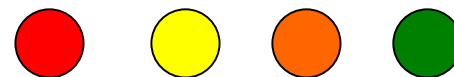


# 时间序列建模与金融量化交易应用

## 期中大作业布置 (C15)

信息科学与技术学院

胡俊峰



# 内容

- 自回归模型与金融量化分析
- 期中大作业布置



# 线性回归-多项式回归

- 一元线性回归
- 一元高次函数回归（拟合）
- N元高次函数回归（略）



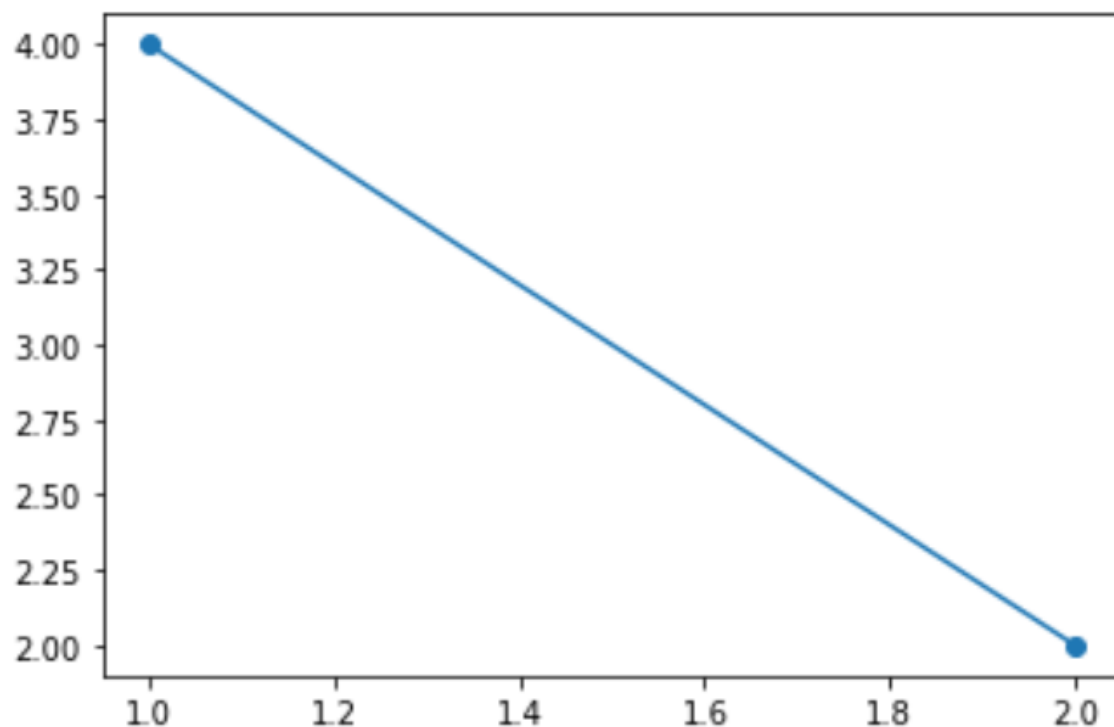
```
from sklearn.linear_model import LinearRegression

x = np.array([1, 2]) # 两个样本点
y = np.array([4, 2])

X = x[:, np.newaxis] # 转换为一列数据

model = LinearRegression().fit(X, y) # 线性回归一条直线
yfit = model.predict(X)

plt.scatter(x, y) # 画散点
plt.plot(x, yfit); # x及模型输出的预测结果 $\hat{y}$  画线
```



## 超定方程组 最小二乘回归

#多个样本点, 最小二乘拟合

```
x = np.array([1, 2, 3, 4, 5, 6])
```

```
y = np.array([4, 2, 1, 3, 7, 3])
```

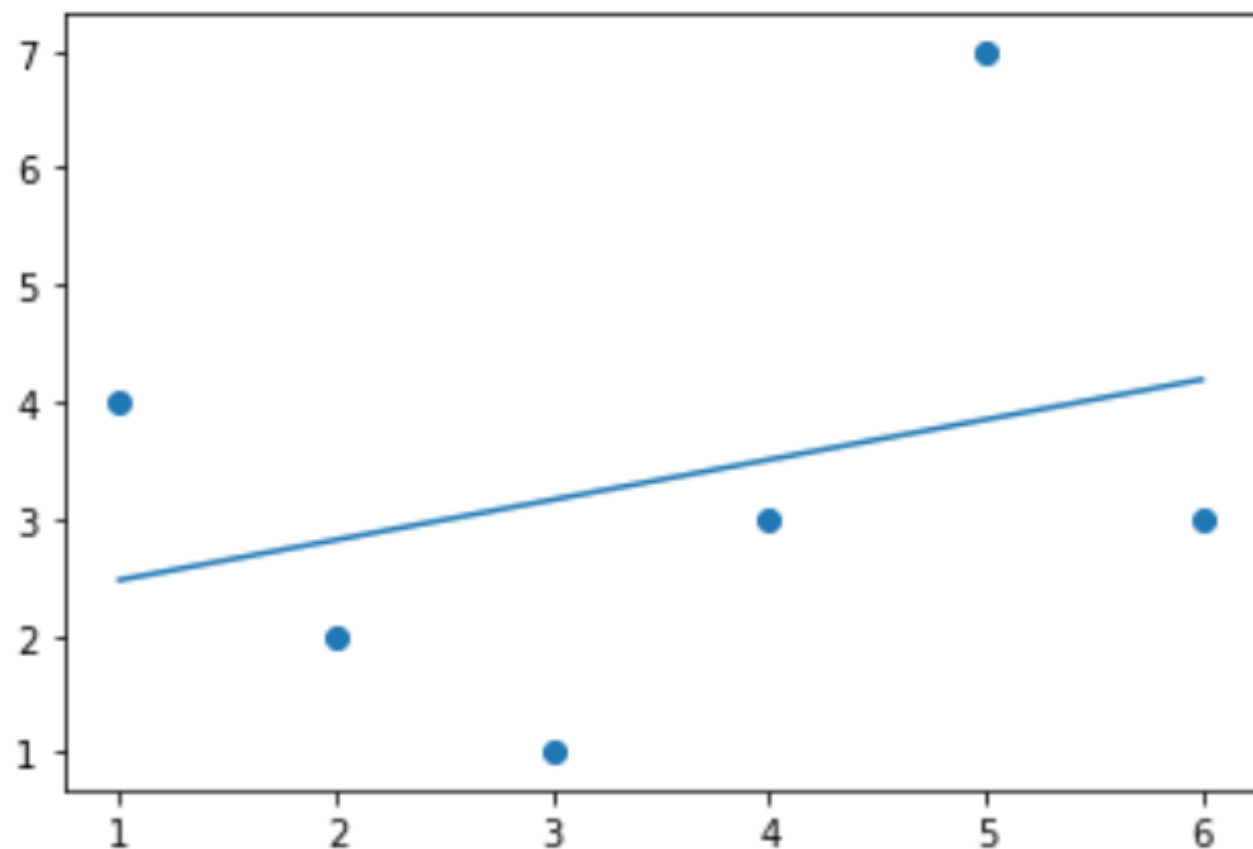
```
X = x[:, np.newaxis]
```

```
model = LinearRegression().fit(X, y) # 简单线性回归
```

```
yfit = model.predict(X)
```

```
plt.scatter(x, y)
```

```
plt.plot(x, yfit);
```



# 线性模型回归与预测

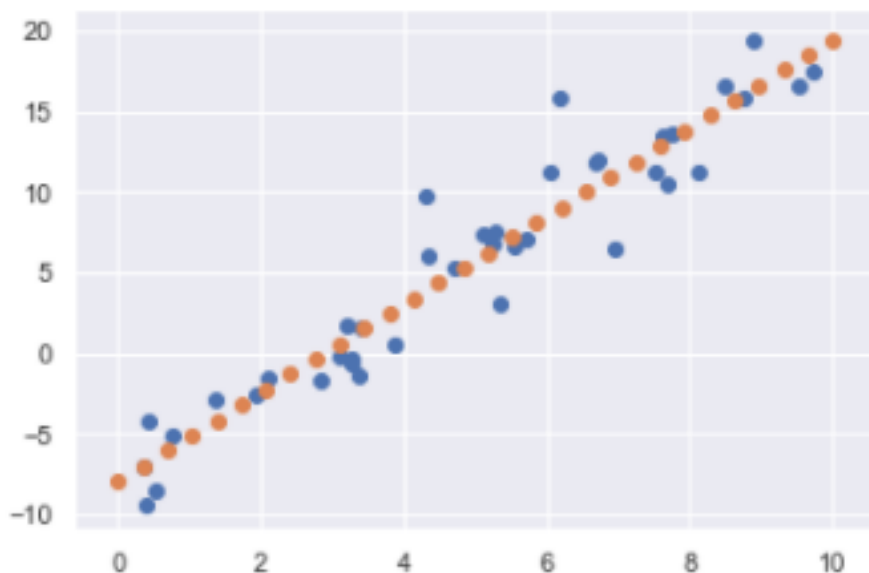
```
: from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)    # 选择线性模型

model.fit(x[:, np.newaxis], y)                  # 模型拟合

xfit = np.linspace(0, 10, 30)                  # 测试数据
yfit = model.predict(xfit[:, np.newaxis])       # 预测结果

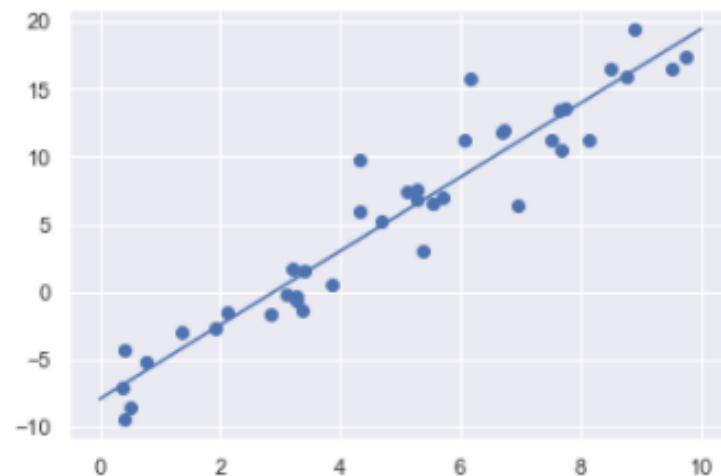
plt.scatter(x, y)
plt.scatter(xfit, yfit)
# plt.plot(xfit, yfit)
```

```
: <matplotlib.collections.PathCollection at 0x25b72915f88>
```



```
plt.plot(xfit, yfit)
```

```
[<matplotlib.lines.Line2D at 0x25b728dc348>]
```



```
print("斜率 slope (相关性):", model.coef_[0])
print("截距 intercept:", model.intercept_)
```

