Time Series Analysis in Python

Generated by Doxygen 1.8.13

Contents

Chapter 1

Time Series Analysis in Python

TSAP is a python package that provides tools for time series analysis in financial data.

Given input of a stock price series, the system will fit time series models, estimate the parameters and do statistical inference. With the identified model, the system predict the future price and assess the prediction accuracy. We can further consider trading strategy and option pricing. Moreover, given the input of multiple stock prices, the system can implement clustering and build a reduced order model for price prediction.

Installation

1. Download TSAP package from GitHub: `git clone https://github.com/APC524/tsap.git`
2. Add the folder `tsap` into your Python search path.

Functionality

TSAP package provides six Python classes.

1. **AR**: the autoregressive model to fit the imput stock price series, computing the log-likelihood and the gradient.
2. **MR**: the moving average model.
3. **Solver**: estimate the model parameters given the model class and the optimization method.
4. **OptionPricing**: calculate the option price given the underlying stock.
5. **Cluster**: impelement the clustering of multiple stock price series.
6. **Reduction**: build a reduced order model for price prediction.

Following is the high-level program structure figure.

Documents and demos

- The `Project Report` explains the detail of the whole project.
- The `User Manual` gives a brief introduction of the functionality of the package.
- The user can also generate Doxygen HTML and LaTeX manuals with `Doxyfile`, using the command `doxygen Doxyfile`.
- In `demo` folder there are several examples showing how to use the package.

Contributors

This is the course project of *APC524/MAE560 Software Engineering for Scientific Computing* (Fall 2016) in Princeton University. The project members are Wenyan Gong, Zongxi Li, Cong Ma, Qingcan Wang, Zhuoran Yang and Hao Zhang. We would appreciate Professor Stone and Assistant Instructor Jeffry and Bernat for their guidance and help.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

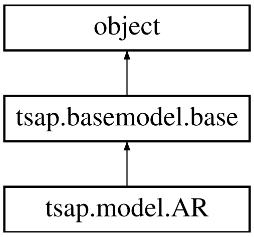
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 tsap.model.AR Class Reference

Inheritance diagram for tsap.model.AR:



Public Member Functions

- def __init__ (self, lag, phi, sigma, intercept)
- def loss (self, X, lag=None, phi=None, sigma=None, intercept=None)
- def predict (self, X, nstep, lag=None, phi=None, sigma=None, intercept=None)

Public Attributes

- **params**

4.1.1 Detailed Description

```
class AR implements the AR model which has __init__ , loss and predict as functions
```

4.1.2 Constructor & Destructor Documentation

4.1.2.1 __init__()

```
def tsap.model.AR.__init__ (
              self,
              lag,
              phi,
              sigma,
              intercept )
```

```
__init__: initialize the model with lag phi sigma and intercept
   Input:
 lag: the number of lag in AR model, dimension 1
 phi: the coefficients for each lag, dimension lag, column vector
 sigma: the standard deviation of the error term, dimension 1
 intercept: the constant component in the AR model, dimension 1
   Output:
 _lag: the number of lag in the AR model
 params: hash table of phi, sigma and intercept
```

4.1.3 Member Function Documentation

4.1.3.1 loss()

```
def tsap.model.AR.loss (
              self,
              X,
              lag = None,
              phi = None,
              sigma = None,
              intercept = None )
```

```
loss: return the loglikelihood and its gradient with respect to phi, sigma and intercept
   Input:
 X: the input time series, each row is about one stock. For one stock, X is a row vector. Note phi is a column
   Output:
  loglikelihood: the loglikelihood that calculated from the input time series
  grads: hash table that records the gradient of phi sigma and intercept
```

```
the number of samples, usually it’s about how many stocks we have
```

4.1.3.2 predict()

```
def tsap.model.AR.predict (
              self,
              X,
              nstep,
              lag = None,
              phi = None,
              sigma = None,
              intercept = None )
```

```
predict: return the predicted series based on the samples given
   Input:
 X: the input time series, each row is about one stock. For one stock, X is a row vector. Note phi is a colum
 nstep: the number of steps to predict
   Output:
  pred_state: the predicted series based on AR model, which is a row vector
```

