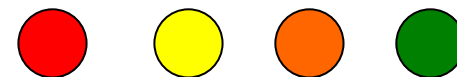


图像压缩编码与DCT (C14)

时间序列基础

信息科学与技术学院

胡俊峰



内容

- 图像聚类问题
- 图像频域特征
- DCT变换与图像压缩



图像向量空间特征、图像聚类

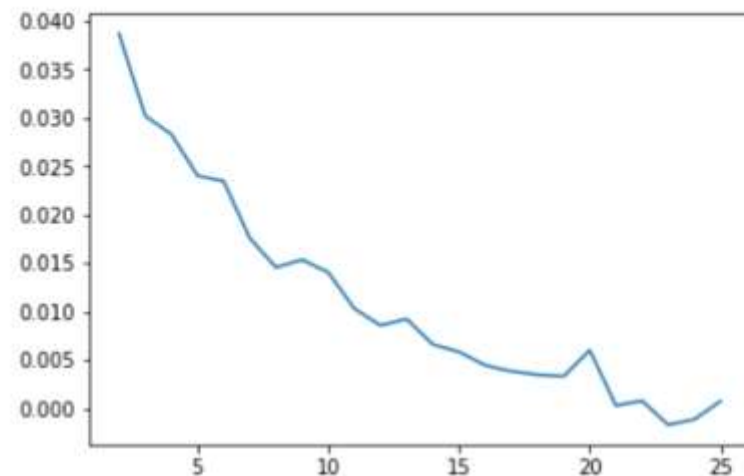
```
1  # from img2vec_pytorch import Img2Vec
2  # img2vec = Img2Vec(cuda=False) # 装入已经预训练好的模型
3  # img_embed = img2vec.get_vec(imgs).squeeze() # 得到图片的嵌入向量 1min
4
5  #img_embed.shape
6  #np.save("data/poster_embed", img_embed) #保存数组
7
8  img_embed = np.load('data/poster_embed.npy')
9  img_embed.shape
```

```
(2938, 512)
```



- 多次尝试找到相对好的聚类结果
- 每次随机种子要设定为不一样

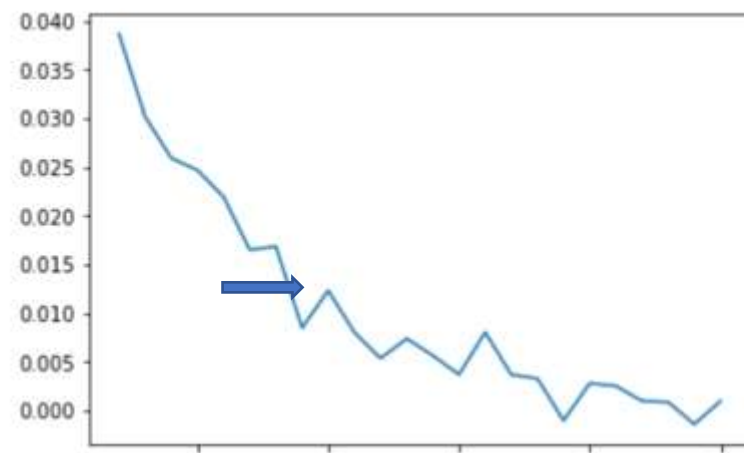
Out[5]: [



In [6]:

```
1 sc = []
2 for N_CL in range(2, 26): # 尝试2-26个类质心,
3     clf = KMeans(n_clusters = N_CL, random_state=2).fit(img_embed)
4     labels = clf.labels_
5     eval_score = metrics.silhouette_score(img_embed, labels, metric='eucli
6     sc.append(eval_score)
7
8 plt.plot(np.arange(2,26), np.array(sc)) # 画出类属相关的loss值
```

Out[6]: [



cluster 0: ['Western', 'Action', 'Crime']

['Animation', 'Children's']



['Drama', 'Thriller']



['Comedy']



['Action', 'Comedy', 'Drama']



['Crime', 'Drama', 'Thriller']



['Adventure', 'Sci-Fi']



['Drama', 'Sci-Fi']



cluster 1: ['Musical', 'Film-Noir', 'Animation']

['Adventure', 'Children's', 'Fantasy']



['Comedy']



['Documentary']



['Drama']



['Drama', 'Musical', 'Adventure', 'Children's', 'Comedy', 'Fantasy', 'Romance']



cluster 2: ['Horror', 'Sci-Fi', 'Thriller']

cluster 2: ['Horror', 'Sci-Fi', 'Thriller']



cluster 3: ['Crime', 'Horror', 'War']



['Drama', 'Romance']



['Drama']



['Adventure', 'Romance']



['Drama']



['Comedy', 'Romance']



['Sci-Fi', 'Thriller']



['Comedy']



cluster 5: ['Comedy', 'Romance', 'Drama']

['Comedy']



['Adventure', 'Children's']



['Action']



['Thriller']



['Comedy', 'Romance']



['Drama']



['Crime', 'Thriller']



cluster 6: ['Romance', 'Drama', 'War']

聚类的评价：纯度

- 每个类与目标类重合最多的数目占比总数：

$$\text{purity}(\Omega, \mathbb{C}) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|$$

参考： <https://blog.csdn.net/fangqi100/article/details/129651239>



对不同类的海报观察分析:

- 年龄偏好?
- 职业偏好?
- 与标注类型对比分析?

```
1 cluster_genres_distrib = [] # 各个聚类的电影类型分布?
2
3 for INDEX in range(10):
4     ind = clf.predict(img_embed) == INDEX # 返回第i个类为True的布尔下标
5
6     imgs = np.array(imgs, dtype=object)
7     example_images = imgs[ind] # 只有numpy数组才能正常用布尔下标
8     img_id = np.array(img_ids)[ind]
9     genres_cls = np.array(gen_labels)[ind]
10    NUMS = 7
11    fig = plt.figure(figsize=(16, 7))
12    for i in range(NUMS):
13        plt.subplot(1, NUMS, i+1); plt.axis('off')
14        plt.imshow(example_images[i])
15        plt.title( data[img_id[i]][-1], size = 8) # 显示电影类型
16
17    perc = np.sum(genres_cls, axis=0) / np.sum(gen_labels, axis=0) # 得到每类
18    cluster_genres_distrib.append(perc)
19
20    fig.text(0.4, 0.8, f'cluster {INDEX}: {[genres[k] for k in np.argsort(-
21    plt.show()
```



性别偏 好电影A:

Sense and Sensibility (1995)



Shakespeare in Love (1998)



Gone with the Wind (1939) Four Weddings and a Funeral (1994)



Emma (1996)



My Fair Lady (1964)



Like Water for Chocolate (Como Agua para Chocolate) (1992)



Pretty Woman (1990)



Breakfast at Tiffany's (1961)



Strictly Ballroom (1992)



Babe (1995)



Clueless (1995)



Sleepless in Seattle (1993)



Dirty Dancing (1987)



My Best Friend's Wedding (1997) When Harry Met Sally... (1989)



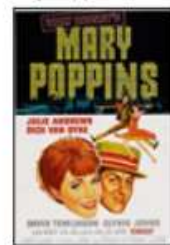
While You Were Sleeping (1995) Little Mermaid, The (1989) American President, The (1995)



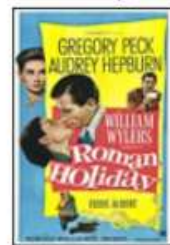
Elizabeth (1998)



Mary Poppins (1964)



Roman Holiday (1953)



Room with a View, A (1986)



Singin' in the Rain (1952)



Moonstruck (1987)



Ever After: A Cinderella Story (2002)



Titanic (1997)



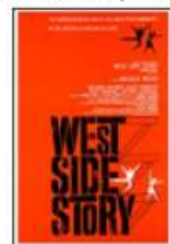
You've Got Mail (1998)



Lady and the Tramp (1955)



West Side Story (1961)



Beauty and the Beast (1991)



Notting Hill (1999)



Much Ado About Nothing (1993)



Sabrina (1954)



Circle of Friends (1995)



Jerry Maguire (1996)



Edward Scissorhands (1990)



Dangerous Liaisons (1988)



Philadelphia Story, The (1940)



Erin Brockovich (2000)

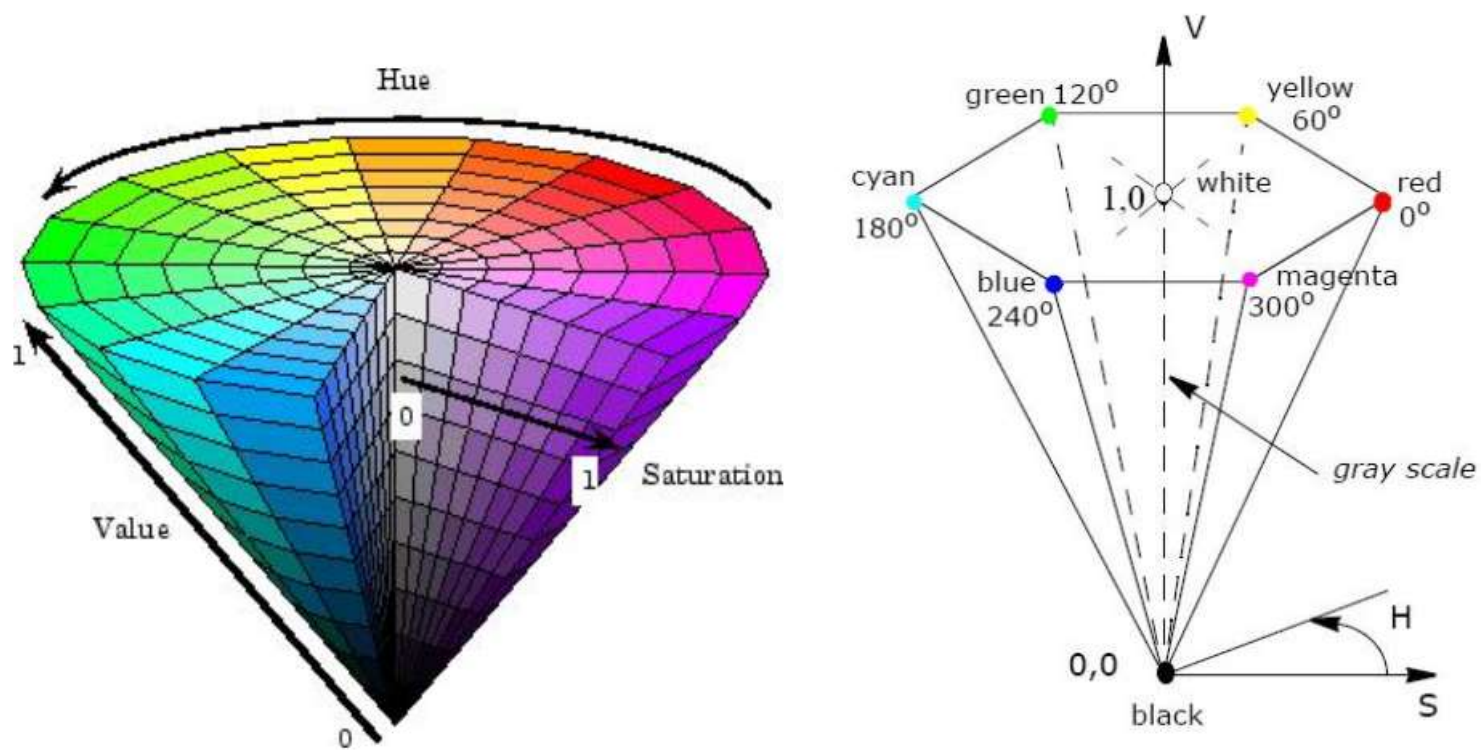


性别偏好电影B:



HSV空间:

在HSV模型中, 颜色是由色度 (Hue), 饱和度 (Saturation), 明度 (Value) 共同组成。如图所示。

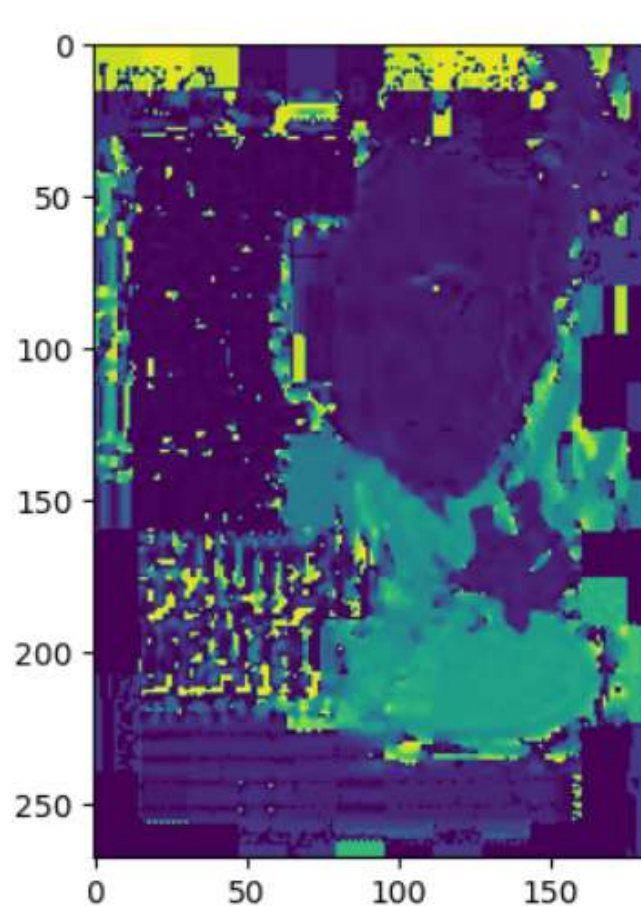


计算图像的明度、饱和度和色调

```
: ▶ hsv=cv2.cvtColor(img,cv2.COLOR_RGB2HSV)  
H, S, V=cv2.split(hsv)
```

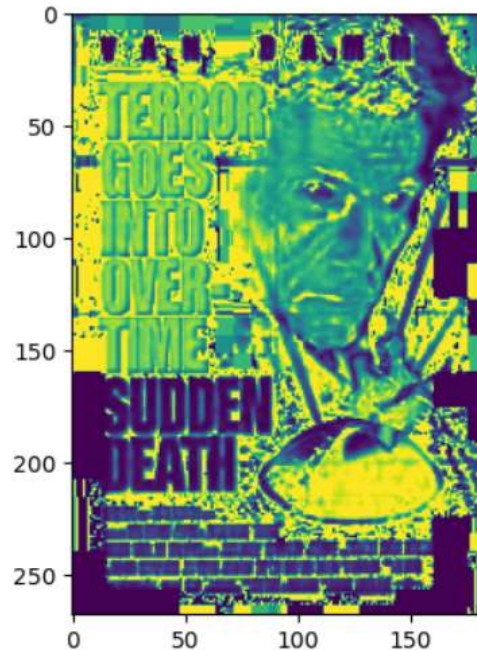
```
: ▶ ## 色调 (H)  
h = H.ravel()[np.flatnonzero(H)]  
average_h = sum(h)/len(h)  
plt.imshow(H)  
print(average_h)
```

47.50274173138476



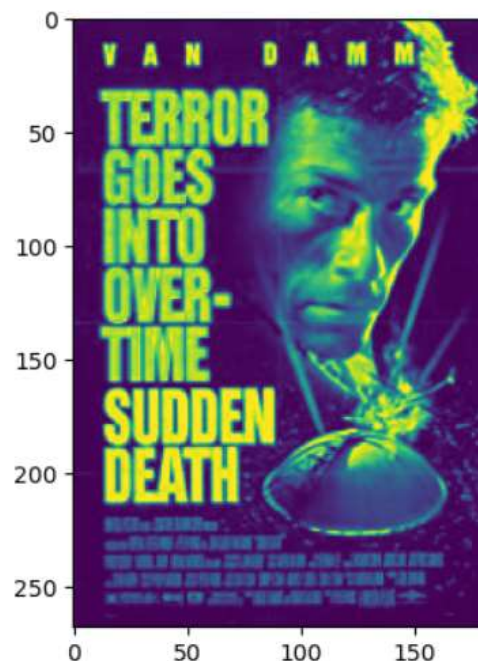
```
## 饱和度 (S)  
s = S.ravel()[np.flatnonzero(S)]  
average_s = sum(s)/len(s)  
plt.imshow(S)  
print(average_s)
```

153.247234236285



```
## 明度 (V)  
v = V.ravel()[np.flatnonzero(V)]  
average_v = sum(v)/len(v)  
plt.imshow(V)  
print(average_v)
```

