Some Methods Based on Ranks (Conover Chapter Five)

STAT 345-01: Nonparametric Statistics [*]

Fall Semester 2018

Contents

[*]Copyright 2018, John T. Whelan, and all that

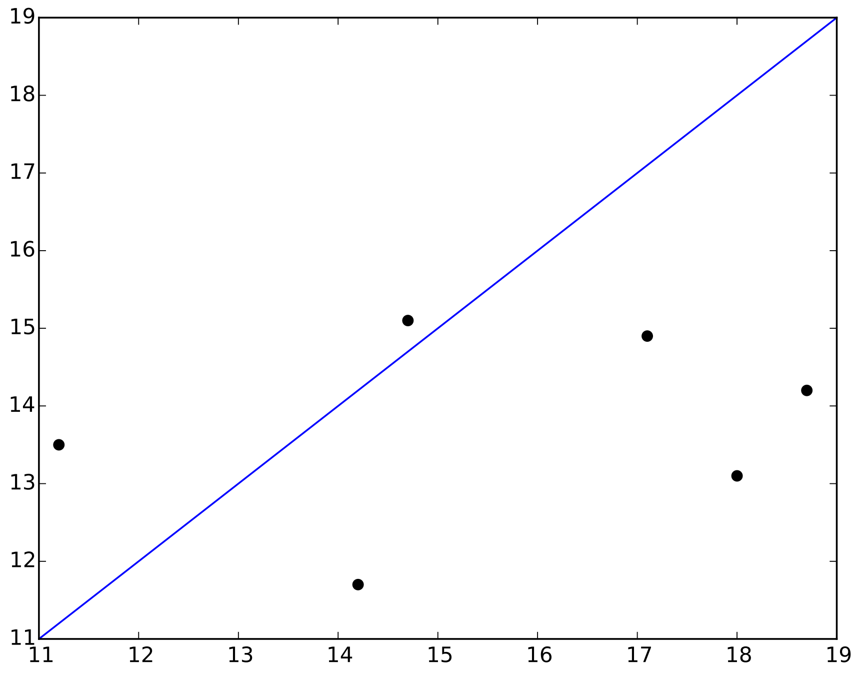
Thursday 20 September 2018
– Read Section 5.7 of Conover; refer to Section 3.1 of Hollander and Section 4.2 of Higgins

1 The Wilcoxon Signed Rank Test

Recall that last week we considered the sign test for paired data, where the random sample is $\{(X_i, Y_i)\}$. The test statistic was constructed from the signs of the differences[1] $D_i = Y_i - X_i$. We noted that this meant discarding a lot of information about the data, in order to avoid being thrown off by outliers. But we can still keep a little more information about the size of the differences. Consider the following dataset:

```
from __future__ import division
import numpy as np
from scipy import stats
xi = np.array([18.0, 18.7, 14.7, 17.1, 11.2, 14.2])
yi = np.array([13.1, 14.2, 15.1, 14.9, 13.5, 11.7])
figure();
plot(xi,yi,'ko');
plot([11,19],[11,19]);
savefig('notes05_sr_scatter.eps',bbox_inches='tight');
```

[1] We're adopting Conover's somewhat unusual convention of calling $y_i > x_i$ a positive difference. Of course, in a given problem you can generally define which quantity is x and which is y, so you should always sanity check the signs rather than applying formulas blindly.



The sign test simply observes that two of the points are above the line ($d_i > 0$) and four are below the line ($d_i < 0$) so the two-sided p-value is

$$P(N_+ \leq 2) + P(N_+ \geq 4) = P(N_+ \neq 3) = 1 - \binom{6}{3}\frac{1}{2^6} = 1 - \frac{20}{64} = 0.6875 \quad (1.1)$$

But looking at the plot, this configuration looks unusual in a way that's not captured by counting the number of positive and negative differences. The four points below the line are generally farther away from it than the two points above the line. The signed rank test tries to address this by ranking the absolute differences $|y_i - x_i|$ from smallest to largest and then, instead of just counting the positive differences, adding up their ranks:

```
In [10]: di = yi - xi; di
```

```
Out[10]: array([-4.9, -4.5,  0.4, -2.2,  2.3, -2.5])

In [11]: idx = np.argsort(np.abs(di))

In [12]: di[idx]
Out[12]: array([ 0.4, -2.2,  2.3, -2.5, -4.5, -4.9])
```

The positive differences are 0.4 (which is the smallest) and 2.3 (which is the third smallest, so in this case the test statistic is $t^+ = 1 + 3 = 4$.

```
In [13]: n = len(xi)

In [14]: ranks = np.arange(n) + 1; ranks
Out[14]: array([1, 2, 3, 4, 5, 6])

In [15]: posranks = ranks[di[idx] >= 0]; posranks
Out[15]: array([1, 3])

In [16]: t = posranks.sum(); t
Out[16]: 4
```

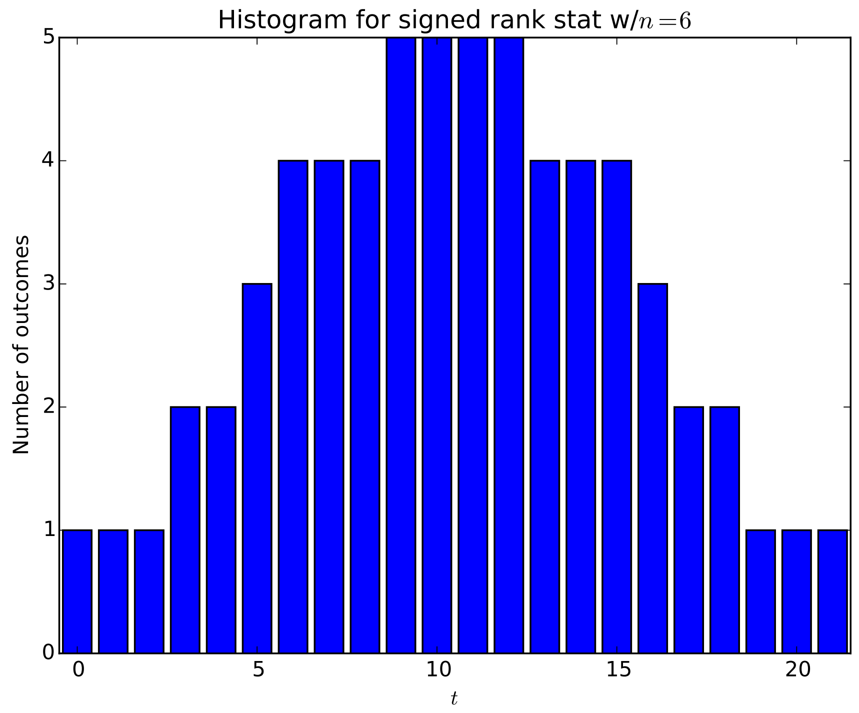
To get a p-value, we need to think about the probability distribution of this statistic under the null hypothesis that each difference is as likely to be positive as negative. If each rank is equally likely to have a $+$ associated with it as a $-$, there are 2^n different outcomes, which are all equally likely. Here, $n = 6$, so $2^n = 2^6 = 64$. Now, we have to ask how many of them give a signed rank statistic of 4 or less. There are few enough of them that we can just list them:

$-$	$-$	$-$	$-$	$-$	$-$	0
$+$	$-$	$-$	$-$	$-$	$-$	1
$-$	$+$	$-$	$-$	$-$	$-$	2
$-$	$-$	$+$	$-$	$-$	$-$	3
$+$	$+$	$-$	$-$	$-$	$-$	1+2=3
$-$	$-$	$-$	$+$	$-$	$-$	4
$+$	$-$	$+$	$-$	$-$	$-$	1+3=4

So there are 7 out of 64 outcomes with $T \leq 4$, and another 7 equally extreme outcomes on the other end of the distribution (more on this in a moment), so the p-value is $\frac{14}{64} = 0.21875$. It's still not in "statistically significant" territory (since this is such a small sample), but it does a better job than the sign test of flagging this dataset as slightly unusual. The null distribution of this statistic is tabulated in Conover's Appendix Table A12. We can cobble something together in Python to enumerate all 64 outcomes, calculate the signed rank statistic, and histogram the results:

```
flags = np.array([[np.right_shift(d,i) % 2
for i in xrange(n)] for d in xrange(2**n)])
ranks*flags
(ranks*flags).shape
wilco = (ranks*flags).sum(axis=-1)
wmin = min(wilco)
wmax = max(wilco)
figure();
counts,bins = np.histogram(wilco,bins=np.arange(wmin
    ,wmax+2))
wvals = bins[:-1]
bar(wvals,counts,align='center');
xlim(wmin-0.5,wmax+0.5);
xlabel('$t$');
ylabel('Number of outcomes');
```

```
title('Histogram for signed rank stat w/$n=%d$' % n)
    ;
savefig('notes05_wilco_hist.eps',bbox_inches='tight
    ');
```



We can divide by $2^n = 64$ to get the pmf, and we can use this (either the histogram or the original array) to get the p value as above:

```
In [33]: np.sum(wilco <= t)
Out[33]: 7

In [34]: 2*np.mean(wilco <= t)
Out[34]: 0.21875
```

Unfortunately, this solution doesn't scale very well. For $n = 30$, there are a billion outcomes, and the associated "giga-" will use up my laptop's memory and grind everything to a halt. Fortunately, SciPy has a function that does the Wilcoxon signed rank test on the original data:

```
In [35]: stats.wilcoxon(xi,yi)
/usr/lib/python2.7/dist-packages/scipy/stats/
    morestats.py:1961:
UserWarning: Warning: sample size too small for
    normal approximation.
  warnings.warn("Warning: sample size too small for
      normal approximation.")
Out[35]: (4.0, 0.17295491798842066)
```

Unfortunately, it doesn't actually calculate the exact p-value, as indicated by the warning and the fact that it gets the wrong answer. It does at least calculate the correct signed rank statistic for us, though. This is one place where R really does win.[2] It implements the null distribution for the signed rank statistic as `dsignrank()`, the cumulative distribution as `psignrank`, and the quantiles as `qsignrank`:

```
> n <- 6
> t <- seq(0,n*(n+1)/2)
> plot(t,dsignrank(t,n))
> 2*psignrank(4,n)
[1] 0.21875
> qsignrank(0.025,n)
[1] 1
> psignrank(1,n)
[1] 0.03125
```

[2]Note that you can actually call R functions from within python using the `rpy2` package, which can be pretty handy in cases like this where one function is missing.

It also has a `wilcox.test()` function which gets the exact p-value for small sample sizes:

```
> x <- c(18.0, 18.7, 14.7, 17.1, 11.2, 14.2)
> y <- c(13.1, 14.2, 15.1, 14.9, 13.5, 11.7)
> wilcox.test(x,y,paired=TRUE)

        Wilcoxon signed rank test

data:  x and y
V = 17, p-value = 0.2188
alternative hypothesis: true location shift is not
   equal to 0
```

The p-value is indeed 0.21875, rounded to four decimal places. Note that R reports the statistic value as 17; that's because R is adding the ranks of the differences where x_i is larger than y_i, so it gets what Conover would call t^-. The two statistics are related in general by

$$T^+ + T^- = \sum_{r=1}^{n} r = \frac{n(n+1)}{2} \tag{1.2}$$

In this case, $\frac{n(n+1)}{2} = \frac{6(7)}{2} = 21$. This is also the maximum possible value of the statistic T^+.

When n becomes large, the distribution of T^+ becomes approximately normal, which simplifies the calculation of p-values for large n. It turns out to be easier to write this in terms of

$$W = T^+ - T^- = 2T^+ - \frac{n(n+1)}{2} \tag{1.3}$$

This is just the sum of *all* the ranks with a + or − sign attached, so for the data above it would be

$$w = +1 - 2 + 3 - 4 - 5 - 6 = +4 - 17 = -13 \tag{1.4}$$

It's easy to see that $E(W = 0)$, and possible to work out[3]

$$\text{Var}(W) = E(W^2) = \sum_{r=1}^{n} r^2 = \frac{n(n+1)(2n+1)}{6} \tag{1.5}$$

So that you can then use the statistic

$$Z = \frac{W}{\sqrt{\frac{n(n+1)(2n+1)}{6}}} \tag{1.6}$$

1.1 Ties

In the sign test, we dealt with the possibility that $X_i = Y_i$ for some observations by ignoring those observations and just applying the sign test to the smaller subsample with those zero differences excluded. We do that in the signed rank test as well, but we also have another kind of tie to contend with: what if two of the differences are non-zero in magnitude: $|d_i| = |d_j|$ for some $i \neq j$? (This is possible if X and Y are discrete, or rounded off to low enough precision to be effectively discrete.) What ranks are assigned to them? The procedure is straightforward: if the two or more absolute differences are equal, assign the average of the relevant ranks to each of them. E.g., if the third and fourth average differences are the same, they each get a rank of 3.5. Assigning a p-value to the statistic is somewhat trickier, though, since the whole argument about possible outcomes assumed there were no ties among the ranks. In fact, you can't really calculate the probability distribution for T^+ in the presesence of ties without knowing how likely it is to have ties in the rankings. But you can still use the normal approximation. The one difference is that the variance of W is no longer a constant,

[3]Hint: let the rth signed rank R_r be $\pm r$ so that $W = \sum_{r=1}^{n} R_r$. Argue that $E(R_r) = 0$, $E(R_r R_s) = 0$ if $r \neq s$, and $E({R_r}^2) = r^2$.

and instead you have to put in the sum of the squares of the ranks, to get a test statistic

$$Z = \frac{W}{\sqrt{\sum_{r=1}^{n} {R_r}^2}} = \frac{\sum_{r=1}^{n} R_r}{\sqrt{\sum_{r=1}^{n} {R_r}^2}} \tag{1.7}$$

For example, with $n = 6$ differences and no ties, the sum of the squared ranks will be

$$\begin{aligned}\sum_{r=1}^{n} (r_r)^2 &= 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 = 91 \\ &= \frac{n(n+1)(2n+1)}{6} = \frac{6(7)(13)}{6}\end{aligned} \tag{1.8}$$

On the other hand, if there's a tie for third and fourth, you'll get

$$\sum_{r=1}^{n} (r_r)^2 = 1^2 + 2^2 + 3.5^2 + 3.5^2 + 5^2 + 6^2 = 90.5 \tag{1.9}$$

Tuesday 25 September 2018 – Read Section 5.7 of Conover; refer to Section 3.3 of Hollander and Section 4.2 of Higgins

1.2 Confidence Interval for the Median Difference

The signed rank statistic can also be used to define a confidence interval on the typical offset between the random variables in a paired distribution. Consider the paired data $\{(x_i, y_i)\}$ from last time, along with the differences $d_i = y_i - x_i$.

```
In [1]: from __future__ import division

In [2]: import numpy as np

In [3]: from scipy import stats

In [4]: xi = np.array([18.0, 18.7, 14.7, 17.1, 11.2, 14.2])

In [5]: yi = np.array([13.1, 14.2, 15.1, 14.9, 13.5, 11.7])

In [6]: di = yi - xi; di
Out[6]: array([-4.9, -4.5,  0.4, -2.2,  2.3, -2.5])

In [7]: n = len(di)
```

