Short Biodata

Kuan-Lun Wang is a doctoral student majoring in generalized pairs trading. The main goal of his research is to develop an algorithmic trading mechanism based on statistical arbitrage. His areas of expertise include automatic search procedures for model selection, multivariate co-integration approach, and structural change test.
Kuan-Lun Wang’s research interests comprise time series models, simulation modeling, and portfolio choice. The central themes of his application are the study of multivariate pairs trading in real time, search for assets with a long-run equilibrium, and building of riskless portfolios. Much of his current work involves conducting structural change analysis and co-integration test of the finite order vector autoregressive process and estimating the probability of mean reversion. Such methods are important in a variety of applications, including economic indicators and hedging. One such application is index funds being tied to indexes with very low costs and risks.

**Assumption (1/4)**

Consider two asset prices has only one same principal factor. That is,

\[
\text{price}_A(t) = a_A + b_A \text{factor}(t) + \epsilon_A(t),
\]
\[
\text{price}_B(t) = a_B + b_B \text{factor}(t) + \epsilon_B(t).
\]

Assume \( \epsilon \) is stable.

**Warning**

Arbitrage pricing theory shows this assumption is not good. But why we assume the above? (Hint: see the first slide.)

https://en.wikipedia.org/wiki/Arbitrage_pricing_theory
Assumption (2/4)

If we buy $b_B$ A share and short $b_A$ at time $t_0$, then our cash flow is

$$-b_B \text{price}_A(t_0) - b_A \text{price}_B(t_0) = \left(-a_k b_B + a_B b_A\right) + \left(-b_k \epsilon_A(t_0) + b_k \epsilon_B(t_0)\right).$$

That is, the portfolio value is

$$\text{price}_{\text{portfolio}}(t) = \left(a_k b_B - a_B b_A\right) + \left(b_B \epsilon_A(t) - b_A \epsilon_B(t)\right).$$

Moreover, the value is independent of factor.

Assumption (3/4)

So, if we know the values $b_A$ and $b_B$, then we can build a portfolio independent of principal factor.

Similarly, we can consider any number of factor.

Why do we need to know this?
Assumption (4/4)

If we buy one this portfolio at time $t_0$ and sell it at time $t_1$, then our cash flow is

$$-(b_B \varepsilon_A(t_0) - b_A \varepsilon_B(t_0)) + (b_B \varepsilon_A(t_1) - b_A \varepsilon_B(t_1)).$$

That shows we only trade a stable noise. We can wait for the low value to buy it or wait for the high value to sell it.
Distance  Python  References

Estimation (2/21)

\[ y_t = a + bx_t + \epsilon_t \]

\[ \Rightarrow y_t = (1 \ x_t) \begin{pmatrix} a \\ b \end{pmatrix} + \epsilon_t \]

Definition: Linear Equation [1]

We define a linear equation in the \( n \) variables \( x_1, x_2, \ldots, x_n \) to be one that can be expressed in the form

\[ a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = b, \]

where \( a_1, a_2, \ldots, a_n \) and \( b \) are constants, and the \( a \)'s are not all zeros.

Estimation (3/21)

\[
\begin{align*}
 y_1 &= a + bx_1 + \epsilon_1 \\
 y_2 &= a + bx_2 + \epsilon_2 \\
 &\vdots \\
 y_T &= a + bx_T + \epsilon_T
\end{align*}
\]

\[ \Rightarrow \begin{pmatrix} y_1 \\ \vdots \\ y_T \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_T \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_T \end{pmatrix} \]

Definition: System of Linear Equations [1]

A finite set of linear equations is called a system of linear equations or, more briefly, a linear system.
### Example: a General Linear System [1]

A general linear system of \( m \) equations in the \( n \) unknowns \( x_1, x_2, \ldots, x_n \) can be written as

\[
\begin{align*}
    a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
    a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
    &\vdots \quad \vdots \quad \vdots \\
    a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m.
\end{align*}
\]

Moreover, it can be written as

\[
Ax = b,
\]

where \( A = [a_{ij}]_{m \times n} \), and \( b = (b_1, b_2, \ldots, b_m)' \).

---

### Definition: Transpose

The **transpose** of an \( m \times n \) matrix \( A \) is the \( n \times m \) matrix \( A' \), which is obtained from \( A \) by writing the rows of \( A \) as the columns of \( A' \).

### Example

\[
A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad \text{and} \quad A' = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}
\]
Definition: Invertible Matrix [1]

If $A$ is a square matrix, and if a matrix $B$ of the same size can be found such that $AB = BA = I$, then $A$ is said to be invertible (or non-singular) and $B$ is called an inverse of $A$. If no such matrix $B$ can be found, then $A$ is said to be singular.