80.84620480382938



图像色彩鲜艳程度

Measuring colorfulness in natural images [David Hasler](#), [Sabine E. Suesstrunk](#)

```
1 def image_colorfulness(img): #假设输入是RGB图像
2     R, G, B = cv2.split(img.astype('float'))
3
4     rg = np.absolute(R - G)
5     yb = np.absolute(0.5 * (R + G) - B)
6
7     (rgMean, rgStd) = (np.mean(rg), np.std(rg))
8     (ybMean, ybStd) = (np.mean(yb), np.std(yb))
9
10    stdRoot = np.sqrt(rgStd ** 2 + ybStd ** 2)
11    meanRoot = np.sqrt(rgMean ** 2 + ybMean ** 2)
12
13    return stdRoot + 0.3 * meanRoot
```

$$\begin{aligned}rg &= R - G \\yb &= \frac{1}{2}(R + G) - B\end{aligned}$$

$$\begin{aligned}\hat{M}^{(3)} &= \sigma_{rgyb} + 0.3 \cdot \mu_{rgyb}, \\ \sigma_{rgyb} &:= \sqrt{\sigma_{rg}^2 + \sigma_{yb}^2}, \\ \mu_{rgyb} &:= \sqrt{\mu_{rg}^2 + \mu_{yb}^2},\end{aligned}$$

图像聚类

对于海报数据（poster文件夹），将每张图片的平均亮度（灰度值）和色彩丰富程度（参考ppt相关部分和[论文链接](#)）作为每张图片的特征，使用该特征对图像尝试聚类。

对于聚类中的每个数据点 X_i ，定义 a_i 为 X_i 到所在聚类的其它数据点的平均距离， b_i 为 X_i 到其它各个聚类的数据点平均距离的最小值，这个数据点的Silhouette可以计算为 $s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$ ，Silhouette score正是所有数据点Silhouette的平均值。直觉上，[Silhoutte](#)描述的是聚类的“合理性”——感兴趣请阅读wiki。

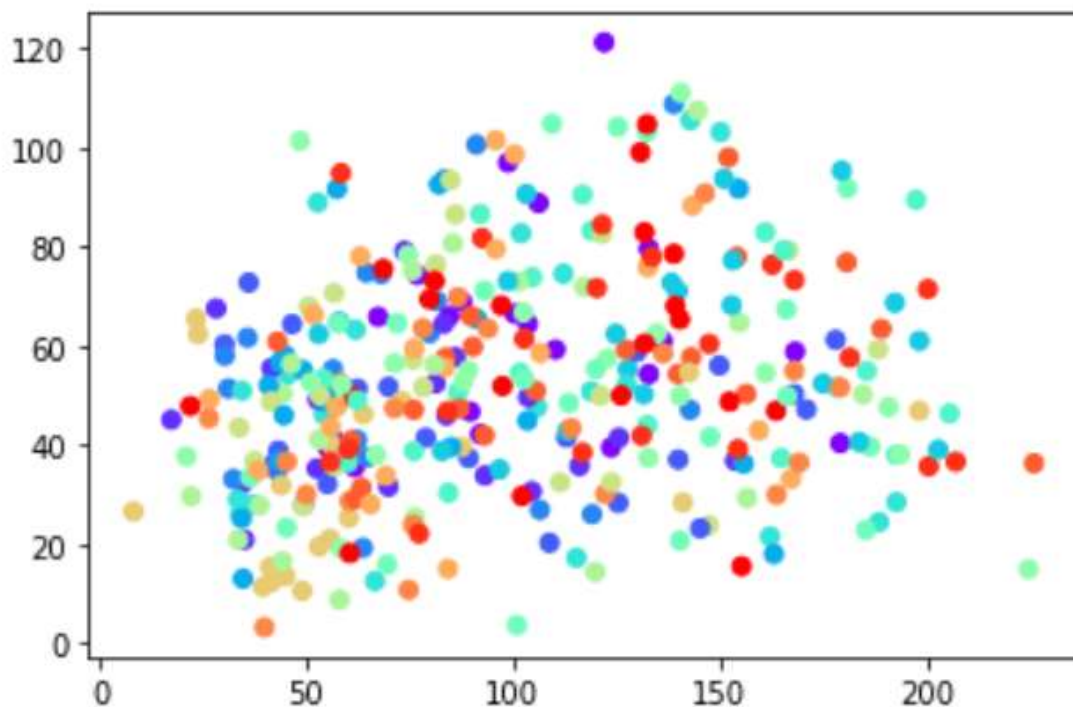
我们可以根据[Silhouette score](#)随聚类数量的变化，来评估何时达到了合理的聚类数。一般而言Silhouette score取极大值，且每个类的样本数均没有特别小的特殊情况时，可以认为这时是一个较为合理的聚类。



直接观察分布

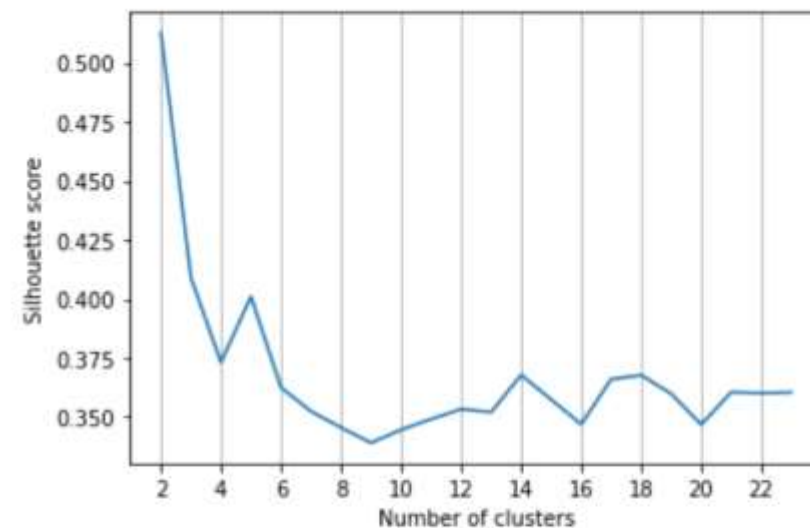
```
1 plt.scatter(features[:,0], features[:,1], c = colors[labels]) # 显示类别
```

<matplotlib.collections.PathCollection at 0x22095be6b20>



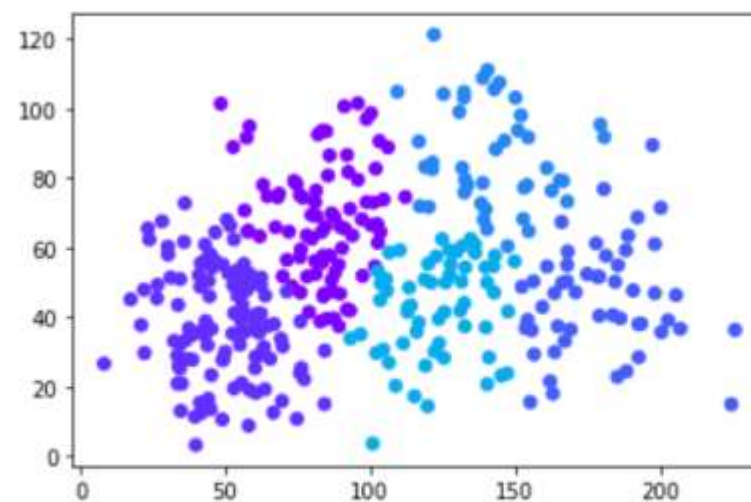
聚类及优化分析:

Out[90]: [



```
In [91]: 1 clf = KMeans(n_clusters=5, random_state=42)
          2 pred = clf.fit_predict(features)
          3
          4 plt.scatter(features[:, 0], features[:, 1], c = colors[pred])
```

Out[91]: <matplotlib.collections.PathCollection at 0x22095ed0820>



图像的DCT变换与压缩编码

- 频域变换
- 降低高频分量的分辨率
- 降阶编码-压缩



Quantization

- Quantization is the process of approximating a continuous (or range of values) by a (much) smaller range of values

$$Q(x, \Delta) = \text{Round} \left(\frac{x + 0.5}{\Delta} \right)$$

- Where $\text{Round}(y)$ rounds y to the nearest integer
- Δ is the quantization stepsize



量化Quantization

- Quantization plays an important role in lossy compression
 - This is where the loss happens
 - Example: $\Delta=2$



Fundamentals of images

- Here is an image represented with 8-bits per pixel



Fundamentals of images

- Here is the same image at 7-bits per pixel



Fundamentals of images

- And at 6-bits per pixel



Fundamentals of images

- And at 5-bits per pixel



Fundamentals of images

- And at 4-bits per pixel



Fundamentals of images

- Do we need all these bits?
 - No!
- The previous example illustrated the eye's sensitivity to luminance
- We can build a perceptual model
 - Only code what is important to the human visual system (HVS)
 - Usually a function of spatial frequency



Fundamentals of Images

- Just as audio has temporal frequencies
- Images have spatial frequencies
- Transforms
 - Fourier transform
 - **Discrete cosine transform**
 - Wavelet transform
 - Hadamard transform



Discrete cosine transform

- Forward DCT

$$S(u) = \frac{C(u)}{2} \sum_{n=0}^{N-1} s(n) \cos\left(\frac{u\pi}{8}(n + 0.5)\right)$$

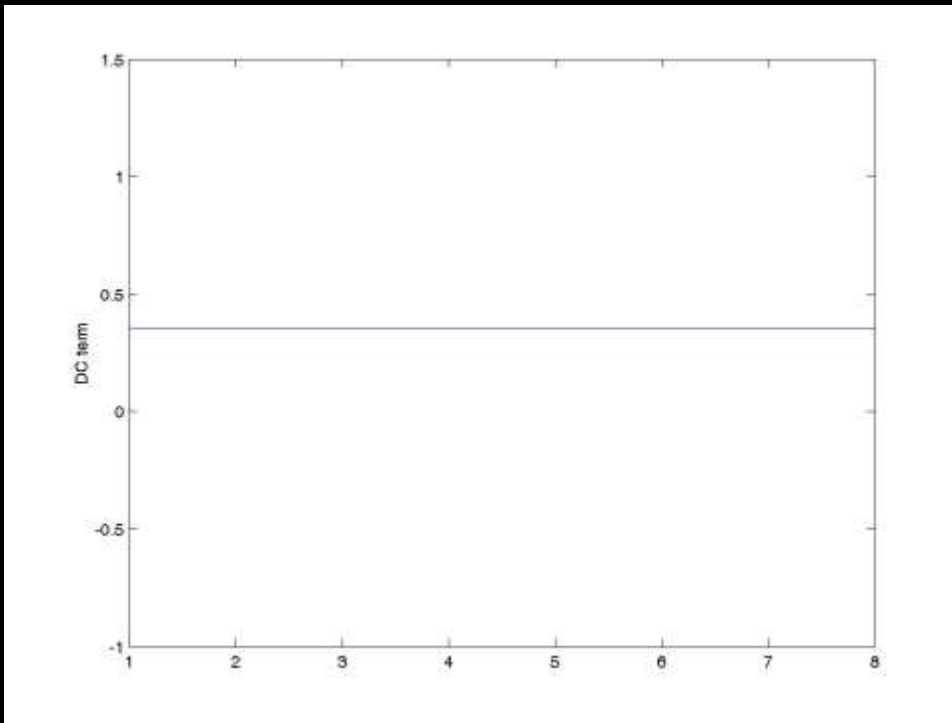
- Inverse DCT

$$s(n) = \frac{C(u)}{2} \sum_{u=0}^{N-1} S(u) \cos\left(\frac{u\pi}{8}(n + 0.5)\right)$$



