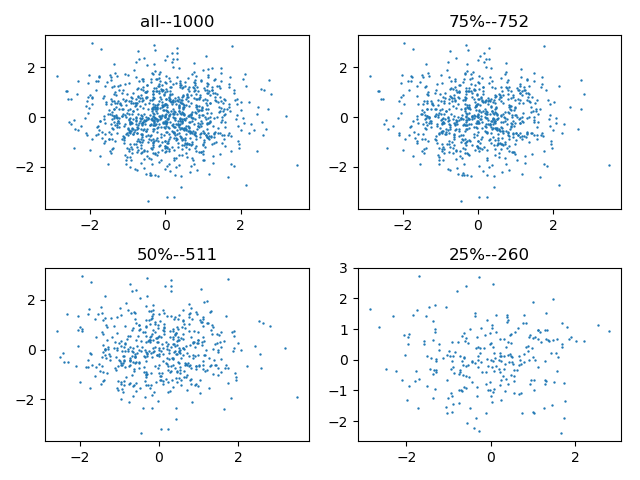
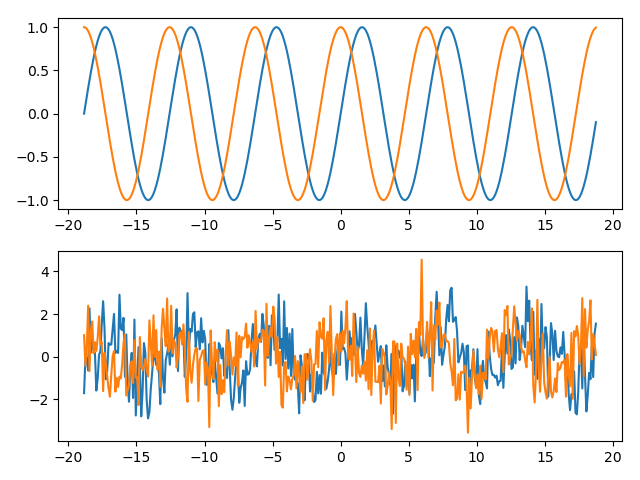
## 1 随机抽样

import matplotlib.pyplot as plt  
import random as rd  
  
x = [rd.gauss(0, 1) for i in range(1000)]  
y = [rd.gauss(0, 1) for i in range(1000)]  
  
x1 = []  
y1 = []  
for j in range(1000):  
 if rd.random() >= 0.25:  
 x1.append(x[j])  
 y1.append(y[j])  
  
x2 = []  
y2 = []  
for j in range(1000):  
 if rd.random() >= 0.5:  
 x2.append(x[j])  
 y2.append(y[j])  
  
x3 = []  
y3 = []  
for j in range(1000):  
 if rd.random() >= 0.75:  
 x3.append(x[j])  
 y3.append(y[j])  
  
plt.subplot(221)  
plt.title("all--" + str(1000))  
plt.scatter(x, y, s=[0.5])  
plt.subplot(222)  
plt.title("75%--" + str(len(x1)))  
plt.scatter(x1, y1, s=[0.5])  
plt.subplot(223)  
plt.title("50%--" + str(len(x2)))  
plt.scatter(x2, y2, s=[0.5])  
plt.subplot(224)  
plt.title("25%--" + str(len(x3)))  
plt.scatter(x3, y3, s=[0.5])  
  
plt.show()



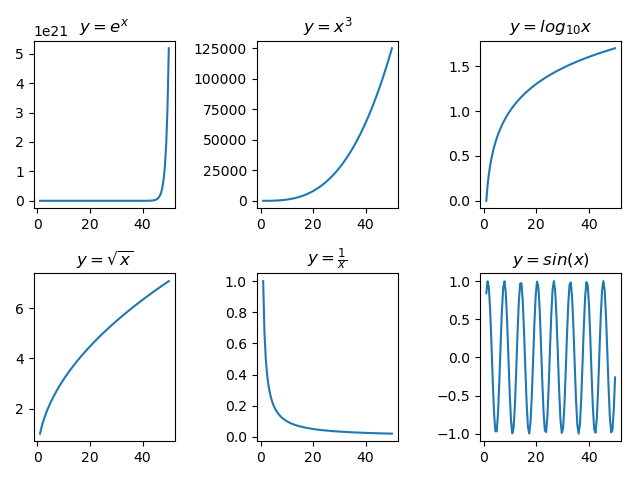
## 2 噪声影响

from math import \*  
import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.arange(-6 \* pi, 6 \* pi, 0.1)  
y = np.sin(x)  
z = np.cos(x)  
  