Example: Invertible Matrix [1]

Let

$$
\begin{pmatrix}
2 & -5 \\
1 & 3
\end{pmatrix}
\quad \text{and} \quad
\begin{pmatrix}
3 & 5 \\
1 & 2
\end{pmatrix}
$$

Then

$$
AB = \begin{pmatrix}
2 & -5 \\
1 & 3
\end{pmatrix}
\begin{pmatrix}
3 & 5 \\
1 & 2
\end{pmatrix}
= \begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
= I
$$

$$
BA = \begin{pmatrix}
3 & 5 \\
1 & 2
\end{pmatrix}
\begin{pmatrix}
2 & -5 \\
1 & 3
\end{pmatrix}
= \begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
= I
$$

Thus, $A$ and $B$ are invertible and each is an inverse of the other.
Theorem: Exactly One Solution [1]

If $A$ is an invertible $n \times n$ matrix, then for each $n \times 1$ matrix $b$, the system of equations $Ax = b$ has exactly one solution, namely, $x = A^{-1}b$.

Consider

\[
\begin{pmatrix}
  y_1 \\
  \vdots \\
  y_T 
\end{pmatrix} = \begin{pmatrix}
  1 & x_1 \\
  \vdots & \vdots \\
  1 & x_T 
\end{pmatrix} \begin{pmatrix}
  a \\
  b 
\end{pmatrix}
\]

$\Rightarrow y = X\beta$

$\Rightarrow X'y = X'X\beta$

$\Rightarrow X'X\beta = X'y$

$\Rightarrow (X'X)^{-1}X'X\beta = (X'X)^{-1}X'y$

$\Rightarrow \beta = (X'X)^{-1}X'y$
However,

\[
\begin{pmatrix}
  y_1 \\
  \vdots \\
  y_T
\end{pmatrix}
= \begin{pmatrix}
  1 & x_1 \\
  \vdots & \vdots \\
  1 & x_T
\end{pmatrix}
\begin{pmatrix}
  a \\
  b
\end{pmatrix}
+ \begin{pmatrix}
  \epsilon_1 \\
  \vdots \\
  \epsilon_T
\end{pmatrix}
\]

\[
\Rightarrow y = X\beta + \epsilon \\
\Rightarrow X'y = X'X\beta + X'\epsilon \\
\Rightarrow (X'X)^{-1}X'y = \beta + (X'X)^{-1}X'\epsilon
\]

**Fundamental Assumptions**

The ordinary linear regression model is described by the equation

\[
y = X\beta + \epsilon,
\]

where

1. \(X\) is a non-stochastic \(n \times p\) matrix with \(p < n\);
2. the matrix \(X\) has rank \(p\), i.e. \(X\) is of full column rank;
3. the elements of the \(n \times 1\) vector \(y\) are observable random vectors;
4. the elements of the \(n \times 1\) vector \(\epsilon\) are non-observable random variables such that \(E[\epsilon] = 0\) and \(\text{Cov}[\epsilon] = \sigma^2 I_n\) with \(\sigma^2 > 0\). We will write \(\epsilon \sim (0, \sigma^2 I_n)\) for short.

The linear regression with fundamental assumptions is also called classical linear regression model.
Figure: A Classical Linear Regression Model Example

Figure: This $X$ can include commodities prices, product prices, etc.
Notation: Unsolvable Equation System

If we assume that the system of equations \( y = X\beta_* \) is solvable with respect to \( \beta_* \), then a solution \( \beta_0 \) clearly satisfies \( \| y - X\beta_0 \|^2 = 0 \). On the other hand, when we assume that \( y = X\beta_* \) is not solvable, then we can nonetheless determine a vector \( \hat{\beta} \) such that

\[
\| y - X\hat{\beta} \|^2 \leq \| y - X\beta_* \|^2
\]

for every vector \( \beta_* \in \mathbb{R}^p \).

Under the linear regression model, we interested in inference about the unknown parameters \( \beta \in \mathbb{R}^p \) and \( \sigma^2 > 0 \). Thus, the parameter space is given by \( \Theta = \mathbb{R}^p \times (0, \infty) \).

Definition: Least-Squares Solution

A vector \( \hat{\beta} \) is called least squares solution of \( y = X\beta_* \) if

\[
\| y - X\hat{\beta} \|^2 \leq \| y - X\beta_* \|^2
\]

for every vector \( \beta_* \in \mathbb{R}^p \).

If we consider \( \epsilon_* = y - X\beta_* \) as the residual vector of the solution \( \beta_* \), then the sum of squared residuals is minimized for \( \beta_* = \hat{\beta} \), so that \( \hat{\beta} \) has the smallest sum of squared residuals.
Theorem: Least-Squares Estimator

Under the linear regression model with fundamental assumptions, the function

\[ f(\beta_*) = \|y - X\beta_*\|^2 = (y - X\beta_*)'(y - X\beta_*) \]

is minimized for \( \beta_* = \hat{\beta} \), where \( \hat{\beta} = (X'X)^{-1}X'y \). Moreover, the vector \( \hat{\beta} \) is called ordinary least-squares estimator of \( \beta \).