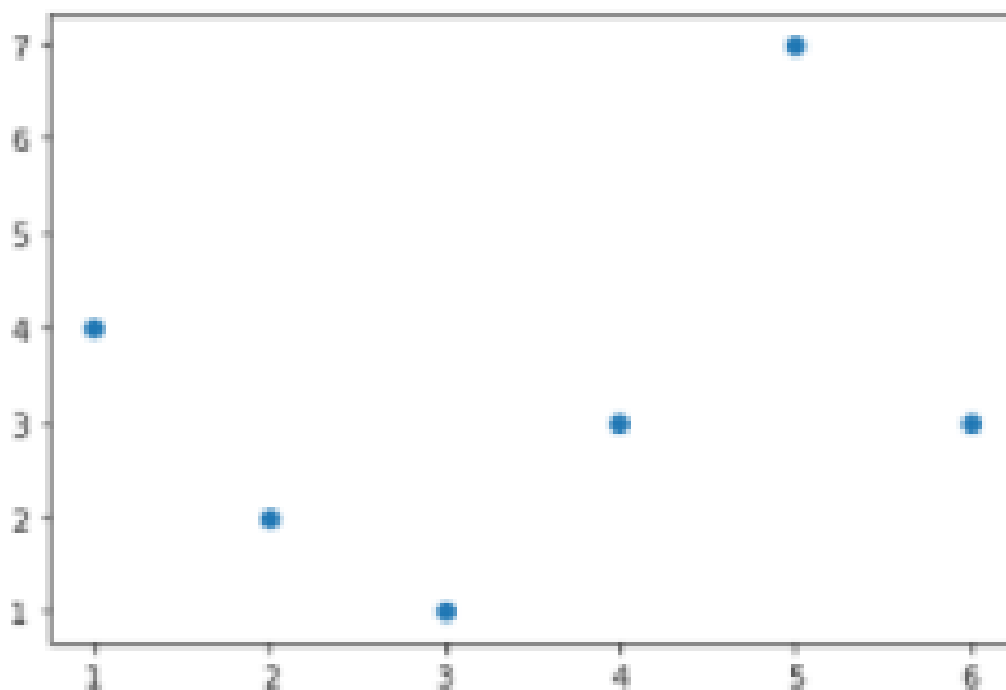
```
斜率 slope (相关性): 2.733634788557487
截距 intercept: -7.911746277704227
```

多元高次线性回归:  $y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \dots$   
 $y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots$

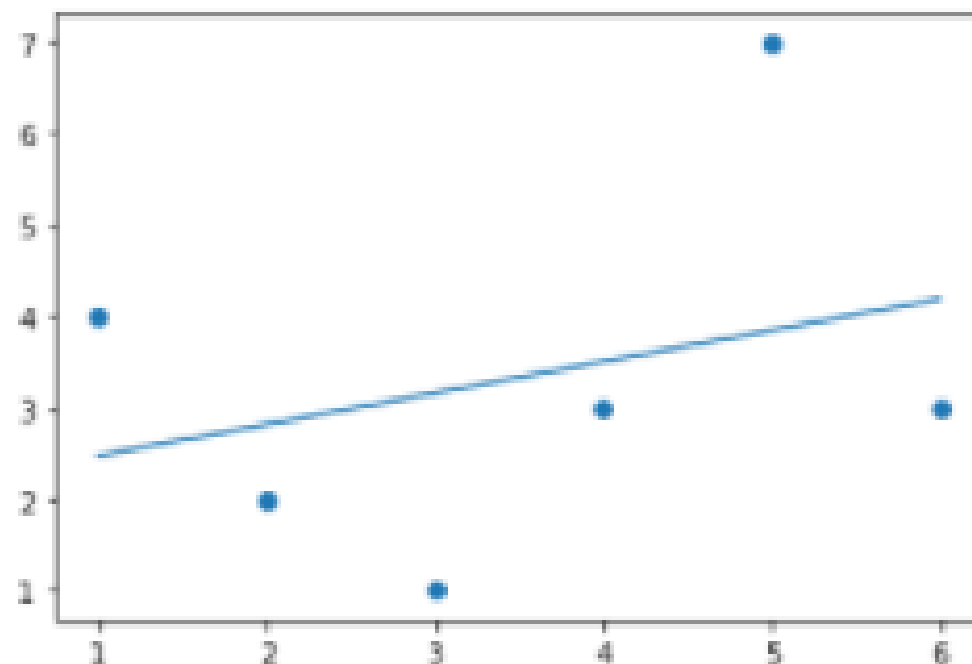
多项式回归

- 一次线性函数拟合非线性分布数据

```
x = np.array([1, 2, 3, 4, 5, 6])  
y = np.array([4, 2, 1, 3, 7, 3])  
plt.scatter(x, y);
```



```
from sklearn.linear_model import LinearRegression  
X = x[:, np.newaxis]  
model = LinearRegression().fit(X, y) # 简单线性回归  
yfit = model.predict(X)  
plt.scatter(x, y)  
plt.plot(x, yfit);
```

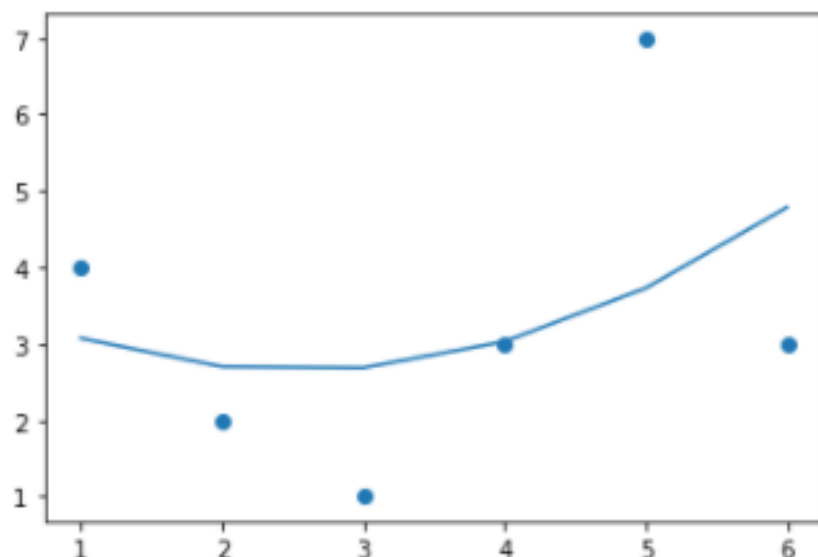


多项式 (线性) 回归:  $y = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots$

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, include_bias=False) # 选择二次函数特征 无偏置项
X2 = poly.fit_transform(X)
print(X2)
```

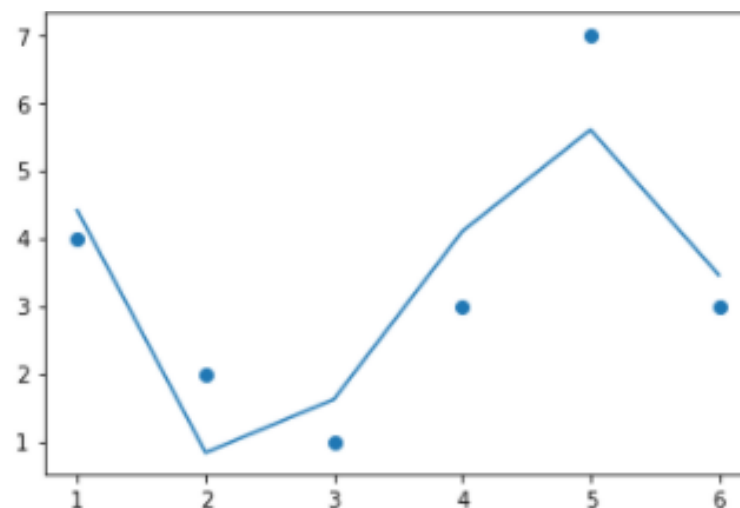
```
[[ 1.  1.]
 [ 2.  4.]
 [ 3.  9.]
 [ 4. 16.]
 [ 5. 25.]
 [ 6. 36.]]
```

```
model = LinearRegression().fit(X2, y)
yfit = model.predict(X2) ←
plt.scatter(x, y)
plt.plot(x, yfit):
```



```
print(model.intercept_)
print(model.coef_)
```

```
15.0000000000000162
[-14.99603175  4.8452381 -0.44444444]
```





# 模型参数与过拟合

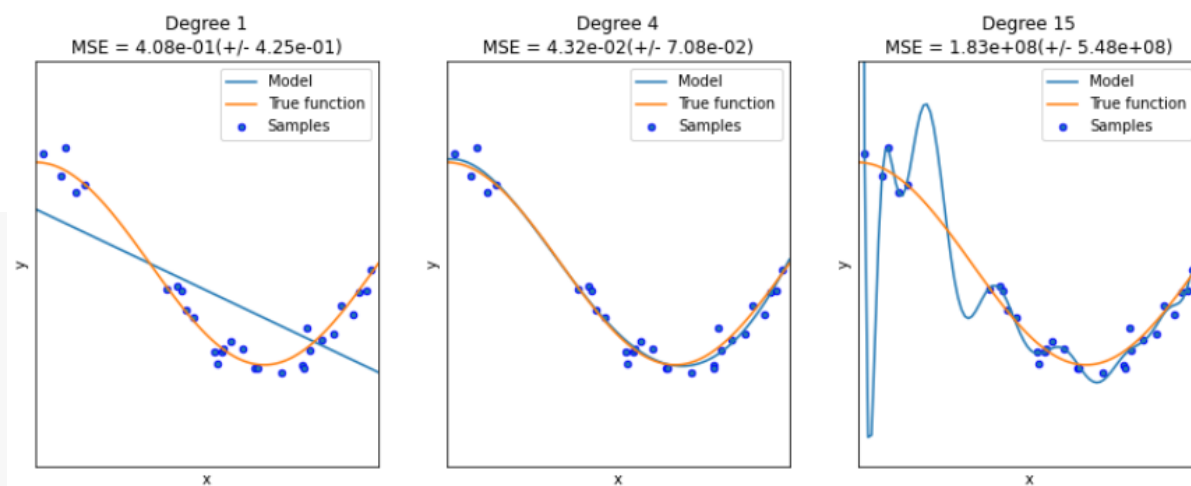
```
n_samples = 30
degrees = [1, 4, 15]
```

```
X = np.sort(np.random.rand(n_samples))
y = true_fun(X) + np.random.randn(n_samples) * 0.1
```

```
plt.figure(figsize=(14, 5))
for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)
    plt.setp(ax, xticks=(), yticks=())
```

```
polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
linear_regression = LinearRegression()
pipeline = Pipeline([
    ("polynomial_features", polynomial_features),
    ("linear_regression", linear_regression),
])
pipeline.fit(X[:, np.newaxis], y)
```

