returncalss=fun()
    a,b,c=returncalss.a,returncalss.b,returncalss.c
    print(a,b,c)  # No Error
```

Wiki: Factory Method Pattern

In class-based programming, the factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

https://en.wikipedia.org/wiki/Factory_method_pattern
Factory Method Pattern (2/2)

Figure: Factory Method Pattern

Python 初階證券交易分析  https://train.csie.ntu.edu.tw/train/course.php?id=2911
Factory Method Pattern A (2/4)

Listing 5: modelFactory

```python
class modelFactory (metaclass = _ABCMeta):
    @_abstractmethod
    def _getEstimatingDict (self):
        pass
    def estimatingCreate (self, estimatingKey):
        estimatingDict = self._getEstimatingDict()
        estimate = estimatingDict[estimatingKey.lower ()]
        return estimate
```

Factory Method Pattern A (3/4)

Listing 6: linearRegression

```python
class linearRegression (_modelFactory):
    def _getEstimatingDict (self):
        returnDict = {'ols': _estimator ().olsEstimate}
        return returnDict
```
Factory Method Pattern A (4/4)

Listing 7: estimatorFactory

```python
class estimatorFactory (metaclass = _ABCMeta):
    pass
```

Listing 8: linearRegressionEstimator

```python
class linearRegressionEstimator (_estimatorFactory):
    def olsEstimate (self , y , x , intercept = True):
        returnModel = _olsEstimator (). estimate (y , x ,
           ← intercept)
        return returnModel
```

Figure: Estimator Factory
Factory Method Pattern B (2/10)

Listing 9: estimatorInterface

```python
class estimatorInterface(metaclass=_ABCMeta):
    @_abstractmethod
def estimate():
    pass
```

Listing 10: linearRegressionEstimatorInterface

```python
class linearRegressionEstimatorInterface(
    _estimatorInterface, metaclass=_ABCMeta):
    pass
```

Factory Method Pattern B (3/10)

Listing 11: olsEstimator

```python
class olsEstimator(_linearRegressionEstimatorInterface):
    def estimate(self,y,x, intercept=True):
        x=_getX(x, intercept)
        y=_matrix(y)
        beta=_estBeta(y,x)
        epsilon=_estEpsilon(y,x, beta)
        y,x,beta, epsilon=_convert2list(y,x,beta, epsilon)
        returnclass=_olsEstimatedLR(y,x,beta, epsilon, intercept)
        return returnclass
```

Python 初階證券交易分析  https://train.csie.ntu.edu.tw/train/course.php?id=2911
Factory Method Pattern B (4/10)

Figure: Return Class

Listing 12: `returnModel`
```python
class returnModel (metaclass=_ABCMeta):
    @_abstractmethod
def __init__(self):
    pass
```

Listing 13: `returnLinearRegression`
```python
class returnLinearRegression(_returnModel):
    pass
```

Listing 14: `estimatedLinearRegression`
```python
class estimatedLinearRegression(_returnLinearRegression):
    pass
```
Listing 15: olsEstimatedLinearRegression

class olsEstimatedLinearRegression:
    def __init__(self, y, x, estBeta, estEpsilon, intercept):
        self.y = y
        self.x = x
        self.estBeta = estBeta
        self.estEpsilon = estEpsilon
        self.intercept = intercept
        esty = (matrix(x) * matrix(estBeta)).tolist()
        self.esty = esty
        self.estEpsilonSigma2 = matrixOperator.norm(matrix(estEpsilon))

Now, we do this class.

Listing 16: olsEstimator

class olsEstimator(_linearRegressionEstimatorInterface):
    def estimate(self, y, x, intercept=True):
        x = getX(x, intercept)
        y = matrix(y)
        beta = estBeta(y, x)
        epsilon = estEpsilon(y, x, beta)
        y, x, beta, epsilon = convert2list(y, x, beta, epsilon)
        returnclass = _olsEstimatedLR(y, x, beta, epsilon, intercept)
        return returnclass
If the linear regression has no intercept, then $x_t = x_t$.
If the linear regression has intercept, then $x_t = (1, x_t)$.

**Listing 17: olsEstimator/_getX**

```python
def _getX(x, intercept):
    x = _matrix(x)
    if intercept:
        x = _append(_ones(_size(x, 0), 1), x, 1)
    return x
```

We only need to use the closed form of OLS.

**Listing 18: olsEstimator/_estBeta**

```python
def _estBeta(y, x):
    beta = _inv(_transpose(x)*x)*_transpose(x)*y
    return beta
```

**Listing 19: olsEstimator/_estEpsilon**

```python
def _estEpsilon(y, x, beta):
    eps = y - x*beta
    return eps
```
The matrix type only in our library. We convert the matrix type to python list type.

**Listing 20: olsEstimator/_estBeta**

```python
1 def _convert2list(y,x,beta, epsilon):
2     y=y.tolist()
3     x=x.tolist()
4     beta=beta.tolist()
5     epsilon=epsilon.tolist()
6     return y,x,beta, epsilon
```

Figure: UML for Mode Factory