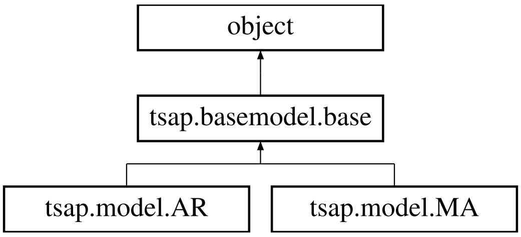
```
parameters
```

The documentation for this class was generated from the following file:

- /u/qingcanw/Programs/tsap/tsap/model.py

4.2 tsap.basemodel.base Class Reference

Inheritance diagram for tsap.basemodel.base:



Public Member Functions

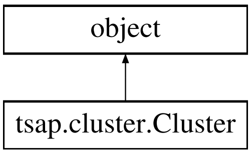
- def **__init__** (self)
- def **loss** (self, X)
- def **predict** (self, X, nstep)

The documentation for this class was generated from the following file:

- /u/qingcanw/Programs/tsap/tsap/basemodel.py

4.3 tsap.cluster.Cluster Class Reference

Inheritance diagram for tsap.cluster.Cluster:



Public Member Functions

- def __init__ (self, X)
- def assign_label (self, Centers)
- def kMeans (self, nClusters, maxIter=300)
- def H_clustering (self, nClusters)
- def **Gaussian_mixture** (self, nClusters, max_iter=300)
- def Spectral (self, nClusters=5, cluster_metric='euclidean', sigma=0.05)

4.3.1 Constructor & Destructor Documentation

4.3.1.1 __init__()

```
def tsap.cluster.Cluster.__init__ (
              self,
              X )
```

```
Return a new object to cluster data based on selected clutering
algorithm.
Example usage: clusterObj = Cluster(X)
X:     numpy array, shape (n_samples, n_features)
```

4.3.2 Member Function Documentation

4.3.2.1 assign_label()

```
def tsap.cluster.Cluster.assign_label (
              self,
              Centers )
```

```
Assign labels to the data points
    Input:
self
Centers:  numpy array, shape (n_clusters, n_features) the centers of each cluster
    Output:
clusters: the index of data points in each class
labels: the label of each class
```

4.3.2.2 H_clustering()

```
def tsap.cluster.Cluster.H_clustering (
              self,
              nClusters )
```

```
Performe hierarchical clustering
```

4.3.2.3 kMeans()

```
def tsap.cluster.Cluster.kMeans (
              self,
              nClusters,
              maxIter = 300 )
```

```
K-means clustering algorithm.

Function usage: kMeans(nClusters, maxIter, nInit)

Inputs:
nClusters : int
    The number of clusters to form as well as the number of
    centroids to generate.
maxIter : int, optional, default 300
    Maximum number of iterations of the k-means algorithm to run.

Returns:
centroid :  float ndarray with shape (k, n_features)
    Centroids found at the last iteration of k-means.
label : integer ndarray with shape (n_samples,)
    label[i] is the code or index of the centroid the i-th
    observation is closest to.
clusters : identity of the data point in the cluster
```

4.3.2.4 Spectral()

```
def tsap.cluster.Cluster.Spectral (
              self,
              nClusters = 5,
              cluster_metric = 'euclidean',
              sigma = 0.05 )
```

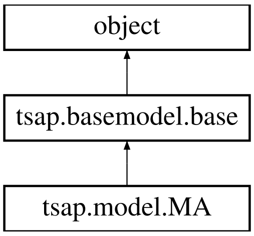
```
Spectral Clustering
    cluster_metric is the metric used to compute the affinity matrix

    sigma is the standard deviation used in the Gaussian kernel
```

The documentation for this class was generated from the following file:

- /u/qingcanw/Programs/tsap/tsap/cluster.py

4.4 tsap.model.MA Class Reference

Inheritance diagram for tsap.model.MA:



Public Member Functions

- def __init__ (self, lag, phi, sigma, intercept)
- def **loss** (self, X, lag=None, phi=None, sigma=None, intercept=None)
- def get_loglikelihood (self, X, lag=None, phi=None, sigma=None, intercept=None)
- def predict (self, X, nstep)

Public Attributes

- **params**

4.4.1 Constructor & Destructor Documentation

4.4.1.1 __init__()

```
def tsap.model.MA.__init__ (
              self,
              lag,
              phi,
              sigma,
              intercept )
```

```
lag, phi, sigma, intercept is the parameter of AR
```

