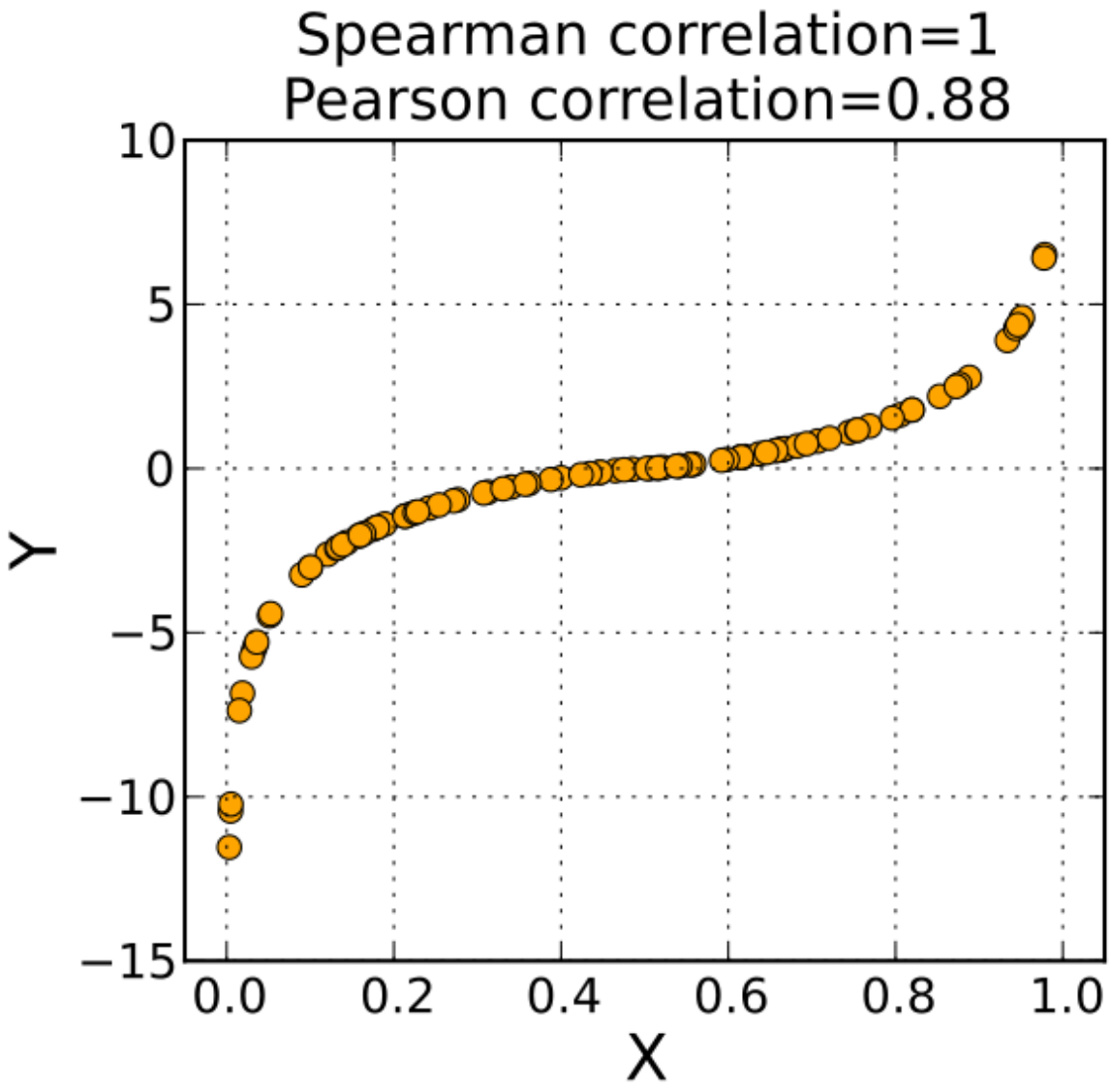
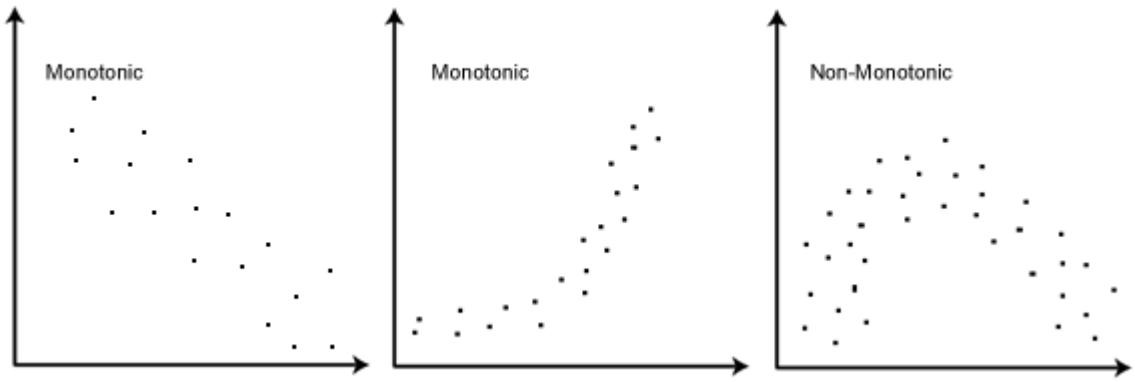
Candidates of measuring independence of two continuous variables

1. 논문 "Efficient test for nonlinear dependence of two continuous variables"

- Pearson correlation coefficient(Pearson product-moment correlation coefficient)
  - $\rho = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$
  - non robust Pearson's r sample statistic
  - misleading values in the presence of outliers
- Spearman’s rank correlation coefficient
  -


  -



- What is the concept of rank?