Proposition: Chain Rule for Vector Differentiation [4]

Let \( \alpha \) and \( \beta \) be \((m \times 1)\) and \((n \times 1)\) vectors, respectively, and suppose \( h(\alpha) \) is \((p \times 1)\) and \( g(\beta) \) is \((m \times 1)\). Then, with \( \alpha = g(\beta) \),

\[ \frac{\partial h(g(\beta))}{\partial \beta'} = \frac{\partial h(\alpha)}{\partial \alpha'} \frac{\partial g(\beta)}{\partial \beta'} = \frac{\partial h(\alpha)}{\partial \alpha'} \frac{\partial g(\beta)}{\partial \beta'} . \]

Rules of Matrix Calculus [4]

- Let \( A \) be an \((m \times n)\) matrix and \( \beta \) be an \((n \times 1)\) vector. Then
  \[ \frac{\partial A\beta}{\partial \beta'} = A' \]
  and \( \frac{\partial \beta'A'}{\partial \beta} = A' \).

- Let \( A \) be \((m \times m)\) and \( \beta \) be \((m \times 1)\). Then
  \[ \frac{\partial \beta'A\beta}{\partial \beta'} (A + \beta'A)^' \beta \text{ and } \frac{\partial \beta'A\beta}{\partial \beta} = \beta'(A' + A) \].
Proof.

By the differentiating the function $f(\beta_*)$ with respect to $\beta_*,$

$$\frac{\partial f(\beta_*)}{\partial \beta_*} = \frac{\partial (y - X\beta_*)'(y - X\beta_*)}{\partial \beta_*} \frac{\partial (y - X\beta_*)}{\partial (y - X\beta_*)}$$

$$= (-2X')(y - X\beta_*)((I' + I))$$

$$= -2X'y + 2X'X\beta.$$ 

and $\partial^2 f(\beta_*)/\partial \beta_*^2 = 2X'X$ is positive definite. The solution is given by $\hat{\beta} = (X'X)^{-1}X'y$ if we put the right-hand side equal to 0 and solve for $\beta_*.$
**Definition: Convergence in Probability**

A series \( Y_1, \ldots, Y_n, \ldots \) of random vectors converges in probability to a fixed \( c \), if

\[
\forall i, \lim_{n \to \infty} \Pr \left[ |Y_{n,i} - c| > \epsilon \right] = 0
\]

for every \( \epsilon > 0 \). The symbol plim denotes convergence in probability.

**Definition: Consistent**

An estimator \( \hat{\beta} \) of \( \beta \) is called consistent for \( \beta \), if \( \text{plim}_{n \to \infty} \hat{\beta} = \beta \) holds true.

**Theorem: Consistency**

Under the linear regression model with fundamental assumptions, if \( \lim_{n \to \infty} \frac{1}{n} X'X = Q \), where \( Q \) is symmetric positive definite, then \( \hat{\beta} \) is consistent for \( \beta \). (We call this sufficient condition the asymptotic assumption.)

**Proof.**

Under the assumptions, \( (X'X)^{-1} = O(n^{-1}) \) and

\[
\text{Cov} \left[ \hat{\beta} \right] = \sigma^2 (X'X)^{-1} = O(n^{-1}).
\]

Since \( E[\hat{\beta}] = \beta \), \( \hat{\beta} \) is consistent for \( \beta \).
How can we measure the principal level of the factor?

\[
\begin{align*}
\text{price}_A(t) &= a_A + b_A \text{factor}(t) + \epsilon_A(t); \\
\text{price}_B(t) &= a_B + b_B \text{factor}(t) + \epsilon_B(t),
\end{align*}
\]

implies

\[
\text{price}_A(t) = \frac{a_A b_B - a_B b_A}{b_B} + \frac{b_A}{b_B} \text{price}_B(t) + \frac{b_B \epsilon_A(t) - b_A \epsilon_B(t)}{b_B}.
\]

The asset prices are linearly dependent. Right?