4.4.2 Member Function Documentation

4.4.2.1 get_loglikelihood()

```
def tsap.model.MA.get_loglikelihood (
              self,
              X,
              lag = None,
              phi = None,
              sigma = None,
              intercept = None )
```

```
X is dataset, right now X is a row vector
```

```
phi is a column vector, and we need to make it into matrix form
```

4.4.2.2 predict()

```
def tsap.model.MA.predict (
              self,
              X,
              nstep )
```

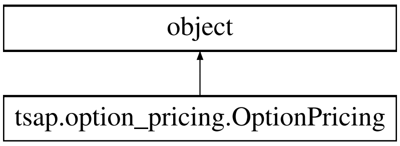
```
X is a row vector
```

The documentation for this class was generated from the following file:

- /u/qingcanw/Programs/tsap/tsap/model.py

4.5 tsap.option_pricing.OptionPricing Class Reference

Inheritance diagram for tsap.option_pricing.OptionPricing:



Public Member Functions

- def **__init__** (self, sigma=0.1, r=0.01, T=1, K=1, Smax=None)
- def solve_black_scholes (self, nS, nt)
- def get_option_price (self, S, t)

Public Attributes

- **sigma**
- **r**
- **T**
- **K**
- **Smax**
- **V**

4.5.1 Detailed Description

```
Callable option pricing object.
Example usage:
optionPriceobj = optionPricing(sigma,r,T,K), declare a class object
V = optionPriceobj.BlackScholesEqn(dS,dt), compute V array in shape [nS,nt]
Vst = optionPriceobj.optionPrice(V,S,t), compute V(S,t) given V array
```

4.5.2 Member Function Documentation

4.5.2.1 get_option_price()

```
def tsap.option_pricing.OptionPricing.get_option_price (
              self,
              S,
              t )
```

```
Compute V at V(S,t) by simple interpolation
Inputs:
V: array in shape [nS, nt]
S: stock price
t: time
Returns:
VSt: option price V(S,t)
```

4.5.2.2 solve_black_scholes()

```
def tsap.option_pricing.OptionPricing.solve_black_scholes (
              self,
              nS,
              nt )
```

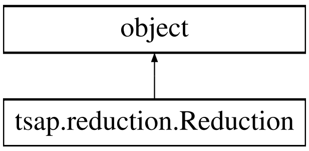
```
V(S, t) satisfies Black-Scholes equation
dVdt + (1/2)*sigma^2*S^2*d2VdS2 + r*S*dVdS - r*V = 0

Inputs:
nS: int, size of grids in S dimension
nt: int, size of grids in t dimension
```

The documentation for this class was generated from the following file:

- /u/qingcanw/Programs/tsap/tsap/option_pricing.py

4.6 tsap.reduction.Reduction Class Reference

Inheritance diagram for tsap.reduction.Reduction:



Public Member Functions

- def **__init__** (self, X)
- def PCA (self, n_components=None)
- def ICA (self, n_components, gfunc='logcosh', tol=1e-4, max_iter=200)
- def DMD (self, n_components=None)

4.6.1 Detailed Description

```
Callable modal reduction object.
Example usage:
xreduction = Reduction(X), X shape [n_features, n_samples], make sure X is
zero-mean
xmean, ux, at, energy_content = xreduction.PCA(n_components=3)
```

4.6.2 Member Function Documentation

4.6.2.1 DMD()

```
def tsap.reduction.Reduction.DMD (
              self,
              n_components = None )
```

```
Dynamic mode decomposition(DMD) of time series data x(k), find square
matrix A such that x(k+1) = Ax(k). Find eigendecomposition of A, and
corresponding DMD modes, and DMD eigenvalues.
```

4.6.2.2 ICA()

```
def tsap.reduction.Reduction.ICA (
              self,
              n_components,
              gfunc = 'logcosh',
              tol = 1e-4,
              max_iter = 200 )
```

```
Independent component analysis(ICA) of data in matrix X
Inputs:
n_components: integer, number of independent components
gfunc: string, ’logcosh’ or ’exp’, default ’logcosh’, Non-gaussian function
tol: float, tolerance of iteration, default 1e-4
max_iter: integer, maximum iteration steps, default 200
Returns:
Ex: array, mean of data
T: array [n_features, n_features], whitening matrix, st, xtilde = Tx
A: array [n_features, n_components], mixing matrix, st, xtilde = As
W: array [n_components, n_features], orthogonal rows, unmixing matrix, st, W = inv(A), s = W*xtilde
S: array, [n_components, n_samples], source data, st, S = W*Xtilde
```