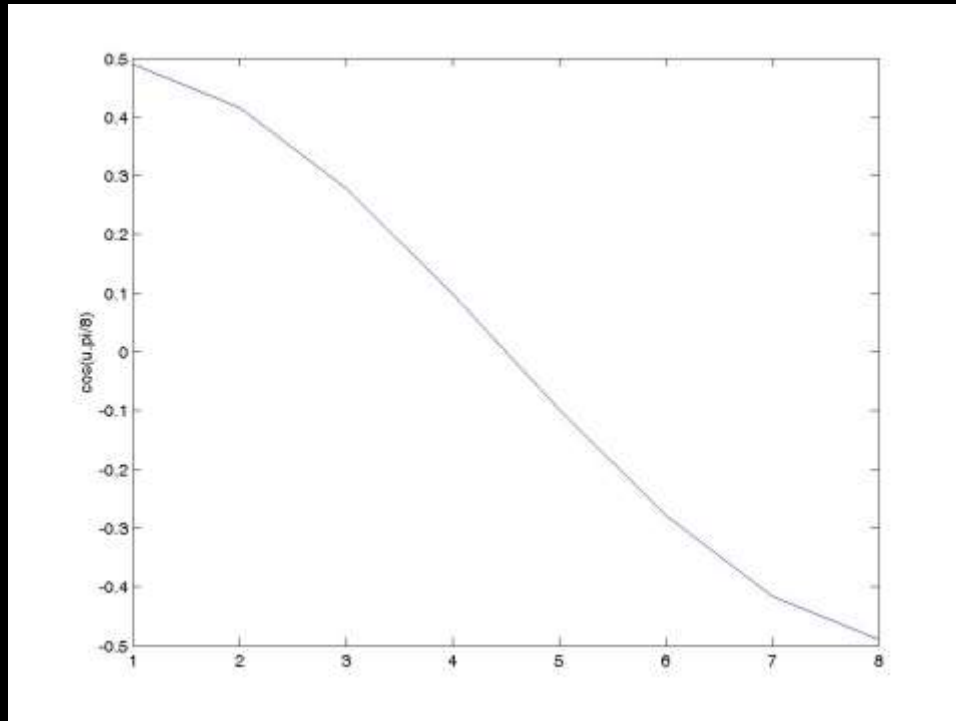
Basis functions

- DC term



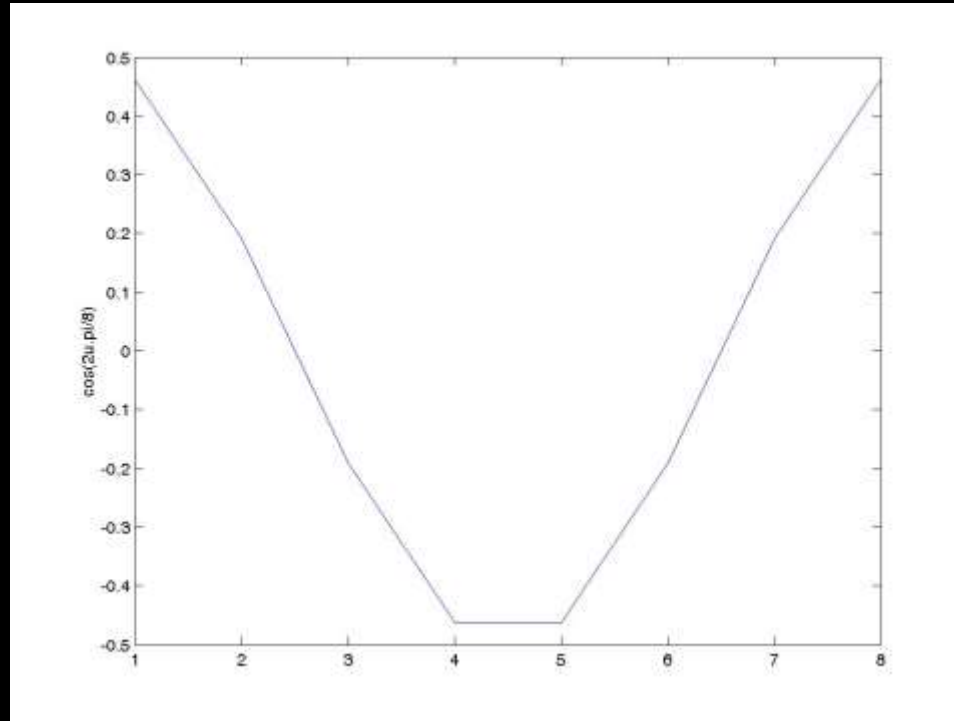
Basis functions

- First term



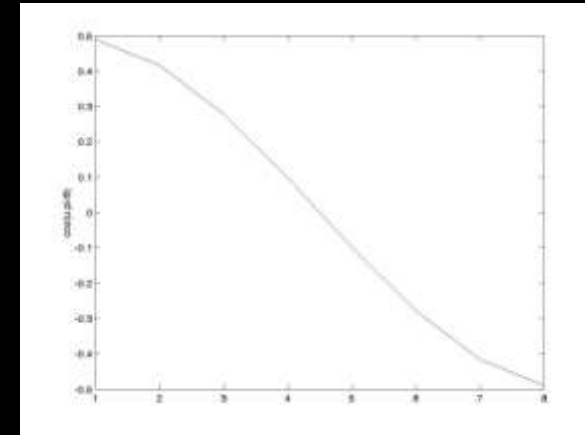
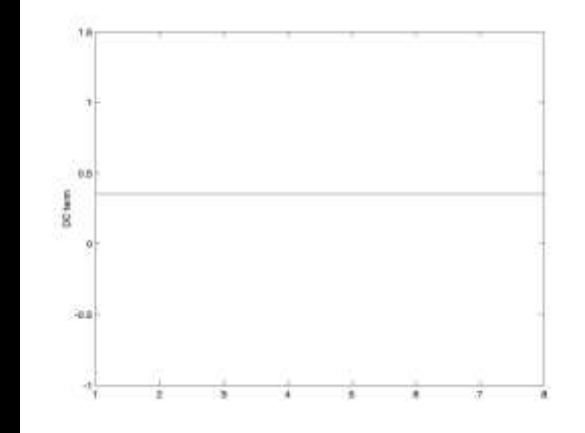
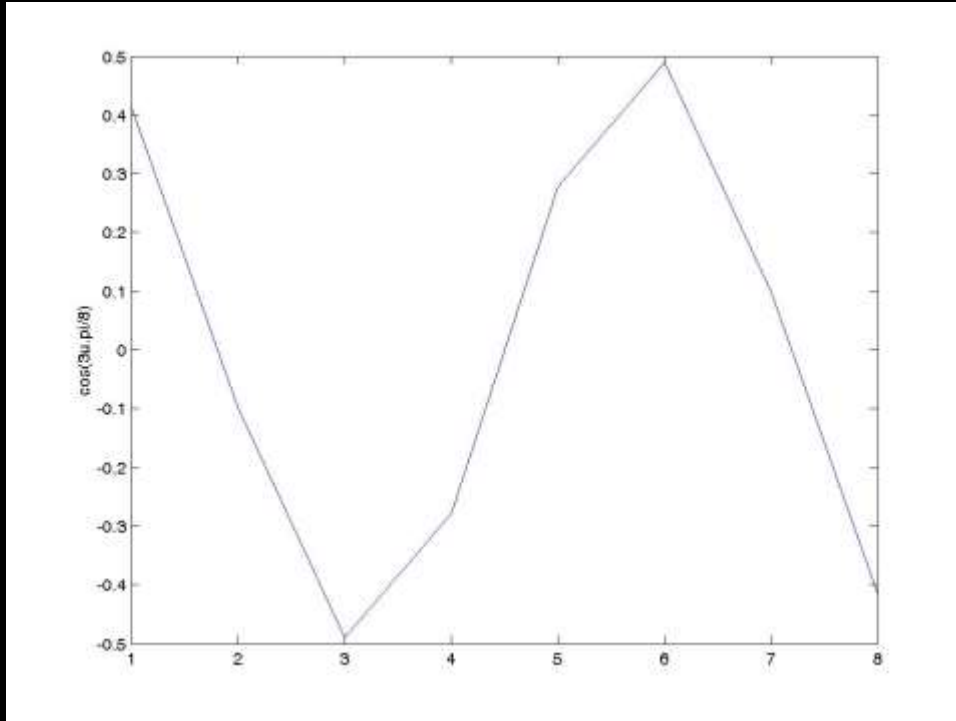
Basis functions

- Second term



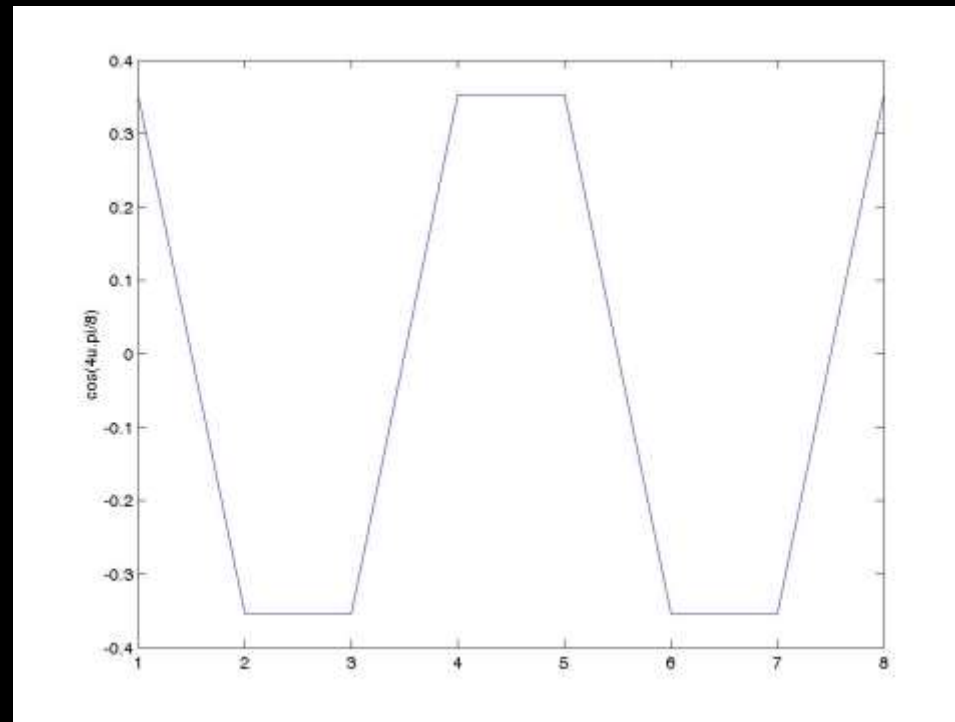
Basis functions

- Third term



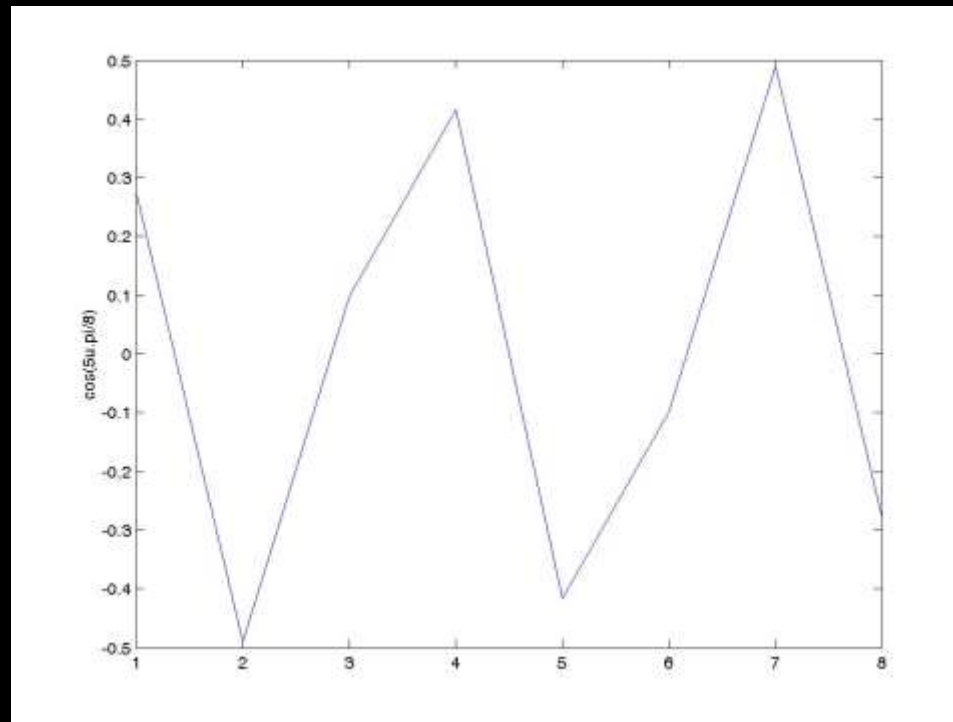
Basis functions

- Fourth term



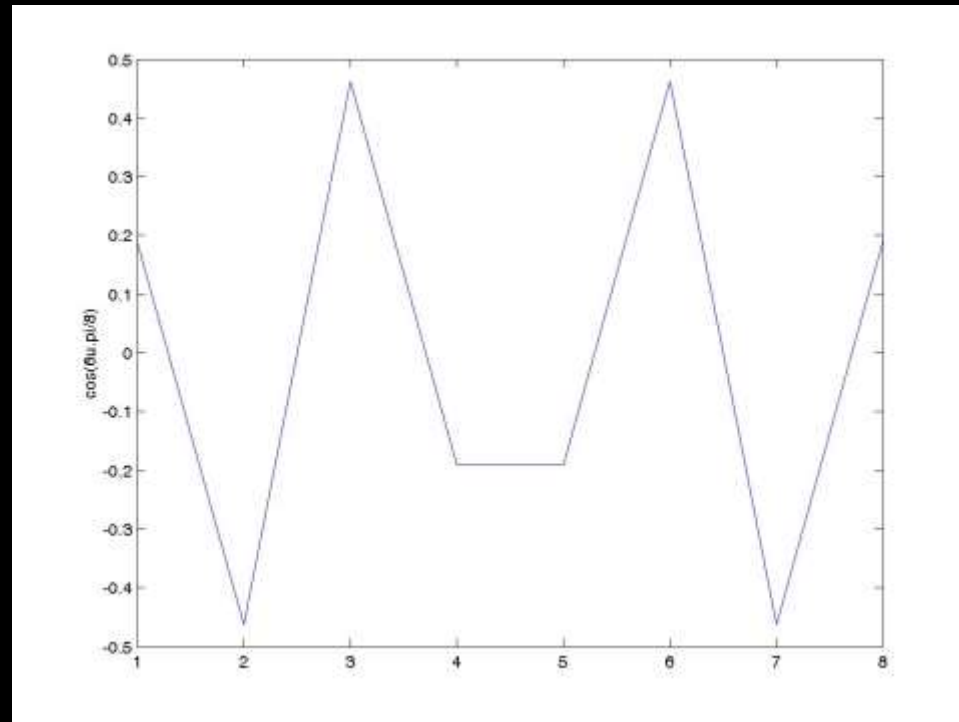
Basis functions

- Fifth term



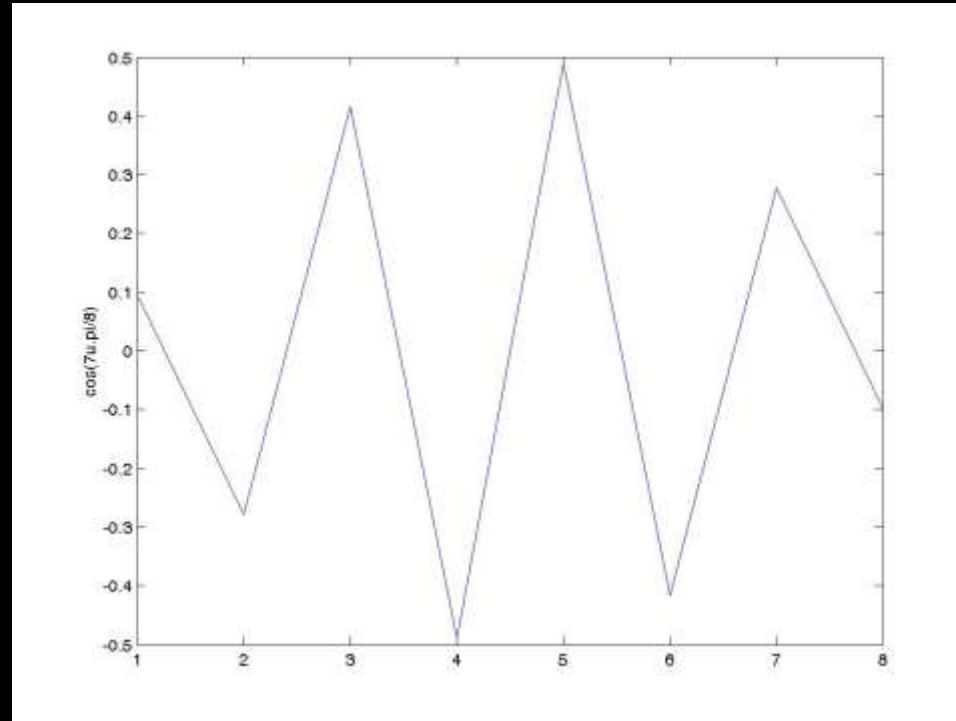
Basis functions

- Sixth term

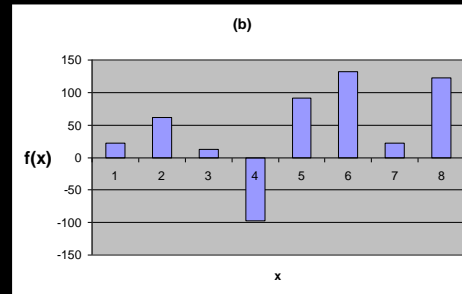


Basis functions

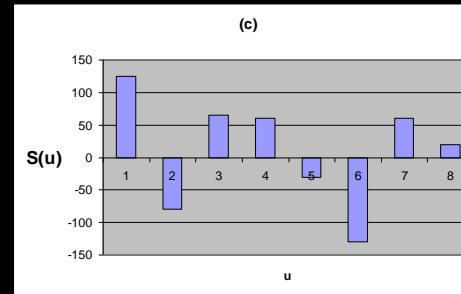
- Seventh term



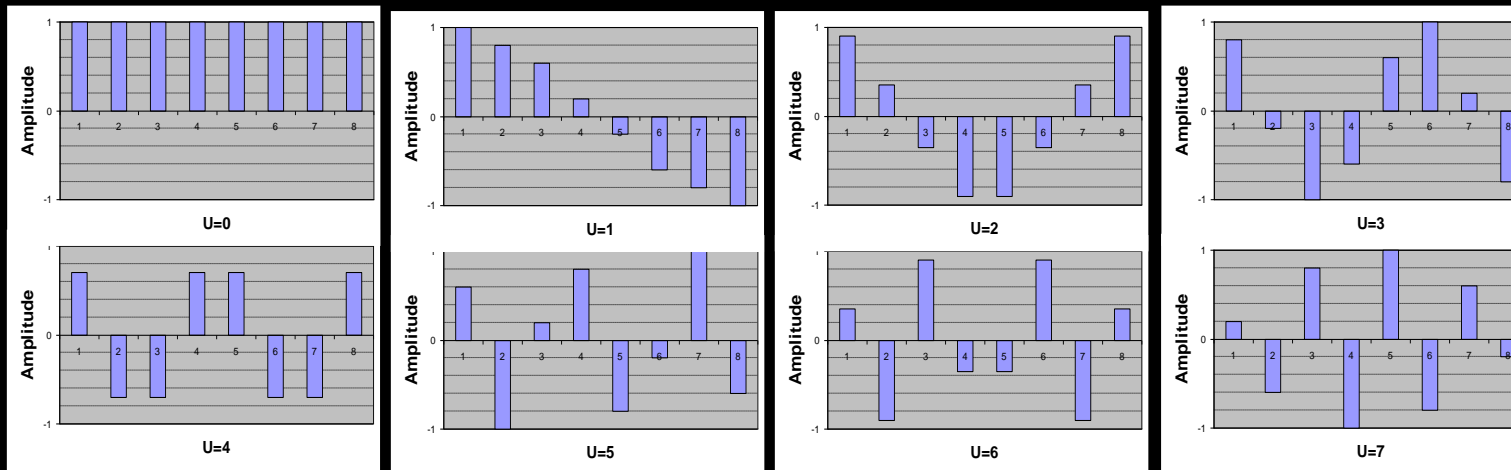
Another example of 1-D DCT decomposition



Before DCT (image data)



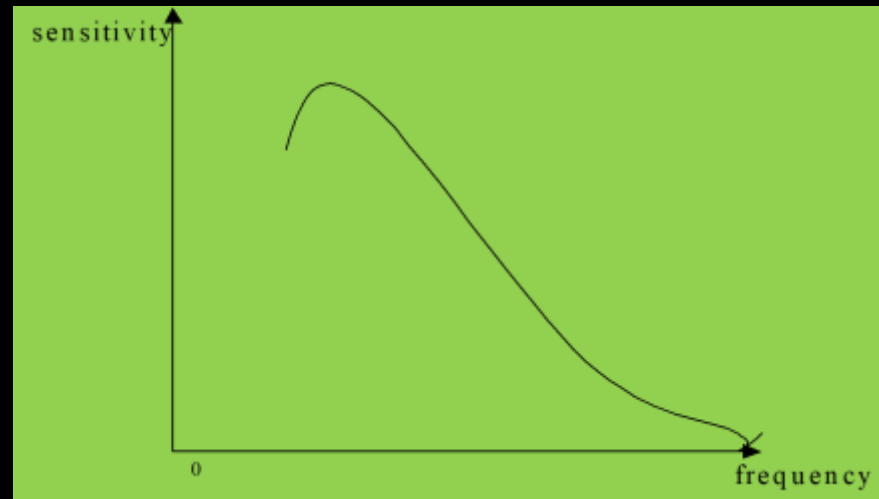
After DCT (coefficients)



The 8 basic functions for 1-D DCT

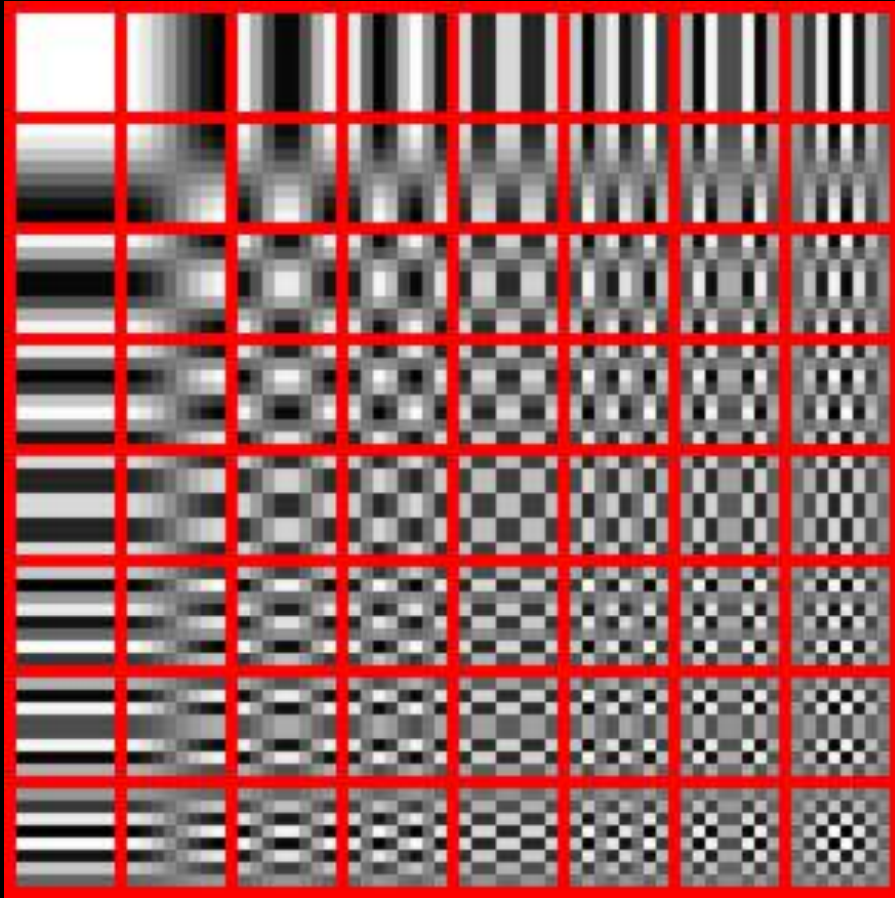
Why does it work?