```
# Evaluate the models using crossvalidation
scores = cross_val_score(
    pipeline, X[:, np.newaxis], y, scoring="neg_mean_squared_error", cv=10
)
```

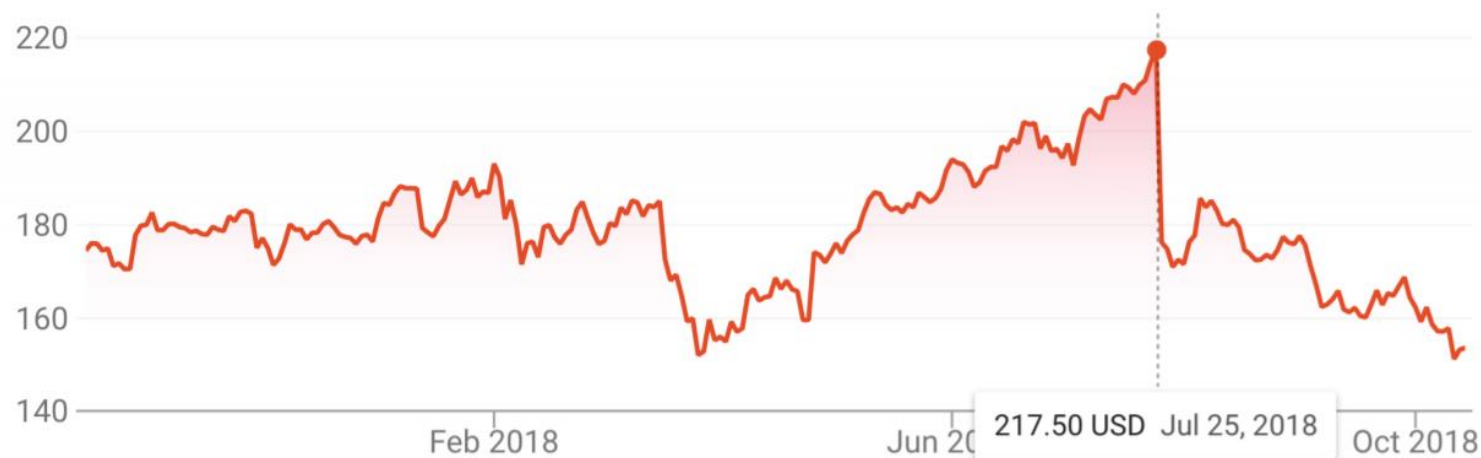




# 量化分析与金融交易模型回归

## 股票交易序列分析建模

- 模型回归拟合
- 自回归模型概念与ARIMA
- 谱特征分解
- 卡尔曼滤波
- HMM
- 时间序列聚类



# 平滑、成分分解与多项式回归

用多项式函数拟合趋势

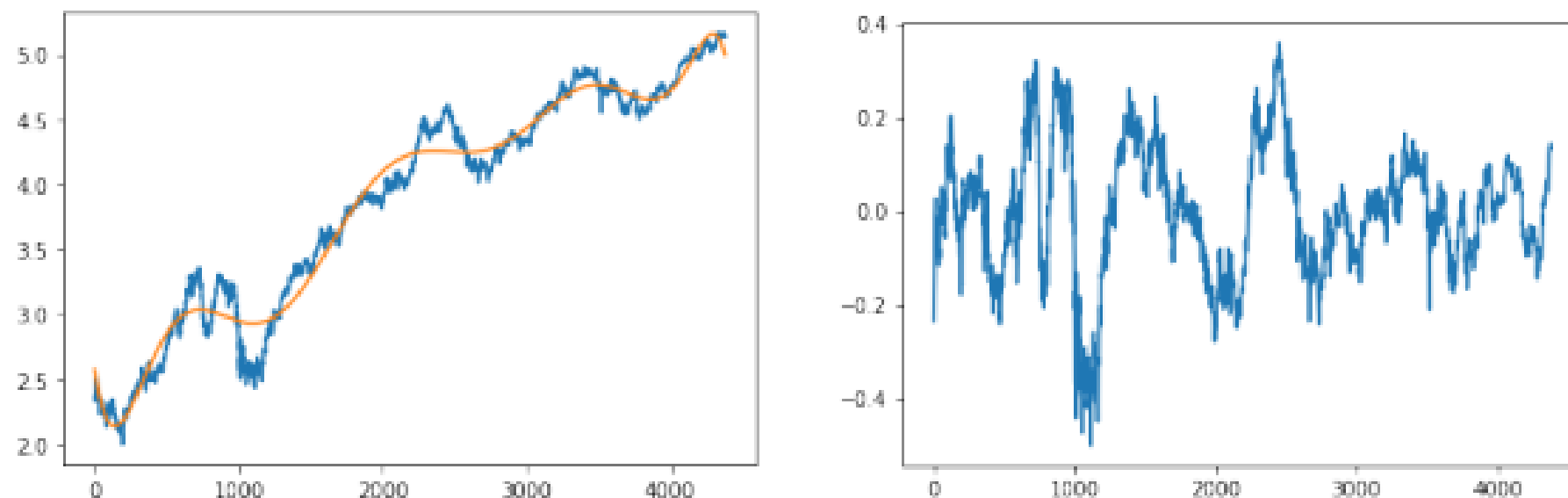


图 1: 趋势与余项

# 序列平稳性定义:

对时间序列 $\{X_t, t \in \mathbb{Z}\}$ , 如果对任意的 $t \in \mathbb{Z}$ 和正整数 $n, k$ ,  $(X_t, \dots, X_{t+n-1})$ 总是与 $(X_{t+k}, \dots, X_{t+n-1+k})$ 同分布, 则称 $\{X_t\}$ 为**严平稳**时间序列。

**弱平稳序列**(宽平稳序列, weakly stationary time series): 如果时间序列 $\{X_t\}$ 存在有限的二阶矩且满足:

- (1)  $EX_t = \mu$ 与 $t$ 无关;
- (2)  $\text{Var}(X_t) = \gamma_0$ 与 $t$ 无关;
- (3)  $\gamma_k = \text{Cov}(X_{t-k}, X_t)$ ,  $k = 1, 2, \dots$ 与 $t$ 无关,

则称 $\{X_t\}$ 为弱平稳序列。




# 平稳性检验 ADF (Augmented Dickey-Fuller Test)

```
from statsmodels.tsa.stattools import adfuller
```

```
adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(stationary)
```


常用返回值的意义:

```
print(adf)    # adf值负值越高, 越确信是平稳的  
print(pvalue) # pvalue越低, 越大概率拒绝非平稳假设  
print(nobs)   # 观察次数越高, 判断的置信度越高
```



```
-10.08442591366971  
1.1655044784188918e-17  
99
```

```
adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(trend_seasonality,  
print("ADF: ", adf)  
print("p-value:", pvalue)
```



```
ADF: 0.29403605928894067  
p-value: 0.9770692037868646
```

# 将序列转换为平稳序列

- 动因：很多模型回归方案在非平稳信号下是无效的
- 序列平稳化一般操作流程：
  - 趋势消除 Remove trend
  - 异方差的消除 Remove heteroscedasticity
  - 自相关性消除 Remove autocorrelation with differencing
  - 去周期性 Remove seasonality
- 循环做上述步骤直到序列特征平稳



# 通过设置回归方案将序列变成平稳序列：

**regression** : {'c','ct','ctt','nc'}

Constant and trend order to include in regression

- 'c' : constant only (default)
- 'ct' : constant and trend
- 'ctt' : constant, and linear and quadratic trend
- 'nc' : no constant, no trend



## 设置回归方案进行平稳化操作:

```
] from statsmodels.tsa.stattools import adfuller
result_nc = adfuller(data, regression = 'nc') #no constant, no trend
result_c = adfuller(data, regression = 'c') # constant
result_ct = adfuller(data, regression = 'ct') # constant + trend
result_ctt = adfuller(data, regression = 'ctt') # constant + trend + quadratic
```

```
print(result_nc)
print(result_c)
print(result_ct)
print(result_ctt)
```



```
(2.6752897748844604, 0.9990797202349937, 27, 2991, {'1%': -2.566487914623174, '5%': -1.94109017771703
83.852960381664)
(1.240364410251056, 0.9962532356713726, 27, 2991, {'1%': -3.4325382049645357, '5%': -2.86250680710657
185.84146337278)
(-0.9794106628299224, 0.9468230455847277, 27, 2991, {'1%': -3.9617999630590077, '5%': -3.411958882024
185.100765326428)
(-4.156338917143881, 0.019856230995530046, 27, 2991, {'1%': -4.3750083727627835, '5%': -3.83436589069
20169.36379510231)
```



# 时间序列自回归模型（线性）的基本流程

- 观察数据特征 — 模型参数分析
- 平稳化、选择回归模型及参数
- 建模 — 预测
- 结果分析与模型调整



# 序列相关性表达

- 皮尔逊相关系数 (Pearson correlation coefficient) : 用来统计变量或序列间的相关性

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y},$$

- 自相关 (Autocorrelation) : 自相关被定义为, 两个随机过程中不同时刻的数值之间的皮尔森相关

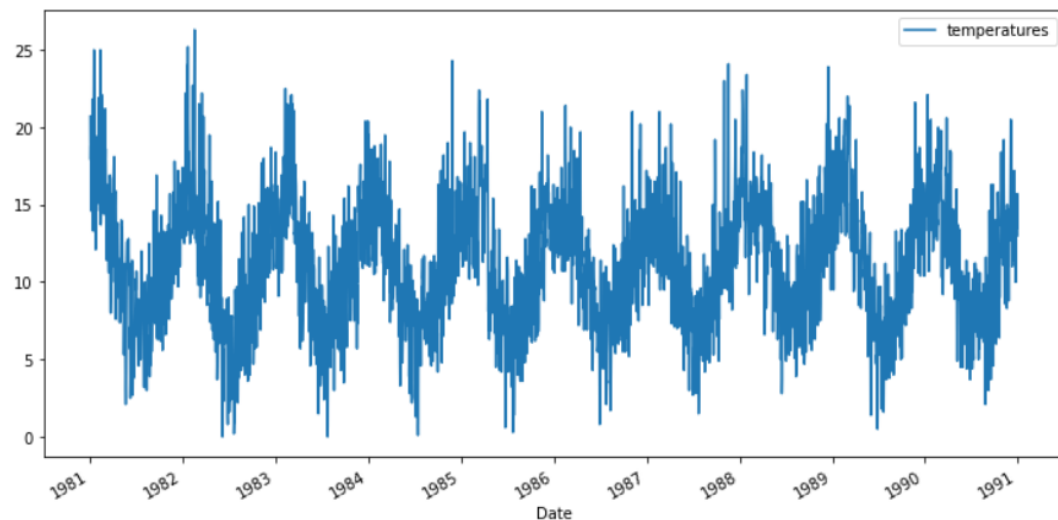
$$R(s,t) = \frac{E[(X_s - \mu_s)(X_t - \mu_t)]}{\sigma_s \sigma_t}$$

- 偏自相关 (Partial Autocorrelation Coefficient) : K线性回归模型的的自相关结果:

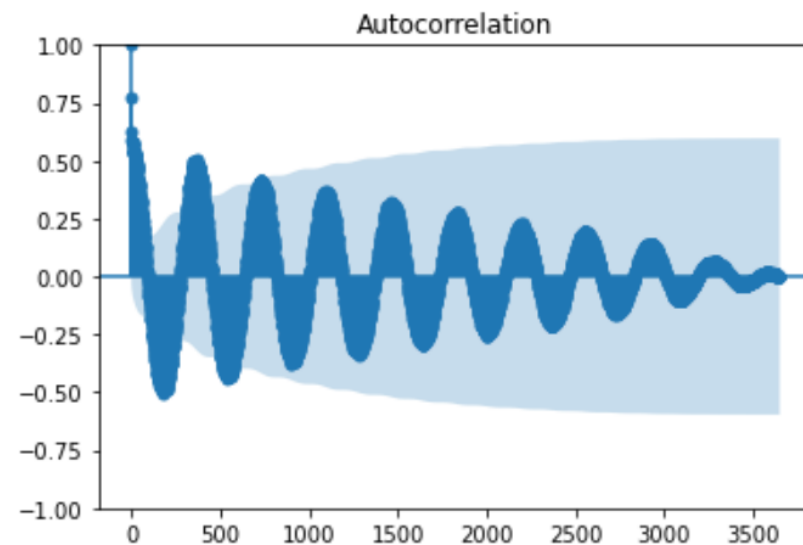
$$x_t = \Phi_{k1} x_{t-1} + \Phi_{k2} x_{t-2} + \dots + \Phi_{kk} x_{t-k} + u_t. \quad (1)$$

# 计算自相关：

- 澳大利亚墨尔本10年最低气温数据

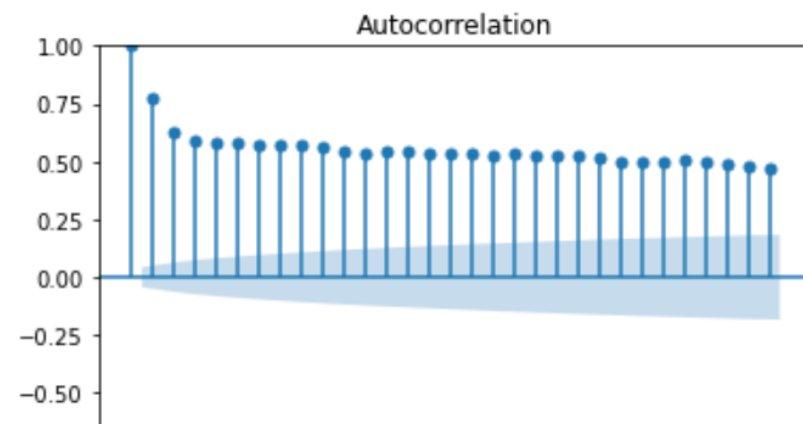