y1 = y + np.random.randn(len(y))  
z1 = z + np.random.randn(len(z))  
  
plt.subplot(211)  
plt.plot(x, y)  
plt.plot(x, z)  
plt.subplot(212)  
plt.plot(x, y1)  
plt.plot(x, z1)  
plt.show()



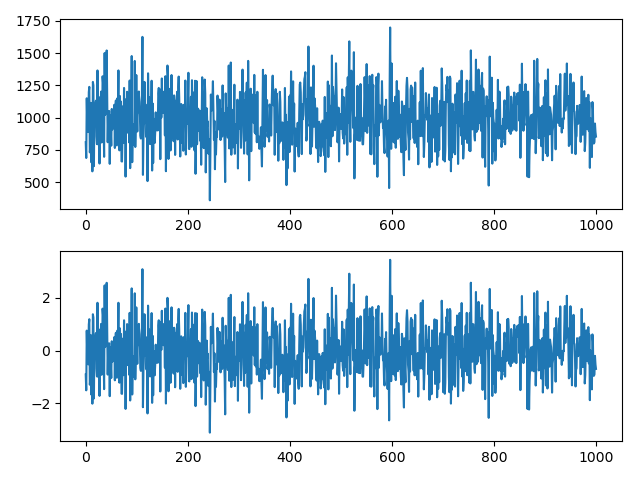
## 3 变量变换

import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.linspace(1, 50, 100)  
y1 = np.exp(x)  
y2 = x \*\* 3  
y3 = np.log10(x)  
y4 = np.sqrt(x)  
y5 = 1 / x  
y6 = np.sin(x)  
  
plt.subplot(231)  
plt.title(r"$y=e^x$")  
plt.plot(x, y1)  
plt.subplot(232)  
plt.title(r"$y=x^3$")  
plt.plot(x, y2)  
plt.subplot(233)  
plt.title(r"$y=log\_{10}x$")  
plt.plot(x, y3)  
plt.subplot(234)  
plt.title(r"$y=\sqrt{x}$")  
plt.plot(x, y4)  
plt.subplot(235)  
plt.title(r"$y=\frac{1}{x}$")  
plt.plot(x, y5)  
plt.subplot(236)  
plt.title(r"$y=sin(x)$")  
plt.plot(x, y6)  
  
plt.show()



## 4 标准化或规范化

from math import \*  
import matplotlib.pyplot as plt  
import numpy as np  
  
y1 = np.random.normal(1000, 200, 1000)  
  
# y2 = (y1 - np.average(y1)) / np.std(y1)  
# y2 = (y1 - np.average(y1)) / (np.std(y1) \* len(y1) / (len(y1) - 1))  
sd = sqrt(sum((y1 - np.average(y1))\*\*2) / (len(y1) - 1))  
y2 = (y1 - np.average(y1)) / sd  
  
print np.mean(y2), np.std(y2)  
  
plt.subplot(211)  
plt.plot(y1)  
plt.subplot(212)  
plt.plot(y2)  
plt.show()



## 5 距离计算

from math import \*  
import numpy as np  
  
  
# 计算街区距离  
def i\_city\_dist(x, y):  
 return sum([abs(x[i] - y[i]) for i in range(len(x))])  
  
  
# 计算欧几里得距离  
def i\_euclid\_dist(x, y):  
 return sqrt(sum([(x[i] - y[i]) \*\* 2 for i in range(len(x))]))  
  
  
# 计算max距离  
def i\_max\_dist(x, y):  
 return max([abs(x[i] - y[i]) for i in range(len(x))])  
  
  
# 计算矩阵行向量之间的距离  
def i\_matrix\_row\_dist(X, dist):  
 rc = np.size(X, axis=0)  
 rel = np.linspace(0.0, 0.0, rc\*rc).reshape(rc, rc)  
 for i in range(rc):  
 for j in range(i, rc):  
 rel[j, i] = rel[i, j] = dist(X[i], X[j])  
 return rel  
  