Last time we considered the null hypothesis that the differences were drawn from a symmetric distribution with zero median, which we evaluated using the signed rank statistic. Consider instead the hypothesis that the differences $\{D_i\}$ come from a symmetric distribution with median θ, i.e., that $\{D_i - \theta\}$ is drawn from a symmetric distribution with median 0. So we apply the signed rank test to $d_i - \theta = y_i - x_i - \theta$, and ask for which values of θ we do not reject H_0 at a certain significance. In particular, if we want a 90% two-sided confidence interval, we should ask for which θ values we get a p-value of more than 0.05. We can get the 5th percentile of the sign rank distribution with $n = 6$ using R, which I'll import into python to avoid switching environments:

```
In [8]: from rpy2.robjects.packages import importr

In [9]: rstats = importr('stats',on_conflict="warn")
/usr/lib/python2.7/dist-packages/rpy2/robjects/
   packages.py:216: UserWarning: Conflict when
   converting R symbol in the package "stats" to a
   Python symbol (format.perc -> format_perc while
   there is already format_perc)
```

```
  warn(msg)

In [10]: print(rstats.qsignrank(0.05,6))
[1] 3
```

We can check the cdf of 3, which is $P(T^+ \leq 3)$, and the cdf of 2, which is $P(T^+ < 3)$

```
In [11]: print(rstats.psignrank(3,6))
[1] 0.078125

In [12]: print(rstats.psignrank(2,6))
[1] 0.046875

In [13]: print(2.*(rstats.psignrank(2,6)[0]))
0.09375
```

So we get a p-value of 0.094 if $T^+ \leq 2$ or $T^+ \geq 17$. Of course we could also tell this from our hand-counting of outcomes which told us there were 3 ways out of 64 to make a signed-rank statistic of 0, 1 or 2:

```
In [14]: 6/64
Out[14]: 0.09375

In [15]: 1-6/64
Out[15]: 0.90625
```

Recall that the sign-rank statistic for the original $\{d_i\}$ (which corresponds to $\theta = 0$ gave us positive ranks of 1 and 3, so $t_+ = 1 + 3 = 4$:

```
In [16]: theta = 0.

In [17]: di[np.argsort(np.abs(di-theta))]-theta
Out[17]: array([ 0.4, -2.2,  2.3, -2.5, -4.5, -4.9])
```

Since $2 < 4 < 19$, that means $\theta = 0$ is in within the confidence interval. We consider what happens to the adjusted differences $d_i - \theta$ as we vary θ:

```
In [18]: theta = 0.025

In [19]: di[np.argsort(np.abs(di-theta))]-theta
Out[19]: array([ 0.375, -2.225, 2.275, -2.525,
    -4.525, -4.925])
```

As we increase θ, the positive differences move towards zero and the negative differences move away from zero. The signed rank statistic will change when they cross each other and change places in the sorted list:

```
In [21]: figure();

In [22]: for i in xrange(n):
    ....:         plot(di[i]-thetavec,thetavec,'g-',lw=1.5);
    ....:         plot(thetavec-di[i],thetavec,'r--',lw=1.5)
    ....:

In [23]: xlim(0,8);

In [24]: ylim(0,0.2);
```

The first time that happens is when $\theta = 0.05$

```
In [25]: theta = 0.05

In [26]: di[np.argsort(np.abs(di-theta))]-theta
Out[26]: array([ 0.35, -2.25,  2.25, -2.55, -4.55, -4.95])
```

because

$$-2.2 - 0.05 = -2.25 = -(2.3 - 0.05) \tag{1.10}$$

this crossing lowers the signed-rank statistic from $1+3=4$ to $1+2=3$. The θ at which it occurs, is when

$$d_i - \theta = -(d_j - \theta) \tag{1.11}$$

where i and j are the indices of the differences $d_i = -2.2$ and $d_j = 2.3$. This can be solved for

$$\theta = \frac{d_i + d_j}{2} \tag{1.12}$$

i.e., the signed rank statistic changes by 1 when θ crosses the average of two of the differences.

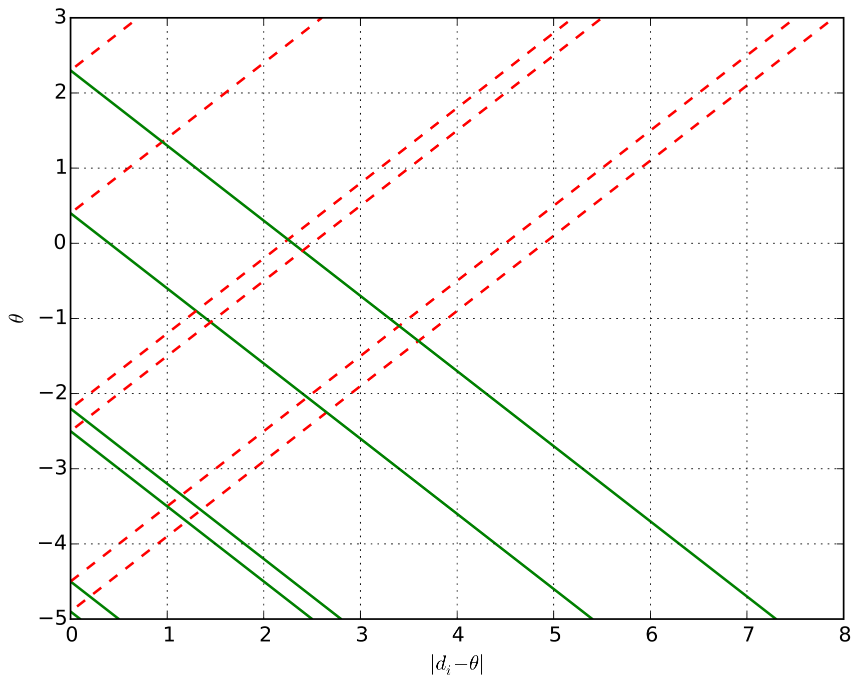
If we keep going, we see that the next time the rank-sum statistic changes is when $\theta = 0.4$:

```
In [27]: ylim(0,1);

In [28]: theta = 0.4

In [29]: di[np.argsort(np.abs(di-theta))]-theta
Out[29]:
array([  3.33066907e-16,   1.90000000e+00,  -2.60000000e+00,
        -2.90000000e+00,  -4.90000000e+00,  -5.30000000e+00])
```

At this point, $d_i - \theta$ changes sign, where i is the index of the difference $d_i = 0.4$, and the signed-rank statistic drops from $1+2=3$ to 2. This means that for $\theta > 0.4$ the signed-rank statistic will be ≤ 2 and the two-sided p-value will be below 10%, so this is the upper end of the 90% confidence interval.

So we see that the value of the signed-rank statistic changes by 1 every time θ crosses the average of two differences $\frac{d_i+d_j}{2}$, or an individual difference d_i (which is the average $\frac{d_i+d_i}{2}$):

```
In [30]: ylim(-5,3);

In [31]: grid(True);

In [32]: xlabel(r'$|d_i-\theta|$');

In [33]: ylabel(r'$\theta$');

In [34]: savefig('notes05_sr_ci.eps',bbox_inches='tight');
```



So if we want to get the values of θ that fall between the three highest and three lowest signed-rank statistic values, we just have to list those values. It's not quite the full list of averages $\frac{d_i+d_j}{2}$ for all i and j, since that would list the differences with $i \neq j$ twice, and they only change the statistic by one, not two:

```
In [35]: 0.5*(di[:,None]+di[None,:])
```

```
Out[35]:
array([[-4.9 , -4.7 , -2.25, -3.55, -1.3 , -3.7 ],
       [-4.7 , -4.5 , -2.05, -3.35, -1.1 , -3.5 ],
       [-2.25, -2.05,  0.4 , -0.9 ,  1.35, -1.05],
       [-3.55, -3.35, -0.9 , -2.2 ,  0.05, -2.35],
       [-1.3 , -1.1 ,  1.35,  0.05,  2.3 , -0.1 ],
       [-3.7 , -3.5 , -1.05, -2.35, -0.1 , -2.5 ]])
```

Instead, we just need the list for $i \leq j$:

```
In [36]: avgs = np.sort(np.array(list(flatten
    ([[0.5*(di[i]+di[j]) for j in xrange(i,n)] for i
    in xrange(n)]))))

In [37]: avgs
Out[37]:
array([-4.9 , -4.7 , -4.5 , -3.7 , -3.55, -3.5 ,
    -3.35, -2.5 , -2.35,
        -2.25, -2.2 , -2.05, -1.3 , -1.1 , -1.05, -0.9
           , -0.1 , 0.05,
         0.4 , 1.35, 2.3 ])
```

Note that in the list of averages $\frac{d_i+d_j}{2}$ for $i = 1, \ldots, n$, $j = i, \ldots, n$, there will be $\frac{n(n+1)}{2}$ individual entries, which is consistent with the fact that the signed-rank statistic can be any value from 0 to $\frac{n(n+1)}{2}$.

So in particular, to get the 90% confidence interval when $n = 6$, i.e., the range of θ values which give a signed-rank statistic greater than 2 and less than 19, we need to take the 2nd and 19th numbers in the sorted list of differences. In our case, that is from -4.5 to 0.4.

```
In [38]: (avgs[2-1],avgs[19-1])
Out[38]: (-4.7000000000000002, 0.40000000000000036)
```

Note that the point estimate, i.e., a 0% confidence interval, would be the middle entry, number $\frac{n(n+1)}{4}$ out of $\frac{n(n+1)}{2}$ if n is even. This is not the sample median of the $\{d_i\}$, but rather the median of the averages. This is known as the **Hodges-Lehmann estimator**, and it's actually estimating something called the **pseudomedian**, which is the median of the averages of pairs of draws from the population or distribution. For a symmetric distribution, the median and pseudomedian (and mean, if it exists) are all the same.

Thursday 27 September 2018 – Read Section 5.1 of Conover; refer to Section 4.1 of Hollander and Sections 2.4-2.6 of Higgins

2 The Wilcoxon-Mann-Whitney Rank Sum Test

Suppose that, instead of having paired data, we have two independent samples $\{X_i | i = 1, \ldots, n\}$ and $\{Y_j | j = 1, \ldots, m\}$[4] from possibly different distributions. Since the data are not paired, the two samples need not be the same size, either. For convenience, we'll call the total size of the two samples together $n + m = N$. We can test the null hypothesis that the two samples are drawn from the same distribution, or more precisely that $P(X > Y) = P(X < Y)$ by combining the two samples into one list, ranking that list, and then using as a test statistic the sum of the ranks of the $\{x_i\}$. As an example, consider the following samples, with $n = 5$, $m = 4$, and $N = 5 + 4 = 9$:

[4]Note that we're following Conover's somewhat unusual convention of calling the size of the *first* sample n and the size of the second sample m, never mind alphabetical order.

```
In [1]: from __future__ import division

In [2]: import numpy as np

In [3]: from scipy import stats

In [4]: xi = np.array([8.56, 5.03, 48.1, 1.31, 4.82])

In [5]: yi = np.array([15.0, 12.3, 28.0, 13.9])

In [6]: n = len(xi); n
Out[6]: 5

In [7]: m = len(yi); m
Out[7]: 4

In [8]: N = n+m; N
Out[8]: 9
```

It's convenient to summarize the data in a table:

rank	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Data	1.31	4.82	5.03	8.56	12.3	13.9	15.0	28.0	48.1
Set	x	x	x	x	**y**	**y**	**y**	**y**	x

```
In [9]: combo = np.concatenate((xi,yi))

In [10]: idx = np.argsort(combo)

In [11]: combo[idx]
Out[11]:
array([ 1.31,  4.82,  5.03,  8.56, 12.3 , 13.9 ,
   15.  , 28.  ,
        48.1 ])

In [12]: xflags = np.concatenate(([True,]*n,[False
   ,]*m))

In [13]: xflags[idx]
Out[13]: array([ True, True, True, True, False,
   False, False, False, True], dtype=bool)

In [14]: ranks = np.arange(N)+1

In [15]: xranks = ranks[xflags[idx]]; xranks
Out[15]: array([1, 2, 3, 4, 9])

In [16]: Wx = np.sum(xranks); Wx
Out[16]: 19
```

The Wilcoxon rank-sum statistic is $w_x = 1+2+3+4+9 = 19$, because the four lowest values in the combined sample (1.31, 4.82, 5.03, and 8.56), and the highest (48.1) come from the $\{x_i\}$. With a little bit of thought, we can see that the smallest possible value for w_x is

$$\min(W_x) = 1 + \cdots + n = \frac{n(n+1)}{2} \tag{2.1}$$

which in this case is 15, while the largest is

$$\max(W_x) = (m+1) + \cdots + N = nm + \frac{n(n+1)}{2} \tag{2.2}$$

which in this case is 35. If H_0 is true, so that the x ranks are equally likely to fall anywhere in the list, the expected average rank should be the middle value $\frac{N+1}{2}$, so the null expectation value of the rank-sum statistic is

$$E(W_x) = \frac{n(N+1)}{2} = \frac{n(n+m+1)}{2} \qquad \text{if } H_0 \text{ true} \tag{2.3}$$

Some related statistics which carry the same information are:

- The sum of the ranks of the $\{y_j\}$ is $w_y = 5+6+7+8 = 26$.
- The Mann-Whitney U statistic[5] is the total over the $\{x_i\}$, of how many of the $\{y_j\}$ each one of them is greater than. So for this data set this is $u_x = 0+0+0+0+4 = 4$.
- The Mann-Whitney U statistic for the $\{y_j\}$ is $u_y = 4+4+4+4 = 20$.

```
In [17]: yflags = np.bitwise_not(xflags)

In [18]: yranks = ranks[yflags[idx]]; yranks
Out[18]: array([5, 6, 7, 8])

In [19]: Wy = np.sum(yranks); Wy
Out[19]: 26

In [20]: Ux = np.sum(xi[None,:] > yi[:,None]); Ux
Out[20]: 4

In [21]: Uy = np.sum(yi[None,:] > xi[:,None]); Uy
Out[21]: 16
```

Some relationships among these statistics fall out:

- The sum of w_x and w_y is the sum of all the ranks, which is

$$w_x + w_y = 1 + \cdots + N = \frac{N(N+1)}{2} \tag{2.4}$$

- The Wilcoxon and Mann-Whitney statistics are related by

$$w_x = u_x + \frac{n(n+1)}{2} \tag{2.5}$$

  Note that this is consistent with the fact that the Mann-Whitney u statistic has a minimum value of 0 and a maximum value of nm.
- The Mann-Whitney U statistics for x and y are related by

$$u_x + u_y = nm \tag{2.6}$$

```
In [22]: Wx + Wy
Out[22]: 45

In [23]: N*(N+1)//2
Out[23]: 45

In [24]: Wx - Ux
Out[24]: 15

In [25]: n*(n+1)//2
Out[25]: 15

In [26]: Wy - Uy
Out[26]: 10

In [27]: m*(m+1)//2
Out[27]: 10

In [28]: Ux + Uy
Out[28]: 20

In [29]: m*n
Out[29]: 20
```

[5]Mann and Whitney publlished their statistic in *Annals of Mathematical Statistics* **18**, 50 (1947), `https://dx.doi.org/10.1214/aoms/1177730491`. It actually cites Wilcoxon's paper *Biometrics Bulletin* **1**, 80 (1945) `https://doi.org/10.2307/3001968` which proposed both the signed rank and rank sum tests, but didn't work out the null distribution.