- Lossy encoding
- HVS is generally more sensitive to low frequencies
- Natural images

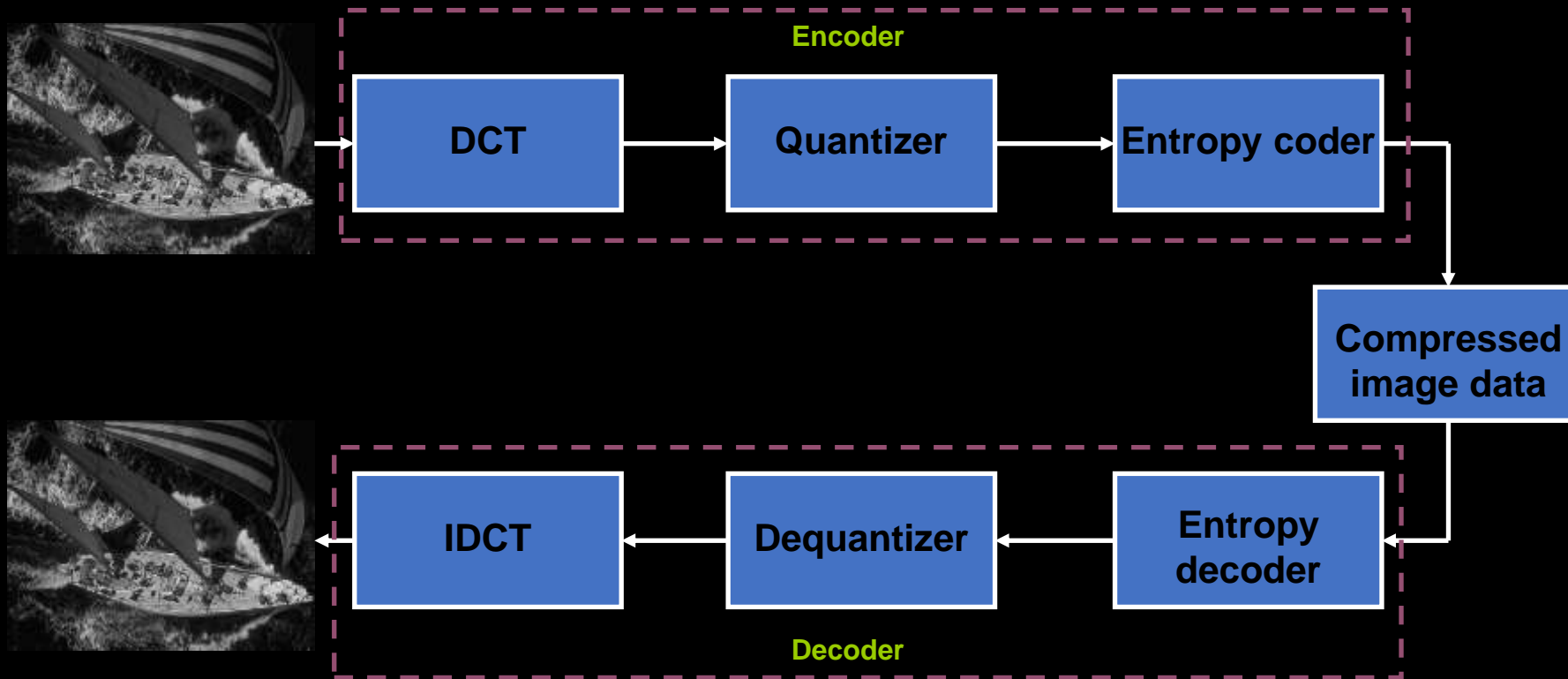


Fundamentals of images

- Basis functions for the 8×8 DCT (courtesy Wikipedia)



Fundamentals of JPEG



Fundamentals of JPEG

- JPEG works on 8×8 blocks
- Extract 8×8 block of pixels
- Convert to DCT domain
- Quantize each coefficient
 - Different stepsize for each coefficient
 - Based on sensitivity of human visual system
- Order coefficients in zig-zag order
- Entropy code the quantized values



Fundamentals of JPEG

- A common quantization table is

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99



CV部分内容总结

- 基础
- 特征分析
- 图像频域变换



内容摘要

- 序列成分分解基础
- 平稳序列与序列平稳化
- 序列谱分析与滤波



时间序列成分分析

- 基本成分：趋势、周期、随机扰动
- 加法、乘法合成与成分分解



时间序列基本成分：趋势、周期、随机扰动

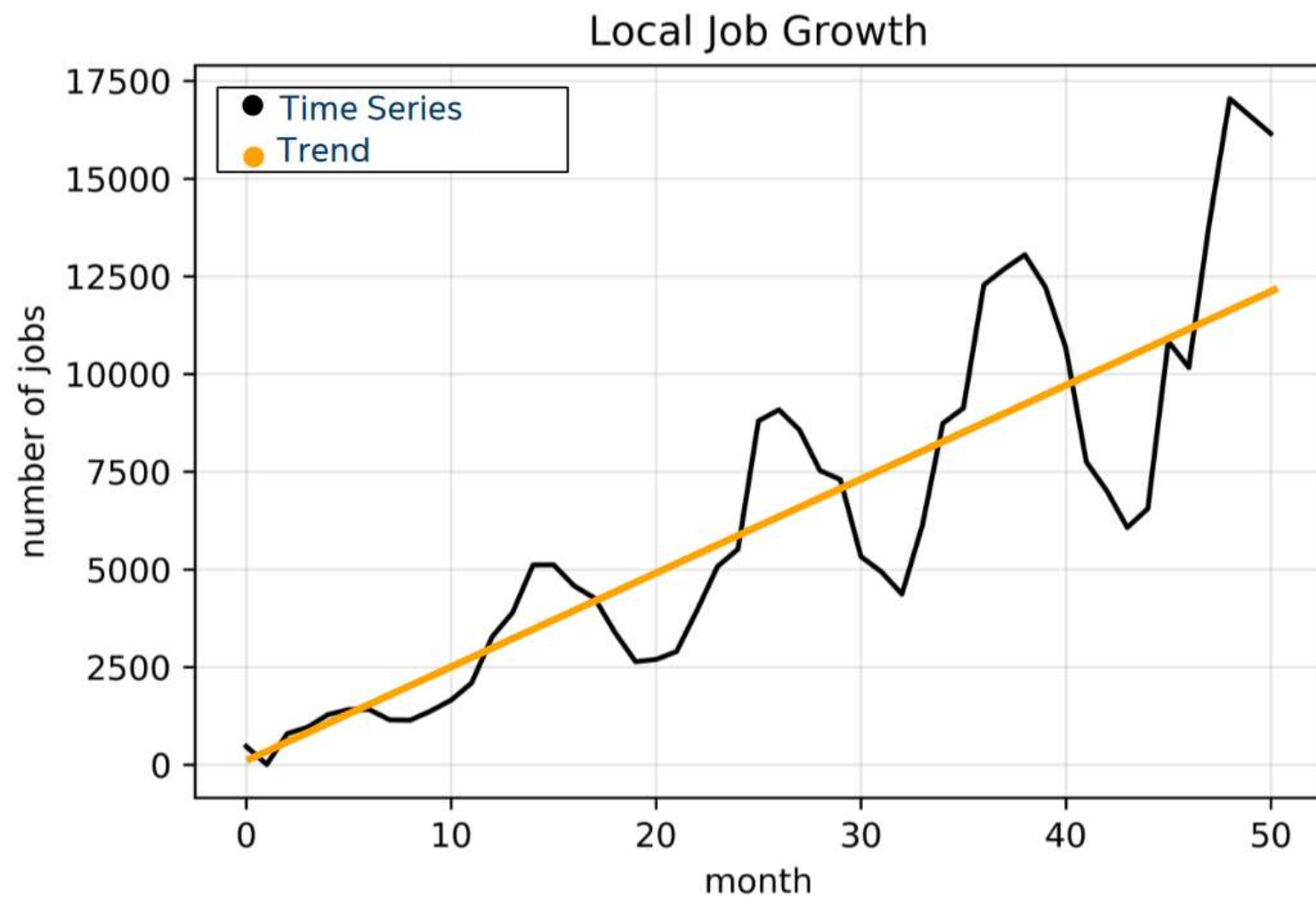
► 模型假设：

$$y_t = S_t + T_t + R_t \quad (y_t = S_t * T_t * R_t)$$

- 直流分量(level)
- 趋势 (trend)
- 周期特征 (Seasonality) 、非固定周期 (Cyclic Changes)
- 随机扰动 (Residual-irregular fluctuations, noise)



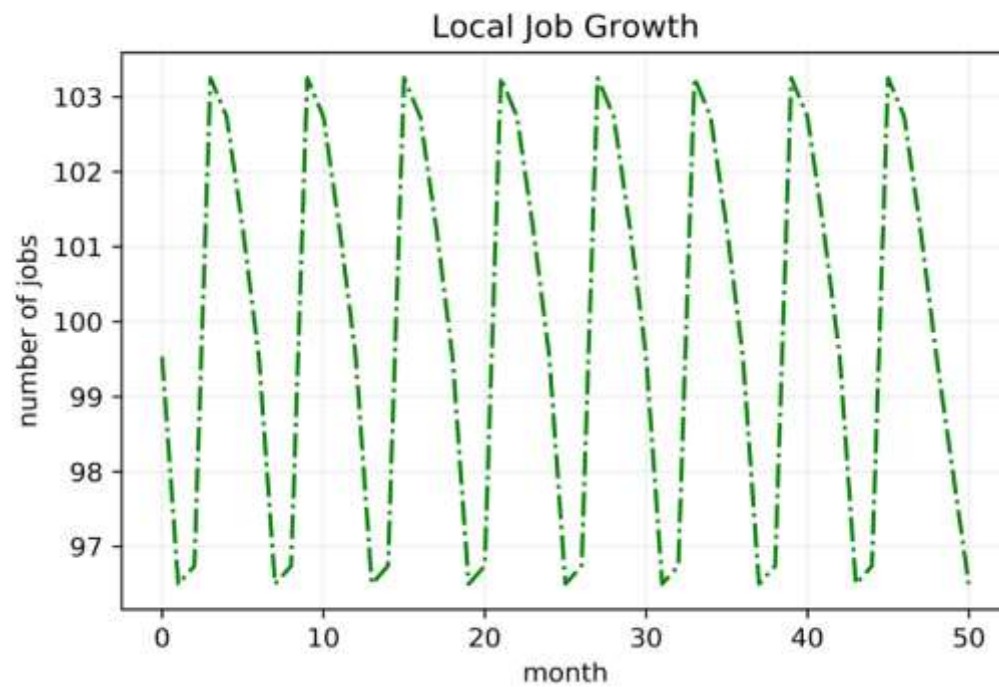
趋势：Trend



周期性：Seasonality

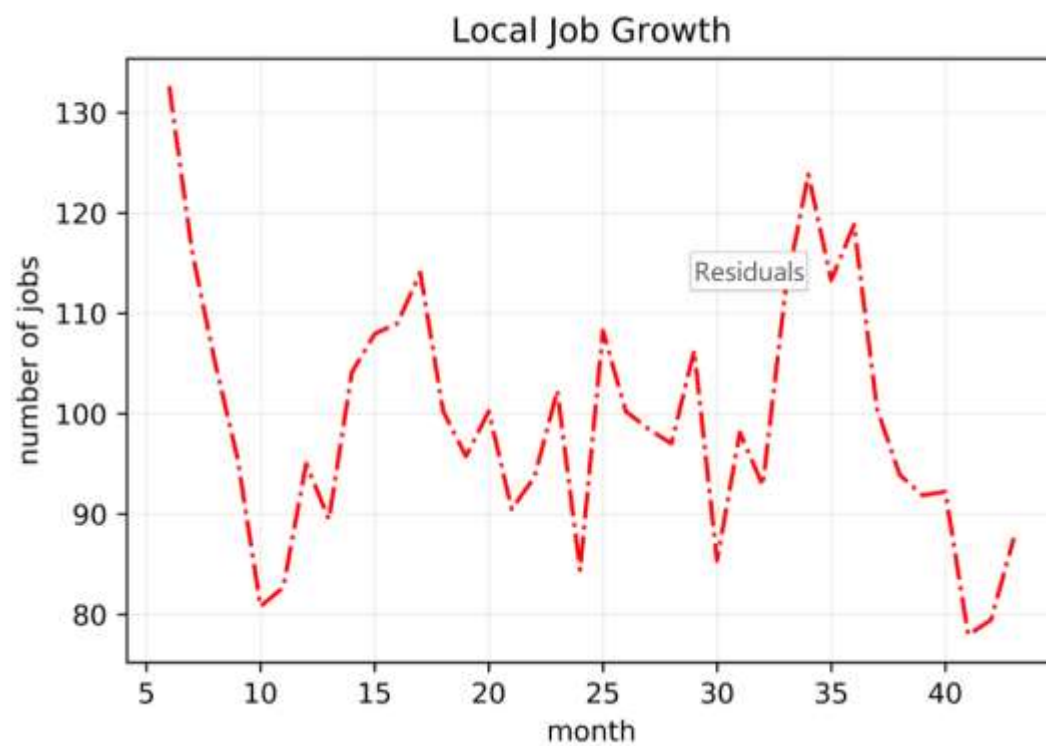
—— 多个圆周运动的叠加投影

•



随机残量 Residuals

- 高斯（随机）噪声



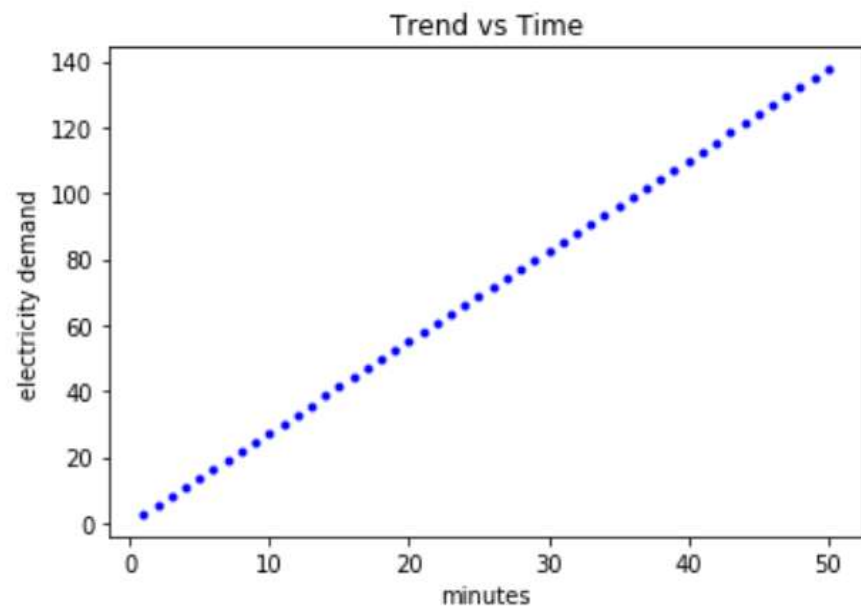
生成信号：趋势、周期波动

```
3 import sys
4 import statsmodels as sm # statistical models
5 import numpy as np
6 import matplotlib
7 import matplotlib.pyplot as plt
```

```
1 time = np.arange(1, 51)
```