```
import statsmodels.api as sm
sm.graphics.tsa.plot_acf(data, lags=3649) # 0-3650
plt.figure(figsize=(12, 6))
plt.show()
```



<Figure size 864x432 with 0 Axes>

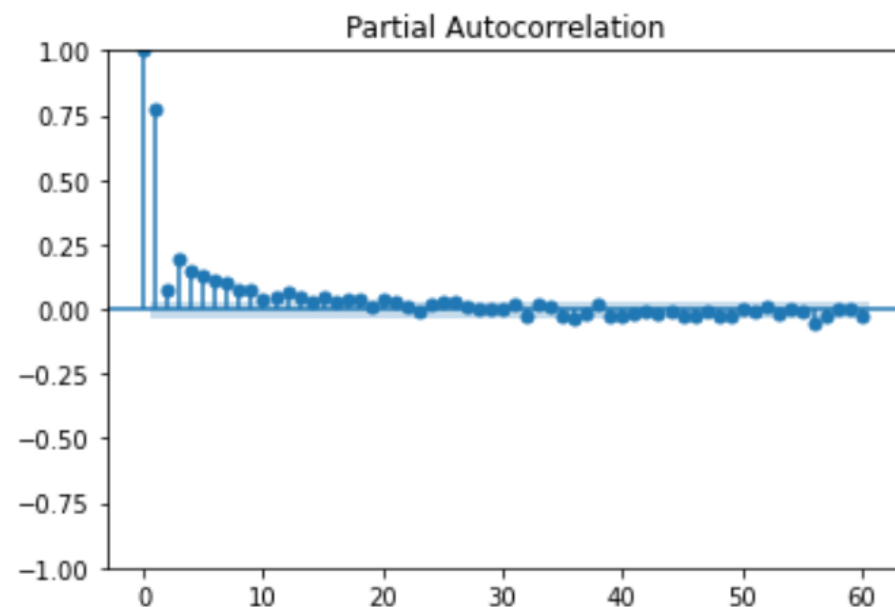
```
sm.graphics.tsa.plot_acf(data, lags=30, alpha = 0.01) # 季节性 alpha: 置信区
plt.figure(figsize=(12, 6))
plt.show()
```



# 计算自偏相关

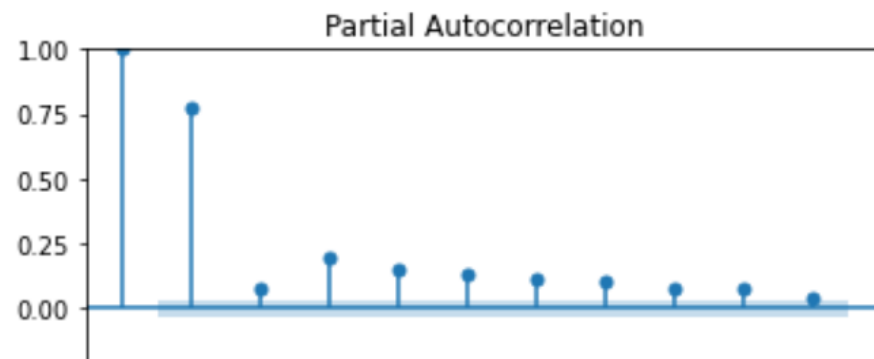
- 偏自相关是剔除干扰后时间序列观察与先前时间步长时间序列观察之间关系的总结。在滞后k处的偏自相关是在消除由于较短滞后条件导致的任何相关性的影响之后产生的相关性

```
: sm.graphics.tsa.plot_pacf(data, lags=60) # 0-3650  
plt.figure(figsize=(12, 6))  
plt.show()
```



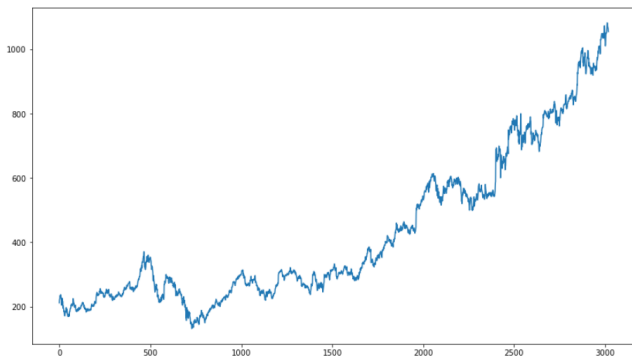
<Figure size 864x432 with 0 Axes>

```
: sm.graphics.tsa.plot_pacf(data, lags=10) # 0-1824  
plt.figure(figsize=(12, 6))  
plt.show()
```

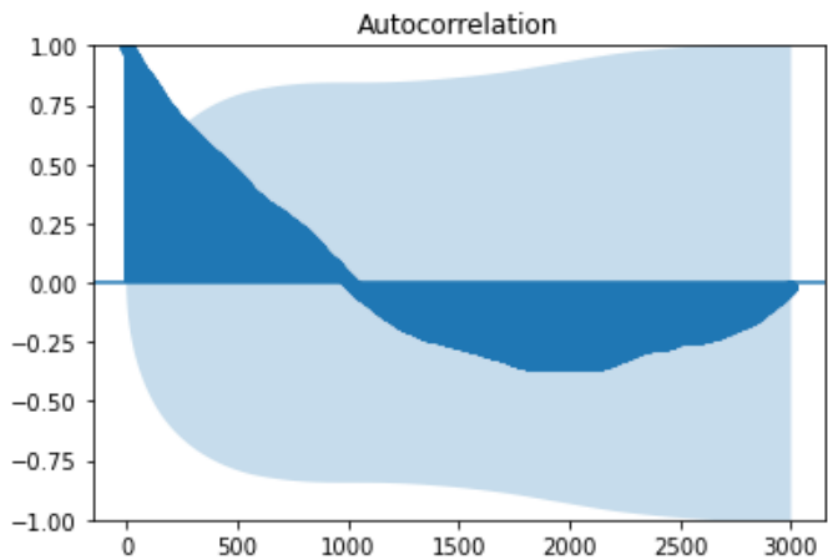


# 交易数据的相关性观察：

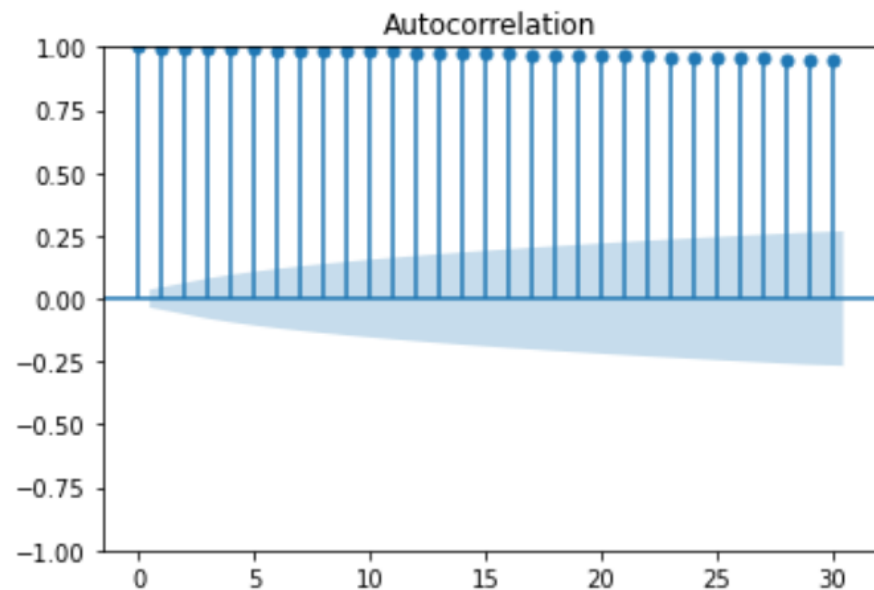
- GOOGL\_2006-01-01\_to\_2018-01-01.csv



```
sm.graphics.tsa.plot_acf(data, lags=3000) # 0-765  
plt.figure(figsize=(12, 6))  
plt.show()
```

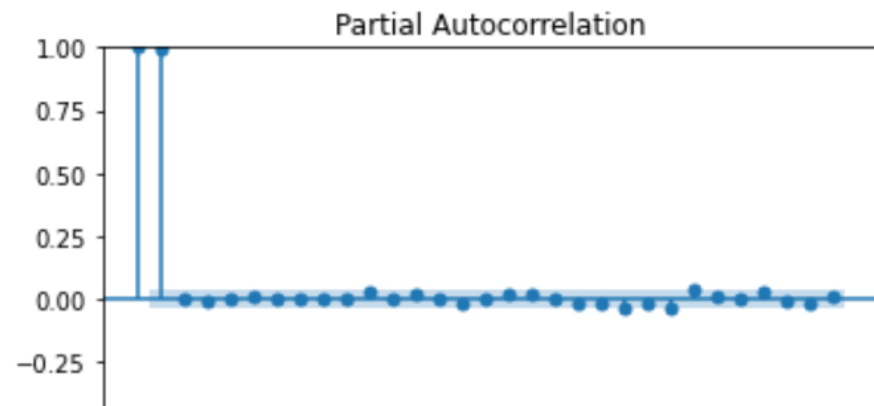


```
sm.graphics.tsa.plot_acf(data, lags=30) # 0-30  
plt.figure(figsize=(12, 6))  
plt.show()
```



<Figure size 864x432 with 0 Axes>

```
sm.graphics.tsa.plot_pacf(data, lags=30) # 0-30  
plt.figure(figsize=(12, 6))  
plt.show()
```



# ARIMA模型 (Autoregressive Integrated Moving Average model)

- ARIMA(p, d, q):
  - AR是“自回归”，p为自回归项数；
  - MA为“滑动平均”，q为滑动平均项数，
  - d为使之成为平稳序列所做的差分次数（阶数 lag）
- 对比ARMA模型：
  - 要求是平稳序列（先通过差分消除趋势、周期）
  - 经过adf检验确定置信度  $P < 5\%$