If the sample size is large, we can use the Central Limit Theorem to convert the rank-sum statistic into a z score (perhaps with the appropriate continuity correction) and compare it to the standard normal percentiles. If the size is small, the p-value comes down to a bit of counting.

Returning to our example, since $W = 19$ is less than $\frac{n(N+1)}{2} = 25$, this statistic is on the low side. That means that the xs seem to appear earlier in the list than the ys do, so we are in the direction indicated by the alternative hypothesis, which says the $\{Y_j\}$ distribution are "stochastically larger" than the $\{X_i\}$, i.e.,

$$H_1 : \quad P(Y_j > X_i) > P(Y_j < X_i) \tag{2.7}$$

But how unlikely is so low a rank score given the null hypothesis? If the samples really are drawn from the same distribution, then any set of 5 out of the 9 possible ranks is equally likely. There are

$$\binom{9}{5} = \frac{9!}{5!4!} = \frac{9 \times 8 \times 7 \times 6}{4 \times 3 \times 2 \times 1} = 126 \tag{2.8}$$

possibilities. If we tabulate the possible sets of ranks corresponding to each statistic value, we find

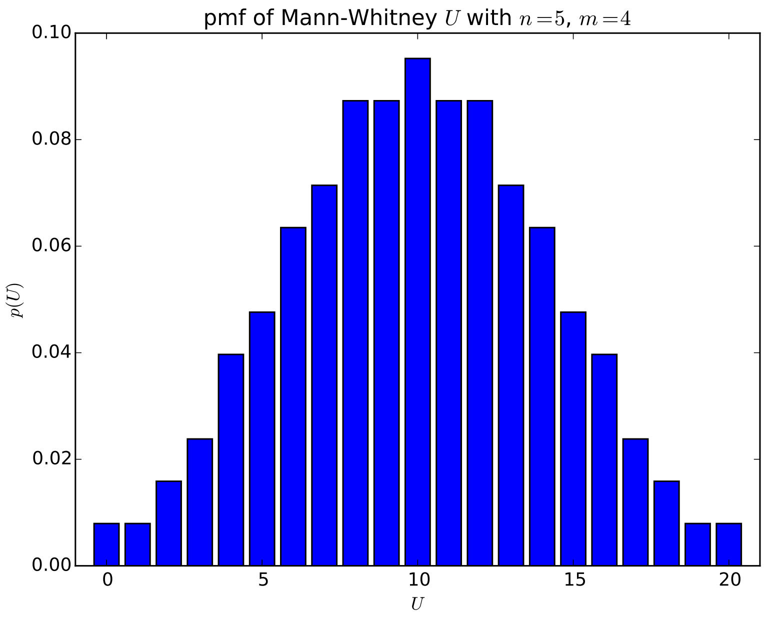
U_x	W_x	Ranks	Prob
0	15	12345	1/126
1	16	12346	1/126
2	17	12347, 12356	2/126
3	18	12348, 12357, 12456	3/126
4	19	12349, 12358, 12367, 12457, 13456	5/126

So there's a $12/126 \approx 9.5\%$ chance of getting a W statistic this low by chance if the null hypothesis is true and the samples are actually drawn from the same distribution, and a one-tailed test would reject H_0 at the 10% level but not the 5% level.

We can code up the counting of permutations in python, but as with the signed rank statistic, R already has a set of distribution functions for the Mann-Whitney U statistic:

```
In [30]: U = np.arange(n*m+1)

In [31]: from rpy2.robjects.packages import importr

In [32]: rstats = importr('stats',on_conflict="warn
    ")
/usr/lib/python2.7/dist-packages/rpy2/robjects/
    packages.py:216: UserWarning: Conflict when
    converting R symbol in the package "stats" to a
    Python symbol (format.perc -> format_perc while
    there is already format_perc)
  warn(msg)

In [33]: rstats.dwilcox(19-15,5,4)
Out[33]:
<FloatVector - Python:0x7fbd97381248 / R:0x4e8b928>
[0.039683]

In [34]: 126 * rstats.dwilcox(19-15,5,4)[0]
Out[34]: 5.0

In [35]: rstats.pwilcox(19-15,5,4)
Out[35]:
<FloatVector - Python:0x7fbd97557d88 / R:0x51e76d8>
[0.095238]

In [36]: import rpy2.robjects.numpy2ri as rpyn

In [37]: Upmf = np.array(rstats.dwilcox(rpyn.
    numpy2ri(U),n,m))
```

```
In [38]: figure();

In [39]: bar(U,Upmf,align='center');

In [40]: xlabel(r'$U$');

In [41]: ylabel(r'$p(U)$');

In [42]: title(r'pmf of Mann-Whitney $U$ with $n=%d$
    , $m=%d$' % (n,m));

In [43]: xlim(-1,n*m+1);

In [44]: savefig('notes05_mw_pmf.eps',bbox_inches='
    tight');
```



Despite the names `dwilcox`, `pwilcox` etc, the R functions relate to the distribution of the Mann-Whitney statistic U_x, not the Wilcoxon rank-sum statistic $W_x = U_x + \frac{n(n+1)}{2}$.

2.1 Normal Approximation

If the sample sizes are large, we can treat the statistic W_x (or U_x) as approximately normal. As noted above, $E(W_x) = \frac{n(N+1)}{2}$, and similarly

$$E(U_x) = \frac{nm}{2} \tag{2.9}$$

The variance is a little trickier to work out, but if we consider that

$$W_x = \sum_{i=1}^{n} R_i \tag{2.10}$$

where R_i is the rank (in the combined list) of X_i, then $\{R_i\}$ is a sample of size n drawn *without replacement* from the integers $1, \ldots, N$, then it turns out that Conover worked out back in section 1.4 that $\operatorname{Var}(R_i) = \frac{(N+1)(N-1)}{12}$ and $\operatorname{Cov}(R_i, R_{i'}) = -\frac{(N+1)}{12}$ (where $i \neq i'$), so

$$\begin{aligned}\operatorname{Var}(W_x) &= n \operatorname{Var}(R_i) + n(n-1) \operatorname{Cov}(R_i, R_{i'}) \\ &= n[(N-1) - (n-1)]\frac{N+1}{12} = \frac{nm(N+1)}{12}\end{aligned} \tag{2.11}$$

Each of the related statistics (U_x, W_y and W_x) will have the same variance.

We can apply the normal approximation to the example at hand:

```
In [45]: mu = n*(N+1)//2; mu
Out[45]: 25

In [46]: sigma = np.sqrt(n*m*(N+1)/12.); sigma**2
```

```
Out[46]: 16.666666666666668

In [47]: print(sigma)
4.08248290464

In [48]: z = (Wx - mu)/sigma; z
Out[48]: -1.4696938456699067

In [49]: stats.norm.cdf(z)
Out[49]: 0.070822345147568397
```

Note that since the p-value is $P(W \leq 19|H_0) = P(W \leq 19.5|H_0)$, we could include a continuity correction:

```
In [50]: stats.norm.cdf((Wx + 0.5 - mu)/sigma)
Out[50]: 0.088954797493491222
```

We can compare this approximate p-value of 8.9% to the exact value of 9.5% calculated above.

As in the case of the signed-rank statistic, we have to modify the procedure if it happens that any of the x and/or y values are exactly equal. First, in the calculation of the rank-sum, all the "tied" values are assigned the average value of the ranks spanned by the tie. For example, if the second, third, fourth and fifth-smallest values in the combined list are all the same, we assign them all the rank 3.5. This reduces the variance of the statistic, so that

$$\mathrm{Var}(W_x) = \frac{nm}{12}\left((N+1) - \sum_i \frac{\tau_i^3 - \tau_i}{N(N-1)}\right) \tag{2.12}$$

where τ_i is the number of values included in the ith tie.

2.2 Rank-Sum Intervals

As with the signed-rank statistic, we can use the rank-sum test to produce a confidence interval for the offset between the distributions or populations from which the $\{X_i\}$ and $\{Y_j\}$ are drawn. We can consider a rank-sum test of the null hypothesis

$$H_0 : P(X - \theta > Y) = P(X - \theta < Y) \tag{2.13}$$

and ask for what values of θ this would not be rejected at significance α; that will provide a $1-\alpha$ confidence interval. This is easiest to describe in terms of the Mann-Whitney U, the number of i,j pairs for which

$$x_i - \theta > y_j \tag{2.14}$$

Following the same argument we used in the signed-rank case, the statistic will change every time $\theta = x_i - y_j$ for some i and j, so the procedure says to rank the nm differences $x_i - y_j$ and set an interval using these differences.

Using our example data, supposing we want a 90% confidence interval on the location difference between the x and y populations. We start by finding the 5th percentile of the Mann-Whitney U distribution:

```
In [51]: rstats.qwilcox(0.05,5,4)
Out[51]:
<FloatVector - Python:0x7f9d32ba6cb0 / R:0x53ee648>
[3.000000]

In [52]: rstats.pwilcox(3,5,4)
Out[52]:
<FloatVector - Python:0x7f9d31ee69e0 / R:0x53ee468>
[0.055556]

In [53]: rstats.pwilcox(2,5,4)
Out[53]:
<FloatVector - Python:0x7f9d31ee67e8 / R:0x607a078>
[0.031746]
```

So we see that

$$P(2 < U_x < 18) = 1 - \frac{8}{126} approx 93.7\% \qquad (2.15)$$

We collect and sort the differences $x_i - y_j$, of which there are $nm = 20$, which also makes sense, since the Mann-Whitney statistic can be between 0 and 20:

```
In [54]: diffs = np.sort(xi)[:,None]-np.sort(yi)[
    None,:]; diffs
Out[54]:
array([[-10.99, -12.59, -13.69, -26.69],
       [ -7.48, -9.08, -10.18, -23.18],
       [ -7.27, -8.87, -9.97, -22.97],
       [ -3.74, -5.34, -6.44, -19.44],
       [ 35.8 , 34.2 , 33.1 , 20.1 ]])
```

	12.3	**13.9**	**15.0**	**28.0**
1.31	-10.99	-12.59	-13.69	-26.69
4.82	-7.48	-9.08	-10.18	-23.18
5.03	-7.27	-8.87	-9.97	-22.97
8.56	-3.74	-5.34	-6.44	-19.44
48.1	35.8	34.2	33.1	20.1

```
In [55]: sorteddiffs = np.sort(list(flatten(diffs)))
    ; sorteddiffs
Out[55]:
array([-26.69, -23.18, -22.97, -19.44, -13.69,
    -12.59, -10.99, -10.18,
        -9.97, -9.08, -8.87, -7.48, -7.27, -6.44,
           -5.34, -3.74,
        20.1 , 33.1 , 34.2 , 35.8 ])
```

If $\theta < -26.69$, $u_{x-\theta} = 0$; if $-26.69 < \theta < -23.18$, $u_{x-\theta} = 1$; if $-23.18 < \theta < -22.97 <$, $u_{x-\theta} = 2$. Similarly, if $35.8 < \theta$, $u_{x-\theta} = 20$, if $34.2 < \theta < 35.8$, $u_{x-\theta} = 19$; if $33.1 < \theta < -34.2$, $u_{x-\theta} = 18$. So the range of θ values for which $2 < u_{x-\theta} < 18$ is:

```
In [56]: (sorteddiffs[2],sorteddiffs[-2-1])
Out[56]: (-22.969999999999999, 33.100000000000001)
```

Tuesday 2 October 2018 – Read Section 5.2 of Conover; refer to Section 6.1 of Hollander and Section 3.2 of Higgins

3 The Kruskal-Wallis Test

As a generalization of the two-sample Wilcoxon rank-sum test, suppose we have k samples, and let the ith sample $\{x_{ij}\}$ have size n_i. The total number of data points in all of the samples is $N = \sum_{i=1}^{k} n_i$. We combine and sort all N values, and let the rank of x_{ij} be R_{ij}. If we write the sum of the ranks in the ith sample as $R_i = \sum_{j=1}^{n_i} R_{ij} = n_i\overline{R}_i$, we have k statistics $\{R_i\}$; however they can be described by only $k-1$ quantities, since they obey the constraint $\sum_{i=1}^{k} R_i = \frac{N(N+1)}{2}$. (In the case where $k = 2$, we already saw that the sum of the ranks in the second group, R_2, which we called W_y, was determined by the sum of the ranks in the first group, R_1, which we called W_x, so there was only $k - 1 = 1$ independent statistic.) To convert these $k-1$ independent numbers into a single statistic, we consider the normal approximation that we expect will apply when all of the $\{n_i\}$ are reasonably large.

If we consider the k statistics $\{R_i\}$, they have expectation values

$$E(R_i) = n_i\frac{N+1}{2} \qquad (3.1)$$

variances

$$\mathrm{Var}(R_i) = n_i(N-n_i)\frac{N+1}{12} \qquad (3.2)$$

and covariances

$$\operatorname{Cov}(R_i, R_\ell) = -n_i n_\ell \frac{N+1}{12} \qquad \text{if } i \neq \ell \tag{3.3}$$

The variances and covariances can be summarized into a variance-covariance matrix with elements (for $i = 1, \ldots k$ and $\ell = 1, \ldots k$)

$$\operatorname{Cov}(R_i, R_\ell) = (N\delta_{i\ell} n_i - n_i n_\ell)\frac{N+1}{12} \tag{3.4}$$

where

$$\delta_{i\ell} = \begin{cases} 1 & \text{if } i = \ell \\ 0 & \text{if } i \neq \ell \end{cases} \tag{3.5}$$

is the Kronecker delta (the elements of the identity matrix). We could turn any one of them into a standard normal by shifting and scaling, but what about the others. Recall that if we have n independent standard normal random variables Z_i, then

$$W = \sum_{i=1}^{n} (Z_i)^2 \tag{3.6}$$

is a chi-squared random variable with n degrees of freedom, $W \sim \chi^2(n)$. So if $\{X_i\}$ are independent normal random variables with $E(X_i) = \mu_i$ and $\operatorname{Var}(X_i) = \sigma_i^2$, then

$$W = \sum_{i=1}^{n} \left(\frac{X_i - \mu_i}{\sigma_i^2}\right)^2 \tag{3.7}$$

Now, in this case the random variables are not independent, so this construction doesn't quite work. But there is a somewhat analogous situation; suppose $\{X_i\}$ are a random sample from a normal distribution with variance σ^2. Then if we define

$$Y_i = X_i - \overline{X} = X_i - \frac{1}{n}\sum_{k=1}^{n} X_k \tag{3.8}$$

we have n correlated random variables $\{Y_i\}$ related by one constraint $\sum_{i=1}^{n} Y_i = 0$. One of the results of Student's theorem, which underpins the confidence intervals for mean and variance when both are unknown is that[6]

$$\sum_{i=1}^{n}\left(\frac{Y_i}{\sigma}\right)^2 = \sum_{i=1}^{n}\left(\frac{X_i - \overline{X}}{\sigma}\right)^2 \sim \chi^2(k-1) \tag{3.9}$$

The one constraint among the k $\{Y_i\}$ causes the number of degrees of freedom to be $k-1$ rather than k.