---

**Pearson Correlation Coefficient [2]**

If the standard deviations \( \sigma_1 \) and \( \sigma_2 \) are positive, then

\[
\rho = \frac{\text{Cov}(X_1, X_2)}{\sigma_1 \sigma_2}
\]

is called the Pearson correlation coefficient (PCC) of \( X_1 \) and \( X_2 \).
**Figure**: Examples of scatter diagrams with different values of correlation coefficient $\rho$.
Definition: Expectation [2] –Continued

If $X$ is a discrete random variable with p.m.f. $p(x)$ and

$$\sum_{x} |x| p(x) < \infty,$$

then the expectation of $X$ is

$$E(X) = \sum_{x} x p(x).$$


$$\text{Cov}(X, Y) = E((X - E(X))(Y - E(Y))).$$

The PCC measures the variance of $\epsilon$ in the following regression:

$$y = a + bx + \epsilon,$$

where $\epsilon \sim (0, \sigma_\epsilon^2)$. 
Function relation: \( y = f(x) \). i.e., \( x \rightarrow y \)

Statistical relation: \((y, x)\). i.e., \( x \leftarrow y \)
- \( x \) maybe imply \( y \).
- \( y = f(x) + \epsilon \), where \( \epsilon \) is a small error.
- if we known more information of \( x \) and \( y \), then we guess there is a function relation.

\[
\begin{align*}
\text{Var}[\text{price}_A(t)] &= \text{Var}[a_A + b_A \text{factor}(t) + \epsilon_A(t)] \\
&= b_A^2 \text{Var}[\text{factor}(t)] + \text{Var}[\epsilon_A(t)] \\
\text{Var}[\text{price}_B(t)] &= b_B^2 \text{Var}[\text{factor}(t)] + \text{Var}[\epsilon_B(t)] \\
\text{Cov}[\text{price}_A, \text{price}_B] &= \text{Cov}[b_A \text{factor}, b_B \text{factor}] \\
&+ \text{Cov}[b_A \text{factor}, \epsilon_B(t)] \\
&+ \text{Cov}[\epsilon_A(t), b_B \text{factor}] \\
&+ \text{Cov}[\epsilon_A(t), \epsilon_B] \\
&= b_A b_B \text{Var}[\text{factor}] + 0 + 0 + 0
\end{align*}
\]
Then
\[
|\rho| = \frac{\text{Cov}[\text{price}_A, \text{price}_B]}{\sqrt{\text{Var}[\text{price}_A] \text{Var}[\text{price}_B]}} = \sqrt{\frac{b_A^2 \text{Var}[	ext{factor}(t)]}{b_A^2 \text{Var}[	ext{factor}(t)] + \text{Var}[\epsilon_A]}} \sqrt{\frac{b_B^2 \text{Var}[	ext{factor}(t)]}{b_B^2 \text{Var}[	ext{factor}(t)] + \text{Var}[\epsilon_B]}}
\]

Clearly, \text{factor} is a high principal level factor if \( |\rho| \to 1 \).
Bernoulli Trial

In the theory of probability and statistics, a Bernoulli trial (or binomial trial) is a random experiment with exactly two possible outcomes, “success” and “failure”, in which the probability of success is the same every time the experiment is conducted. It is named after Jacob Bernoulli, a 17th-century Swiss mathematician, who analyzed them in his Ars Conjectandi (1713).

https://en.wikipedia.org/wiki/Bernoulli_trial
Example: Bernoulli Trial [3]

Out of millions of instant lottery tickets, suppose that 20% are winners. If five such tickets are purchased, then \((0, 0, 0, 1, 0)\) is a possible observed sequence in which the fourth ticket is a winner and the other four are losers. Assuming independence among winning and losing tickets, we observe that the probability of this outcome is

\[
(0.8)(0.8)(0.8)(0.2)(0.8) = (0.2)(0.8)^4.
\]
Negative Binomial Distribution

In probability theory and statistics, the negative binomial distribution is a discrete probability distribution of the number of successes in a sequence of independent and identically distributed Bernoulli trials before a specified (non-random) number of failures (denoted $r$) occurs. For example, if we define a 1 as failure, all non-1s as successes, and we throw a dice repeatedly until 1 appears the third time ($r = $ three failures), then the probability distribution of the number of non-1s that appeared will be a negative binomial distribution.

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

We observe a sequence of Bernoulli trials until exactly $r$ successes occur, where $r$ is a fixed positive integer. Let the random variable $X$ denote the number of trials needed to observe the $r$th success. The p.m.f. of $X$ is

\[
\binom{x - 1}{r - 1} p^r (1 - p)^{x-r}.
\]

We say that $X$ has a negative binomial distribution.
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1    | Search some pairs has high PCC in Top rank;  
      | $|\rho| \text{ vs. } \rho$ |
| 2    | Estimate the pairs share rate;  
      | How many samples do we need? |
| 3    | Use a technical indicator. |

Writing Process:
- Estimate the pairs share rate;
- Use a technical indicator;
- Search some pairs has high PCC in Top rank;
Kuan-Lun WANG  
Fundamental Analysis of Securities Trading

Third-Party (1/13)

- NumPy (license, free)
- SciPy (license, free)
- Matplotlib (license, free)

We must check the license!!!

Third-Party (2/13)

What is the adapter pattern?