  
X = np.array([[0, 2], [2, 0], [3, 1], [5, 1]])  
print(X)  
print("City Block Distance:")  
print(i\_matrix\_row\_dist(X, i\_city\_dist))  
print("Euclid Distance:")  
print(i\_matrix\_row\_dist(X, i\_euclid\_dist))  
print("Distance with Element Max:")  
print(i\_matrix\_row\_dist(X, i\_max\_dist))

D:\2020下课程\数据挖掘\数据挖掘实验程序\k-means\venv\Scripts\python.exe C:/Users/sdeve/PycharmProjects/pythonProject/main.py

[[0 2]

[2 0]

[3 1]

[5 1]]

City Block Distance:

[[0. 4. 4. 6.]

[4. 0. 2. 4.]

[4. 2. 0. 2.]

[6. 4. 2. 0.]]

Euclid Distance:

[[0. 2.82842712 3.16227766 5.09901951]

[2.82842712 0. 1.41421356 3.16227766]

[3.16227766 1.41421356 0. 2. ]

[5.09901951 3.16227766 2. 0. ]]

Distance with Element Max:

[[0. 2. 3. 5.]

[2. 0. 1. 3.]

[3. 1. 0. 2.]

[5. 3. 2. 0.]]

Process finished with exit code 0

## 6 二元变量相似度

import numpy as np  
import random as rd  
  
  
def smc(x, y):  
 f = np.linspace(0, 0, 4, dtype="int32").reshape(2, 2)  
 for i in range(len(x)):  
 f[x[i], y[i]] += 1  
 print(f)  
 return float(f[1, 1] + f[0, 0]) / (f[0, 0] + f[0, 1] + f[1, 0] + f[1, 1])  
  
  
def jc(x, y):  
 f = np.linspace(0, 0, 4, dtype="int32").reshape(2, 2)  
 for i in range(len(x)):  
 f[x[i], y[i]] += 1  
 print(f)  
 return float(f[1, 1]) / (f[0, 1] + f[1, 0] + f[1, 1])  
  
  
a = np.linspace(0, 0, 100, dtype='int32')  
b = a.copy()  
for i in range(100):  
 if rd.random() > 0.5:  
 a[i] = 1  
for i in range(100):  
 if rd.random() > 0.5:  
 b[i] = 1  
print(a)  
print()  
print(b)  
print()  
print(smc(a, b))  
print()  
print(jc(a, b))

D:\2020下课程\数据挖掘\数据挖掘实验程序\k-means\venv\Scripts\python.exe C:/Users/sdeve/PycharmProjects/pythonProject/main.py

[0 1 0 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 0 0

1 1 0 1 1 1 0 0 1 0 1 0 1 0 0 1 1 0 1 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0

0 1 1 0 1 0 0 0 1 1 0 0 1 0 1 1 1 1 1 0 1 1 0 0 1 0]

[0 1 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 1 0 1 1

1 1 0 0 0 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 0 0

0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 1 1 1]

[[18 29]

[25 28]]

0.46

[[18 29]

[25 28]]

0.34146341463414637

## 7 余弦相似度

from math import \*  
  
  
# 计算两个向量的内积  
def i\_inner\_prod(x, y):  
 return sum([x[i] \* y[i] for i in range(len(x))])  
  
  
# 计算向量的模  
def i\_vec\_mode(x):  
 return sqrt(i\_inner\_prod(x, x))  
  
  
# 计算夹角余弦值  
def i\_vec\_cos(x, y):  
 return float(i\_inner\_prod(x, y)) / (i\_vec\_mode(x) \* i\_vec\_mode(y))  
  
  
a = [3, 2, 0, 5, 0, 0, 0, 2, 0, 0]  
b = [1, 0, 0, 0, 0, 0, 0, 1, 0, 2]  
print(r"Vector Angel Cosine: {}".format(i\_vec\_cos(a, b)))

D:\2020下课程\数据挖掘\数据挖掘实验程序\k-means\venv\Scripts\python.exe C:/Users/sdeve/PycharmProjects/pythonProject/main.py