If you go through a similar calculation with the rank-sums $\{R_i\}$, you find[7]

$$T = \frac{12}{N(N+1)}\sum_{i=1}^{k}\frac{1}{n_i}\left(R_i - n_i\frac{N+1}{2}\right)^2 \sim \chi^2(k-1) \tag{3.10}$$

If there are ties, the variances of the various statistics are reduced, and the statistic has to be modified. Of the various equivalent forms, the simplest is probably

$$T = (N-1)\frac{\sum_{i=1}^{k}\left(R_i - n_i\frac{N+1}{2}\right)^2/n_i}{\sum_{i=1}^{k}\sum_{j=1}^{n_i}\left(R_{ij} - \frac{N+1}{2}\right)^2} \sim \chi^2(k-1) \tag{3.11}$$

```
In [1]: from __future__ import division

In [2]: import numpy as np

In [3]: from scipy import stats
```

[6]For a derivation of this result, see e.g., section 6.3 of `https://ccrg.rit.edu/~whelan/courses/2015_3fa_STAT_405/notes03.pdf`.

[7]The key step in the derivation is to write the variance-covariance matrix (3.4) as $\frac{N(N+1)}{12\sqrt{n_i n_\ell}}\left(\delta_{ij} - \frac{\sqrt{n_i n_\ell}}{N}\right)$ and recognize the expression in parentheses as a projection operator with one zero eigenvalue and $k-1$ unit eigenvalues.

```
In [4]: xij = [np.array([ 14.97, 5.80, 25.03, 5.50
   ]),
   ...:       np.array([ 5.83, 13.96, 21.96]),
   ...:       np.array([ 17.89, 23.03, 61.09, 18.62,
       55.51])]

In [5]: ni = np.array([len(x) for x in xij]); ni
Out[5]: array([4, 3, 5])

In [6]: k = len(ni); k
Out[6]: 3

In [7]: N = np.sum(ni); N
Out[7]: 12

In [8]: group = np.concatenate([(i,)*ni[i] for i in
   xrange(k)])

In [9]: xijflat = np.concatenate(xij)

In [10]: idx = np.argsort(xijflat)

In [11]: xijflat[idx]
Out[11]:
array([ 5.5 ,  5.8 ,  5.83, 13.96, 14.97, 17.89,
   18.62, 21.96,
       23.03, 25.03, 55.51, 61.09])

In [12]: group[idx]
Out[12]: array([0, 0, 1, 1, 0, 2, 2, 1, 2, 0, 2, 2])

In [13]: ranks = np.arange(N)+1; ranks
Out[13]: array([ 1, 2, 3, 4,  5,  6,  7,  8,  9, 10,
   11, 12])

In [14]: Rij = [ranks[group[idx]==i] for i in xrange
   (k)]; Rij
Out[14]: [array([ 1, 2, 5, 10]), array([3, 4, 8]),
   array([ 6, 7, 9, 11, 12])]

In [15]: Ri = np.array([np.sum(rij) for rij in Rij])
   ; Ri
Out[15]: array([18, 15, 45])

In [16]: Ri/ni
Out[16]: array([ 4.5, 5. , 9. ])

In [17]: 0.5*(N+1)
Out[17]: 6.5

In [18]: T = 12/(N*(N+1)) * np.sum((Ri-ni*0.5*(N+1))
   **2/ni); T
Out[18]: 4.1538461538461542

In [19]: stats.chi2(df=k-1).sf(T)
Out[19]: 0.12531520484413722
```

Thursday 4 October 2018
– Refer to Section 2.4 of Conover

4 Comparisons of Power and Efficiency

Now that we've considered a number of nonparametric (and parametric) tests, it's a good time to pause and consider some of the metrics we use to compare them. Recall that the significance α of a test is the probability of making a type I error (false alarm) and rejecting H_0, if H_0 is true:

$$\alpha = P(\text{reject } H_0|H_0) \tag{4.1}$$

while the power or efficiency is the probability of rejecting H_0 if the alternative hypothesis H_1 is true:

$$\gamma = P(\text{reject } H_0|H_1) \tag{4.2}$$

For concreteness, suppose that a hypothesis tests constructs a test statistic $T(\mathbf{x})$ from the observed data $\mathbf{x}$ and rejects H_0 if $T(\mathbf{x})$ is greater than or equal to some threshold value c.[8] Then we can write

$$\alpha = P(T(\mathbf{X}) \geq c|H_0) \tag{4.3a}$$
$$\gamma = P(T(\mathbf{X}) \geq c|H_1) \tag{4.3b}$$
$$\tag{4.3c}$$

Written in this form, we can think of the statistic $T(\mathbf{x})$ as defining a family of tests with different values of c; typically, we choose the c that gets us as close as possible to our desired significance α and see what the resulting γ is.

For concreteness, consider two examples:

[8] Basically all tests can be put into this form; e.g., a two-tailed test that rejects H_0 if $z \geq c$ or $z \leq -c$ can be rewritten to use the statistic $|z|$ and reject H_0 if $|z| \geq c$.

- The t-test, designed to test the null hypothesis H_0 that the sampling distribution has zero mean, against a one-sided alternative hypothesis H_1 that the mean of the sampling distribution is positive. The test statistic is

  $$T_t(\mathbf{x}) = \frac{\overline{x}}{\sqrt{s^2/n}} \tag{4.4}$$

  where n is the size of the sample $\mathbf{x} \equiv \{x_i\}$, $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the sample mean, and $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$ is the sample variance.
- The sign test (which is a special case of the quantile test in which $p^* = 0.5$ and $x^* = 0$), designed to test the null hypothesis H_0 that the sampling distribution has zero median, against a one-sided alternative hypothesis H_1 that the median of the sampling distribution is positive. The test statistic is the number of positive values

  $$T_q(\mathbf{x}) = \#_i(x_i > 0) = \sum_{i=1}^{n} I_{x>0}(x_i) \tag{4.5}$$

  where $I_{x>0}(x_i)$ is the **indicator function** which is 1 if $x > 0$ and 0 if $x < 0$.

In the situation of interest for robust inference, both H_0 and H_1 are composite hypotheses. The null hypotheses (H_0) above specify the mean or median of the sampling distribution, but not the form of the distribution itself. Is it normal, or Student-t, or Laplace (double exponential)? (For all of these distributions, the mean and median are the same.) Is it some distribution which violates the assumptions of one of the tests, like the Cauchy distribution (which has a median but whose mean is undefined), or is it some distribution for which the mean and median are not the same, like a gamma distribution? The alternative hypotheses (H_1) likewise don't specify the form of the distribution, but additionally they don't specify the location parameter which H_0 sets to zero. So in addition to the unspecified form, there's a parameter θ, which we'll refer to as the size of the effect, a measure of the amount by which H_1 differs from H_0. We'll assume for simplicity that $\theta = 0$ corresponds to null hypothesis H_0 and that H_1 specifies $\theta > 0$.

We can also see that the definitions of $T(\mathbf{x})$ above depend on the size n of the data sample (which is after all a property of $\mathbf{x}$, but one that we assume we know before we've collected the data). It's also reasonable to assume that the threshold c of interest will be set in a way that depends on n.

So, taking all this into account, the significance α of a test depends on:

- The choice of test statistic $T(\mathbf{x})$
- The form of null sampling distribution
- The choice of threshold c
- The sample size n

The power γ depends on all of that plus the effect size θ.

Now we can consider a comparison between two families of tests with statistics $T_1(\mathbf{x})$ and $T_2(\mathbf{x})$. The comparison is to be done under the assumption of a null sampling distribution, so there will be one comparison assuming a normal sampling distribution, one for Laplace, one for Cauchy, etc. We can write the significance and power of each test as[9]

$$\alpha_1 = \alpha_1(c_1, n_1) \qquad \text{and} \qquad \gamma_1 = \gamma_1(c_1, n_1, \theta) \tag{4.6a}$$

$$\alpha_2 = \alpha_2(c_2, n_2) \qquad \text{and} \qquad \gamma_2 = \gamma_2(c_2, n_2, \theta) \tag{4.6b}$$

In general, if $\alpha_1 = \alpha_2$, $n_1 = n_2$ and $\gamma_1 > \gamma_2$, we say that T_1 gives us a more powerful (or efficient) test than T_2. But this may depend on the form of the sampling distribution and the significance α and sample size n at which the comparison is done.

Note that it doesn't really make sense to compare the thresholds c_1 and c_2, since they're defined for different statistics.

4.1 Power Curves

One comparison we've done on a few homeworks is to fix a value of $n = n_1 = n_2$, and choose c_1 and c_2 to acheive a specified value of $\alpha = \alpha_1 = \alpha_2$ and then plot $\gamma_1(\theta)$ and $\gamma_2(\theta)$ as functions of the effect size θ. For example, on a recent homework, you found, for $n = 100$ and $\alpha = 0.0443$, the following power curves $\gamma_t(\theta)$ and $\gamma_q(\theta)$ for different families of sampling distribution, shown in figure 1.

4.2 ROC Curves

One drawback to comparing tests using power curves is that the comparison has to be done at a specified significance α, which means a specified threshold c_1 or c_2 for each test. An alternative is to consider the families of tests, each of which can be "tuned": lowering the threshold increases the power γ, but it also increases the false alarm probability α. To do this we fix the sample size $n = n_1 = n_2$ and the effect size θ, and consider how, for each test, α and γ change when the threshold c is varied. As noted above, it's not really meaningful to plot e.g., $\gamma_1(c_1)$ and $\gamma_2(c_2)$ on the same set of axes, since the thresholds for the different tests mean different things, and there's nothing special about tests for which $c_1 = c_2$. Instead we want to plot $\gamma_1(c_1)$ versus $\alpha_1(c_1)$ and $\gamma_2(c_2)$ versus $\alpha_2(c_2)$, i.e., we can consider $\gamma_1(c_1)$ and $\alpha_1(c_1)$ as the functions which define a parametrized curve. Such a γ-versus-α plot is known as an ROC curve. The ROC stands for "Receiver Operating Characteristic", and the technique was developed for evaluating radar detection during

[9] We could also allow for different effect sizes θ_1 and θ_2, but all of the comparisons we're interested in will be done at the same effect size θ.

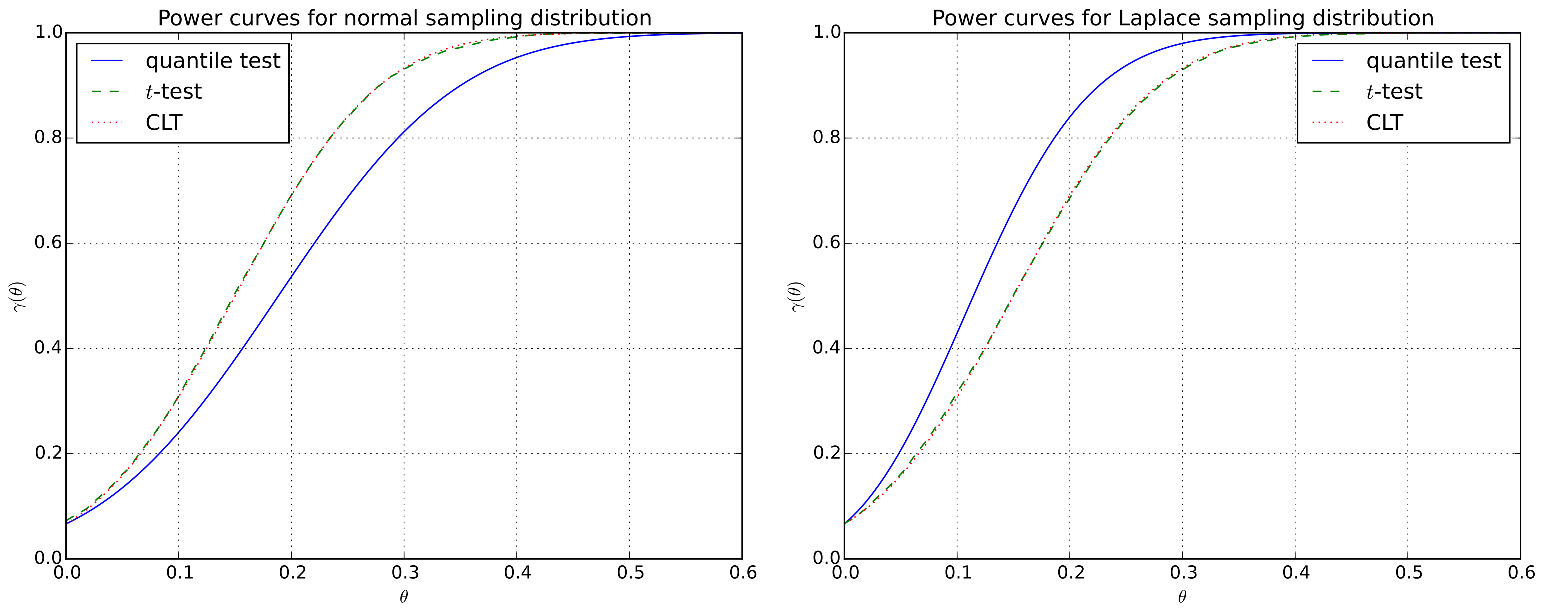


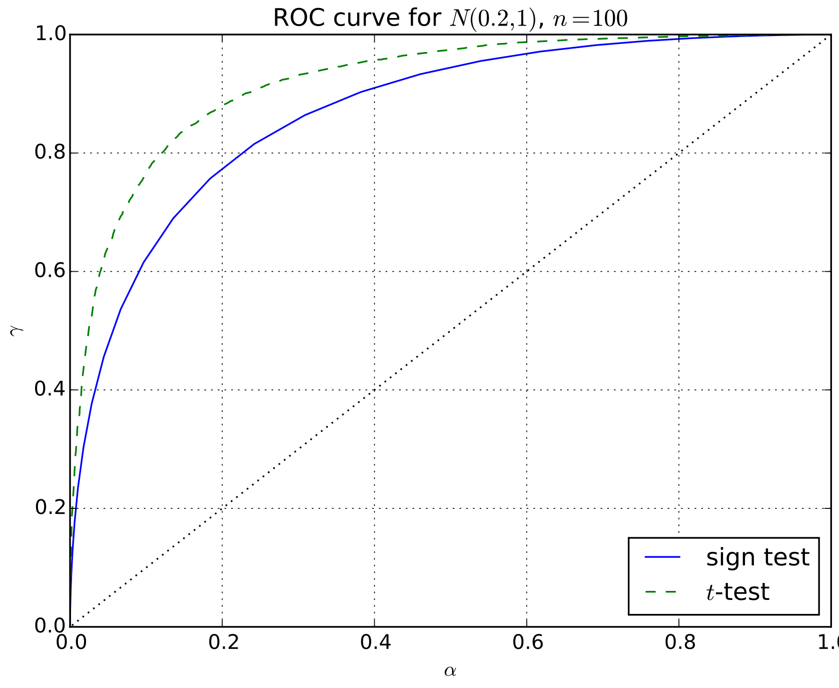
Figure 1: Power curves for one-tailed tests under different sampling distributions. Note that the curve lablled “quantile test” is actually for the sign test.