Put simply, the adapter pattern is used to implement a light wrapper around third-party APIs, one that is contextually relevant for your codebase, and can withstand upstream changes or wholesale replacements of the vendor API without impacting the rest of your application. This manages the risk of integration by providing the rest of your codebase with consistent interface that you control. Additionally, the adapter provides an ideal test seam for stubbing out the service during testing.

https://bocoup.com/blog/adapter-pattern-a-must-for-vendor-service-integrations
The Adapter Pattern

The adapter pattern converts the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.

http://shop.oreilly.com/product/9780596007126.do

(a) Class Adapter

(b) Object Adapter

Figure: The Adapter Pattern
We combine `numpy.mat` and `numpy.array` to a new class. Moreover, for simple usable, we divide operations to a static class. Therefore, we use the object adapter but not the class adapter.

Warning

此堂課課程介紹寫初階課程, 我不知道為何初階課程上了沒多久後就開始要我們寫design pattern的轉接器模式, 這是什麼意思? 具我所知, design pattern是屬於高階的課程範疇而不是初階沒多久就馬上要我們寫design pattern, 不是嗎? 請問這樣王冠倫老師的課程安排沒有任何問題嗎?

If you don’t want to learn the technology, you can pause it. We not to test it in our class.

For example, we want to use:

- `matrix1 * matrix2 := matrix1 * matrix2`
  - `numpy: mat(matrix1) * mat(matrix2)`
- `transpose(matrix1) := transpose(matrix1)`
  - `numpy: mat(matrix1).transpose()`
- `matrix1 .* matrix2 := pointTimes(matrix1,matrix2)`
  - `numpy: array(matrix1) * array(matrix2)`
- `[matrix1;matrix2] := append(matrix1,matrix2,0)`
  - `numpy: concatenate((mat(matrix1),mat(matrix2)),axis=0)`
Listing 3: matrixTarget

```python
class matrixTarget(metaclass=abc.ABCMeta):
    def __init__(self,matrix):
        self._matrix=matrix
    def tolist(self):
        matrix=self._matrix
        return matrix
    @abc.abstractmethod
    # operations
```

Listing 4: matrixAdapter

```python
class matrixAdapter(_matrixTarget):
    # interface realize
```

Mapping Operators to Functions

This table shows how abstract operations correspond to operator symbols in the Python syntax and the functions in the operator module.

[https://docs.python.org/3/library/operator.html](https://docs.python.org/3/library/operator.html)
Abstract operation list of matrixTarget:

- `__repr__(self)`
- `__str__(self)`
- `__pos__(self)`
- `__add__(self, other)`
- `__radd__(self, other)`
- `__neg__(self)`
- `__sub__(self, other)`
- `__rsub__(self, other)`
- `__mul__(self, other)`
- `__rmul__(self, other)`
- `__pow__(self, other)`
- `__lt__(self, other)`
- `__le__(self, other)`
- `__eq__(self, other)`
- `__ne__(self, other)`
- `__gt__(self, other)`
- `__ge__(self, other)`
- `__getitem__(self, index)`
- `__len__(self)`
- `__abs__(self)`

Listing 5: matrixAdapter

```python
1 class matrixOperatorTarget(metaclass=abc.ABCMeta):
2     @abc.abstractmethod
3     # operations
```

Listing 6: matrixOperatorAdapter

```python
1 class matrixOperatorAdapter(_matrixOperatorTarget):
2     # interface realize
```
Abstract operation list of `matrixOperatorTarget`:

- `inv()`
- `det()`
- `pointTimes()`
- `size()`
- `norm()`
- `pointPower()`
- `append()`
- `vec()`
- `eig()`
- `transpose()`
- `kron()`
- `choleski()`

Two examples of interface realization:

**Listing 7: matrixAdapter**

```python
from numpy import mat as _matAdaptee

class matrixAdapter(_matrixStandard,_matrixTarget):
    def __add__(self,other):
        numpyMatrix0=_matAdaptee(self._matrix)
        if type(other)==matrixAdapter:
            numpyMatrix0=numpyMatrix0+_matAdaptee(other._matrix)
        else:
            numpyMatrix0=numpyMatrix0+other

        matrix=matrixAdapter(numpyMatrix0.tolist())
        return matrix
```
Listing 8: matrixAdapter

```python
class matrixOperatorAdapter(_matrixOperatorTarget):
    def append(matrix, other, dim):
        if type(other) != _matrixAdapter:
            other = _matrixAdapter(other)
        # convert matrix to mat
        a = _matrixConvert(matrix, _matAdaptee)
        b = _matrixConvert(other, _matAdaptee)
        if dim == 0:
            c = _concatenateAdaptee((a, b), axis=0)
        elif dim == 1:
            c = _concatenateAdaptee((a, b), axis=1)
        matrix = _matrixConvert(c, _matrixAdapter)
        return matrix
```

In usually, we generate, estimate, or test a model.