4.6.2.3 PCA()

```
def tsap.reduction.Reduction.PCA (
              self,
              n_components = None )
```

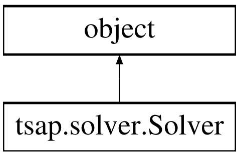
```
Principal component analysis (PCA) of data in matrix
Inputs:
n_components: integer, number of principal components
Returns:
ux: principal components
at: principal components coefficients
energy_content: energy content percentage in the principal components
```

The documentation for this class was generated from the following file:

- /u/qingcanw/Programs/tsap/tsap/reduction.py

4.7 tsap.solver.Solver Class Reference

Inheritance diagram for tsap.solver.Solver:



Public Member Functions

- def **__init__** (self, model, data, kwargs)
- def **train** (self)

Public Attributes

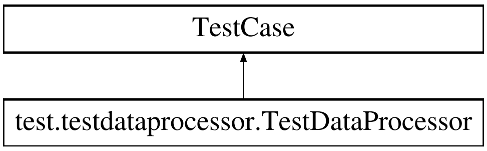
- **model**
- **X**
- **update_rule**
- **optim_config**
- **batch_size**
- **num_epochs**
- **print_every**
- **epoch**
- **loss_history**
- **optim_configs**

The documentation for this class was generated from the following file:

- /u/qingcanw/Programs/tsap/tsap/solver.py

4.8 test.testdataprocessor.TestDataProcessor Class Reference

Inheritance diagram for test.testdataprocessor.TestDataProcessor:



Public Member Functions

- def testGetReturn1 (self)
- def testGetReturn2 (self)
- def testGetReturn3 (self)
- def testGetReturn4 (self)
- def testGetPrice1 (self)
- def testGetPrice2 (self)
- def testGetPrice3 (self)
- def testMaxDrawdown1 (self)
- def testMaxDrawdown2 (self)
- def testMaxDrawdown3 (self)
- def testGetIndicator1 (self)
- def testGetIndicator2 (self)
- def testGetIndicator3 (self)
- def testGetIndicator4 (self)

4.8.1 Member Function Documentation

4.8.1.1 testGetIndicator1()

```
def test.testdataprocessor.TestDataProcessor.testGetIndicator1 (
              self )
```

```
test get_indicator with upper trend
```

4.8.1.2 testGetIndicator2()

```
def test.testdataprocessor.TestDataProcessor.testGetIndicator2 (
              self )
```

```
test get_indicator with lower trend
```

4.8.1.3 testGetIndicator3()

```
def test.testdataprocessor.TestDataProcessor.testGetIndicator3 (
              self )
```

```
test get_indicator without trend, trough before peak
```

4.8.1.4 testGetIndicator4()

```
def test.testdataprocessor.TestDataProcessor.testGetIndicator4 (
              self )
```

```
test get_indicator without trend, trough after peak
```

4.8.1.5 testGetPrice1()

```
def test.testdataprocessor.TestDataProcessor.testGetPrice1 (
              self )
```

```
test get_price with a row vector whose elements are all 1.0
```

4.8.1.6 testGetPrice2()

```
def test.testdataprocessor.TestDataProcessor.testGetPrice2 (
              self )
```

```
test get_price with a row vector whose elements are not the same
```

4.8.1.7 testGetPrice3()

```
def test.testdataprocessor.TestDataProcessor.testGetPrice3 (
              self )
```

```
test get_price with a row vector whose elements are not the same, can be negative
```

4.8.1.8 testGetReturn1()

```
def test.testdataprocessor.TestDataProcessor.testGetReturn1 (
              self )
```

```
test get_return with a row vector whose elements are all 1.0
```

4.8.1.9 testGetReturn2()

```
def test.testdataprocessor.TestDataProcessor.testGetReturn2 (
              self )
```

```
test get_return with a row vector whose elements are not the same
```