World War II. It doesn’t generally appear in statistics textbooks, but it’s a useful complement to the other comparative measures we’re considering today.

```
from __future__ import division
import numpy as np
from scipy import stats
n = 100
theta = 0.2
c_q = np.arange(0,n+2)
c_q
p0 = 0.5
p1 = stats.norm(loc=theta).sf(0)
alpha_q = stats.binom(n,p0).sf(c_q-0.5)
gamma_q = stats.binom(n,p1).sf(c_q-0.5)
figure();
plot(alpha_q,gamma_q,'b-',label='sign test');
xlabel(r'$\alpha$');
ylabel(r'$\gamma$');
title(r'ROC curve for $N(%.1f,1)$, $n=%d$' % (theta,n));
xlim(0,1);
ylim(0,1);
grid(True);
plot(alpha_q,alpha_q,'k:');
N = 10**4
x_ji = stats.norm.rvs(size=(N,n))
xbar_j = np.mean(x_ji,axis=-1)
```

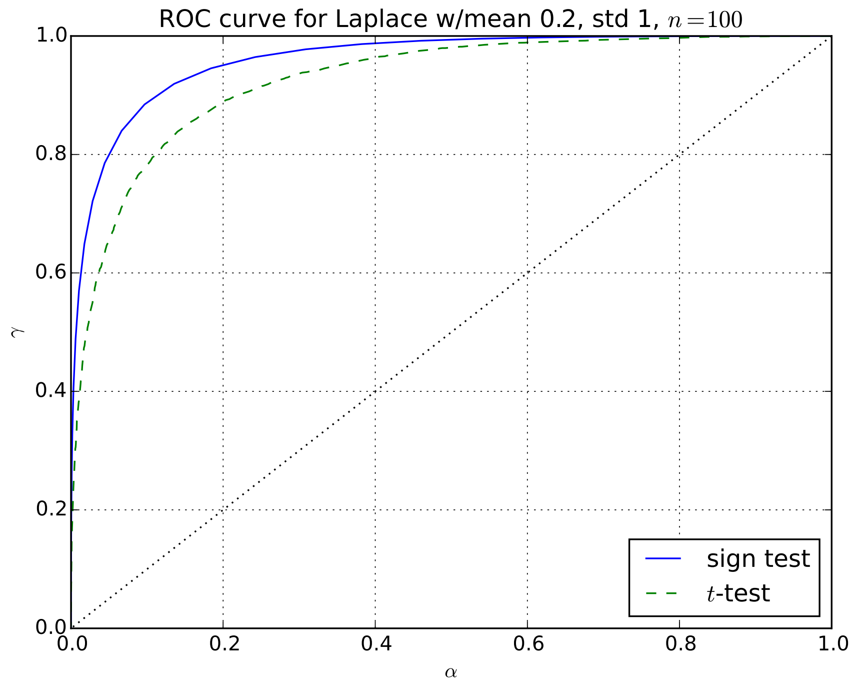
```
s_j = np.std(x_ji,ddof=1,axis=-1)
t_j0 = xbar_j / (s_j/np.sqrt(n))
t_j1 = (xbar_j+theta) / (s_j/np.sqrt(n))
cmin = -3
cmax = 3 + theta * np.sqrt(n)
c_t = np.linspace(cmin,cmax,1000)
alpha_t = np.mean(t_j0[None,:] >= c_t[:,None], axis=-1)
gamma_t = np.mean(t_j1[None,:] >= c_t[:,None], axis=-1)
plot(alpha_t,gamma_t,'g--',label=r'$t$-test');
legend(loc='lower right')
p1 = stats.laplace(loc=theta,scale=np.sqrt(0.5)).sf(0)
alpha_q = stats.binom(n,p0).sf(c_q-0.5)
gamma_q = stats.binom(n,p1).sf(c_q-0.5)
savefig('notes05_roc_gauss.eps',bbox_inches='tight');
```

It's convenient to draw the line $\gamma = \alpha$ on the graph, since any consistent test must lie above and to the right of that line. We also note that a more powerful test will be found above and to the left of a less powerful one.



```
figure();
plot(alpha_q,gamma_q,'b-',label='sign test');
xlabel(r'$\alpha$');
ylabel(r'$\gamma$');
title(r'ROC curve for Laplace w/mean %.1f, std 1, $n=%d$'
      % (theta,n));
xlim(0,1);
ylim(0,1);
grid(True);
plot(alpha_q,alpha_q,'k:')
N = 10**4
x_ji = stats.laplace(scale=np.sqrt(0.5)).rvs(size=(N,n))
xbar_j = np.mean(x_ji,axis=-1)
s_j = np.std(x_ji,ddof=1,axis=-1)
t_j0 = xbar_j / (s_j/np.sqrt(n))
```

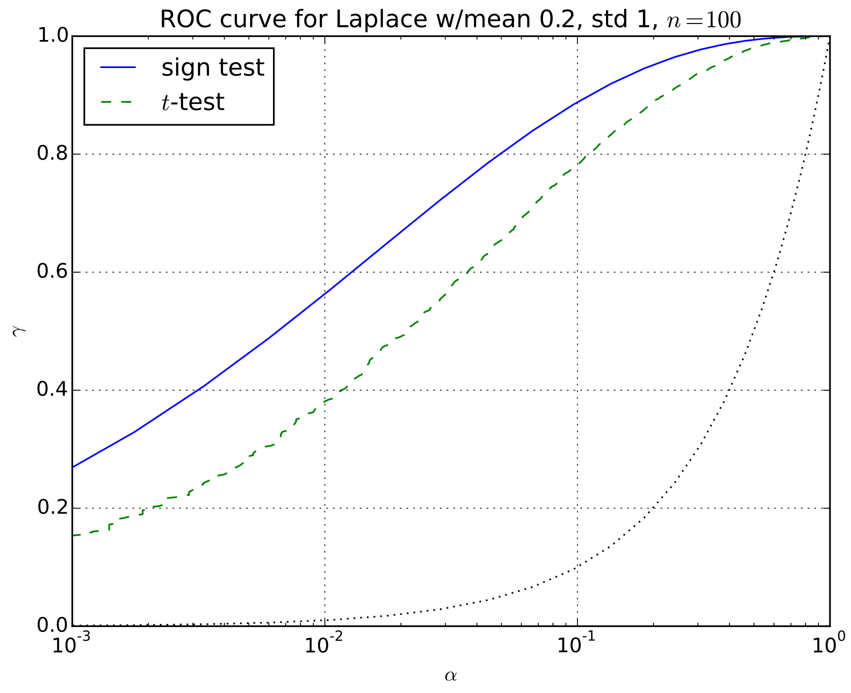
```
t_j1 = (xbar_j+theta) / (s_j/np.sqrt(n))
cmin = -3
cmax = 3 + theta * np.sqrt(n)
c_t = np.linspace(cmin,cmax,1000)
alpha_t = np.mean(t_j0[None,:] >= c_t[:,None], axis=-1)
gamma_t = np.mean(t_j1[None,:] >= c_t[:,None], axis=-1)
plot(alpha_t,gamma_t,'g--',label=r'$t$-test');
legend(loc='lower right')
savefig('notes05_roc_laplace.eps',bbox_inches='tight');
```



We see that, for a normal sampling distribution the t-test has a higher power γ at any given significance α, while for Laplace, the situation is reversed.

Since hypothesis tests get more interesting the lower the false alarm probability α, it's sometimes helpful to do the ROC plot with α on a log scale:

```
figure();
semilogx(alpha_q,gamma_q,'b-',label='sign test');
semilogx(alpha_t,gamma_t,'g--',label=r'$t$-test');
semilogx(alpha_q,alpha_q,'k:');
xlabel(r'$\alpha$');
ylabel(r'$\gamma$');
title(r'ROC curve for Laplace w/mean %.1f, std 1, $n=%d$'
      % (theta,n));
legend(loc='upper left');
xlim(1e-3,1);
ylim(0,1);
grid(True);
savefig('notes05_roc_laplace_log.eps',bbox_inches='tight');
```



4.3 Asymptotic Relative Efficiency

The final measure we'll consider is the one mentioned most frequently in Conover, and defined in Section 2.4. We consider two tests with the same significance $\alpha = \alpha_1 = \alpha_2$ and power $\gamma = \gamma_1 = \gamma_2$. In order to acheive this, the tests must be applied to samples of different sizes $n_1 \neq n_2$. The **relative efficiency** of test 2 relative to test 1 is $\frac{n_1}{n_2}$. (The more efficient test can acheive the same significance and power with a smaller sample size.) In the limit that the sample size becomes large, this is known as the asymptotic relative efficiency (A.R.E.).

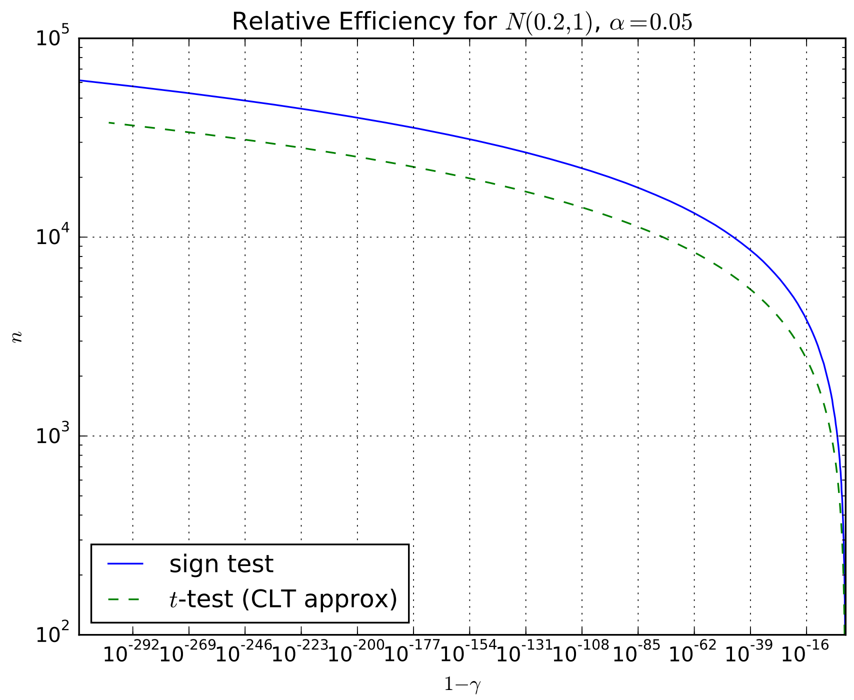
This is sort of tricky to estimate numerically, since you don't know ahead of time what sample size you'll need for each test. One trick we can use is to fix the significance α and effect size θ, and then calculate the power γ for the test at a bunch of different sample sizes n. We can then plot n versus γ and compare the n for different tests at the same γ. As n becomes large, we know γ will get close to 1, so the best approach is actually to calculate the false dismissal probability $\beta = 1 - \gamma$ and then plot β on a logarithmic scale. We also want the scale for n to be logarithmic, since we want to eyeball the ratio of different n at the same γ value, which will correspond to the separation on a log scale. We first plot this for the sign test, where the threshold c_q is the $1-\alpha$ quantile of the $\text{Bin}(n, 0.5)$ distribution (i.e., a different threshold for every n), and the false dismissal probability $\beta = 1-\gamma$ is the probability that a $\text{Bin}(n, p_\theta)$ random variable will fall below that threshold, where p_θ is the fraction of the sampling distribution to the right of the origin when the location parameter is θ. We do this for a $N(\theta, 1)$ sampling distribution:

```
from __future__ import division
import numpy as np
from scipy import stats
alpha = 0.05
theta = 0.2
p0 = 0.5
p1 = stats.norm(loc=theta).sf(0); p1
n = np.array(np.logspace(2,5,100),dtype=int)
c_q = stats.binom(n,p0).isf(alpha)
beta_q = stats.binom(n,p1).cdf(c_q)
figure();
loglog(beta_q,n,'b-',label='sign test');
title(r'Relative Efficiency for $N(%.1f,1)$, $\alpha
   =%g$' % (theta,alpha));
xlabel(r'$1-\gamma$');
ylabel(r'$n$');
grid(True);
```

Now for the t-test, we really don't want to do a Monte Carlo which we'd have to re-do for each sample size n we're trying out. Instead, we rely on the central limit theorem, which tells us that, for large n, the t-statistic is approximately a $N(\theta\sqrt{n}, 1)$ random variable, and

$$P(T \leq c) \approx \Phi\left(c - \theta\sqrt{n}\right) \tag{4.7}$$

```
c_z = stats.norm.isf(alpha)
beta_z = stats.norm(loc=theta*np.sqrt(n)).cdf(c_z)
loglog(beta_z,n,'g--',label=r'$t$-test (CLT approx)');
legend(loc='lower left');
savefig('notes05_re_gauss.eps',bbox_inches='tight');
```



We can see that, as the curves go to large n (in the upper left-hand part of the figure, they seem to have a constant difference between them, and by eyeballing the fact that the sign test curve hits the left edge at around 6×10^4 and the t-test curve would be extrapolated to around 4×10^4, it looks like we might expect the A.R.E. of the t-test relative to the sign test to be 1.5 or so. We also notice that neither of the curves actually makes it up to 10^5 because the β calculation is underflowing, and the binomial cdf in the sign test does so sooner than the normal cdf in the t test. This suggests that we should use some sort of asymptotic approximation, and in fact this is used in the actual computation of the limiting case.