```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(data, order=(1,1,2)).fit() # 一阶差分消除趋势, 2阶平滑, 1阶回归
pred = model.predict(start = 0, end = 1500)

#通过获取回归系数计算
#获取回归方程的系数
coefs = model.params
print(f"coefs:{coefs}")

plt.figure(figsize=(14, 8))
plt.plot(data, label = 'data') # 原信号
plt.plot(pred, label = 'pred') # 预测信号
plt.legend()
plt.show()
```

```
coefs:[ 8.09237500e-01 -8.14015475e-01 -8.76044700e-03  5.02559182e+01]
```



# 指数滑动平均 (Exponential Moving Average, EMA)

与均值滑动平均(Equally Weighted Moving Average) 不同，窗口内的各权值按指数递减，距离越远，权值越小。

EMA的计算公式为：  $EMA_t = \alpha y_t + (1 - \alpha)EMA_{t-1}$

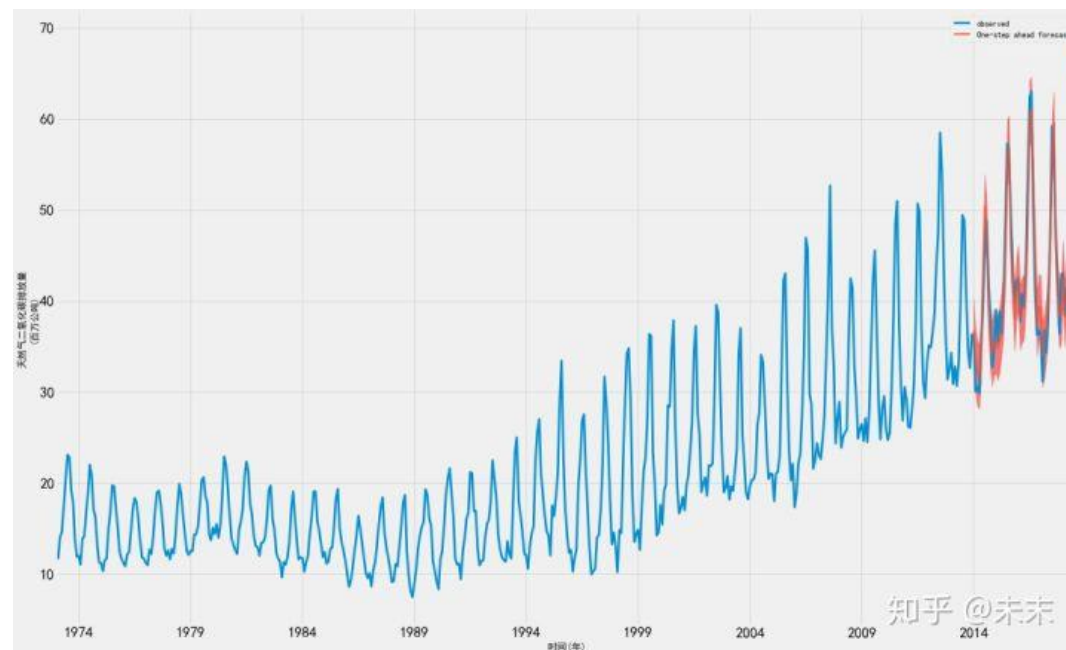
其中， $\alpha \in (0, 1)$ 表示权重的衰减程度， $\alpha$  越大，当前抽样值的权重越高，滑动平均的实时性越强，相反， $\alpha$  越小，平稳性更高。 $\alpha$  实际上控制权重计算中有效项的个数，即指数平滑有效窗口的大小。





## 例子2：二氧化碳排放预测模型

- 基于美国能源情报署(Energy Information Administration)和杰森•麦克尼尔(JasonMcNeill)提供的1973年1月到2018年9月美国发电产生的二氧化碳的数据进行建模分析。
- 1、数据清洗
- 2、可视化处理
- 3、平稳性检验
- 4、序列平稳化
- 5、SARIMA模型（季节模型）
- 6、模型预测



## What is the goal?

- Estimate the variables of interest at any given time—your “state.” (In the previous example, these variables are the position and velocity of the car.)
- At each time point, the Kalman filter estimates your state by calculating the optimal compromise between two sources of data:
  - the model with information from the previous time point
  - the latest measurement

- The compromise is determined by the noise.
- If the model has relatively large errors, more weight is given to the latest measurement in making the current estimate.
- If the measurement has larger errors, more weight is given to the model in making the current estimate.
- Therefore, you need to estimate not only your state but also the errors (the covariance) for both the model and the measurements. These must be updated at each time step, too.

## 模型使用前提 (Assumptions) :

The Kalman filter approach is based on three fundamental assumptions:

- The system can be described or approximated by a linear model.
- All noise (from both the system and the measurements) is white.
- All noise is Gaussian.

Variable at current time is a linear function of variable and noise at previous time

Nonlinear systems can often be approximated by linear models around a current estimate (for example, extended Kalman filter).

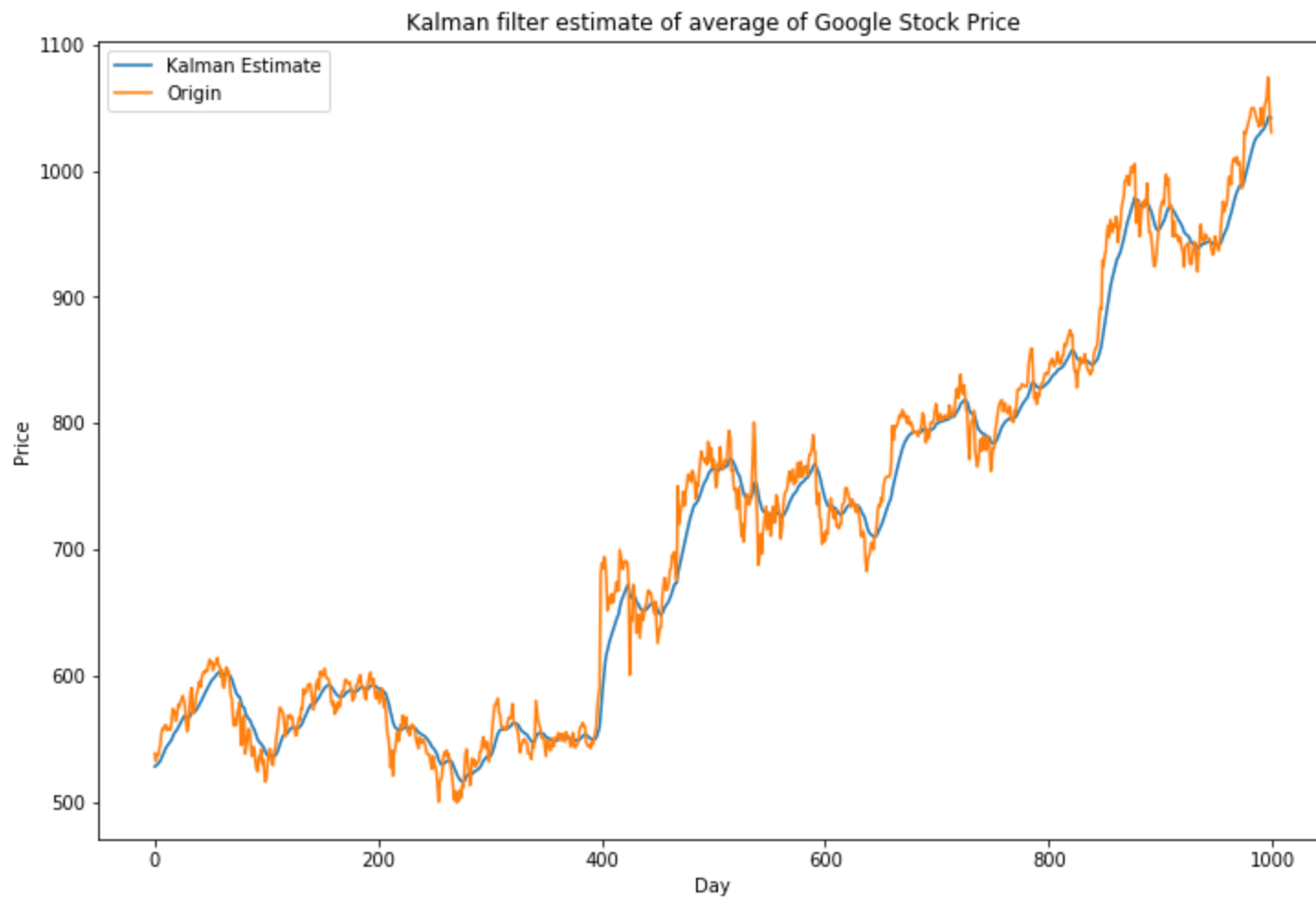
# Kalman Filter: The Workflow

1. Make initial estimates of state vector and covariance matrix (time  $k=0$ ).
2. Predict state and covariance for next time step.
3. Compute the Kalman gain.
4. Make a measurement.
5. Update estimates of state and covariance.
6. Repeat steps 2-5.



# 序列分析与金融交易模型回归

## 卡尔曼滤波



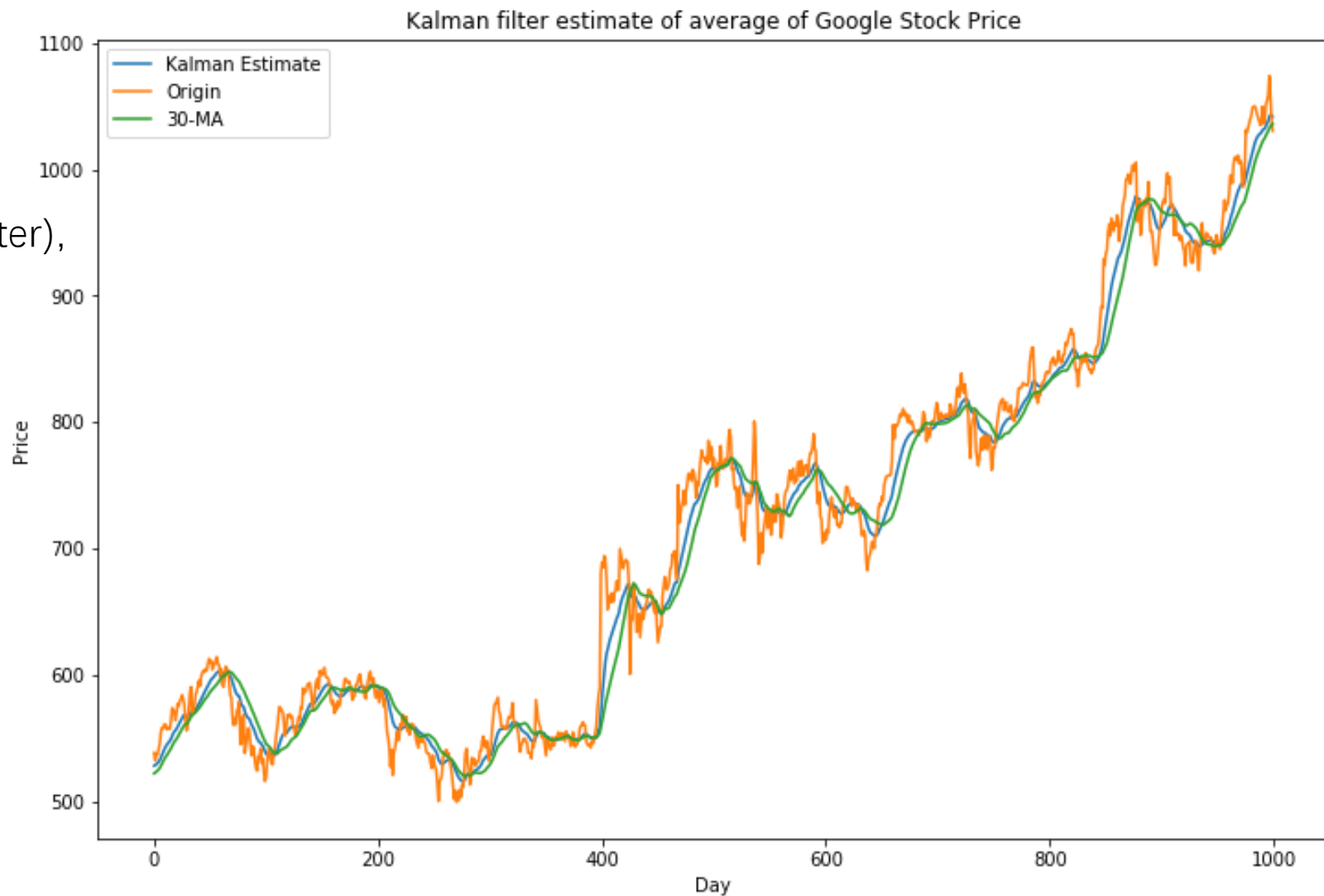


# 序列分析与金融交易模型回归

## 其他滤波

Baxter-King (`statsmodels.tsa.filters.bkfilter`)

Hodrick-Prescott (`statsmodels.tsa.filters.hpfilter`),

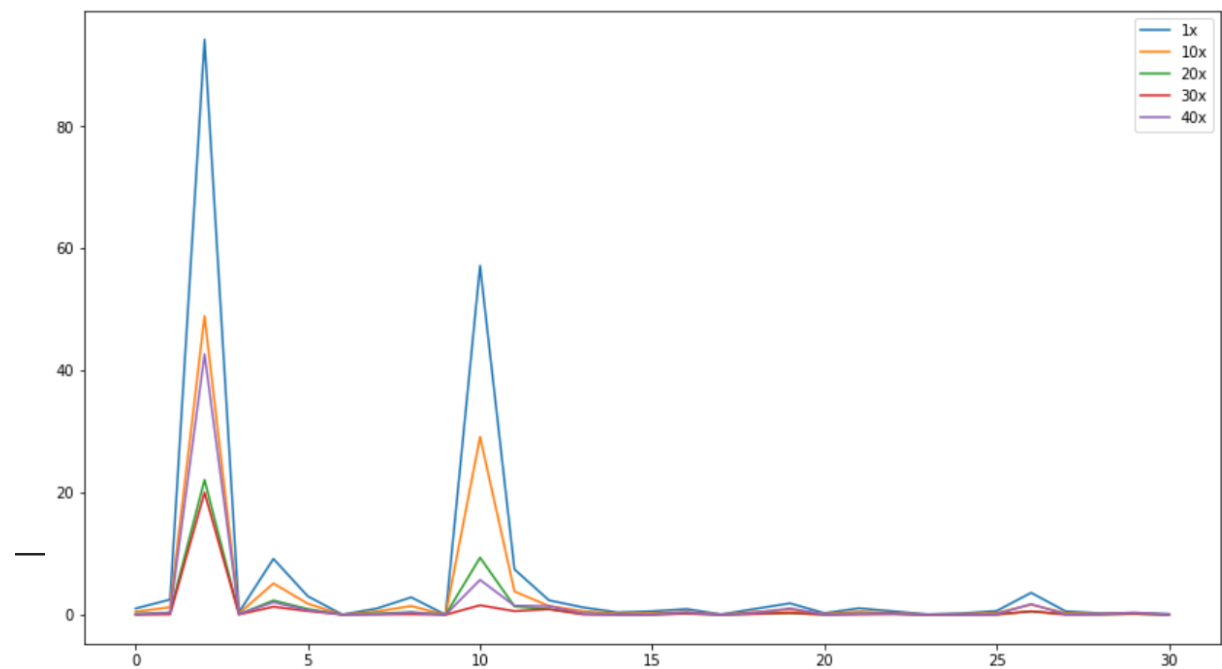
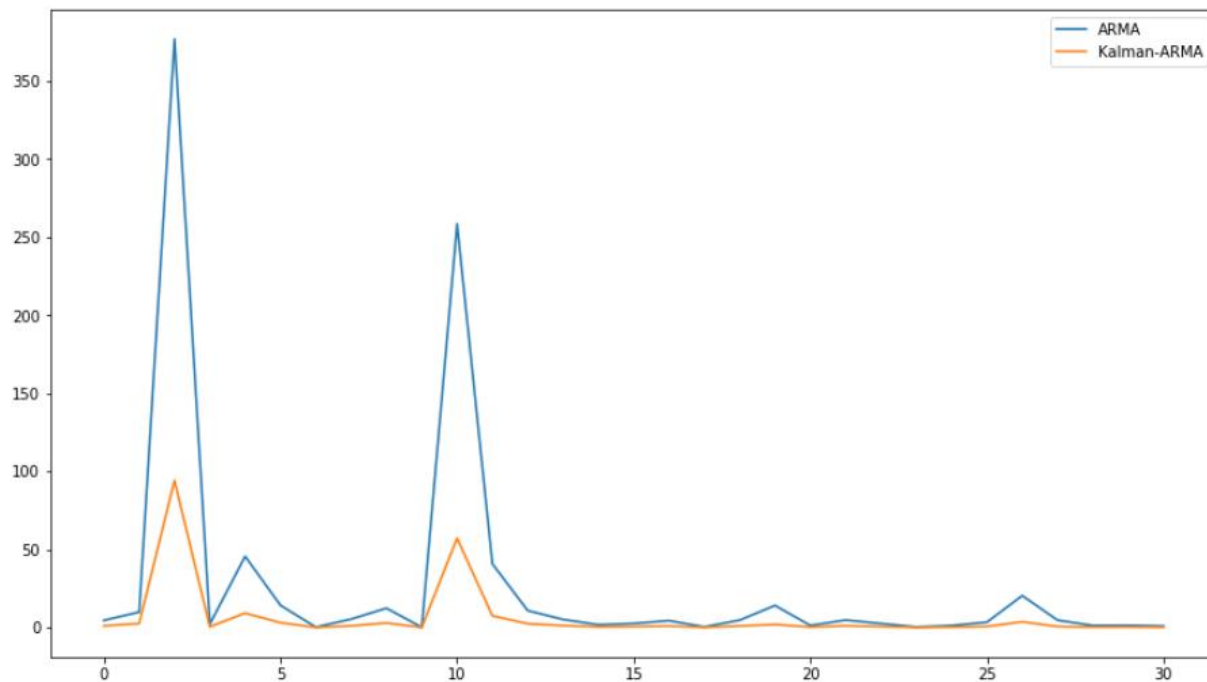


# Kalman-ARMA:

$F(x)$ 进行Kalman滤波后得到 $H(x)$ ，用ARMA预测出 $H(x)$ 二阶差分后的结果，用 $H(x)$ 的增量直接拟合 $F(x)$ 的增量。