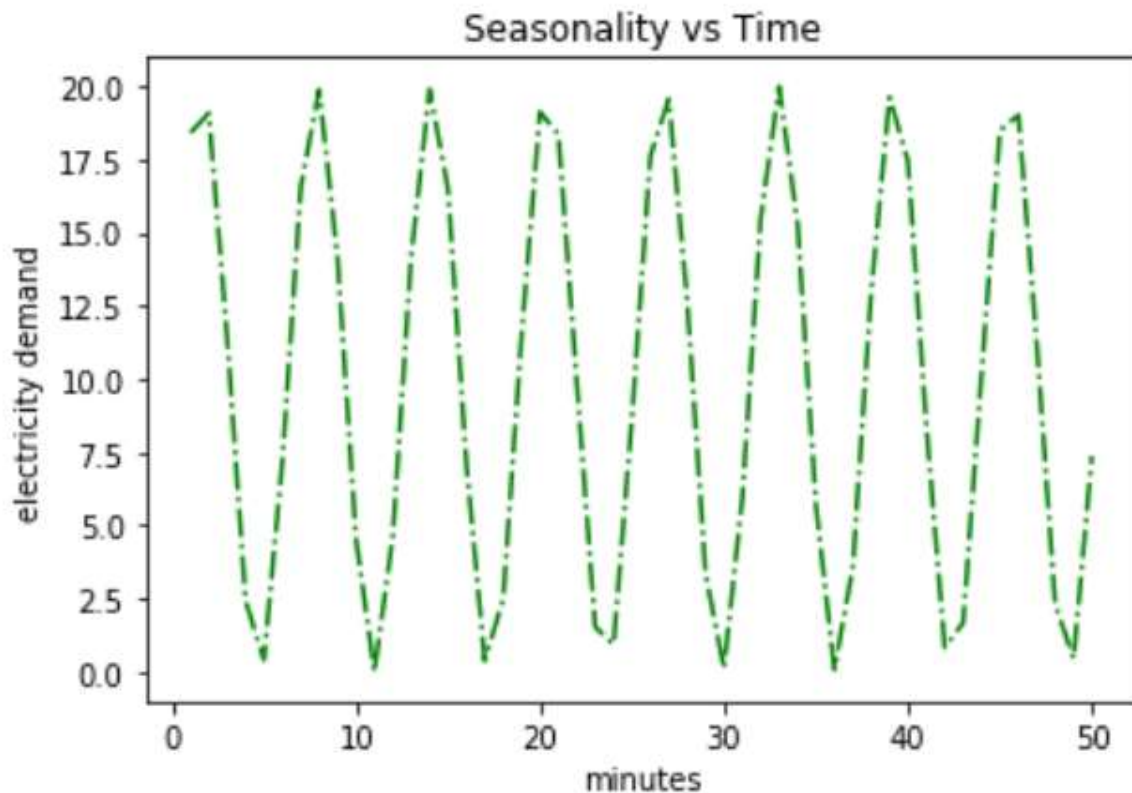
```
1 trend = time * 2.75
```

```
1 plt.plot(time, trend, 'b.')
2 plt.title("Trend vs Time")
3 plt.xlabel("minutes")
4 plt.ylabel("electricity demand");
```



```
1 seasonal = 10 + np.sin(time) * 10
```

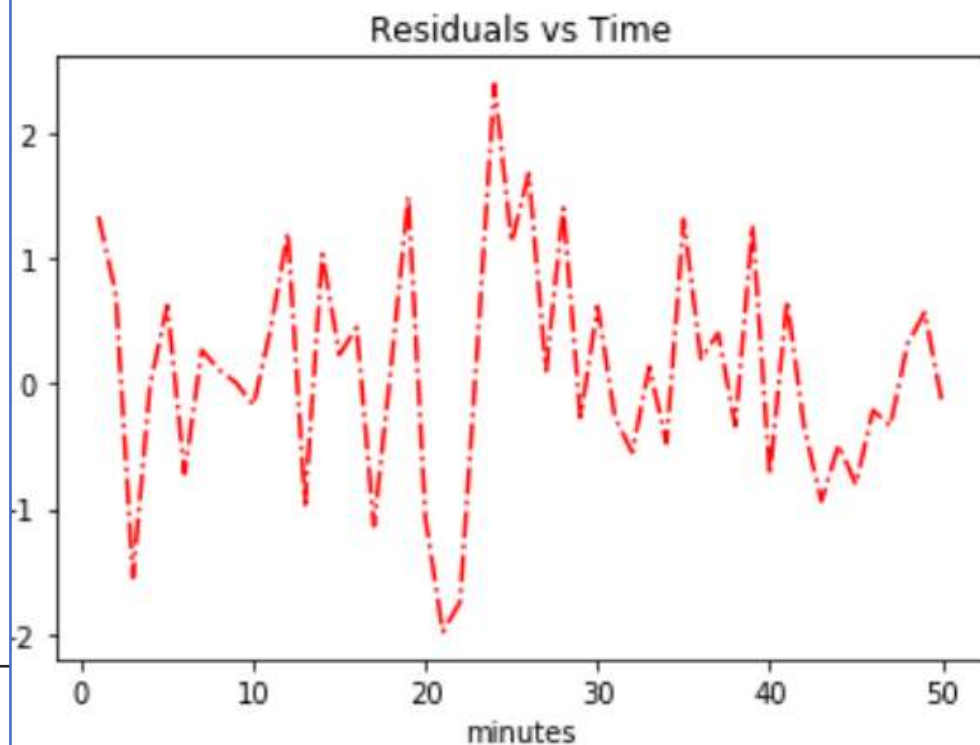
```
1 plt.plot(time, seasonal, 'g-.')
2 plt.title("Seasonality vs Time")
3 plt.xlabel("minutes")
4 plt.ylabel("electricity demand");
```



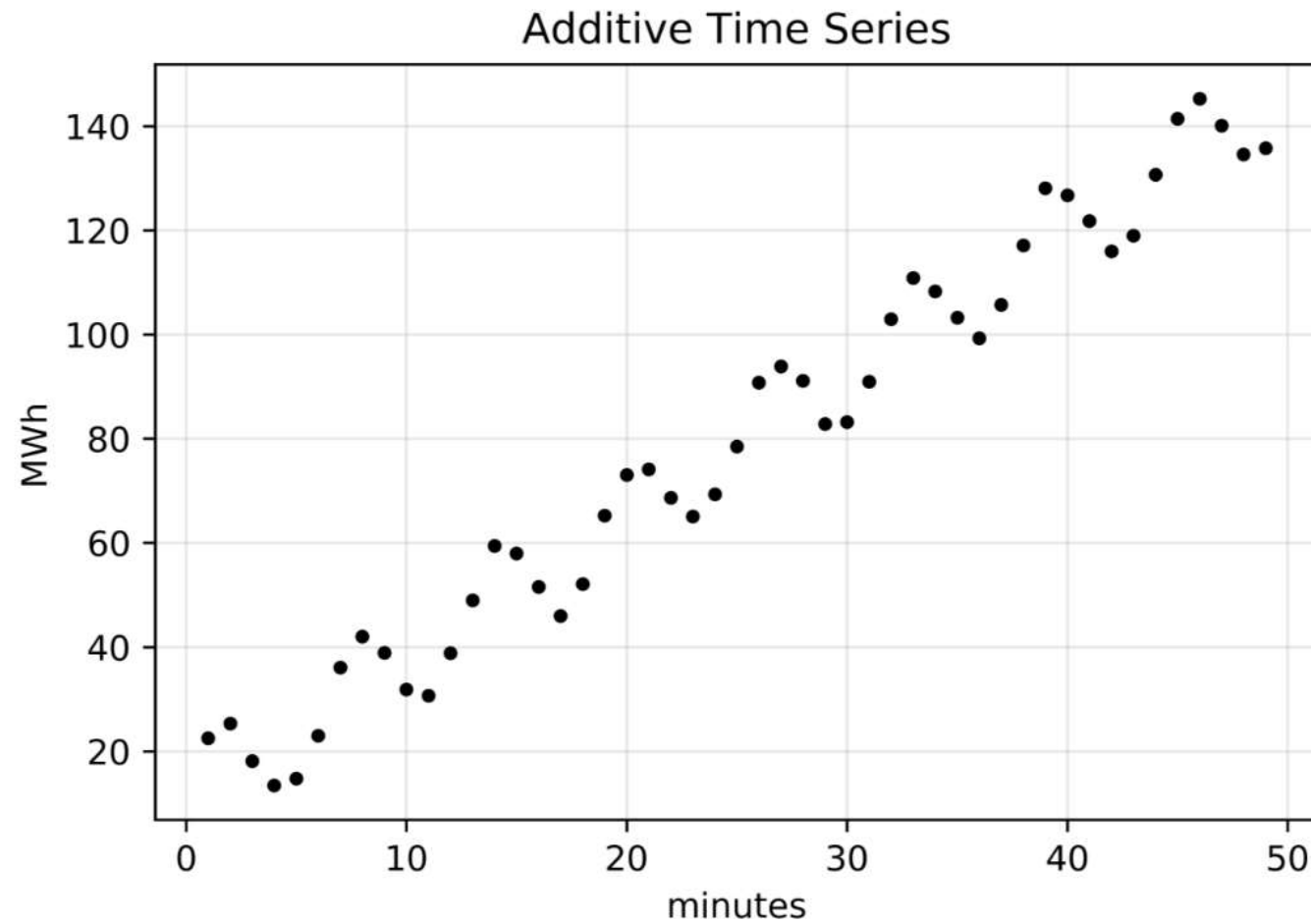
生成噪声残量

```
np.random.seed(10) # reproducible results  
residual = np.random.normal(loc=0.0, scale=1, size=len(time)) # 均值=0, 标准差=1, list-size
```

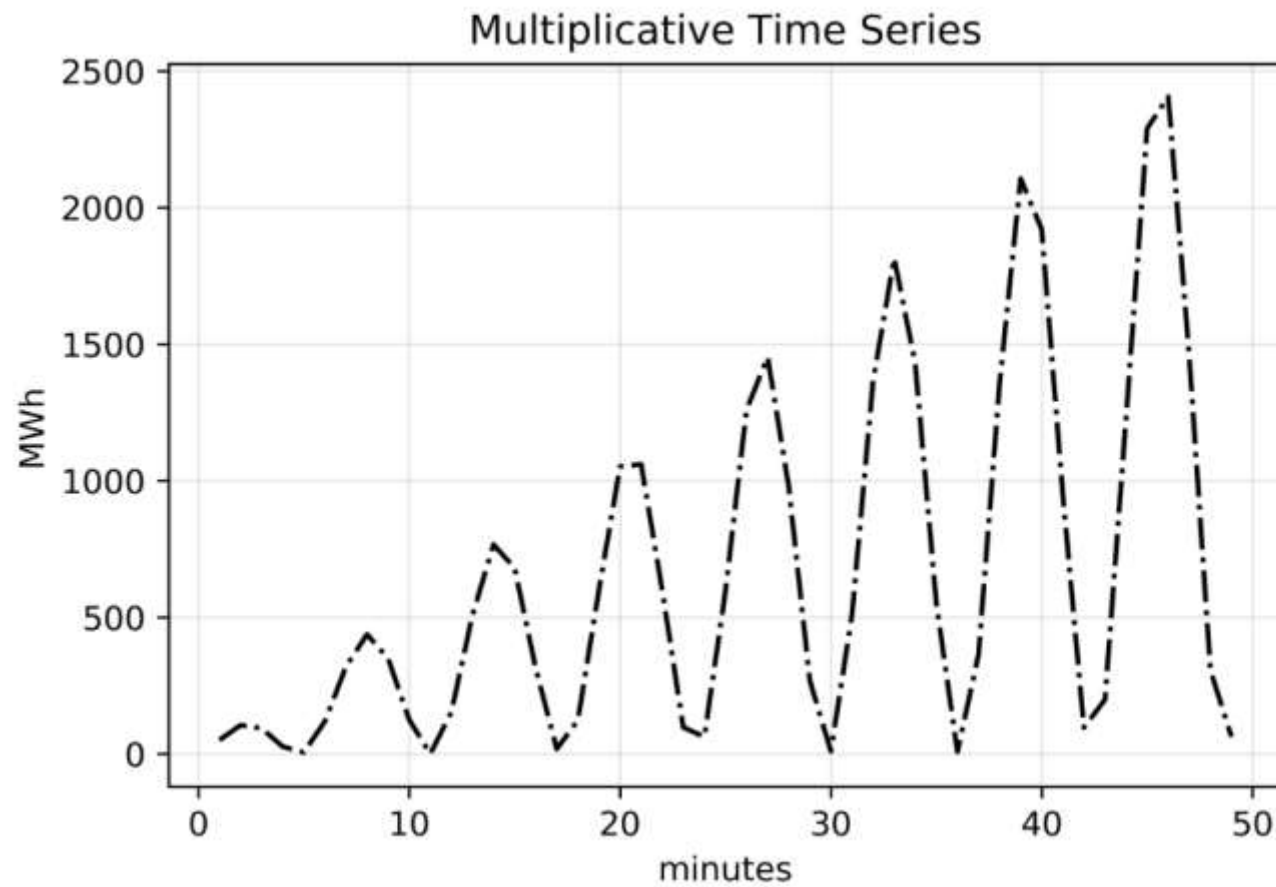
```
plt.plot(time, residual, 'r-.')  
plt.title("Residuals vs Time")  
plt.xlabel("minutes")  
plt.ylabel("electricity demand");
```



信号的加法模型



乘法模型



成分叠加:

Additive Time Series

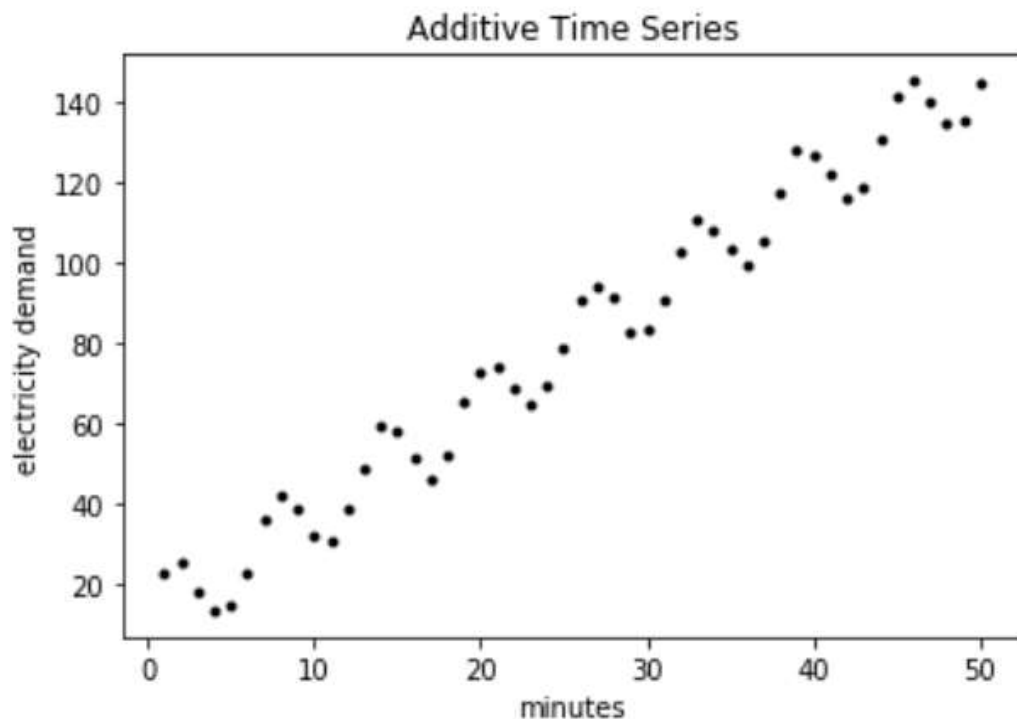
Remember the equation for additive time series is simply: $O_t = T_t + S_t + R_t$

O_t is the output; T_t is the trend S_t is the seasonality

R_t is the residual t is a variable representing a particular point in time

```
1 additive = trend + seasonal + residual
```

```
1 plt.plot(time, additive, 'k.')
2 plt.title("Additive Time Series")
3 plt.xlabel("minutes")
4 plt.ylabel("electricity demand");
```