Listing 9: modelStandard

```python
class modelStandard(metaclass=_ABCMeta):
    @abstractmethod
    def generatingCreate():
        pass
    @abstractmethod
    def estimatingCreate():
        pass
    @abstractmethod
    def testingCreate():
        pass
```

Kuan-Lun WANG  Fundamental Analysis of Securities Trading
Listing 10: linearRegression

```python
class linearRegression(_modelStandard):
    def generatingCreate():
        pass
    def estimatingCreate(estimateName):
        returnFun={'ols':_olsEstimate
        returnFun=returnFun[estimateName.lower()]
        return returnFun
    def testingCreate():
        pass
```

Define a return type for this model.

Listing 11: estimatedModel

```python
class estimatedModel(metaclass=_ABCMeta):
    @abstractmethod
    def __init__(self):
        pass
```

Listing 12: estimatedLinearRegression

```python
class estimatedLinearRegression(_estimatedModel):
    pass
```
One of estimator is the OLS estimator.

**Listing 13: `olsEstimatedLinearRegression`**

```python
class olsEstimatedLinearRegression(
    _estimatedLinearRegression):
    def __init__(self, y, x, estBeta, estEpsilon, intercept):
        self.y = y
        self.x = x
        self.estBeta = estBeta
        self.estEpsilon = estEpsilon
        self.intercept = intercept
```

**Listing 14: `olsEstimate`**

```python
def olsEstimate(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 = _olsEstimatedLinearRegression(y, x, beta, epsilon, intercept)
    return returnclass
```
### Listing 15: `getX`

```python
def _getX(x, intercept):
    x = _matrix(x)
    if intercept:
        T = _size(x, 0)
        one = [[1] for t in range(T)]
        one = _matrix(one)
        x = _matrixOperator.append(one, x, dim=1)
    return x
```

### Listing 16: `estBeta`

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

### Listing 17: `estEpsilon`

```python
def _estEpsilon(y, x, beta):
    eps = y - x * beta
    return eps
```

### Listing 18: `convert2list`

```python
def _convert2list(y, x, beta, epsilon):
    y = y.tolist()
    x = x.tolist()
    beta = beta.tolist()
    epsilon = epsilon.tolist()
    return y, x, beta, epsilon
```
Listing 19: portfolio

```python
class portfolio(metaclass=ABCMeta):
    @abstractmethod
def __init__(self):
    pass
```

Listing 20: portfolio

```java
public abstract class portfolio
|- public abstract class buildedPortfolio < portfolio
|- public class buildedCorrelatedPortfolio
    ↪ buildedPortfolio
```

Listing 21: buildedPortfolio

```python
class buildedPortfolio(_portfolio):
    pass
```

Listing 22: buildedCorrelatedPortfolio

```python
class buildedCorrelatedPortfolio(_buildedPortfolio):
def __init__(self,assetsValue,weightVector,
             portfolioValue,rho,securityCodes=None):
    self.assetsValue=assetsValue
    self.weightVector=weightVector
    self.portfolioValue=portfolioValue
    self.rho=rho
    self.securityCodes=securityCodes
```
Listing 23: buildedPortfolioEstimate

def buildedPortfolioEstimate(securityCodes, prices):
    y, x, w, rho = _getWeight(prices)
    value = _getPortfolioValue(y, x, w)
    return_class = _buildedCorrelatedPortfolio(prices, w, value, rho, securityCodes)
    return return_class

Listing 24: _getPortfolioValue

def _getPortfolioValue(y, x, w):
    value = [w[0] * y[t][0] + w[1] * x[t][0] for t in range(len(y))]
    return value

Listing 25: _getWeight

def _getWeight(prices):
    x, y = _getXY(prices)
    rho = _corrcoef([xx[0] for xx in x], [yy[0] for yy in y])
    estimate = _linearRegression.estimatingCreate('ols')
    estimatedLinearRegression = estimate(y, x)
    beta = estimatedLinearRegression.estBeta
    w = [1, -beta[1][0]]
    return y, x, w, rho
**Listing 26: _getXY**

```python
def _getXY(prices):
    x = [[p[0]] for p in prices]
    y = [[p[1]] for p in prices]
    return x, y
```

**Listing 27: portfolio**

```java
public abstract class portfolio

|- public abstract class searchedPortfolio < portfolio

|- public class searchedCorrelatedPortfolio

  ↪ searchedPortfolio
```

**Listing 28: searchedPortfolio**

```python
class searchedPortfolio(_portfolio):
    pass
```
Listing 29: searchedCorrelatedPortfolio

class searchedCorrelatedPortfolio(_searchedPortfolio):
    def __init__(self, assetsValue, weightVector, portfolioValue, rho, securityCodes):
        self.assetsValue = assetsValue
        self.weightVector = weightVector
        self.portfolioValue = portfolioValue
        self.rho = rho
        self.securityCodes = securityCodes