	Maths (mark)	Rank (English)	Rank (maths)
56	66	9	4
75	70	3	2
45	40	10	10
71	60	4	7
61	65	6.5	5
64	56	5	9
58	59	8	8
80	77	1	1
76	67	2	3
61	63	6.5	6

- The **Spearman correlation** between two variables is equal to the Pearson correlation between the rank values of those two variables
- $rs = \rho_{rg_X, rg_Y} = \frac{cov(rg_X, rg_Y)}{\sigma_{rg_X} \sigma_{rg_Y}}$
- $\rho = \frac{\sum(x_i - \hat{x})(y_i - \hat{y})}{\sqrt{\sum(x_i - \hat{x})^2 \sum(y_i - \hat{y})^2}}$ where i = paired score.

- CANOVA

Table 1 Simulation power in nine simple functions

N = 50, x ~ U(−1,1)	CANOVA2	CANOVA4	CANOVA8	CANOVA12	Pearson	Kendall	Spearman	Distance	Hoeffding	MIC
y = 0 + N(0,1)	0.051	0.048	0.048	0.050	0.047	0.048	0.049	0.039	0.059	0.051
y = x + N(0,1)	0.564	0.798	0.889	0.902	**0.972**	0.962	0.961	0.950	0.953	0.591
$y = 0.5;*(x+1)^2 + N(0,1)$	0.606	0.836	0.904	0.918	**0.968**	0.953	0.962	0.964	0.953	0.633
y = sin(Pi*x) + N(0,1)	0.758	0.941	0.966	0.962	0.936	0.918	0.930	**0.969**	**0.969**	0.829
y = sin(2*Pi*x) + N(0,1)	0.713	**0.886**	0.812	0.294	0.318	0.328	0.320	0.341	0.405	0.579
y = sin(3*Pi*x) + N(0,1)	0.677	**0.796**	0.254	0.076	0.178	0.192	0.199	0.186	0.219	0.423
y = cos(Pi*x) + N(0,1)	0.784	0.940	**0.973**	0.942	0.067	0.076	0.083	0.660	0.710	0.660
y = cos(2*Pi*x) + N(0,1)	0.738	**0.891**	0.754	0.142	0.045	0.054	0.053	0.100	0.129	0.548
y = cos(3*Pi*x) + N(0,1)	0.673	**0.751**	0.160	0.031	0.053	0.054	0.057	0.074	0.090	0.371

```
#!/usr/bin/python

import sys
import numpy as np
import multiprocessing


class Data:
    def __init__(self,x,y):
    self.x = x
    self.y = y

def sort_data(data_list):
    o = sorted(data_list,key=lambda Data:Data.x)
    ##insert sorting
    tied_x=[];i=0;l=len(o)
    while i<l-1:
        if o[i].x == o[i+1].x:
            t=[i,i+1]
            while 1:
                i +=  1;
                if i+1 == l:tied_x.append(t);break
                if o[i].x == o[i+1].x:t.append(i+1)
                else: i -= 1;tied_x.append(t);break
        i += 1
    return o,tied_x

def shuffle_Y(o,tied_x):
    for i in tied_x:
        Y=[]
        for j in i:
            Y.append(o[j].y)
        new=np.random.permutation(Y)
        k=0
        for j in i:
            o[j].y=new[k]
            k += 1
    return o

def show(o):
    x=[];y=[]
```

```
    for i in o:
        x.append(str(i.x))
        y.append(str(i.y))
    print '\t'.join(x)
    print '\t'.join(y)

def cal_observe_W(o,k=2):
    w=0;rank=0
    for i in o:
        r1=max(rank-k+1,0)
        #r2=min(len(o),rank+k)
        for j in range(r1,rank):
            w += (i.y-o[j].y)*(i.y-o[j].y)
        rank += 1
    return w

def Permutation(o,loop,k):
    w=[]
    for i in range(loop):
        Y=[];n=o
        for i in o:
            Y.append(i.y)
        new=np.random.permutation(Y)
        for i,j in zip(n,new):
        i.y=j
        w.append(cal_observe_W(n,k))
    return w

def canova(x,y,permutation=10000,shuffle_times=100,k=2,threads=10):
    data_list = list_to_data(x,y)
    o,tie=sort_data(data_list)
    w=[]
    if len(tie) != 0:
        for i in range(shuffle_times):
            n=shuffle_Y(o,tie)
            w.append(cal_observe_W(n,k))
    else: w=[cal_observe_W(o,k)]
    aver_w=float(sum(w))/float(len(w))
    #permutation
    #multiprocessing
    pool = multiprocessing.Pool(processes=threads)
    each_loop = int(permutation/threads)
    result = []
    for i in range(threads-1):
        result.append(pool.apply_async(Permutation, (o,each_loop,k, )))
    rest = permutation - each_loop*(threads-1)
    result.append(pool.apply_async(Permutation, (o,rest,k, )))
    pool.close()
    pool.join()

    count = 0
    for res in result:
    for i in res.get():
```

```
        if i<=aver_w: count +=1
    p=float(count)/float(permutation)
    return p

def list_to_data(x,y):
    o=[]
    for i,j in zip(x,y):
    o.append(Data(int(i),int(j)))
    return o

if __name__ == '__main__':
    '''
    This is a canova python package
    '''
    canova()
```