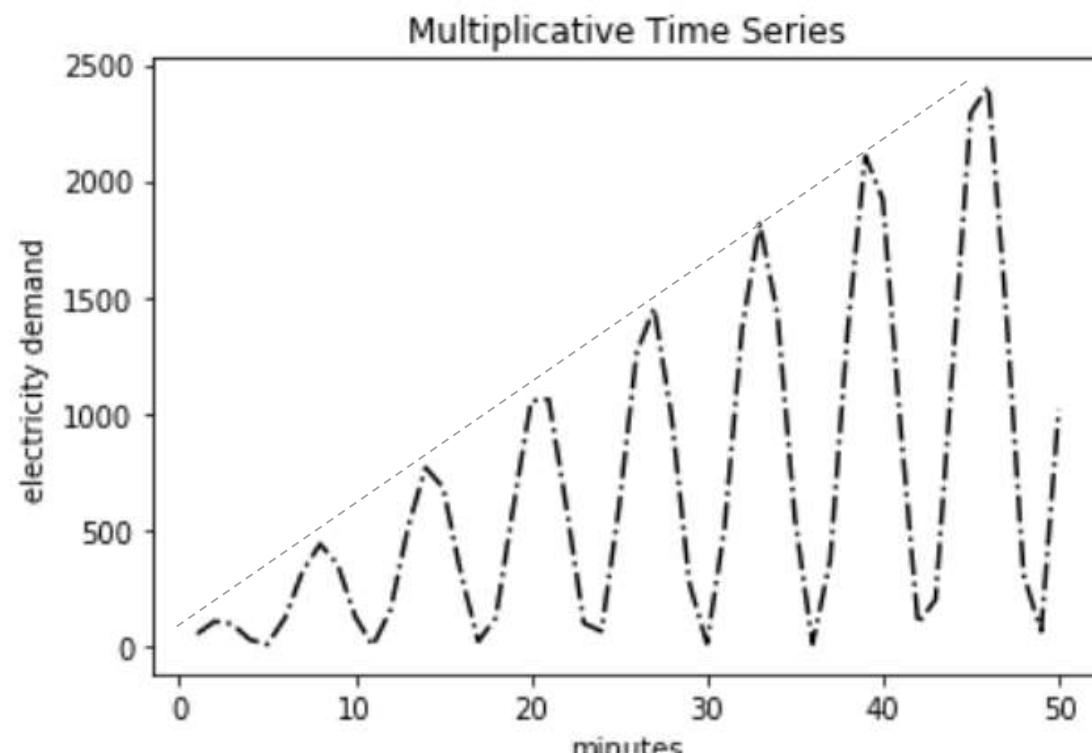
成分相乘：

Multiplicative Time Series

Remember the equation for multiplicative time series is simply: $O_t = T_t * S_t * R_t$

```
1 # ignoring residual to make pattern more apparent
2 ignored_residual = np.ones_like(residual)
3 multiplicative = trend * seasonal * ignored_residual
```

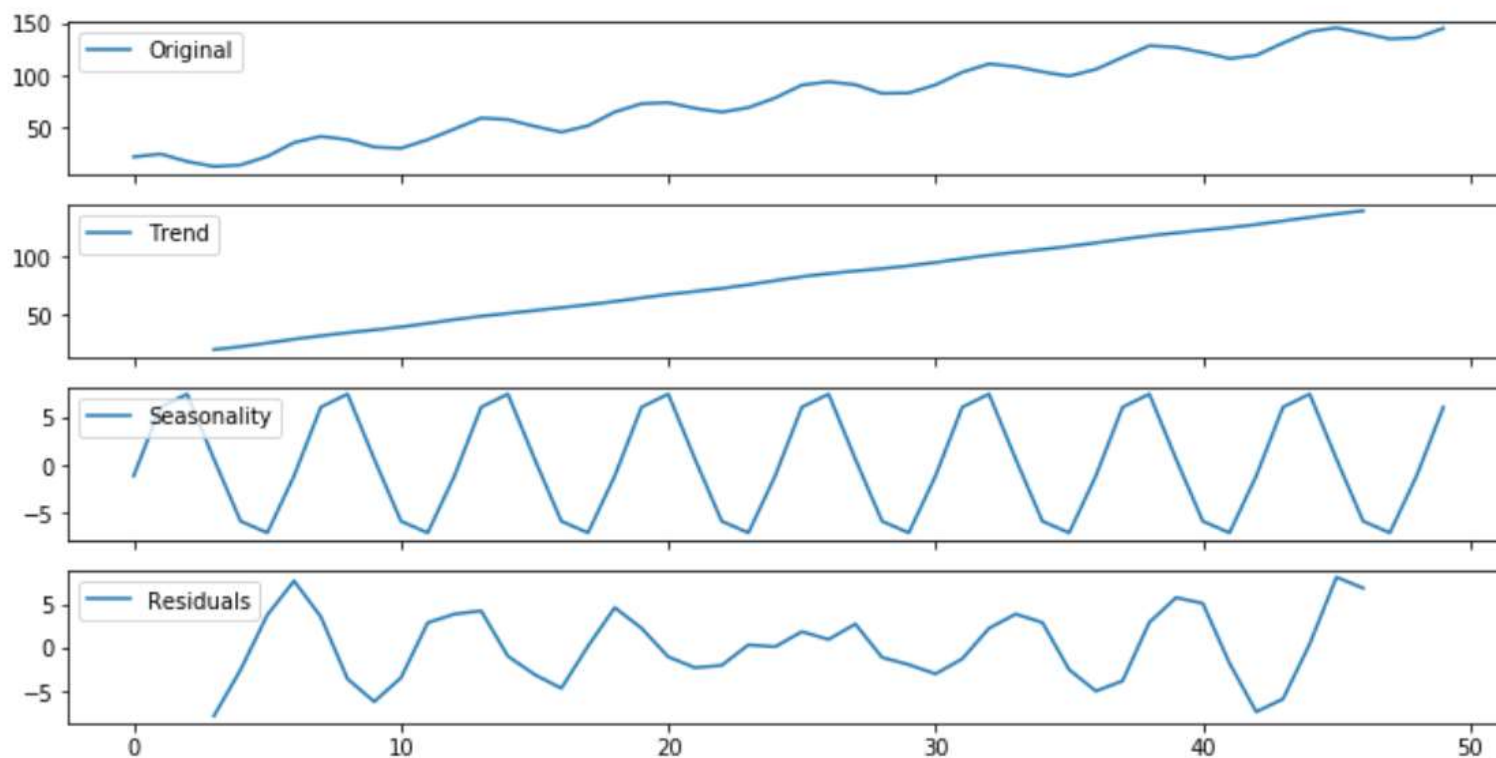
```
1 plt.plot(time, multiplicative, 'k-.')
2 plt.title("Multiplicative Time Series")
3 plt.xlabel("minutes")
4 plt.ylabel("electricity demand");
```



Additive Decomposition

加和 (additive) 成分分解

```
1 from statsmodels.tsa.seasonal import seasonal_decompose
2
3 ss_decomposition = seasonal_decompose(x=additive, model='additive', freq=6)
4 #####
5 estimated_trend = ss_decomposition.trend
6 estimated_seasonal = ss_decomposition.seasonal
7 estimated_residual = ss_decomposition.resid
```



信号数据平滑与滤波（去除噪声）

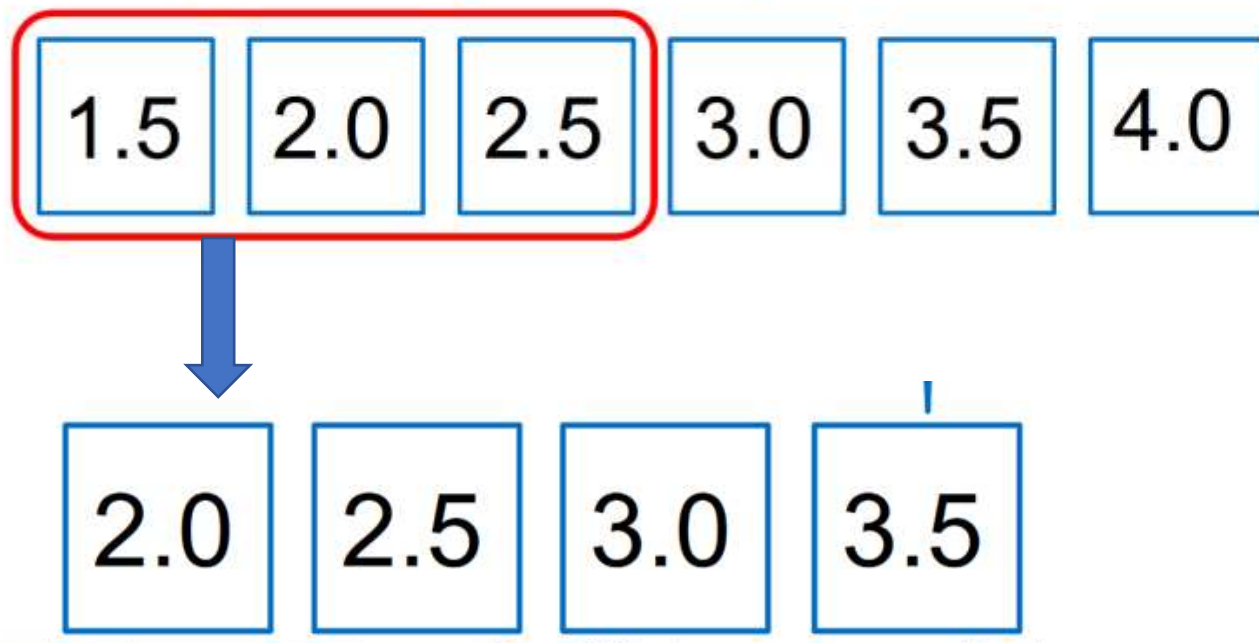
- moving average（滑窗滤波）
 - 等权均值滤波 Equally weighted
 - 距离加权滤波（指数） Exponentially weighted
 - 卡尔曼滤波（待续）



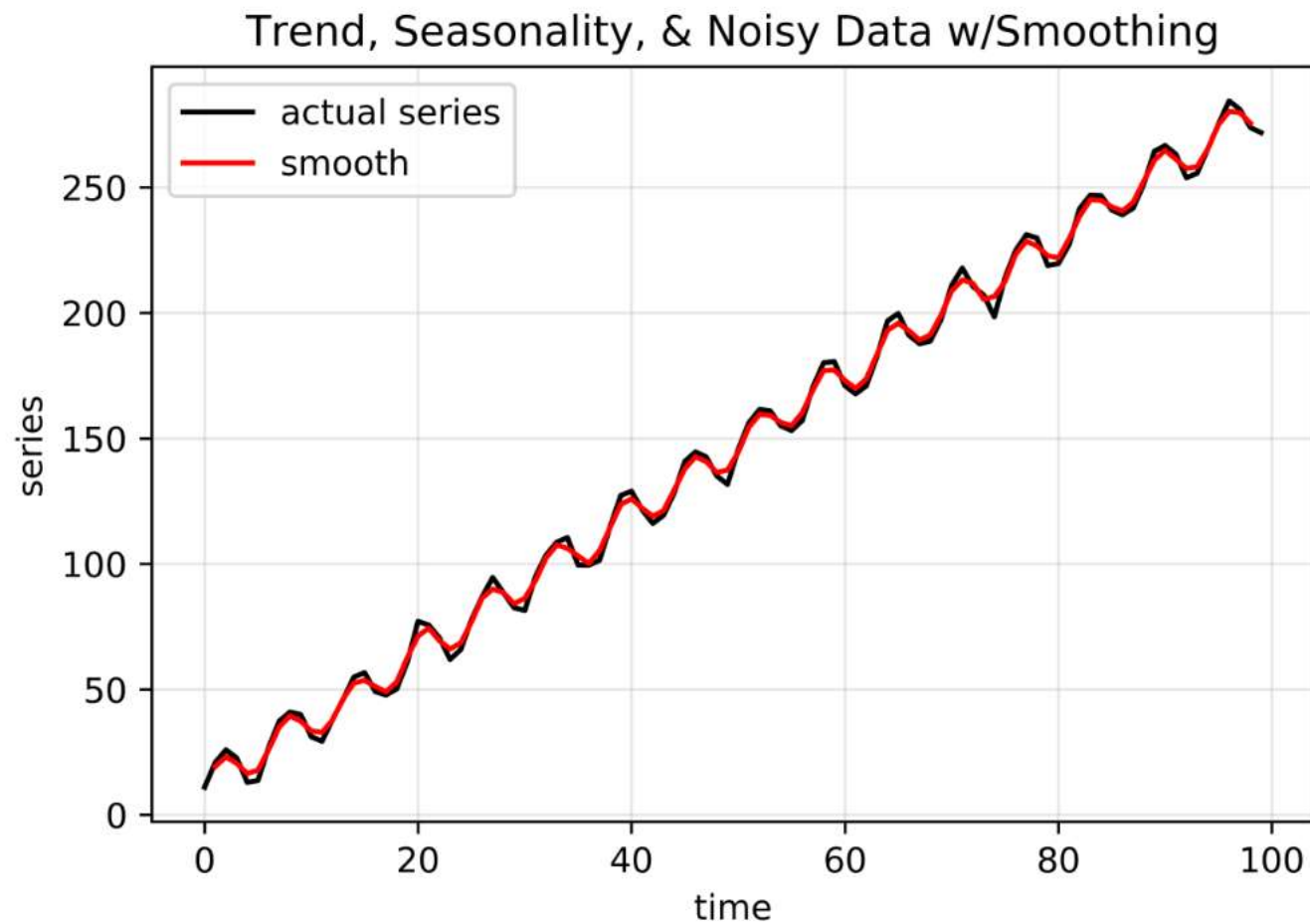
均值滑窗：

Equally Weighted Moving Average – Example

The first step in calculating moving average is to select a window size (we'll use 3).



均值滑窗滤波效果：



距离加权（指数）滑窗滤波： —— 相对均值滑窗能更多的保留趋势信息

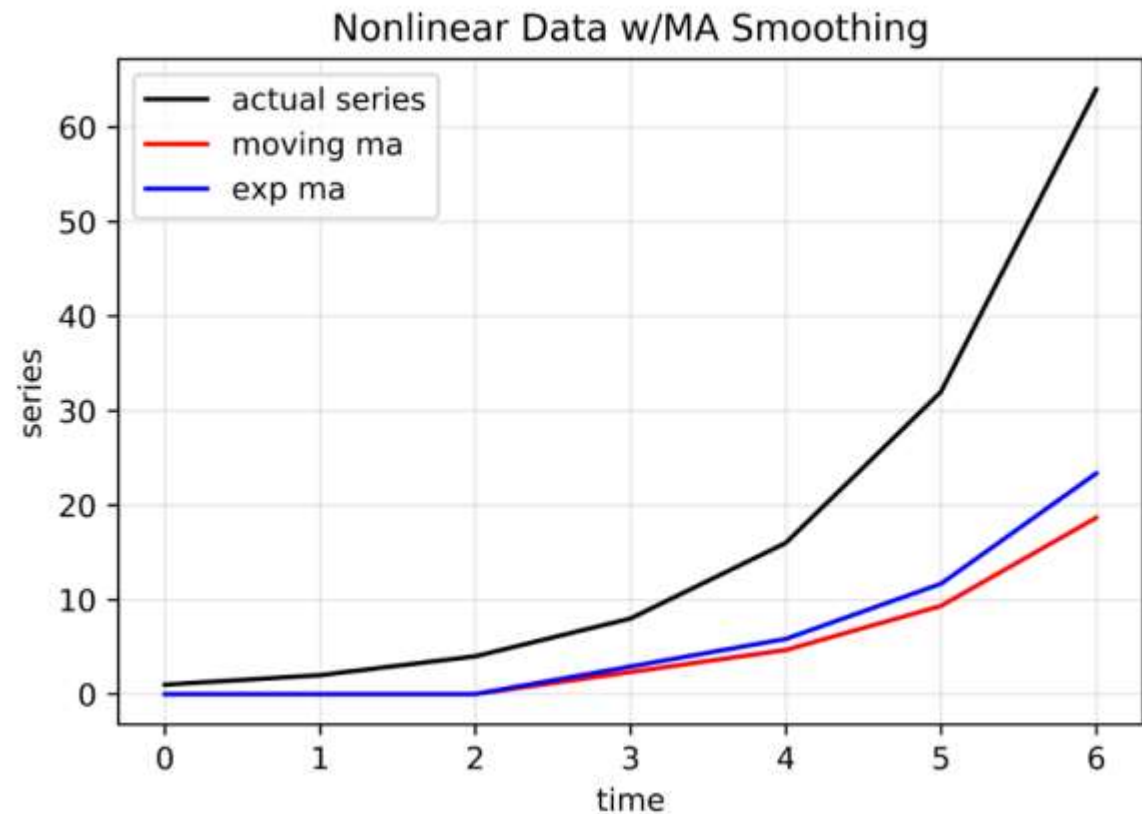
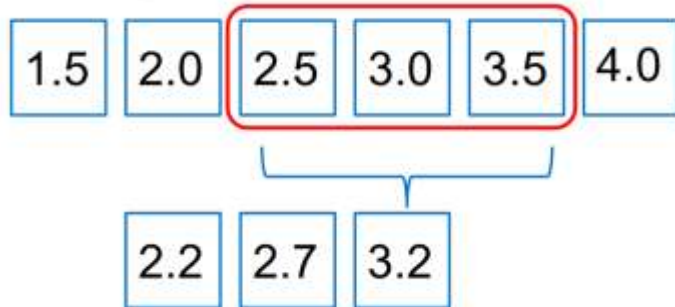
•

$$w + w^2 + w^3 = 1$$

$$w = w_{t-1} \sim 0.543$$

$$w^2 = w_{t-2} \sim 0.294$$

$$w^3 = w_{t-3} \sim 0.160$$



平稳序列 (stationarity Time Series)