The calculation of A.R.E. is complicated in general, but it's made somewhat simpler if we recall that for large sample sizes most test statistics are approximately normally distributed. So for the sign test, the statistic is

$$Z_q = \frac{N_+ - n/2}{\sqrt{n}/2} \tag{4.8}$$

while for the t test, it is

$$Z_t = \frac{\overline{X}}{\sqrt{S^2/n}} \tag{4.9}$$

under H_0, both are standard normal, and so we reject if the statistic exceeds $z_{1-\alpha}$. If the location parameter is θ under H_1, then $E(N_+) = np_\theta$ and $V(N_+) = np_\theta(1-p_\theta)$, so

$$E(Z_q) = \frac{np_\theta - n/2}{\sqrt{n}/2} = \sqrt{n}(2p_\theta - 1) \tag{4.10a}$$

$$\text{Var}(Z_q) = \frac{4}{n} np_\theta(1-p_\theta) = 4p_\theta(1-p_\theta) \tag{4.10b}$$

and

$$\gamma_q \approx P(Z_q > z_{1-\alpha}) = 1 - \Phi\left(\frac{z_{1-\alpha} - \sqrt{n_q}(2p_\theta - 1)}{\sqrt{4p_\theta(1-p_\theta)}}\right) \tag{4.11}$$

where

$$\Phi(z) = \frac{1}{2\pi}\int_{-\infty}^{z} e^{-t^2/2}\,dt \tag{4.12}$$

is the standard normal cdf. Similarly, if the sampling distribution is $N(\theta, 1)$, then $E(\overline{X}) = \theta$ and $V(\overline{X}) = 1/n$, and

$$E(Z_t) = \theta\sqrt{n} \qquad \text{and} \qquad \text{Var}(Z_t) = 1 \tag{4.13}$$

so

$$\gamma_t \approx P(Z_t > z_{1-\alpha}) = 1 - \Phi\left(z_{1-\alpha} - \theta\sqrt{n_t}\right) \tag{4.14}$$

Now, in order to ensure that $\gamma_q = \gamma_t$, we must have

$$\frac{\sqrt{n_q}(2p_\theta - 1) - z_{1-\alpha}}{\sqrt{4p_\theta(1-p_\theta)}} \approx \theta\sqrt{n_t} - z_{1-\alpha} \tag{4.15}$$

It seems like in general n_q/n_t will depend on both θ and α, and the θ dependence doesn't look like it will go away even when $n \to \infty$. But, since we're supposed to carry out the comparison at fixed α and γ, as we send the sample size to infinity, we have to send the effect size θ to zero, which means $p_\theta \to \frac{1}{2}$. Thus we can (to leading order) set the denominator on the left-hand size to one, which leaves us with

$$\sqrt{n_q}(2p_\theta - 1) - z_{1-\alpha} \approx \theta\sqrt{n_t} - z_{1-\alpha} \tag{4.16}$$

and

$$\frac{n_q}{n_t} = \lim_{\theta\to 0}\left(\frac{\theta}{2p_\theta - 1}\right)^2 \tag{4.17}$$

we can use l'Hôpital's rule to deal with the limit, or just expand the form of p_θ to first order for small θ. If the sampling distribution is normal,

$$\begin{aligned} p_\theta &= \frac{1}{\sqrt{2\pi}}\int_0^\infty e^{-(x-\theta)^2/2}\,dx = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\theta} e^{-z^2/2}\,dz \\ &= \frac{1}{2} + \frac{1}{\sqrt{2\pi}}\int_0^\theta e^{-z^2/2}\,dz \approx \frac{1}{2} + \frac{\theta}{\sqrt{2\pi}}e^{-0^2/2} \end{aligned} \tag{4.18}$$

so

$$\lim_{\theta\to 0}\frac{\theta}{2p_\theta - 1} = \lim_{\theta\to 0}\frac{\theta}{2\theta/\sqrt{2\pi}} = \sqrt{\frac{\pi}{2}} \tag{4.19}$$

and

$$\text{A.R.E.} = \frac{\pi}{2} \approx 1.57 \tag{4.20}$$

which is about the value we eyeballed from the plot.

Tuesday 16 October 2018
– Read Section 5.3 of Conover

5 The Conover Squared-Ranks Test for Equal Variances

So far our rank-based statistical measures have focused on whether two (or more) samples (either independent or paired) were drawn from distributions with the same location parameter (mean or median). Now let's consider the question of the scale parameter, which can be described by the variance or some more general dispersion. We consider a case where we have two independent samples $\{x_i | i = 1, \ldots, n\}$ and $\{y_j | j = 1, \ldots, m\}$. We let the null hypothesis be that these samples were drawn from equally spread out probability distributions (with possibly different centers). The alternative hypothesis can be one-sided (that e.g., the first distribution is more spread out than the second) or two-sided (that the dispersions are simply different).

As a reminder, the parametric test in this situation is the F-test: calculate the sample variance of each sample

$$s_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2 \tag{5.1a}$$

$$s_y^2 = \frac{1}{m-1}\sum_{j=1}^{m}(y_j - \overline{y})^2 \tag{5.1b}$$

where $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\overline{y} = \frac{1}{m}\sum_{j=1}^{m} y_j$, and threshold on values of the statistic

$$F = \frac{s_x^2}{s_y^2} \tag{5.2}$$

To do a rank-based nonparametric test, the basic idea is to take the quantity we're summing or averaging and then replace it with the ranks of that quantity. In this case we write

$$U_i^2 = (x_i - \overline{x})^2 \qquad \text{and} \qquad V_j^2 = (y_j - \overline{y})^2 \tag{5.3}$$

Since the F test considers the relationship of

$$s_x^2 = \frac{1}{n-1}\sum_{i=1}^{n} U_i^2 \tag{5.4a}$$

$$s_y^2 = \frac{1}{m-1}\sum_{j=1}^{m} V_j^2 \tag{5.4b}$$

we should then take all of the $\{U_i^2\}$ and $\{V_j^2\}$, rank them in order, and note the ranks $\{R(U_i^2)\}$ and $\{R(V_j^2)\}$. An equivalent by easier computation is to take the square roots

$$U_i = |x_i - \overline{x}| \qquad \text{and} \qquad V_j = |y_j - \overline{y}| \tag{5.5}$$

and rank those instead. Since the mapping from $U_i \to U_i^2$ is monotonic, it won't change the order, and thus we'll get the exact same set of ranks if we use $\{R(U_i)\}$ and $\{R(V_j)\}$. So the obvious thing to do would be to sum the ranks of the $\{U_i\}$. It turns out that you generally get a more powerful test if you square the ranks before adding them, so the test statistic is

$$T_u = \sum_{i=1}^{n} [R(U_i)]^2 \tag{5.6}$$

It's not really obvious that this is the best thing to do, but it sort of makes sense since parametric measures of variance include a sum of squares. You will investigate on the homework and see empirically that $\sum_{i=1}^{n} [R(U_i)]^2$ tends to lead to a more powerful test statistic than $\sum_{i=1}^{n} R(U_i)$.

Note that, if there are no ties,

$$\sum_{i=1}^{n} [R(U_i)]^2 + \sum_{j=1}^{m} [R(V_j)]^2 = \sum_{r=1}^{N} r^2 = \frac{N(N+1)(2N+1)}{6} \tag{5.7}$$

(The arithmetic identity used is called Faulhaber's formula, and we've actually already used it in calculating the variance of some rank-based statistics.) This means that the sum of the $[R(U_i)]^2$ carries the same information as the sum of the $[R(V_j)]^2$, which is the justification behind using $[R(U_i)]^2$ as the test statistic. If there are ties, we have to assign the average ranks in the usual way. But note that, unlike in the case of the sum of ranks, the sum of squared ranks does not add up to the same constant with and without ties. This is because e.g.,

$$2.5 + 2.5 = 5 = 2 + 3 \tag{5.8}$$

but

$$(2.5)^2 + (2.5)^2 = 6.25 + 6.25 = 12.5 \neq 2^2 + 3^2 = 4 + 9 = 13 \tag{5.9}$$

So in fact if there are ties, the test statistic should be constructed in a way that involves both sets of squared ranks, i.e., T_x as well as

$$T_y = \sum_{j=1}^{m} [R(V_j)]^2 \tag{5.10}$$

As usual, this is done with an eye towards the normal approximation. The expectation value of the T_u statistic is n times the averaged squared rank:

$$E(T_u) = n\overline{R^2} \tag{5.11}$$

where

$$\overline{R^2} = \frac{1}{N}\left(\sum_{i=1}^{n} [R(U_i)]^2 + \sum_{j=1}^{m} [R(V_j)]^2\right) = \frac{T_u + T_v}{N} \tag{5.12}$$

If there are no ties, $\overline{R^2} = \frac{(N+1)(2N+1)}{6}$. If there are ties, $\overline{R^2}$ is determined from the data. You might reasonably complain that $E(T_u)$ should be a fixed number, but like the variance of other rank-based statistics, it depends on the observed ranks, specifically on how many ties there are and where. The "dirty secret" of methods like this is that this is actually a conditional expectation value, conditional upon the full set of N available ranks with the actual observed ties taken into account. I.e., our statements about expectations, significance, power, etc, all pertain to repeating the experiment but only considering repetitions that have this particular set of ranks including ties. This is kind of unsatisfying from a frequentist point of view, but it's necessary to be able to describe the properties of the test in a distribution-free way, since you'd have to know the underlying distribution to know how likely ties are. Note that this is no big deal to a Bayesian, who is always constructing probabilities for unknown quantities conditional upon the data which were actually observed.

In any event, we can see that

$$T_u - n\overline{R^2} = \frac{(N-n)T_u - nT_v}{N} = \frac{mT_u - nT_v}{N} = T_v - m\overline{R^2} \tag{5.13}$$

so a statistic based on $T_u - n\overline{R^2}$ will not only have zero expectation value; it will actually treat the two data samples in a reasonably symmetric way. It's thus not too surprising that the variance of this quantity treats the two samples symmetrically, i.e.,

$$\operatorname{Var}(T_u - n\overline{R^2}) = \frac{nm}{N-1}\left(\overline{R^4} - \left[\overline{R^2}\right]^2\right) \tag{5.14}$$

where

$$\overline{R^4} = \frac{1}{N}\left(\sum_{i=1}^{4}[R(U_i)]^2 + \sum_{j=1}^{m}[R(V_j)]^4\right) \tag{5.15}$$

which is again equal to a constant ($\frac{1}{N}\sum_{r=1}^{N} r^4 = \frac{(N+1)(2N+1)(3N^2+3N-1)}{30}$) if there are no ties. The statistic

$$T_1 = \frac{T_u - n\overline{R^2}}{\sqrt{\frac{nm}{N-1}\left(\overline{R^4} - \left[\overline{R^2}\right]^2\right)}} \tag{5.16}$$

is approximately standard normal (under H_0) for large samples, and can be used to construct hypothesis tests, p-values, etc.

```
from __future__ import division
import numpy as np
from scipy import stats
import itertools
xi = np.array([15.0, 12.3, 18.0, 13.9])
yj = np.array([18.56, 25.03, 48.1, 1.31, 4.82])
n = len(xi); n
m = len(yj); m
N = n+m; N
xbar = np.mean(xi); xbar
ybar = np.mean(yj); ybar
Ui = np.abs(xi - xbar); Ui
Vj = np.abs(yj - ybar); Vj
UVcombo = np.concatenate((Ui,Vj))
xflags = np.concatenate(([True,]*n,[False,]*m))
yflags = np.bitwise_not(xflags)
ranks = stats.rankdata(UVcombo)
Uranks = ranks[xflags]; Uranks
Vranks = ranks[yflags]; Vranks
np.sum(Uranks)
np.sum(Uranks**2)
np.sort([np.sum(np.array(foo)**2) for foo in
   itertools.combinations(range(1,N+1),n)])
np.sort([np.sum(np.array(foo)) for foo in itertools.
   combinations(range(1,N+1),n)])
```

5.1 Extension to k-Sample Case

The squared ranks test can be extended to the case of k independent samples, just as the Kruskal-Wallis test is defined as the k-sample generalization of the Wilcoxon rank-sum test. The test statistic is written in terms of the squared ranks

$$S_i = \sum_{j=1}^{n_i} [R(|x_{ij} - \overline{x}_i|)]^2 \tag{5.17}$$

as

$$\frac{\sum_{i=1}^{k} \frac{S_i^2}{n_i} - N(\overline{R^2})^2}{\sqrt{\frac{1}{N-1}\left(\overline{R^4} - \left[\overline{R^2}\right]^2\right)}} \tag{5.18}$$

and its null distribution is approximately $\chi^2(k-1)$.

Thursday 18 October 2018 – Read Section 5.4 of Conover

6 Measures of Correlation

So far, we've considered inferential problems using independent samples $\{x_i|i = 1,\ldots,n\}$ and $\{y_j|j = 1,\ldots,m\}$ (the Wilcoxon-Mann-Whitney rank sum test and the Conover test for variances, as well as the Kruskal-Wallis test for more than two samples), and paired data $\{(x_i, y_i)|i = 1,\ldots,m\}$ (the sign test and the Wilcoxon signed-rank test). In our paired data analysis so far we were interested in the difference of location parameters, and so the paired data were immediately converted to a set of differences $d_i = y_i - x_i$. In fact, all of the paired methods considered so far can be applied to a single sample $\{x_i\}$ which is just analyzed in the way that the differences $\{d_i\}$ are in the standard paired methods.

Now we're going to consider a question that uses the full set of data $\{(x_i.y_i)\}$ rather than just the differences $y_i - x_i$: how correlated are the data? We are really interested in the correlation of the paired populations or bivariate distribution $f(x, y)$ from which the paired sample is drawn, but of course it's the sample we'll use to address it. Qualitatively the question is: does X tend to be large when Y is large, and X large when Y is small? Recall that if $\mu_X = E(X)$ and $\mu_Y = E(Y)$ are the means of the random variables, $\sigma_X^2 = \text{Var}(X) = E([X-\mu_X]^2) = E([X]^2)-\mu_X^2$ and $\sigma_Y^2 = \text{Var}(Y) = E([Y - \mu_Y]^2) = E([Y]^2) - \mu_Y^2$ are the variances and $\text{Cov}(X, Y) = E([X - \mu_X][Y - \mu_Y]) = E(XY) - \mu_X\mu_Y$ is the covariance, the correlation is defined as

$$\text{Cov}(X, Y) = \frac{\text{Corr}(X, Y)}{\sigma_X\sigma_Y} \tag{6.1}$$

If the random variables X and Y are independent (so that the joint distribution can be written $f(x, y) = f_X(x)f_Y(y)$), there will be zero correlation, but that's not the only way to get $\text{Cov}(X, Y) = 0$. It's not hard to show that $-1 \le \text{Cov}(X, Y) \le 1$.

6.1 Pearson's r

The standard parametric estimate of the correlation is to divide the sample covariance

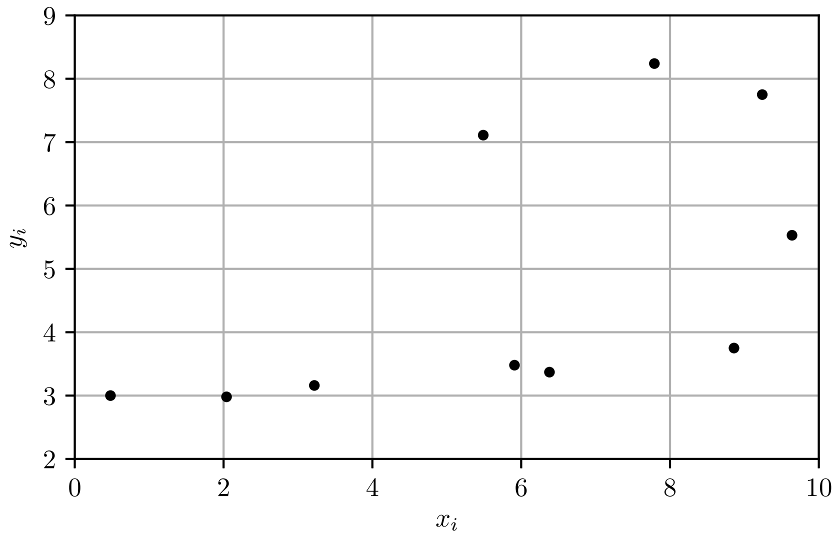
$$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}) \tag{6.2}$$

by the product of the sample standard deviations s_x and s_y; the ratio

$$r = \frac{s_{xy}}{s_x s_y} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}} \tag{6.3}$$

is known as Pearson's correlation coëfficient or "Pearson's r". It's not too hard to see that $-1 \le r \le 1$ as well.