Listing 30: searchedPortfolioEstimate

def searchedPortfolioEstimate(securityCodes, bigPriceTable, searchingNum, searchedTopRank, successProbability=0.95, checkProbability=0.05):
    takeSampleSize = _takeSampleSize(...)
    searchedPortfolioList = _topRankSearch(...)
    return searchedPortfolioList
Listing 31: _topRankSearch

```python
def _topRankSearch(securityCodes, bigPriceTable,
                   searchingNum, takeSampleSize):
    pts = _getPTSlist(takeSampleSize, len(bigPriceTable))
    buildedPortfolioEstimate = _buildingPortfolio.create
                                ('meanReverting.pts.correlation')
    buildedPortfolio = []
    for pts0 in pts:
        buildedPortfolio0 = _getBuildedPortfolio(
            securityCodes, pts0, bigPriceTable,
            buildedPortfolioEstimate)
        buildedPortfolio.append(buildedPortfolio0)
    ...
```

Listing 32: _topRankSearch

```python
def _topRankSearch(securityCodes, bigPriceTable,
                   searchingNum, takeSampleSize):
    ...
    rho = [buildedPortfolio0.rho for buildedPortfolio0
           in buildedPortfolio]
    indexList = _sortList(rho, returnType='index')
    indexList = indexList[-searchingNum:]
    searchedPortfolio = [_bp2sp(buildedPortfolio[i]) for
                         i in indexList]
    return searchedPortfolio
```
Listing 33: searchedPortfolioEstimate

```python
def _bp2sp(bp):
    sp = _searchedCorrelatedPortfolio(assetsValue=bp.assetsValue,
                                      weightVector=bp.weightVector,
                                      portfolioValue=bp.portfolioValue,
                                      rho=bp.rho,
                                      securityCodes=bp.securityCodes)
    return sp
```

Kuan-Lun WANG  
Fundamental Analysis of Securities Trading
Listing 34: _getBuildedPortfolio

```python
def _getBuildedPortfolio(securityCodes, pts, bigPriceTable, buildedPortfolioEstimate):
    price = []
    for t in range(len(bigPriceTable)):
        price0 = [bigPriceTable[t][pts[0]], bigPriceTable[t][pts[1]]]
        price.append(price0)
    price = _missingValues.deleteMissing(price)
    ptsName = [securityCodes[p] for p in pts]
    buildedPortfolio = buildedPortfolioEstimate(ptsName, price)
    return buildedPortfolio
```

Listing 35: _getPTSlist

```python
def _getPTSlist(num, Num):
    pts = []
onoff = True
    while onoff:
        pts0 = _randomPTS(Num)
        if pts0 not in pts:
            pts.append(pts0)
        if len(pts) == num:
            onoff = False
    return pts
```
Listing 36: \_randomPTS

```
def \_randomPTS(num):
    a=[\_randomIndex(num),\_randomIndex(num)]
    if a[0]>a[1]:
        a=[a[0],a[1]]
    if a[0]==a[1]:
        a=\_randomPTS(num)
    return a
```

Listing 37: \_randomIndex

```
def \_randomIndex(Num):
    num=\_floor(\_random()\*Num)
    return num
```
Listing 38: _takeSampleSize

```python
1 def _takeSampleSize(bigPriceTable, searchingNum,
2          searchedTopRank, successProbability,
3          checkProbability):
4     allSampleSize = \_nchoosek(len(bigPriceTable[0]), 2)
5     needSampleSize = searchingNum
6     needProbability = searchedTopRank
7     successProbability = successProbability
8     checkProbability = checkProbability
9     takeSampleSize = \_bernoulliSample(allSampleSize,
10           needSampleSize, needProbability,
11           successProbability, checkProbability)
12 return takeSampleSize
```

Usually, a technical indicator defines overbought and oversold implies mean reversing indicator. In this class, we use Bollinger bands because it is simple. (If there are some open sources, then use it.)
**Listing 39: technicalIndicatorsStandard**

```python
class technicalIndicatorsStandard(metaclass=abc.ABCMeta):
    def __init__(self, value0, parameter=None):
        value, index = self._init(value0, parameter)
        self.value = value
        self.index = index
        @abc.abstractmethod
    def _init(self, value0, parameter):
        return value, index
```