- 平稳序列概念
- 平稳序列判定方法
- 序列平稳化



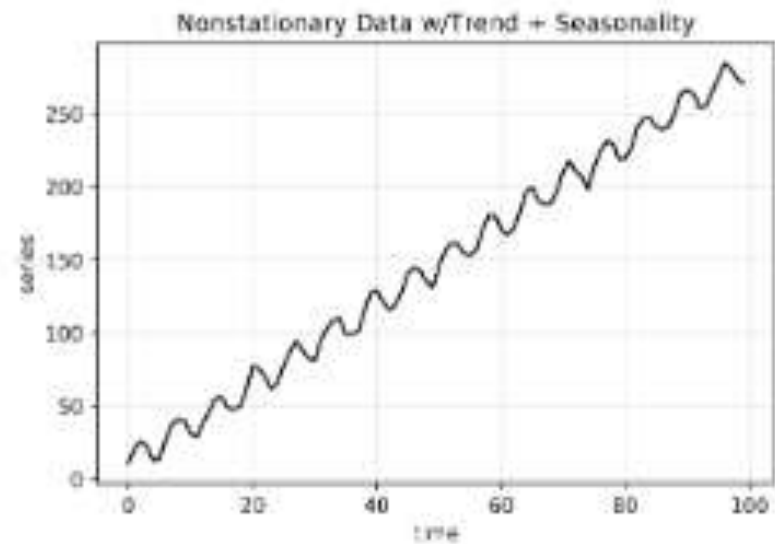
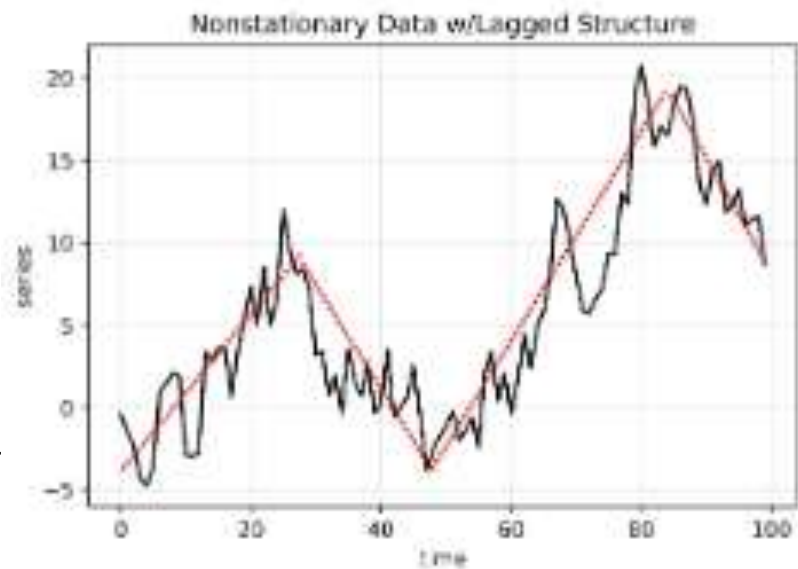
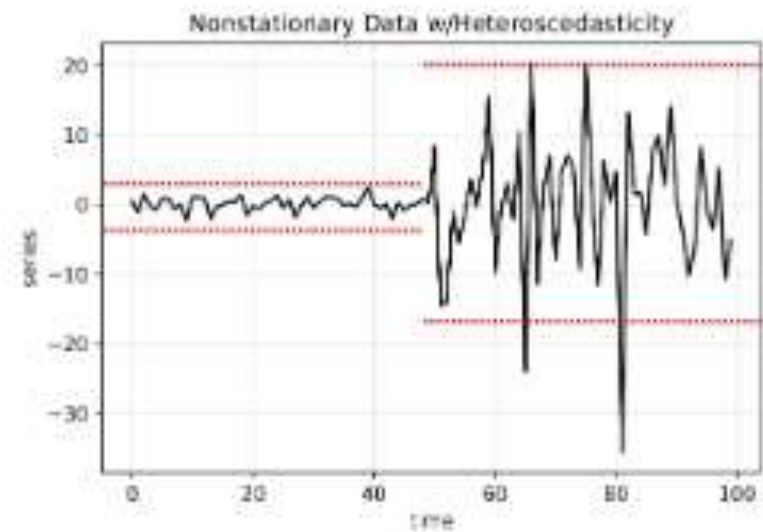
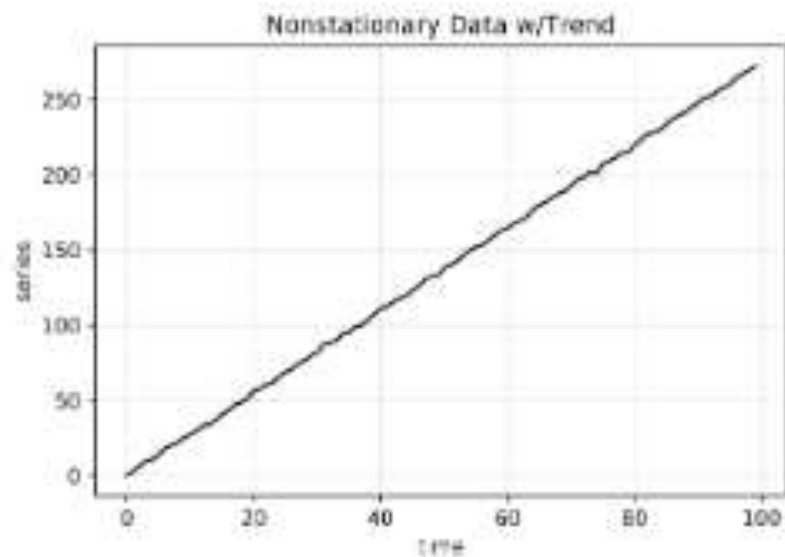
平稳 (Stationarity) 序列的概念

A stationary time series is a time series where there are **no changes** in the underlying system: 信号系统物理特征无变化

- Constant mean (no trend 没有趋势性变化)
- Constant variance (no heteroscedasticity) 自方差稳定
- Constant autocorrelation structure (自协方差稳定)
- No periodic component (no seasonality 没有周期波动)

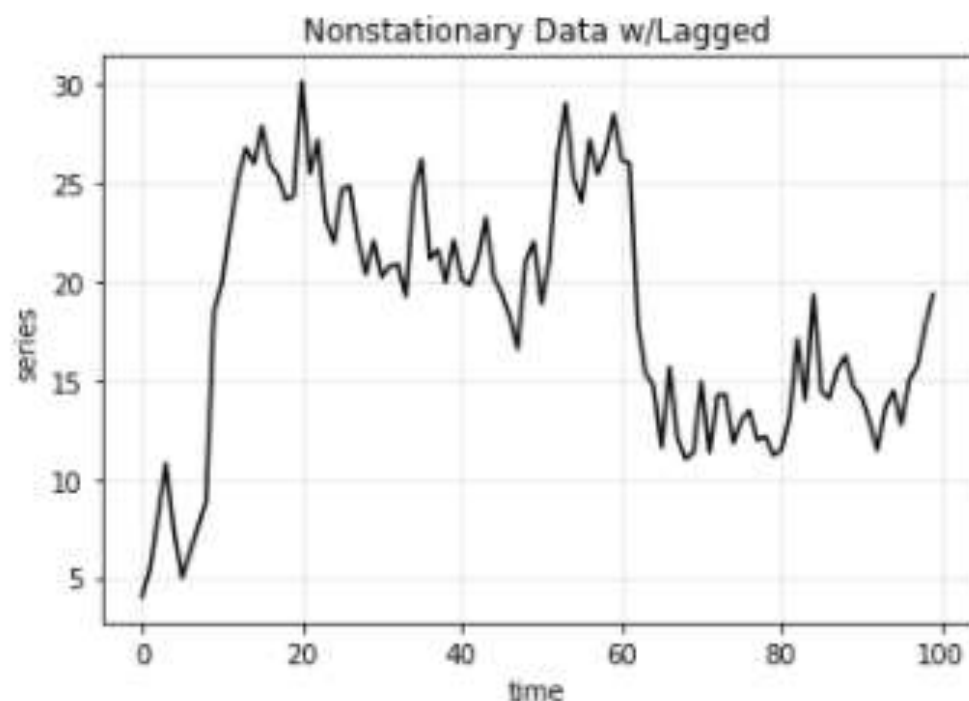


几个非平稳的例子：



自相关结构 Autocorrelation Structure

```
: seed = 3.14 # seed to start series 类比随机序列发生器的种子  
# create autocorrelated data  
lagged = np.empty_like(time, dtype='float') # 随机初始化一个100的序列  
  
for t in time: # 按照随机量做步长叠加一个随机信号  
    lagged[t] = seed + np.random.normal(loc=0, scale=2.5, size=1) # 加入带延迟lag的随机量  
    seed = lagged[t]  
  
: run_sequence_plot(time, lagged,  
                    title="Nonstationary Data w/Lagged")
```



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。



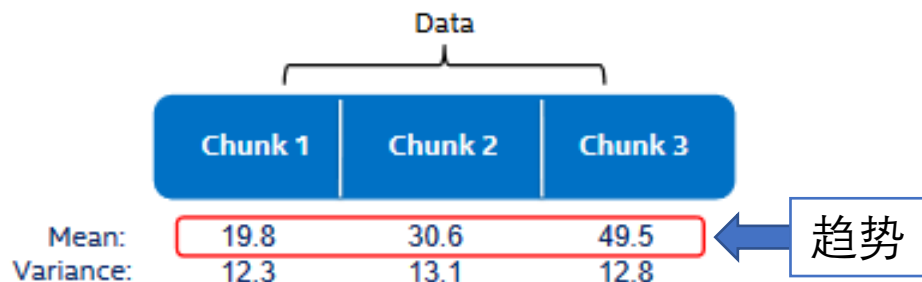
平稳性检验

- 滑窗（观测）检验
- ADF检验



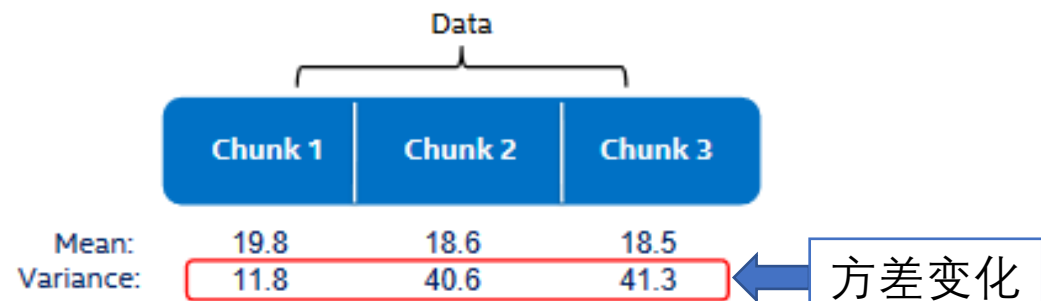
平稳性滑窗观测:

Nonstationary



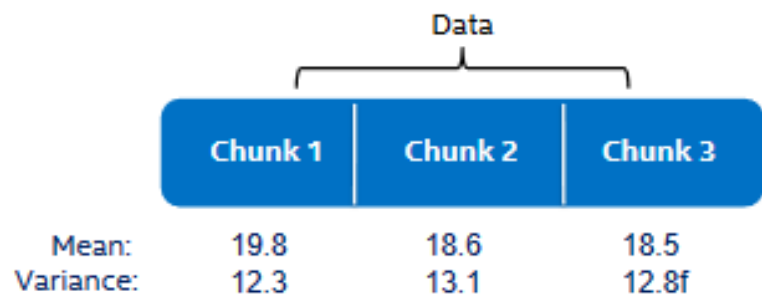
There are large deviations in the mean between chunks.

Nonstationary

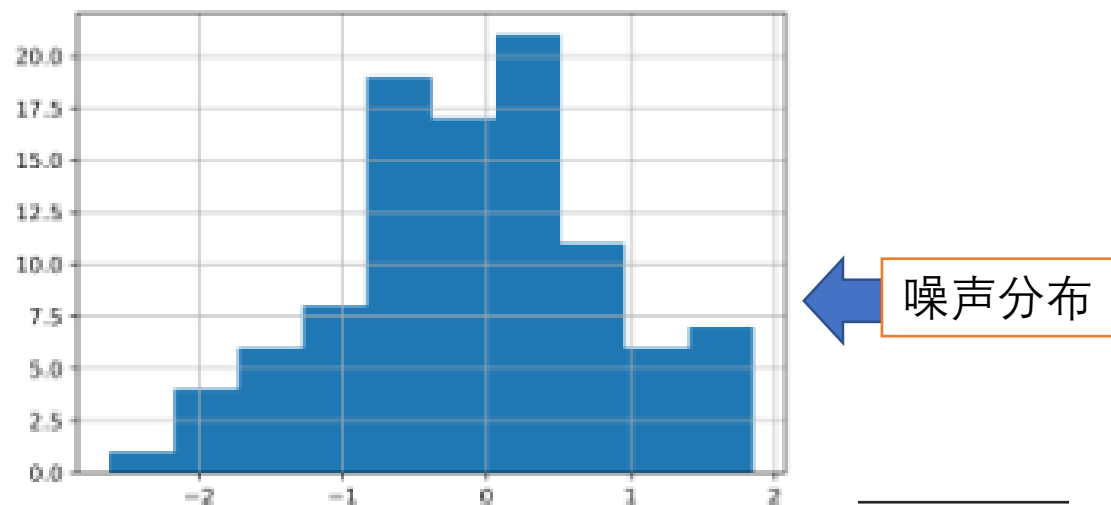


There are large deviations in the variance between chunks.

Stationary



The chunks have similar mean and variance.




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



例子：通过差分运算消除趋势

#对于这个序列 简单的差分就可以

```
def toStationary(data, alpha):
```

```
    p = 0
```

```
    while(True):
```

```
        data = np.diff(data) ←
```

```
        p = p + 1
```

```
        result = adfuller(data)
```

```
        if result[1] < alpha:
```

```
            break
```

```
    return data, p
```

```
dataZ, p = toStationary(data, 0.05)
```

```
adfuller(dataZ)
```

默认有一个常数项阈值

```
(-10.599917193507997, ←
```

```
6.209303164631373e-19,
```

```
26,
```

```
2991,
```

```
['10', 2.4225282040645257
```



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)
```

```
statsmodels.tsa.stattools.adfuller(x, maxlag=None, regression='c', autolag='AIC',
store=False, regresults=False)[source]
```

Parameters

x : array_like, 1d

The data series to test.

maxlag : int

Maximum lag which is included in test, default $12 \cdot (\text{nobs}/100)^{1/4}$.

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.

autolag : {"AIC", "BIC", "t-stat", None}

Method to use when automatically determining the lag length among the values

- If "AIC" (default) or "BIC", then the number of lags is chosen to minimize the
- "t-stat" based choice of maxlag. Starts with maxlag and drops a lag until the significant using a 5%-sized test.
- If None, then the number of included lags is set to maxlag.

store : bool

If True, then a result instance is returned additionally to the adf statistic. Default is

regresults : bool, optional

If True, the full regression results are returned. Default is False.

Returns

adf : float

The test statistic.

pvalue : float

Mackinnon's approximate p-value based on Mackinnon (1994, 2010).

usedlag : int

The number of lags used.

nobs : int

The number of observations used for the ADF regression and calculation of the critical values.

critical values : dict

Critical values for the test statistic at the 1 %, 5 %, and 10 % levels. Based on Mackinnon (2010).

icbest : float

The maximized information criterion if autolag is not None.



resstore : ResultStore, optional

时序自相关性与Arma模型

- MA (moving Aveage) 模型
- ACF (Autocorrelation Function) 基础
- AR (Autoregressive) 模型

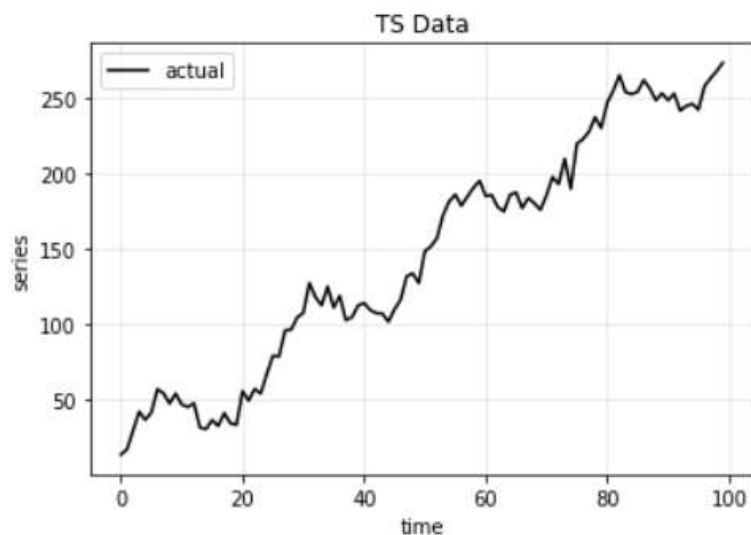


移动窗口平滑模型： Moving Average Smoothing

$$\hat{y}_{t+1} = \frac{y_t + y_{t-1} + \dots + y_{t-m+1}}{m}$$