We can compute the correlation coëfficient for a paired data set like this:



```
In [1]: from __future__ import division

In [2]: import numpy as np

In [3]: from scipy import stats

In [4]: x = np.array([9.64, 5.91, 3.22, 2.04, 5.49,
   9.24, 6.38, 7.79, 0.48, 8.86])

In [5]: y = np.array([5.53, 3.48, 3.16, 2.98, 7.11,
   7.75, 3.37, 8.24, 3.00, 3.75])

In [6]: xbar = np.mean(x); xbar
Out[6]: 5.9049999999999994

In [7]: ybar = np.mean(y); ybar
Out[7]: 4.8369999999999997

In [8]: r = np.sum((x-xbar)*(y-ybar))/np.sqrt(np.sum
   ((x-xbar)**2)*np.sum((y-ybar)**2)); r
Out[8]: 0.59010021965957937
```

Note that numpy and scipy actually have functions that calculate the correlation coefficient directly:

```
In [9]: np.corrcoef(x,y)
Out[9]:
array([[ 1.        ,  0.59010022],
       [ 0.59010022,  1.        ]])

In [10]: stats.pearsonr(x,y)
Out[10]: (0.59010021965957937, 0.072524064987892614)
```

6.2 Spearman's ρ

A simple non-parametric measure of correlation is to compute the correlation coëfficient not of the data themselves, but of the ranks of the data. We do the ranking within each of the data samples, i.e., the ranks of the $\{x_i\}$ are $\{R^x_i\}$ and the ranks of the $\{y_i\}$ are $\{R^y_i\}$. Conover calls these $R(X_i)$ and $R(Y_i)$, respectively, but it's important to note they are two different sets of ranks, each from 1 to n, and *not* a single ranking of the $\{x_i\}$ and the $\{y_i\}$ together. Since each set of ranks has the same mean (even if there are ties)

$$\overline{R} = \frac{1}{n}\sum_{i=1}^{n} R^x_i = \frac{1}{n}\sum_{i=1}^{n} R^y_i = \frac{n+1}{2} \qquad (6.4)$$

the rank correlation coëfficient, known as Spearman's ρ, is

$$\rho = \frac{\sum_{i=1}^{n}(R_i^x - \overline{R})(R_i^y - \overline{R})}{\sqrt{\sum_{i=1}^{n}(R_i^x - \overline{R})^2}\sqrt{\sum_{i=1}^{n}(R_i^y - \overline{R})^2}} \tag{6.5}$$

Note that if there are no ties, the sums in the denominator don't depend on the data, e.g.,

$$\begin{aligned}\sum_{i=1}^{n}(R_i^x - \overline{R})^2 &= \sum_{i=1}^{n}(R_i^x)^2 - 2\overline{R}\sum_{i=1}^{n} R_i^x + n\overline{R}^2 = \sum_{r=1}^{n} r^2 - n\overline{R}^2 \\ &= \frac{n(n+1)(2n+1)}{6} - n\frac{(n+1)^2}{4} = \frac{n(n+1)(4n+2-3n-3}{12} \\ &= \frac{n(n+1)(n-1)}{12} = \sum_{i=1}^{n}(R_i^y - \overline{R})^2 \end{aligned} \tag{6.6}$$

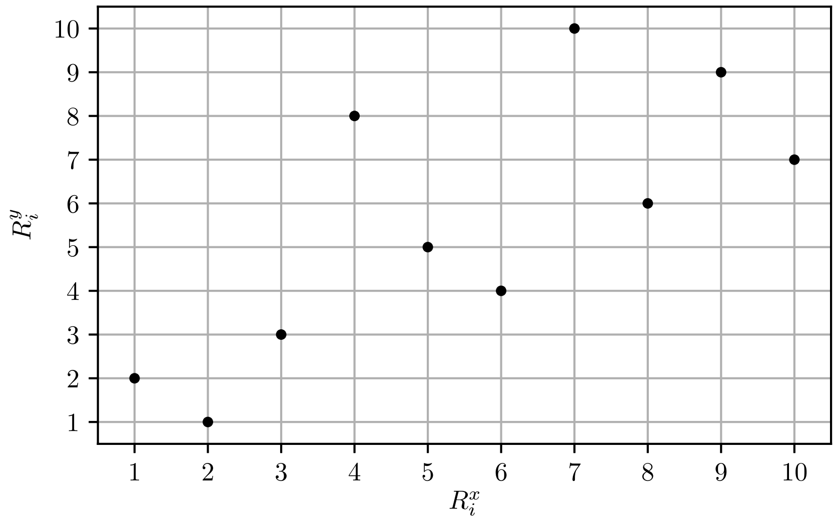
Note that we can also simplify the numerator by noting

$$\begin{aligned}\sum_{i=1}^{n}(R_i^x - R_i^y)^2 &= \sum_{i=1}^{n}([R_i^x - \overline{R}] - [R_i^y - \overline{R}])^2 \\ &= \sum_{i=1}^{n}(R_i^x - \overline{R})^2 + \sum_{i=1}^{n}(R_i^y - \overline{R})^2 - 2\sum_{i=1}^{n}(R_i^x - \overline{R})(R_i^y - \overline{R})\end{aligned} \tag{6.7}$$

so, if there are no ties,

$$\rho = 1 - \frac{6}{n(n^2-1)}\sum_{i=1}^{n}(R_i^x - R_i^y)^2 \qquad \text{(no ties)} \tag{6.8}$$

Spearman's ρ is the correlation coëfficient calculated on the ranks of the data; we can plot the ranks of the same data set as before:



and compute ρ as follows:

```
In [11]: Rx = stats.rankdata(x)

In [12]: Ry = stats.rankdata(y)

In [13]: Rbar = np.mean(Rx); Rbar
Out[13]: 5.5

In [14]: np.mean(Ry)
Out[14]: 5.5

In [15]: rho = np.sum((Rx-Rbar)*(Ry-Rbar))/np.sqrt(
   np.sum((Rx-Rbar)**2)*np.sum((Ry-Rbar)**2)); rho
Out[15]: 0.73333333333333328
```

We can check the simplifying expressions since there are no ties:

```
In [16]: np.sum((Rx-Rbar)**2)
Out[16]: 82.5
```

```
In [17]: np.sum((Ry-Rbar)**2)
Out[17]: 82.5

In [18]: n = len(x)

In [19]: (n*(n**2-1))/12.
Out[19]: 82.5

In [20]: 1 - 6/(n*(n**2-1)) * np.sum((Rx-Ry)**2)
Out[20]: 0.73333333333333339
```

And as before numpy and scipy let us calculate the rank correlation coefficient (but note that the function is called `spearmanr()` and not `spearmanrho()`.

```
In [21]: np.corrcoef(Rx,Ry)
Out[21]:
array([[ 1.        ,  0.73333333],
       [ 0.73333333,  1.        ]])

In [22]: stats.spearmanr(x,y)
Out[22]: (0.73333333333333317, 0.015800596250571581)
```

To perform a hypothesis test, we need the null distribution of ρ. If there's no correlation between the data sets, than the point with $R^x_i = 1$ is equally likely to be paired with any of the n y ranks; the point with $R^x_i = 2$ is equally likely to be paired with any of the $n-1$ y ranks, etc. There are $n!$ different permutations, In the case of this data, there are $10! = 3\,628\,800$.

```
In [23]: from scipy.special import factorial

In [24]: factorial(10)
Out[24]: array(3628800.0)
```

It takes maybe a minute, but python can loop through all of these permutations and give us a null distribution that we can use to calculate p-values.

```
In [26]: import itertools

In [27]: sumsq = np.array([np.sum((ranks-perm)**2)
   ....:  for perm in itertools.permutations(ranks)
      ], dtype=int)

In [28]: nullrho = 1. - 6./(n*(n**2-1.)) * sumsq

In [29]: 2.*np.mean(nullrho >= rho)
Out[29]: 0.020233134920634922
```

For larger sample sizes, we'd like to use a normal approximation. Clearly $E(\rho) = 0$; it turns out that $\text{Var}(\rho) = \frac{1}{n-1}$, which we can verify:

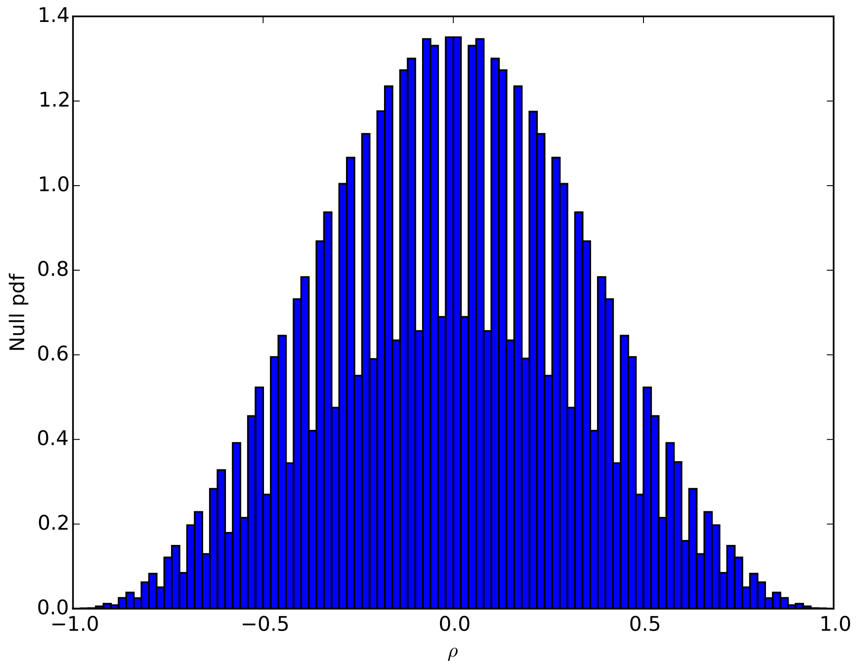
```
In [30]: np.mean(nullrho)
Out[30]: -1.0827317878249146e-16

In [31]: np.std(nullrho)
Out[31]: 0.33333333333333331

In [32]: 1./np.sqrt(n-1)
Out[32]: 0.33333333333333331
```

We can compare the exact two-tailed p-value above to the normal approximation value:

```
In [33]: 2*stats.norm.sf(rho*np.sqrt(n-1))
Out[33]: 0.027806895026997232
```

It's not a great approximation 2.8% versus 2.0%, but it's okay. Note that both of these are different from the 1.6% returned by scipy. In any event, we know that the disribution can't quite be normal because the probability for ρ to be above 1 or below -1 is zero. We can look at how normal things seem with a histogram:

```
In [34]: hist(nullrho,bins=100,normed=True);

In [35]: xlabel(r'$\rho$');

In [36]: ylabel('Null pdf')
Out[36]: <matplotlib.text.Text at 0x7f52d56ba850>

In [37]: savefig('notes05_rhohist.eps',bbox_inches='tight');
```



We see that there is also some discretization effect!

6.3 Kendall's τ

A measure of correlation or association coming from a different starting point is the so-called Kendall τ. This is defined by considering all of the $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs of data points, and counting how many of them are "concordant" and how many are "discordant". We say that:

- (x_i, y_i) and (x_j, y_j) are **concordant** if $x_i > x_j$ and $y_i > y_j$, *or* $x_i < x_j$ and $y_i < y_j$
- (x_i, y_i) and (x_j, y_j) are **disccordant** if $x_i > x_j$ and $y_i < y_j$, *or* $x_i < x_j$ and $y_i > y_j$
- (x_i, y_i) and (x_j, y_j) are neither concordant or discordant if $x_i = x_j$ and $y_i = y_j$.

The ideas of concordant and discordant pairs are easy to grasp if we visualize them on a scatter plot of the data, as shown in figure 2. Although the classification into concordant and discordant pairs is not explicitly written in terms of the ranks, we can see that, since only depends on the ordering of the $\{x_i\}$ and the ordering of the $\{y_i\}$, it can be determined from the ranks alone, as shown in figure 3. The statistic is constructed from N_c, the number of concordant pairs, and N_d, the number of discordant pairs. If there are no ties,

$$N_c + N_d = \frac{n(n-1)}{2} \tag{6.9}$$

and we can define the Kendall tau statistic

$$\tau = \frac{N_c - N_d}{N_c + N_d} = \frac{2(N_c - N_d)}{n(n-1)} \qquad \text{no ties} \tag{6.10}$$

where the denominator was chosen so that $-1 \le \tau \le 1$.

There are actually a few different prescriptions to handle ties. Conover advocates considering any pair with $x_i \neq x_j$ and $y_i = y_j$

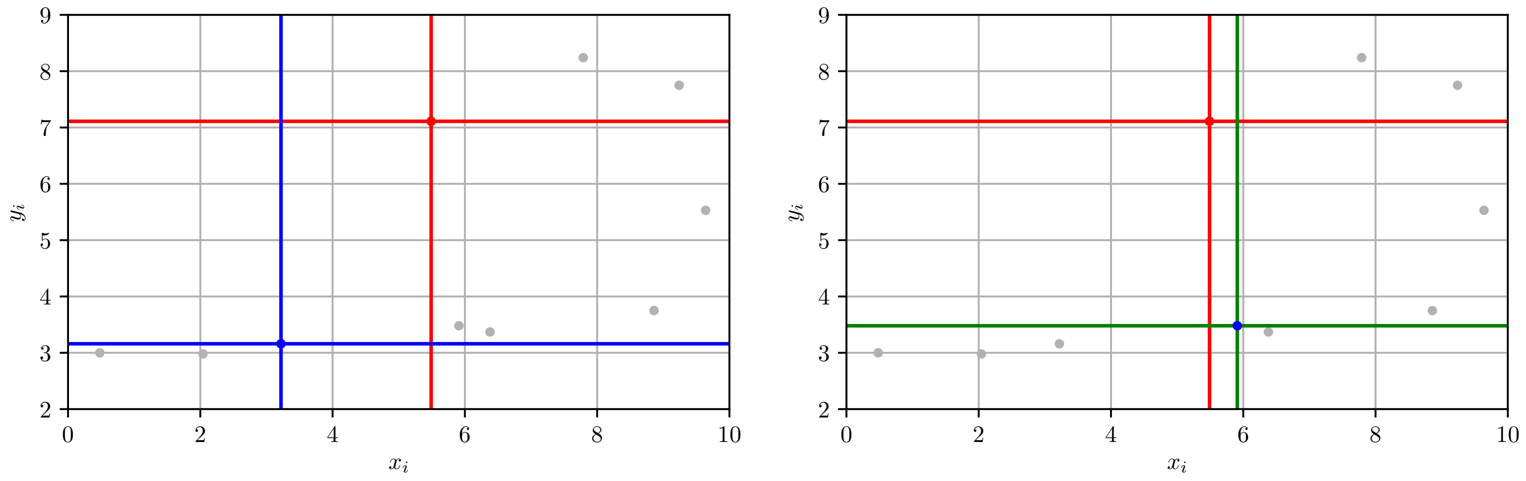


Figure 2: Concordant (left) and discordant (right) pairs. A concordant pair is one where the pair of data points is "northeast" and "southwest" of each other, as you'd expect for positively correlated populations. A discordant pair is one where the pair of data points is "northwest" and "southeast" of each other, as you'd expect for negatively correlated populations.

to be hald-concordant and half-discordant, and thus contributed $\frac{1}{2}$ to N_c and $\frac{1}{2}$ to N_d. This won't change the numerator of τ, since these contributions cancel out of $N_c - N_d$, but it will change the denominator.