Vector Angel Cosine: 0.314970394174356

Process finished with exit code 0

## 8 多维向量距离计算

from math import \*  
import numpy as np  
  
  
# 注意，按照numpy的默认习惯，每行是一个变量，每列是一个观测值  
# 注意，下面的函数一般都需要numpy的array  
  
# 计算两个矩阵的乘积  
# numpy的对应函数 np.linalg.dot(X, Y)  
def i\_matrix\_mul(X, Y):  
 rrow = np.size(X, axis=0)  
 rcol = np.size(Y, axis=1)  
 rmid = np.size(X, axis=1)  
 rel = np.linspace(0.0, 0.0, rrow \* rcol).reshape(rrow, rcol)  
 for r in range(rrow):  
 for c in range(rcol):  
 rel[r, c] = 0.0  
 for mid in range(rmid):  
 rel[r, c] += X[r, mid] \* Y[mid, c]  
 return rel  
  
  
# 计算期望值（均值），对应np.average  
def i\_mean(x):  
 return sum(x) / len(x)  
  
  
# 计算样本方差  
# 注意，numpy.var计算的是总体的，使用时需要调整  
def i\_var(x):  
 x\_mean = i\_mean(x)  
 return sum((x - x\_mean) \*\* 2) / (len(x) - 1)  
  
  
# 计算样本标准差  
# 注意，numpy.std计算的是总体的，使用时需要调整  
def i\_std(x):  
 x\_mean = i\_mean(x)  
 return sqrt(i\_var(x))  
  
  
# 序列标准化  
def i\_norm\_vec(x):  
 x\_mean = i\_mean(x)  
 x\_std = i\_std(x)  
 return (x - x\_mean) / x\_std  
  
  
# 计算两个向量的协方差  
# numpy的对应方法 np.cov(x, y)，但是返回协方差矩阵  
def i\_cov(x, y):  
 x\_av = np.average(x)  
 y\_av = np.average(y)  
 return sum((x - x\_av) \* (y - y\_av)) / (len(x) - 1)  
  
  
# 计算一个numpy矩阵的协方差阵（每行是一个变量）  
# numpy的对应方法 np.cov(X)  
def i\_cov\_matrix(X):  
 vc = np.size(X, axis=0)  
 cm = np.linspace(0.0, 0.0, vc \*\* 2).reshape(vc, vc)  
 for i in range(vc):  
 for j in range(i, vc):  
 cm[j, i] = cm[i, j] = i\_cov(X[i, ...], X[j, ...])  
 return cm  
  
  
# 计算方差矩阵，一般不用这个函数，这里为实验演示添加  
def i\_var\_matrix(X):  
 vc = np.size(X, axis=0)  
 cm = np.linspace(0.0, 0.0, vc \*\* 2).reshape(vc, vc)  
 for i in range(vc):  
 cm[i, i] = i\_cov(X[i, ...], X[i, ...])  
 return cm  
  
  
# 计算两个向量的相关系数（先标准化在计算协方差）  
# numpy的对应方法 np.corrcoef(x, y)，  
def i\_cor(x, y):  
 x\_s = i\_norm\_vec(x)  
 y\_s = i\_norm\_vec(y)  
 return i\_cov(x\_s, y\_s)  
  
  
# 计算一个numpy矩阵的相关系数阵（每行是一个变量）  
# numpy的对应方法 np.corrcoef(X)  
def i\_cor\_matrix(X):  
 vc = np.size(X, axis=0)  
 cm = np.linspace(0.0, 0.0, vc \*\* 2).reshape(vc, vc)  
 for i in range(vc):  
 for j in range(i, vc):  
 cm[i, j] = i\_cor(X[i, ...], X[j, ...])  
 return cm  
  