```
: plt.figure(figsize=(14, 8))  
plt.plot(ARMAmse, label="ARMA")  
plt.plot(kalmanmse, label="Kalman-ARMA")  
plt.legend()  
plt.show()
```

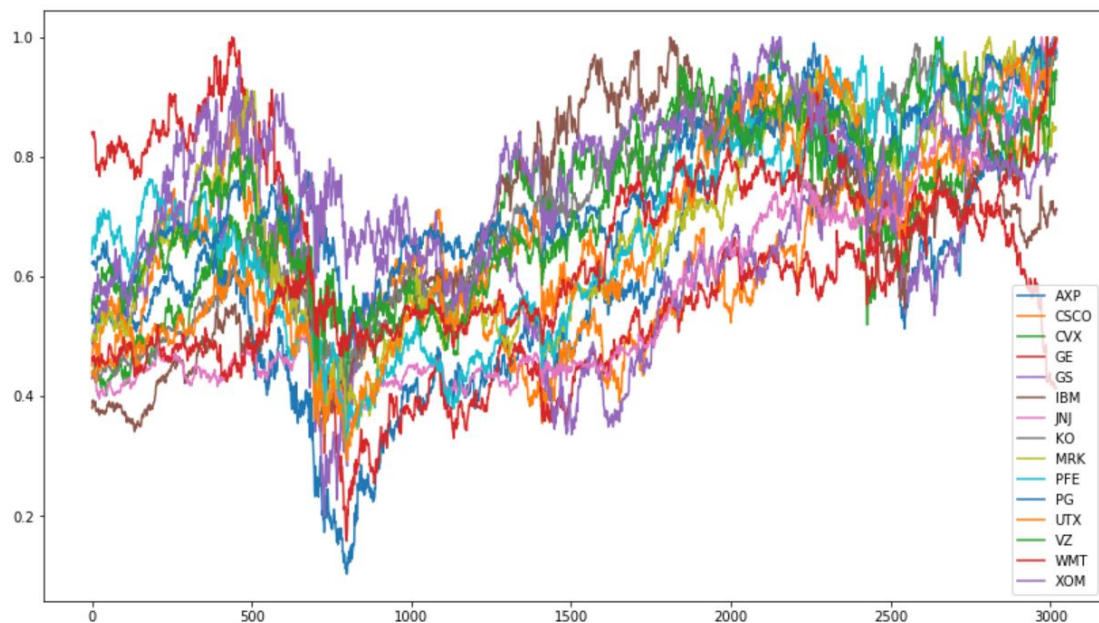
K-ARMA可以获得较好小的MSE



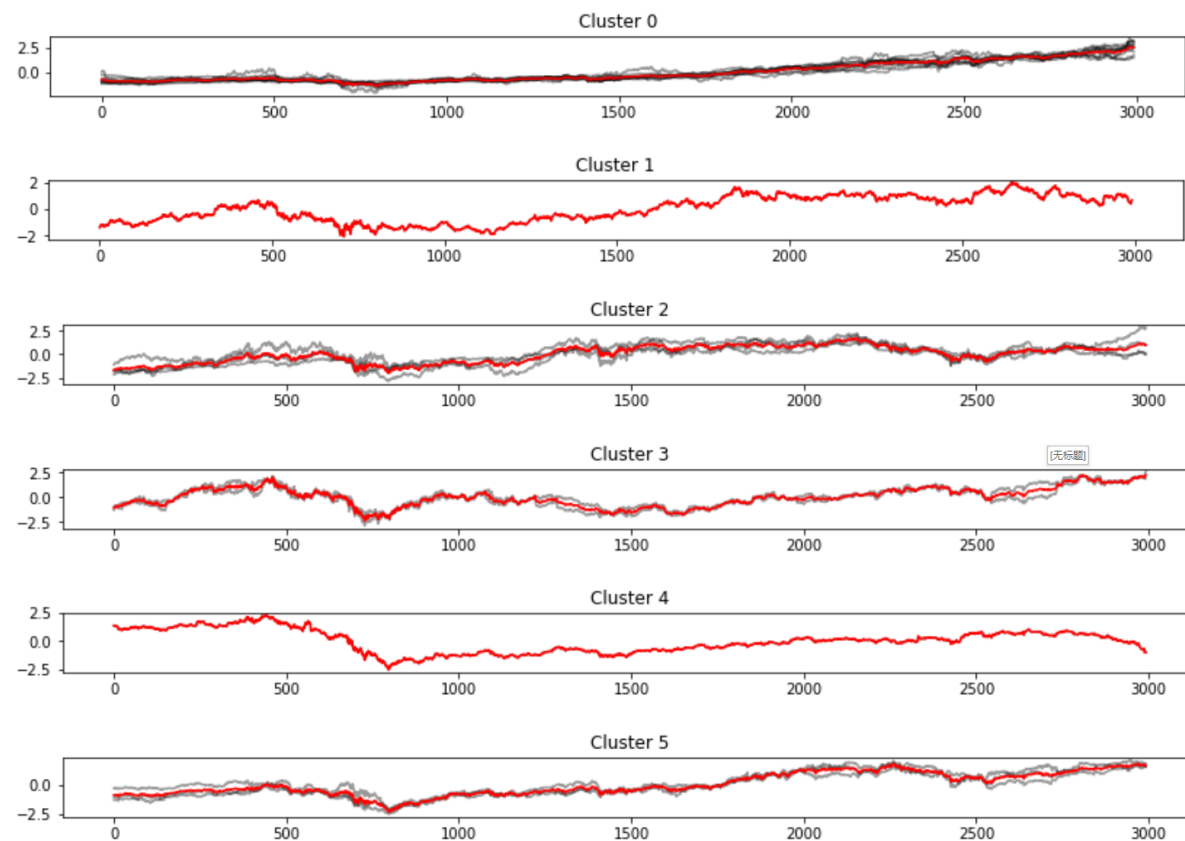


# 聚类与进一步的分析:

```
km = TimeSeriesKMeans(n_clusters=clunum,random_state=seed)
y_pred=km.fit_predict(X_train)
```



通过归一化、PAA降维、Kmeans聚类后可以发现，2008年金融危机对所有的公司造成了不同的影响，股票大致可以据此分为两类，一类在金融危机时受到重创，10年之后的市值才与金融危机前的市值基本持平；另一类在金融危机时受到的打击相对较轻，在金融危机后很快实现了稳定发展。

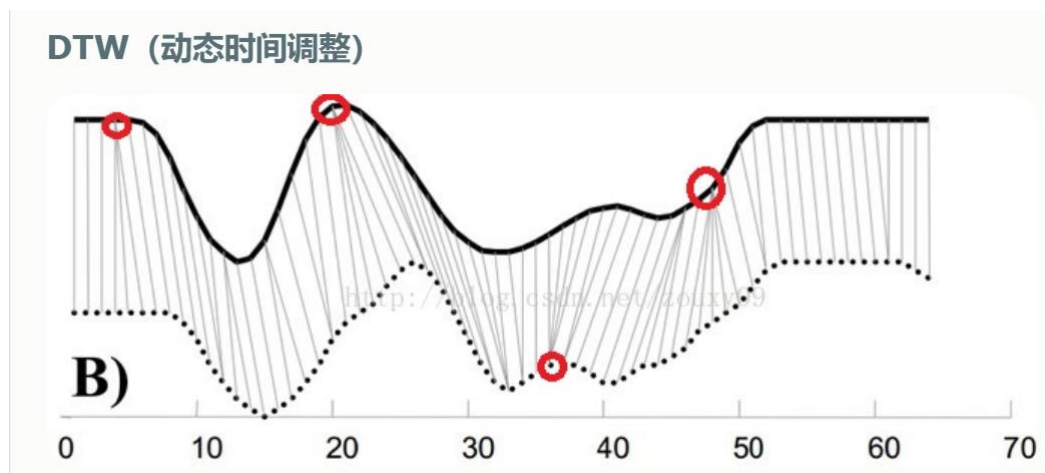


聚类后一起预测对某些单只股票能获得优异的效果



# 可以采用不同距离量

- 欧氏距离
- DTW距离



<https://www.cnblogs.com/zhxuxu/p/10492578.html>

# 向量自回归模型 (VAR模型)

- VAR模型描述在同一样本期间的n个变量（内生变量）可以作为它们过去值的线性函数
- $Y_t = c + A_1(y_{t-1}) + A_2(y_{t-2}) + \dots + A_p(y_{t-p}) + e_t$ ,

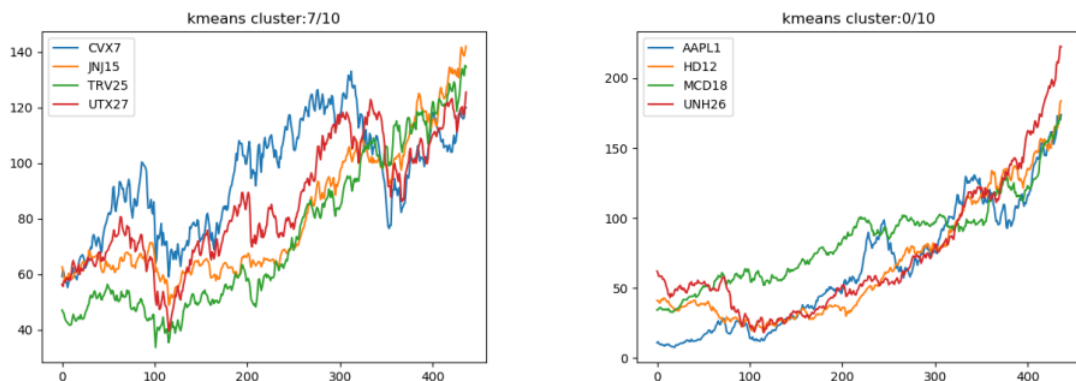


图 11: 按照 DTW 距离聚类得到的两个次大的类

进一步加入Kshape 可以消除波动程度（方差）、相位偏移在聚类中的影响,



# Granger 因果测试

在时间序列情形下，两个经济变量  $X$ 、 $Y$  之间的 Granger 因果关系定义为：若在包含了变量  $X$ 、 $Y$  的过去信息的条件下，对变量  $Y$  的预测效果要优于只单独由  $Y$  的过去信息对  $Y$  进行的预测效果，即变量  $X$  有助于解释变量  $Y$  的将来变化，则认为变量  $X$  是引致变量  $Y$  的 Granger 原因。进行 Granger 因果关系检验的一个前提条件是时间序列必须具有平稳性，否则可能会出现虚假回归问题。



# 用神经网络模型建模（假定信号有相关性）

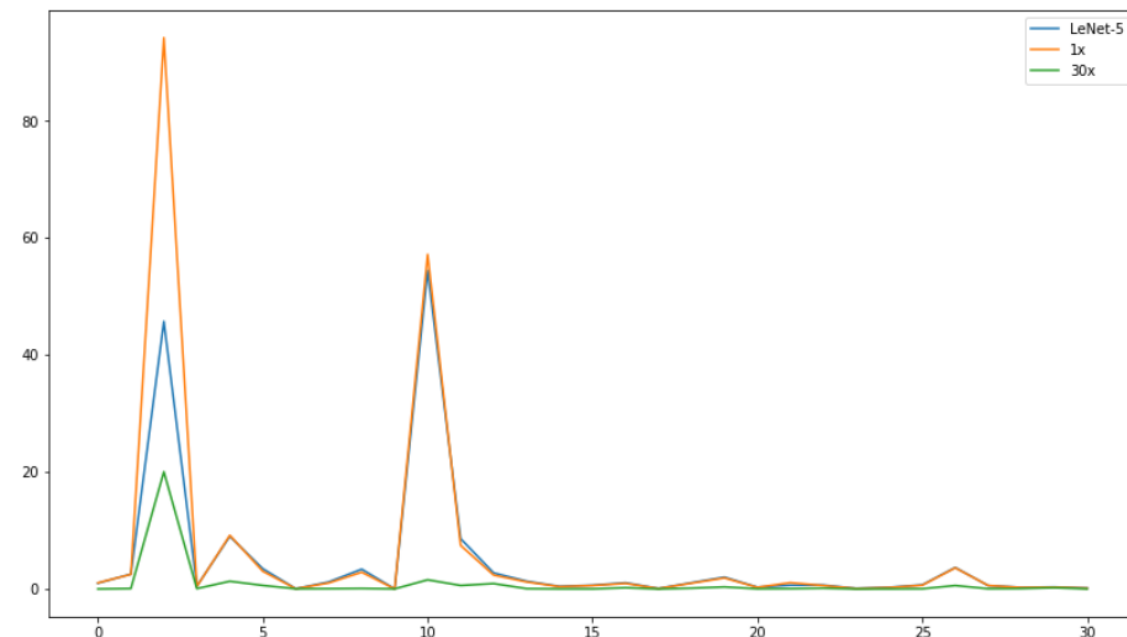
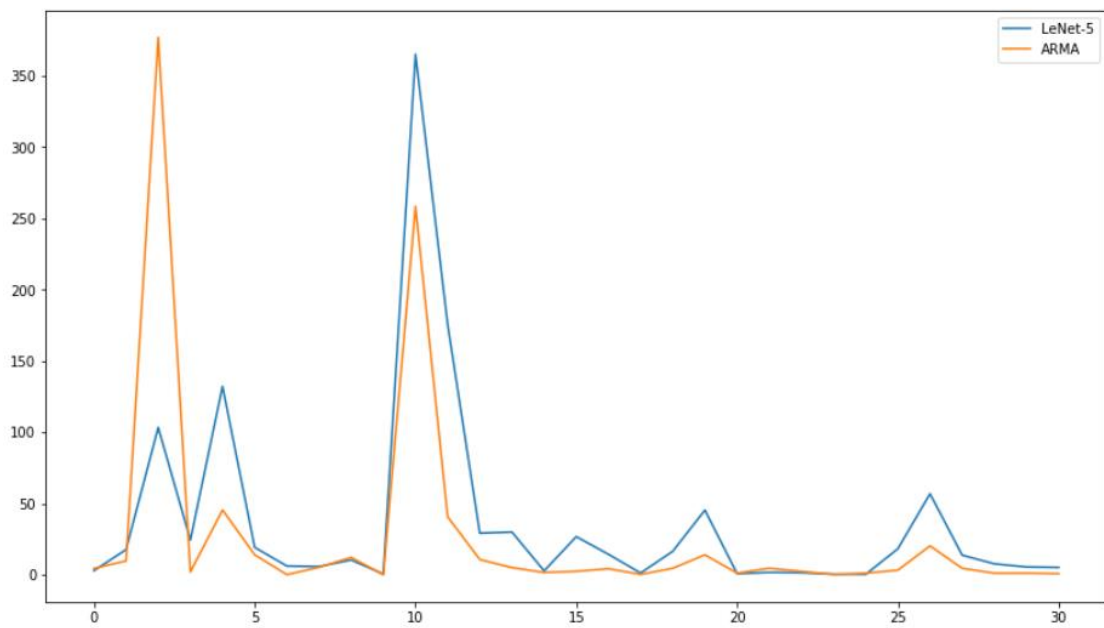
- 采用2D卷积平面，方案类似图像分类问题
- 采用3D卷积立方体，类似视频处理



# 采用CNN网络对多只 (30) 股票进行综合预测

- 使用3层30\*30卷积神经网络对所有的Close进行同时预测。结果仅与ARMA相当，无法和ARMA预测Kalman滤波后的函数得到的结果相提并论。

在Kalman滤波的基础上，使用神经网络对所有的Close增量进行同时预测。CNN比参数优化前的ARMA预测Kalman滤波后的函数效果更优秀，但比不上参数优化后ARMA预测Kalman滤波后的函数的效果。





# 序列分析与金融交易模型回归

## 特征提取

传统的统计分布的特征:

最大值, 最小值, 均值, 方差,  
自相关函数, 偏自相关函数, 偏  
度和峰度

$$\text{skewness}(X) = E\left[\left(\frac{X - \mu}{\sigma}\right)^3\right] = \frac{1}{T} \sum_{i=1}^T \frac{(x_i - \mu)^3}{\sigma^3},$$

$$\text{kurtosis}(X) = E\left[\left(\frac{X - \mu}{\sigma}\right)^4\right] = \frac{1}{T} \sum_{i=1}^T \frac{(x_i - \mu)^4}{\sigma^4}.$$

# 通过AIC 或者BIC准则来选择阶数

- AIC 赤池信息量 (akaike information criterion)
  - $AIC = -2 \ln(L) + 2k$
  - AIC越小，模型越好，通常选择AIC最小的模型
- BIC 贝叶斯信息量 (bayesian information criterion)
  - $BIC = -2 \ln(L) + \ln(n) \cdot k$

L是在该模型下的最大似然，n是数据数量，k是模型的变量个数