4.8.1.10 testGetReturn3()

```
def test.testdataprocessor.TestDataProcessor.testGetReturn3 (
              self )
```

```
test get_return with a matrix
```

4.8.1.11 testGetReturn4()

```
def test.testdataprocessor.TestDataProcessor.testGetReturn4 (
              self )
```

```
test get_return with a larger matrix
```

4.8.1.12 testMaxDrawdown1()

```
def test.testdataprocessor.TestDataProcessor.testMaxDrawdown1 (
              self )
```

```
test max_drawdown with upper trend
```

4.8.1.13 testMaxDrawdown2()

```
def test.testdataprocessor.TestDataProcessor.testMaxDrawdown2 (
              self )
```

```
test max_drawdown with lower trend
```

4.8.1.14 testMaxDrawdown3()

```
def test.testdataprocessor.TestDataProcessor.testMaxDrawdown3 (
              self )
```

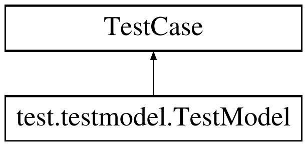
```
test max_drawdown with peak and trough
```

The documentation for this class was generated from the following file:

- /u/qingcanw/Programs/tsap/test/testdataprocessor.py

4.9 test.testmodel.TestModel Class Reference

Inheritance diagram for test.testmodel.TestModel:



Public Member Functions

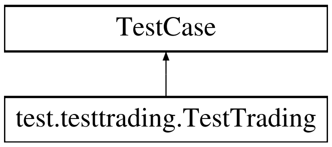
- def **testARloglklh1** (self)
- def **testARlklh2** (self)
- def **testARgrad1** (self)
- def **testARgrad2** (self)
- def **testMAloglklh1** (self)
- def **testMAloglklh2** (self)

The documentation for this class was generated from the following file:

- /u/qingcanw/Programs/tsap/test/testmodel.py

4.10 test.testtrading.TestTrading Class Reference

Inheritance diagram for test.testtrading.TestTrading:



Public Member Functions

- def testSignalGeneration1 (self)
- def testSignalGeneration2 (self)
- def testSignalGeneration3 (self)
- def testSignalGeneration4 (self)
- def testSignalGeneration5 (self)
- def testProfitLoss1 (self)
- def testProfitLoss2 (self)
- def testProfitLoss3 (self)
- def testProfitLoss4 (self)
- def testTrade1 (self)

4.10.1 Member Function Documentation

4.10.1.1 testProfitLoss1()

```
def test.testtrading.TestTrading.testProfitLoss1 (
              self )
```

```
test profit_loss with upper trend, this is immediate buy
```

4.10.1.2 testProfitLoss2()

```
def test.testtrading.TestTrading.testProfitLoss2 (
              self )
```

```
test profit_loss with upper trend, this is immediate buy
```

4.10.1.3 testProfitLoss3()

```
def test.testtrading.TestTrading.testProfitLoss3 (
              self )
```

```
test profit_loss with longer holding period
```

4.10.1.4 testProfitLoss4()

```
def test.testtrading.TestTrading.testProfitLoss4 (
              self )
```

```
test profit_loss with longer holding period, multiple trades and more money
```

4.10.1.5 testSignalGeneration1()

```
def test.testtrading.TestTrading.testSignalGeneration1 (
              self )
```

```
test signal_generation with upper trend
```

4.10.1.6 testSignalGeneration2()

```
def test.testtrading.TestTrading.testSignalGeneration2 (
              self )
```

```
test signal_generation with lower trend
```

4.10.1.7 testSignalGeneration3()

```
def test.testtrading.TestTrading.testSignalGeneration3 (
              self )
```

```
test signal_generation without trend
```

4.10.1.8 testSignalGeneration4()

```
def test.testtrading.TestTrading.testSignalGeneration4 (
              self )
```

```
test signal_generation with bigger window
```

4.10.1.9 testSignalGeneration5()

```
def test.testtrading.TestTrading.testSignalGeneration5 (
              self )
```

```
test signal_generation with bigger holding period
```

4.10.1.10 testTrade1()

```
def test.testtrading.TestTrading.testTrade1 (
              self )
```

```
test trade with a very simple model
```

The documentation for this class was generated from the following file:

- /u/qingcanw/Programs/tsap/test/testtrading.py