  
# 计算矩阵的逆，注意，未检查奇异矩阵  
# numpy的对应函数np.linalg.inv(Y)  
# 其实当不能从下面找到行交换时，就可以判断为奇异矩阵了  
def i\_matrix\_inv(Y):  
 X = np.copy(Y) # 不破坏原来的矩阵  
 n = np.size(X, axis=0)  
 rel = np.identity(n)  
 "先变为右上三角矩阵"  
 for i in range(n):  
 "如果（i，i）位置为0则从下面行向上交换"  
 if abs(X[i, i]) < 1e-5:  
 for r in range(i + 1, n):  
 if X[r, i] > 1e-5:  
 X[i, ...], X[r, ...] = X[r, ...], X[i, ...]  
 rel[i, ...], rel[r, ...] = rel[r, ...], rel[i, ...] # 伴随矩阵要一块变化  
 break  
 "这里如果找不到，可以触发异常，是奇异矩阵"  
 "当前行（i，i）位置变1"  
 t = float(X[i, i])  
 for j in range(i, n): # 只需要处理i后面的列就可以，因为前面已经都变成0了  
 X[i, j] /= t  
 for j in range(n):  
 rel[i, j] /= t # 伴随矩阵要一块变化且要变化所有列  
 "将所有下面行的相应位置变为0"  
 for row in range(i + 1, n):  
 rh = float(X[row, i])  
 for col in range(i, n): # 这里不必处理所有的列，因为前面已经都变成0了  
 X[row, col] -= rh \* X[i, col]  
 for col in range(n):  
 rel[row, col] -= rh \* rel[i, col] # 伴随矩阵要一块变化并且要处理所有的列  
 "再变为单位矩阵（把右上三角都变成0），实际上不用做，伴随矩阵记录变化就可以"  
 for i in range(n - 1, 0, -1):  
 for row in range(i - 1, -1, -1):  
 rh = float(X[row, i])  
 for col in range(n):  
 rel[row, col] -= rh \* rel[i, col] # 伴随矩阵变化，要处理所有的列  
 return rel  
  
  
# 计算欧几里得距离  
# 也可以用numpy的计算模的函数完成计算np.linalg.norm(x-y)  
def dist(x, y):  
 return sqrt(sum([(x[i] - y[i]) \*\* 2 for i in range(len(x))]))  
  
  
# 距离计算的例子，生成两个相关的数据序列，观察其不同的距离  
  
np.random.seed(856479)  
a = np.random.randn(100) # 标准正态分布的序列  
b = 50\*a + 10000 # 期望值为10000，方差为2500的正态分布，且与a完全线性相关  
m = np.vstack((a, b))  
  
# 首尾两个点计算距离  
fp = np.array([a[0], b[0]])  
lp = np.array([a[99], b[99]])  
print(fp, lp)  
print()  
  
# 完全未调整的距离  
print(r"Unadjust distance: {}".format(dist(fp, lp)))  
print()  
  
#计算相关矩阵  
dv = (fp - lp).reshape(1, 2)  
dv\_t = dv.T  
print(dv)  
print(dv\_t)  
print()  
  
# 经过方差矩阵调整的距离  
print("Adjusted only by variance:")  
m\_var = i\_var\_matrix(m)  
m\_var\_inv = i\_matrix\_inv(m\_var)  
print(m\_var)  
print(m\_var\_inv)  
print()  
print(np.sqrt(i\_matrix\_mul(i\_matrix\_mul(dv, m\_var\_inv), dv\_t)))  
print()  
  
# 经过协方差矩阵调整的距离  
print("Adjusted by covariance:")  
m\_cov = i\_cov\_matrix(m)  
m\_cov\_inv = i\_matrix\_inv(m\_cov)  
print(m\_cov)  
print(m\_cov\_inv)  
print()  
print(np.sqrt(i\_matrix\_mul(i\_matrix\_mul(dv, m\_cov\_inv), dv\_t)))

D:\2020下课程\数据挖掘\数据挖掘实验程序\k-means\venv\Scripts\python.exe C:/Users/sdeve/PycharmProjects/pythonProject/main.py

[-6.06541164e-01 9.96967294e+03] [1.97400852e+00 1.00987004e+04]

Unadjust distance: 129.05328700872207

[[ -2.58054968 -129.02748409]]

[[ -2.58054968]

[-129.02748409]]

Adjusted only by variance:

[[1.13089792e+00 0.00000000e+00]

[0.00000000e+00 2.82724480e+03]]

[[8.84253107e-01 0.00000000e+00]

[0.00000000e+00 3.53701243e-04]]

[[3.43174903]]

Adjusted by covariance:

[[1.13089792e+00 5.65448960e+01]

[5.65448960e+01 2.82724480e+03]]

[[ 1.09951163e+15 -2.19902326e+13]

[-2.19902326e+13 4.39804651e+11]]

[[2.32444099]]

Process finished with exit code 0