```
#定阶数
import statsmodels.tsa.stattools as st
order = st.arma_order_select_ic(X, max_ar=5, max_ma=0, ic=['aic', 'bic'])
print(order.bic_min_order)
print(order.aic_min_order)
```





```

: from statsmodels.tsa.arima_model import ARMA

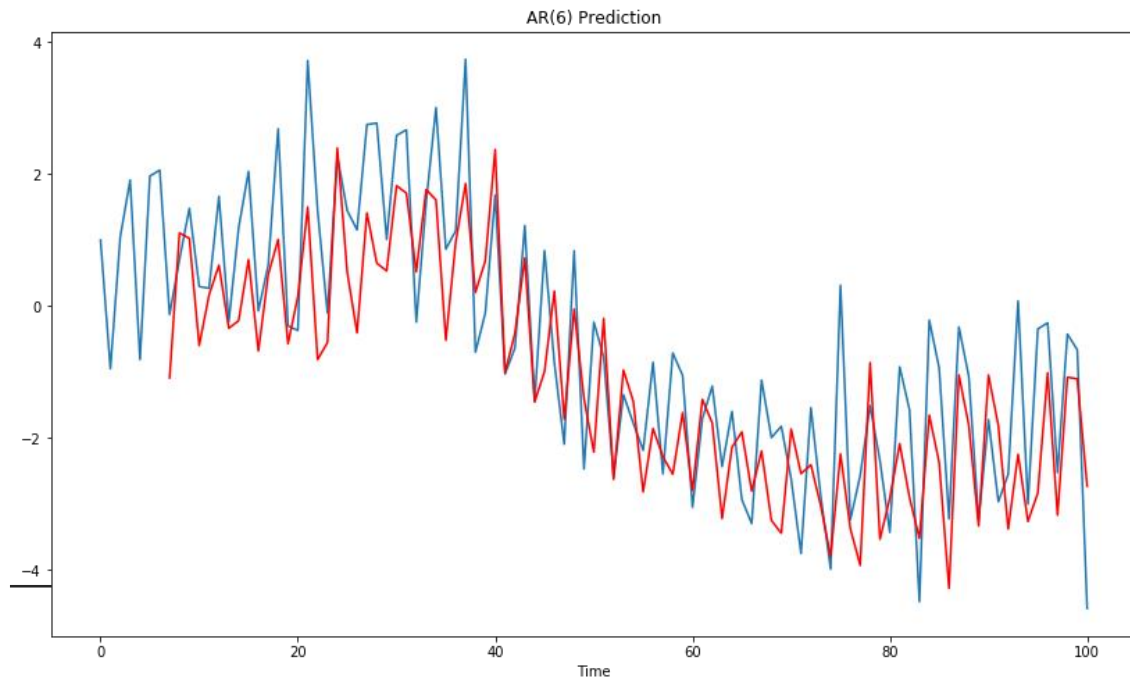
model_fit = ARMA(X, order=(1,0)).fit()

#获取回归方程的系数
coefs = model_fit.params

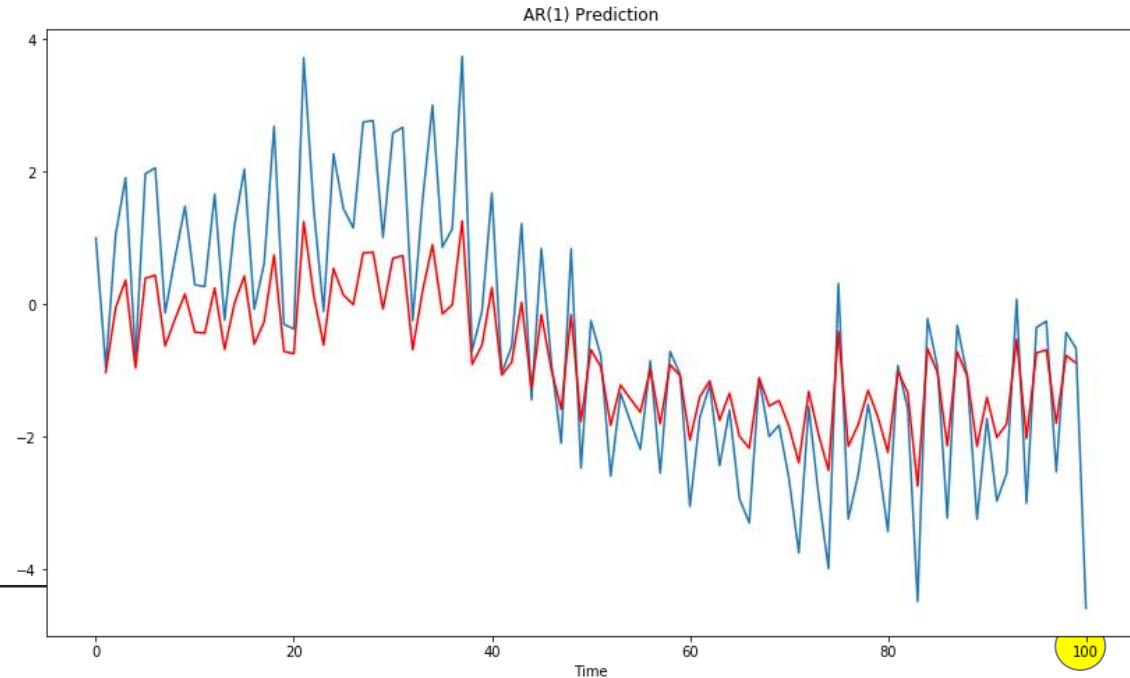
#  $y[t] = coefs[0] + coefs[1]x[t - 1] + \dots$ 

```

AR(6) --- MSE = 1.16



AR(1) --- MSE = 2.91



# 序列特征提取

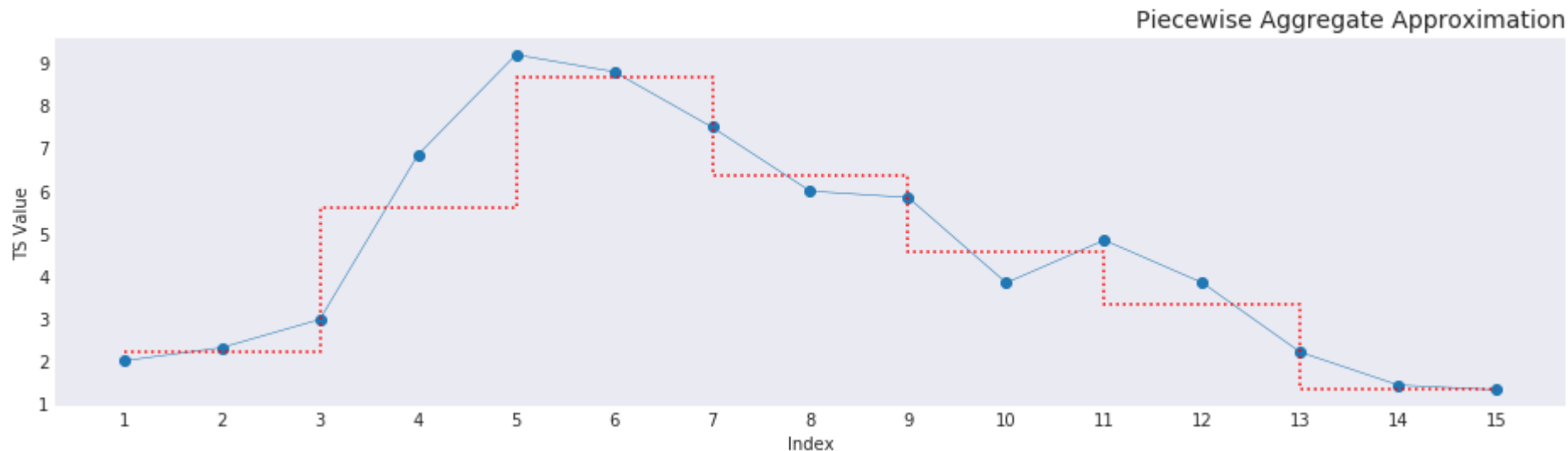
## 时间序列的分段特征

在这种算法中，分段聚合逼近 (Piecewise Aggregate Approximation) 是一种非常经典的算法。假设原始的时间序列是  $C = \{x_1, \dots, x_N\}$ ，定义 PAA 的序列是：

$$\bar{C} = \{\bar{x}_1, \dots, \bar{x}_w\},$$

其中

$$\bar{x}_i = \frac{w}{N} \cdot \sum_{j=\frac{N}{w}(i-1)+1}^{\frac{N}{w}i} x_j.$$



# 高阶选项：RCNN神经网络模型

- 分段词汇化后做word embedding + RNN
- 直接采用RCNN建模





# 序列分析与金融交易模型回归

## 参考资料

<https://online.stat.psu.edu/stat510/lesson/1>

<https://otexts.com/fpp2/> Forecasting: Principles and Practice

[http://www.math.pku.edu.cn/teachers/lidf/course/fts/ftsnotes/html/\\_ftsnotes/index.html](http://www.math.pku.edu.cn/teachers/lidf/course/fts/ftsnotes/html/_ftsnotes/index.html) 金融时间序列分析讲义

<https://nwfsc-timeseries.github.io/atsa-labs/> Applied Time Series Analysis for Fisheries and Environmental Sciences

<https://zr9558.com/2019/01/21/timeseriesclustering-2/> 时间序列的聚类

<https://zhuanlan.zhihu.com/p/39105270> 时间序列的表示与信息提取

Time-series clustering – A decade review 见pdf

# 期中大作业

- 主题一： 量化回测系统与股票策略设计 by 熊江凯助教



## 主题二： 基于观影数据集的数据分析与挖掘

数据环境： movielens数据集