```
# data 有噪声有趋势有偏置的sin函数
noise = np.random.normal(loc=0, scale=6.5, size=len(time))
trend = time * 2.75
seasonality = 10 + np.sin(time * 0.25) * 20
data = trend + seasonality + noise
```

```
: run_sequence_plot(time, data, title="TS Data")
```

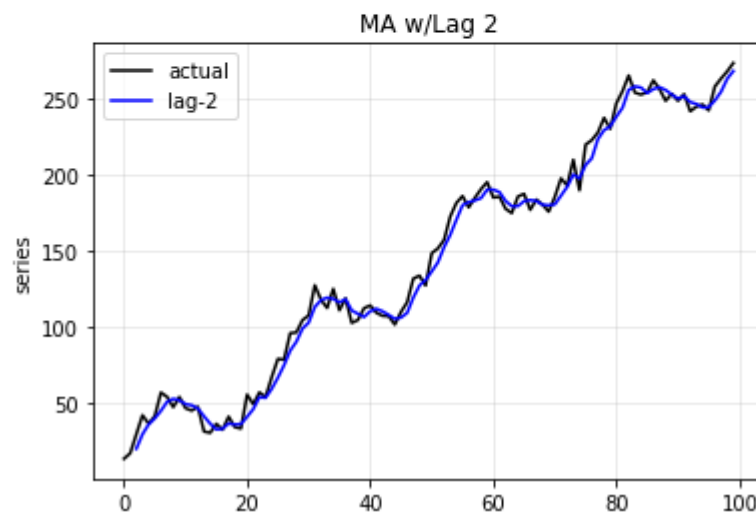


```
: series = pd.Series(data)

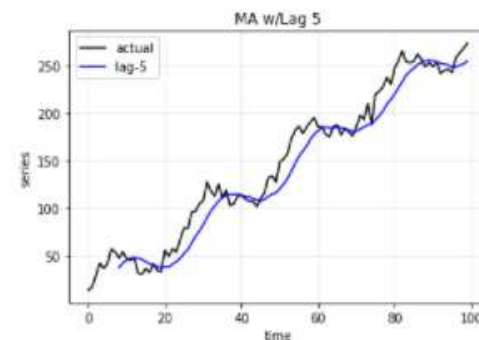
lag_2 = series.rolling(window=3).mean()

lag_5 = series.rolling(window=9).mean()
```

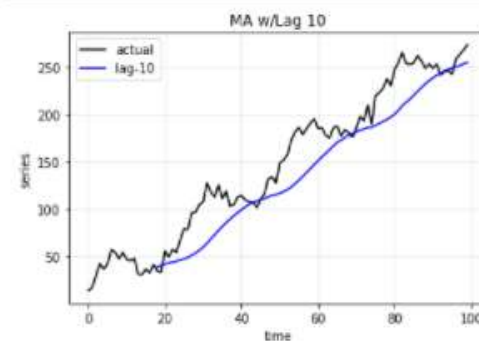
```
: run_sequence_plot(time, data, title="MA w/Lag 2")
plt.plot(time, lag_2, 'b-', label="lag-2")
plt.legend():
```



```
: run_sequence_plot(time, data, title="MA w/Lag 5")
plt.plot(time, lag_5, 'b-', label="lag-5")
plt.legend():
```



```
: run_sequence_plot(time, data, title="MA w/Lag 10")
plt.plot(time, lag_10, 'b-', label="lag-10")
plt.legend():
```



序列自（相关）回归模型：

两个随机变量 X 和 Y 的相关系数定义为

$$\rho(X, Y) = \rho_{xy} = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}} = \frac{E[(X - \mu_x)(Y - \mu_y)]}{\sqrt{E(X - \mu_x)^2 E(Y - \mu_y)^2}}$$

设 $\{X_t\}$ 为弱平稳序列， $\{\gamma_k\}$ 为自协方差函数。则

$$\rho(X_{t-k}, X_t) = \frac{\text{Cov}(X_{t-k}, X_t)}{\sqrt{\text{Var}(X_{t-k})\text{Var}(X_t)}} = \frac{\gamma_k}{\sqrt{\gamma_0\gamma_0}} = \frac{\gamma_k}{\gamma_0}, \quad k = 0, 1, \dots, \forall t$$

记 $\rho_k = \gamma_k/\gamma_0$ ，这是 X_{t-k} 与 X_t 的相关系数且与 t 无关，称 $\{\rho_k, k = 0, 1, \dots\}$ 为时间序列 $\{X_t\}$ 的自相关函数（Autocorrelation function, ACF）。 $\rho_0 = 1$ 。

如果 $\rho_1 \neq 0$ ，则 X_t 与 X_{t-1} 相关，可以用 X_{t-1} 预测 X_t 。最简单的预测为线性组合，如下模型：

$$X_t = \phi_0 + \phi_1 X_{t-1} + \varepsilon_t \quad \leftarrow \boxed{\text{AR(1)模型}} \quad (4.1)$$

称为一阶**自回归**模型(Autoregression model)，记作AR(1)模型。其中 $\{\varepsilon_t\}$ 是零均值独立同分布白噪声序列，方差为 σ^2 ，并设 ε_t 与 X_{t-1}, X_{t-2}, \dots 独立。系数 $|\phi_1| < 1$ 。更一般的定义中仅要求 $\{\varepsilon_t\}$ 是零均值白噪声，不要求独立同分布。

AR (p) 模型: $X_t = \phi_0 + \phi_1 X_{t-1} + \dots + \phi_p X_{t-p} + \varepsilon_t, \phi_p \neq 0$

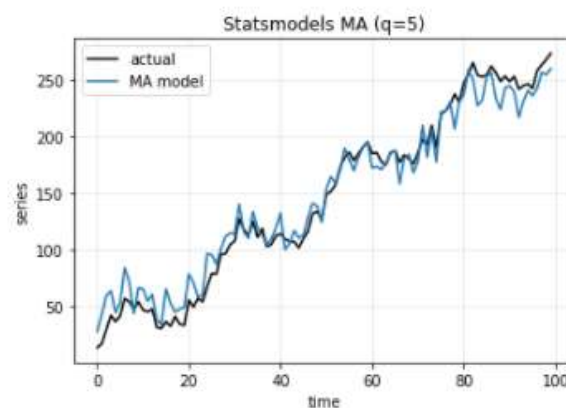
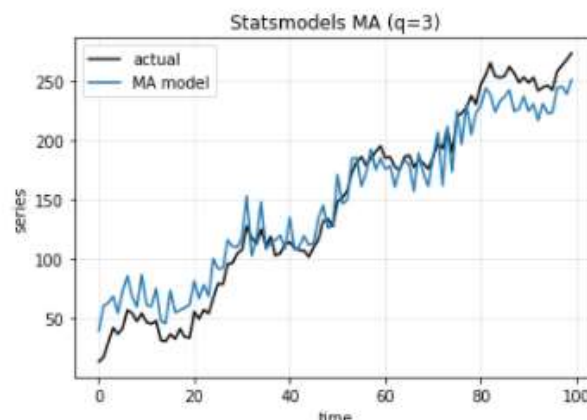
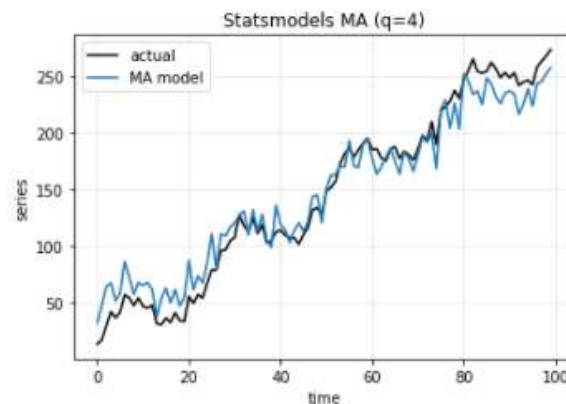
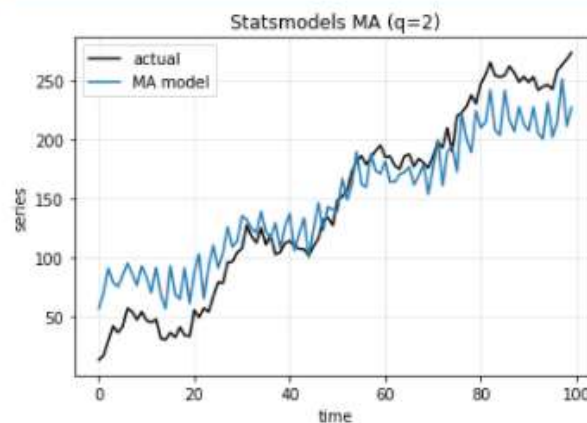
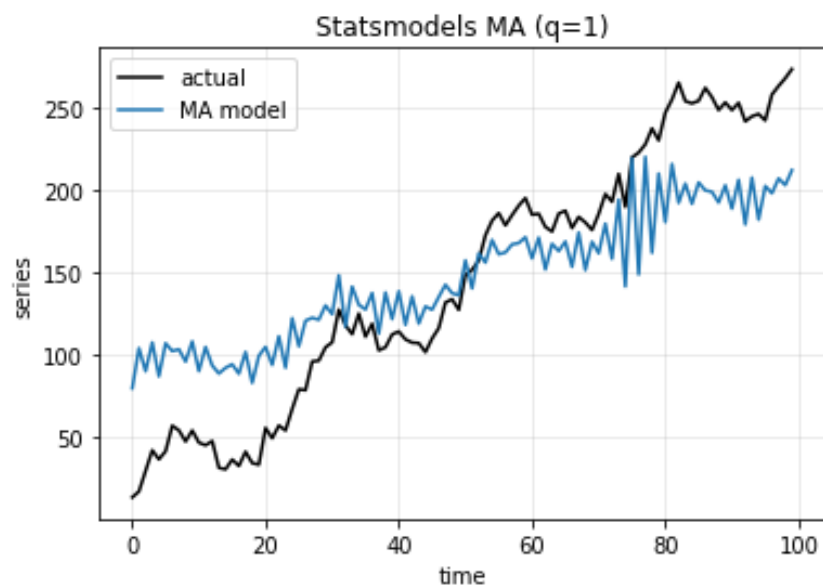
ARMA模型:

$$X_t = \phi_0 + \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + \varepsilon_t, \phi_p \neq 0$$

$$y_t = c + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \cdots + \theta_q e_{t-q}$$

```
from statsmodels.tsa.arima_model import ARMA
```

```
# plot different orders (q's)
for i in range(1,6):      # 采用不同阶的ARMA模型进行回归
    model = ARMA(data, (0,i)).fit()
    run_sequence_plot(time, data, title="Statsmodels MA (q={})".format(i))
    plt.plot(time, model.predict(start=1, end=100), label="MA model")
    plt.legend()
    plt.show();
```



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

#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

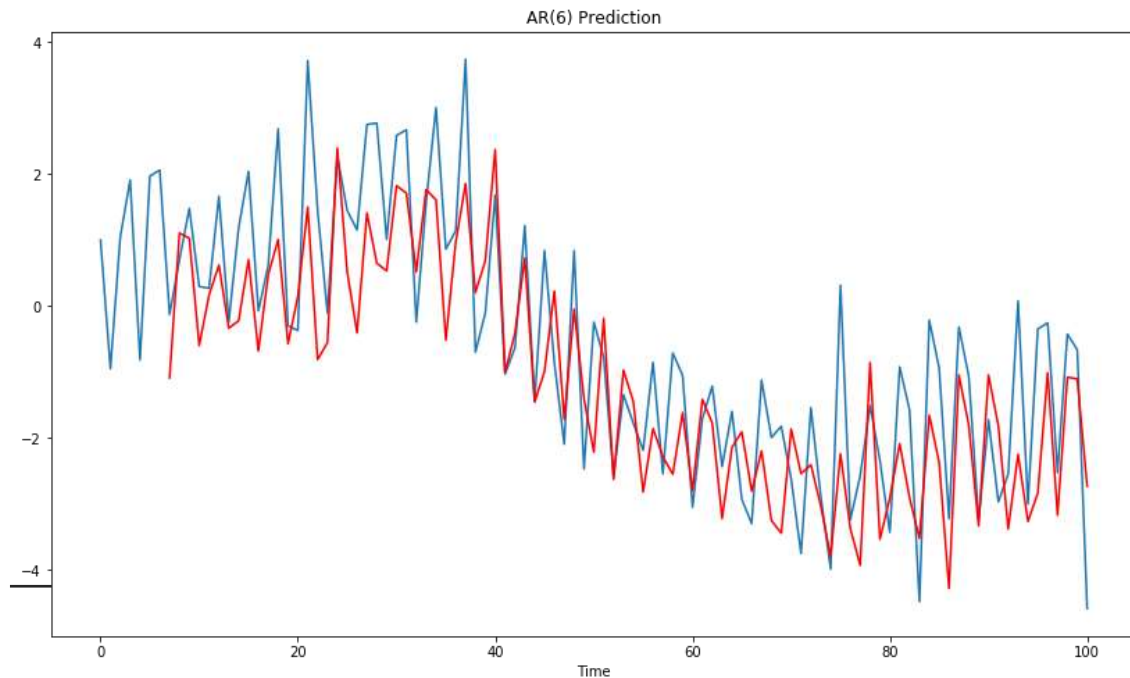
model_fit = ARMA(X, order=(1,0)).fit()

#获取回归方程的系数
coefs = model_fit.params

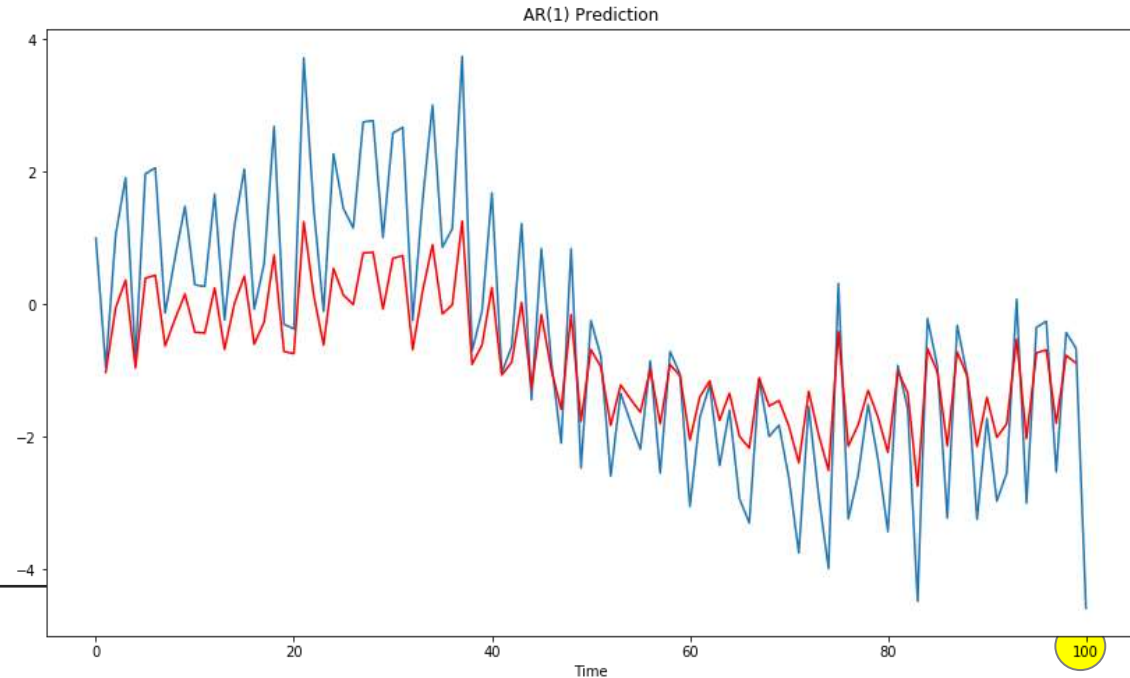
#  $y[t] = coefs[0] + coefs[1]x[t - 1] + \dots$ 

```

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

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