**Listing 40: bollingerBands**

```python
class bollingerBands(_technicalIndicatorsStandard):
    def __init__(self, value0, parameter):
        ma = parameter['ma']
        k = parameter['k']
        value = immutableList(value0)  # copy value0
        upperBB, lowerBB, middleBB = _BBands(value0, ma, k)
        upperBB = [[upperBB[t]] for t in range(len(value))]
        lowerBB = [[lowerBB[t]] for t in range(len(value))]
        middleBB = [[middleBB[t]] for t in range(len(value))]
        index = {'upperBB': upperBB, 'lowerBB': lowerBB, 'middleBB': middleBB}
        return value, index
```
Listing 41: BBands

```python
def BBands(y, ma, k):
    # copy y
    y = _immutableList(y)
    if type(y[0]) == list:
        y = [v[0] for v in vec]
    upperBB = []
    lowerBB = []
    middleBB = []
    for n in range(len(y)):
        if n < ma:
            upperBB0 = None
            lowerBB0 = None
            may0 = None
        else:
            y0 = y[n-ma+1:n+1]
            may0 = sum(y0)/len(y0)
            std0 = statistics.stdev(y0)
            upperBB0 = may0 + k*std0
            lowerBB0 = may0 - k*std0
            upperBB.append(upperBB0)
            lowerBB.append(lowerBB0)
            middleBB.append(may0)
    return upperBB, lowerBB, middleBB
```

Listing 42: BBands

```python
def BBands(y, ma, k):
    ...
    else:
        y0 = y[n-ma+1:n+1]
        may0 = sum(y0)/len(y0)
        std0 = statistics.stdev(y0)
        upperBB0 = may0 + k*std0
        lowerBB0 = may0 - k*std0
        upperBB.append(upperBB0)
        lowerBB.append(lowerBB0)
        middleBB.append(may0)
    return upperBB, lowerBB, middleBB
```
First, we select a database folder path.

Listing 43: Stage 1

```python
1 crawler=twseCrawler(sys.argv[2])  # our database path
2 yyyymmddInterval=[20190101,20190228]
```

Second, we load security code.

Listing 44: Stage 2

```python
1 print('securityCode')
2 securityCode=crawler.create('securityCode')
3 securitycode=securityCode.read('Security Code')
4 [print(v) for v in securitycode[-5:]]
5 print()
```

Warning
The backtesting slide shows we need to use adjusted price or custom price.

Third, we load closing price.

Listing 45: Stage 3

```python
1 print('securityCode')
2 securityCode=crawler.create('securityCode')
3 securityCode=securityCode.read('Security Code')
4 [print(v) for v in securitycode[-5:]]
5 print()
```
Fourth, we search one PTS by PCC.

**Listing 46: Stage 4**

```python
1 searchedPortfolioEstimate = searchingPortfolio.create('meanReverting.pts.correlation')
2 searchedPortfolioList = searchedPortfolioEstimate(
    securitycode, price, searchingNum=1,
    searchedTopRank=0.1)
3 searchedPortfolio = searchedPortfolioList[0]
4 ptsName = searchedPortfolio.securityCodes
5 assetsValue = searchedPortfolio.assetsValue
6 portfolioValue = searchedPortfolio.portfolioValue
7 weightVector = searchedPortfolio.weightVector
8 rho = searchedPortfolio.rho
9 print('{name0} and {name1} has PCC={r:.4f}.'.format(
    name0=ptsName[0], name1=ptsName[1], r=rho))
```

Fifth, we calculate the technical indicator.

**Listing 47: Stage 6**

```python
1 technicalIndicators = bollingerBands(portfolioValue, {'ma': 5, 'k': 2})
2 upperBB = technicalIndicators.index['upperBB']
3 lowerBB = technicalIndicators.index['lowerBB']
4 middleBB = technicalIndicators.index['middleBB']
```
Sixth, we plot the figure.

**Listing 48: Stage 6**

```python
fig, axs = subplots(2, 1, constrained_layout=True)
pricey = [price[0] for price in assetsValue]
pricex = [price[1] for price in assetsValue]
date = [n for n in range(len(pricex))]
axs[0].plot(date, zscore(pricey))
axs[0].plot(date, zscore(pricex))
axs[0].set_title('(a) Securities Value')
axs[0].legend(tuple(ptsName), loc='upper right')
axs[0].set_xlabel('From s to e'.format(s=yyyymmddInterval[0], e=yyyymmddInterval[1]))
axs[0].set_ylabel('Z-Score')
axs[0].set_xlim([date[0], date[-1]])
...
```

**Listing 49: Stage 6**

```python
... 
axs[1].set_title('(b) Portfolios Value {w[0]}:{w[1]:.4f}'.format(w=weightVector))
axs[1].plot(date, portfolioValue)
axs[1].plot(date, upperBB)
axs[1].plot(date, lowerBB)
axs[1].plot(date, middleBB)
axs[1].set_ylabel('Value')
axs[1].set_xlabel('From s to e'.format(s=yyyymmddInterval[0], e=yyyymmddInterval[1]))
axs[1].set_xlim([date[0], date[-1]])
show()
```
Figure: A Random Searching PTS by PCC