We can count the number of concordant and discordant pairs, and compute τ, for the data above. We pause for a moment to consider how to automatically identify concordant and discordant pairs. Consider the first two points:

```
In [38]: x[:2],y[:2]
Out[38]: (array([ 9.64,  5.91]), array([ 5.53,  3.48]))

In [39]: (x[0]>x[1]),(y[0]>y[1])
Out[39]: (True, True)
```

We see that $x_1 > x_2$ and $y_1 > y_2$, so this is a conccordant pair. We'd like to combine the information in such a way that two trues or two falses will flag the pair as concordant. We can accomplish this with the logical "exclusive or", which is true if exactly one of the two statements is true, but false if they are both true or both false. So we want "NOT ($x_i > x_j$ XOR $y_i > y_j$)" for concordant and "NOT ($x_i > x_j$ XOR $y_i < y_j$)" for discordant. (This will actually still have trouble with ties, since it will mark a pair with $x_i = x_j$ and $y_i = y_j$ as both concordant and disccordant.)

```
In [40]: True^True
Out[40]: False

In [41]: not(True^True)
Out[41]: True
```

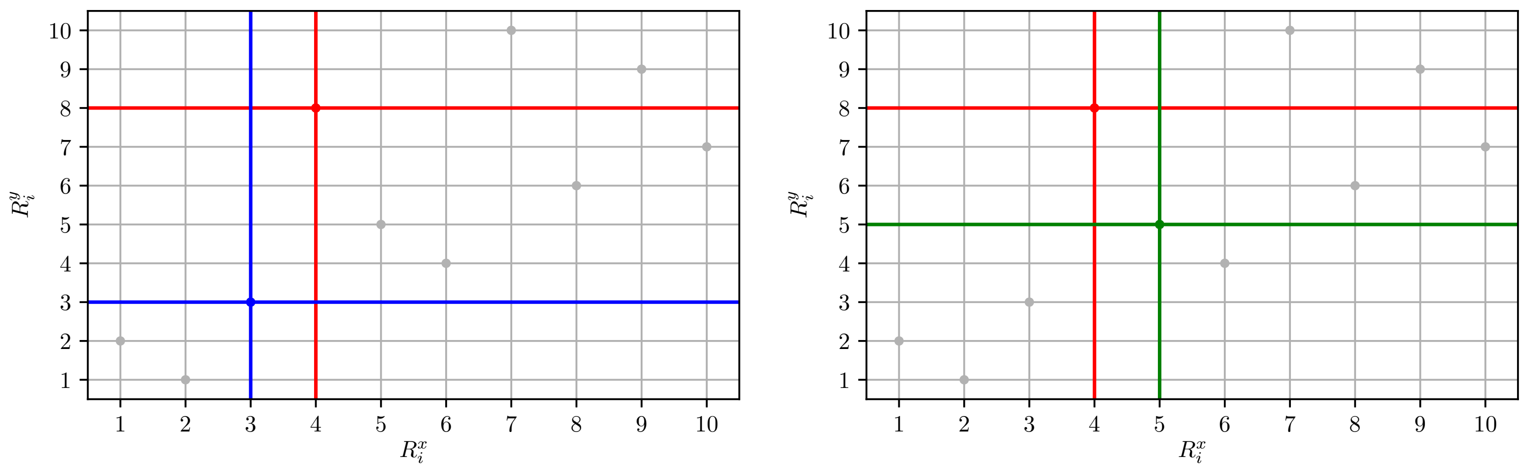


Figure 3: The concordant (left) and discordant (right) pairs from figure 2, shown in terms of ranks.

```
In [42]: not(True^False)
Out[42]: False

In [43]: not(False^True)
Out[43]: False

In [44]: not(False^False)
Out[44]: True

In [45]: not((x[0]>x[1])^(y[0]>y[1]))
Out[45]: True
```

We sum these up for all the pairs, and find $N_c = 35$ and $N_d = 10$:

```
In [46]: Nc = np.sum([not((x[i]>x[j])^(y[i]>y[j]))
   for (i,j) in itertools.combinations(xrange(n),2)
   ]); Nc
Out[46]: 35
```

```
In [47]: Nd = np.sum([not((x[i]>x[j])^(y[i]<y[j]))
   for (i,j) in itertools.combinations(xrange(n),2)
   ]); Nd
Out[47]: 10

In [48]: np.sum([not((Rx[i]>Rx[j])^(Ry[i]>Ry[j]))
   ....:  for (i,j) in itertools.combinations(xrange
      (n),2)])
Out[48]: 35

In [49]: np.sum([not((Rx[i]>Rx[j])^(Ry[i]<Ry[j]))
   ....:  for (i,j) in itertools.combinations(xrange
      (n),2)])
Out[49]: 10
```

From these, we compute $\tau = \frac{35-10}{45} = \frac{5}{9}$:

```
In [50]: tau = (Nc-Nd)/(Nc+Nd); tau
```

```
Out[50]: 0.55555555555555558
```

And of course there's a scipy function which does it for us and estimates a two-sided p-value of 2.53%:

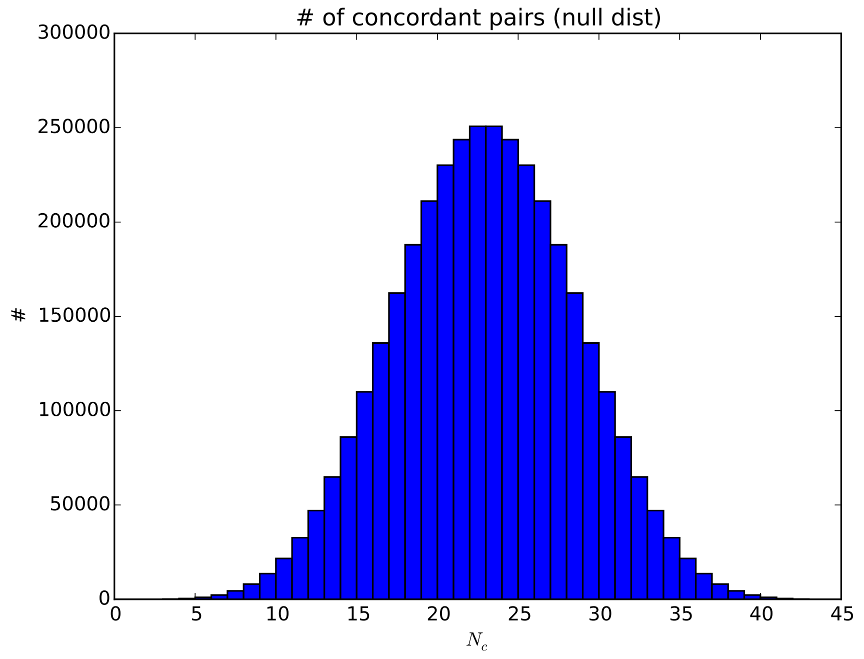
```
In [51]: stats.kendalltau(x,y)
Out[51]: (0.55555555555555569, 0.025347322573947558)
```

Again, to get the null distribution, we assume any of the 10! rank pairings is equally likely. This takes a little longer to run in python, but it's still bearable:

```
In [52]: Ncdist = np.array([np.sum([not((ranks[i]>
   ranks[j])^(perm[i]>perm[j])) for (i,j) in
   itertools.combinations(xrange(n),2)]) for perm in
    itertools.permutations(ranks)])

In [53]: Npairs = n*(n-1)//2; Npairs
Out[53]: 45

In [54]: Nddist = Npairs - Ncdist
```

We can plot a histogram of this distribution:

```
In [55]: figure();

In [56]: hist(Ncdist,bins=np.arange(Npairs+1))
Out[56]:
(array([ 1.00000000e+00, 9.00000000e+00, 4.40000000e
   +01,
         1.55000000e+02, 4.40000000e+02, 1.06800000e
            +03,
         2.29800000e+03, 4.48900000e+03, 8.09500000e
            +03,
         1.36400000e+04, 2.16700000e+04, 3.26830000e
            +04,
         4.70430000e+04, 6.48890000e+04, 8.60540000e
            +04,
         1.10010000e+05, 1.35853000e+05, 1.62337000e
            +05,
         1.87959000e+05, 2.11089000e+05, 2.30131000e
            +05,
         2.43694000e+05, 2.50749000e+05, 2.50749000e
            +05,
         2.43694000e+05, 2.30131000e+05, 2.11089000e
            +05,
         1.87959000e+05, 1.62337000e+05, 1.35853000e
            +05,
         1.10010000e+05, 8.60540000e+04, 6.48890000e
            +04,
         4.70430000e+04, 3.26830000e+04, 2.16700000e
            +04,
         1.36400000e+04, 8.09500000e+03, 4.48900000e
            +03,
         2.29800000e+03, 1.06800000e+03, 4.40000000e
            +02,
         1.55000000e+02, 4.40000000e+01, 1.00000000e
            +01]),
 array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11,
    12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
           28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
           45]),
 <a list of 45 Patch objects>)

In [57]: xlabel(r'$N_c$');

In [58]: ylabel('#');
In [59]: title('# of concordant pairs (null dist)')
Out[59]: <matplotlib.text.Text at 0x7f058a875e50>

In [60]: savefig('notes05_tauhist.eps',bbox_inches='
   tight');
```



and from this we get the null distribution of τ statistic:

```
In [61]: taudist = (Ncdist-Nddist)/Npairs

In [62]: min(taudist),max(taudist)
Out[62]: (-1.0, 1.0)
```

It should be apparent that the null distribution has $E(\tau) = 0$. The variance can be computed theoretically to be $\operatorname{Var}(\tau) = \frac{2(2n+5)}{9n(n-1)}$, which we can verify for $n = 10$:

```
In [63]: np.mean(taudist)
Out[63]: 3.0577147711722124e-17

In [64]: np.var(taudist)
Out[64]: 0.061728395061728412

In [65]: 2*(2*n+5)/(9*n*(n-1))
Out[65]: 0.06172839506172839

In [66]: sigtau = np.sqrt(2*(2*n+5)/(9*n*(n-1))); sigtau
Out[66]: 0.24845199749997662
```

Now we can compare the exact p-value with the value from the normal distribution:

```
In [67]: 2*np.mean(Ncdist>=Nc)
Out[67]: 0.028609457671957671

In [68]: 2*np.mean(taudist>=tau)
Out[68]: 0.028609457671957671

In [69]: 2*stats.norm.sf(tau/sigtau)
Out[69]: 0.025347318677468252
```

Note that the p-value reported by `stats.kendalltau()` above is the 2.53% from the normal approximation; the exact p-value is 2.86%.

Thursday 18 October 2018
– Read Section 5.8 of Conover

7 The Friedman and Quade Tests

7.1 The Complete Block Design

So far in our study of rank-based tests we've considered both paired and independent samples. We can categorize them as:

- Independent samples
  - Two independent samples: Wilcoxon rank sum (Mann-Whitney), Conover squared ranks
  - k independent samples: Kruskal-Wallis (generalization of rank sum), Conover squared ranks
- Paired samples
  - Two paired samples: Sign test, Wilcoxon signed rank (also correlation coëfficients)

We now consider the extension of the paired-samples case to the case where instead of pairs of observations we have a set of k observations. Each set of observations is called a *block*, and we refer to the number of blocks as b (this was called n in the paired-samples case). The k observations in the block are sometimes referred to as *treatments*, a nomenclature which comes from an experimental design where you have b groups of k subjects each. The subjects within the "block" are assumed to be identical, but the different blocks may not be. One subject from each block receives each of the k treatments. This is known as a *randomized complete block design*. (The "randomized" part is because it's randomly selected which of the k members of each block receives which treatment.) The output is a $b \times k$ matrix of observations:

$$\{X_{ij}\} = \begin{pmatrix} X_{11} & X_{12} & \cdots & X_{1k} \\ X_{21} & X_{22} & \cdots & X_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ X_{b1} & X_{b2} & \cdots & X_{bk} \end{pmatrix} \tag{7.1}$$

The null hypothesis is that, within a block, each of the treatments is as likely to give a larger or smaller result than another, $P(X_{ij} > X_{i\ell} = P(X_{ij} < X_{i\ell})$. We define $\{R_{ij}|j = 1, \dots, k\}$ to be the ranks of the responses to the k treatments within block i, and let

$$R_j = \sum_{i=1}^{b} R_{ij} \tag{7.2}$$

be the sum of ranks for treatment j. Note that the minimum possible value for R_j is b and the maximum is kb. To give a specific example:

```
In [1]: from __future__ import division

In [2]: import numpy as np

In [3]: from scipy import stats

In [4]: Xij = np.array([[  2.  ,  19.86,   9.17],
   ...:                 [  1.05,   3.1 ,   3.34],
   ...:                 [  0.14,  25.4 ,  26.59],
   ...:                 [ 14.6 ,   3.93,  10.95]])

In [5]: b,k = np.shape(Xij)

In [6]: Rij = stats.mstats.rankdata(Xij,axis=-1); Rij
Out[6]:
```

```
array([[ 1.,  3.,  2.],
       [ 1.,  2.,  3.],
       [ 1.,  2.,  3.],
       [ 3.,  1.,  2.]])

In [7]: Rj = np.sum(Rij,axis=0); Rj
Out[7]: array([  6.,   8.,  10.])
```

The sums of the ranks for each treatment are

$$R_1 = 1+1+1+3 = 6 \tag{7.3a}$$
$$R_2 = 3+2+2+1 = 8 \tag{7.3b}$$
$$R_3 = 2+3+3+2 = 10 \tag{7.3c}$$

7.2 The Friedman Test

The simplest test observes that since, under the null hypothesis, $E(R_{ij}) = \frac{k+1}{2}$ and, if there are no ties, $\operatorname{Var}(R_{ij}) = \frac{k(k+1)}{12}$, and since the ranks of a given treatment within different blocks can be treated as independent random variables, we have $E(R_j) = \frac{b(k+1)}{2}$ and, if there are no ties, $\operatorname{Var}(R_j) = \frac{bk(k+1)}{12}$. This means that

$$T_1 = \frac{12}{bk(k+1)} \sum_{j=1}^{k} \left(R_j - \frac{b(k+1)}{2} \right)^2 \tag{7.4}$$

should be approximately chi-squared distributed with $k-1$ degrees of freedom, under the null hypothesis. (The number of degrees of freedom is $k-1$ because of the constraint that $\sum_{j=1}^{k} R_j = \frac{bk(k+1)}{2}$.) This test is known as the Friedman test, after economist Milton Friedman.

We can evaluate this for the data above, and get the following

```
In [8]: T1 = (12/(b*k*(k+1)))*np.sum((Rj-0.5*b*(k+1))**2); T1
Out[8]: 2.0

In [9]: stats.chi2(df=k-1).sf(T1)
Out[9]: 0.36787944117144233

In [10]: stats.friedmanchisquare(Xij[:,0],Xij[:,1],Xij[:,2]
Out[10]: (2.0, 0.36787944117144233)
```

but note that the normal approximation that leads to the approximate chi-squared distribution is not very good for this small sample size. Conover asserts that we get a better approximation with the transformed statistic[10]

$$T_2 = \frac{(b-1)T_1}{b(k-1) - T_1} \tag{7.5}$$

which should have an F distribution with degree-of-freedom parameters $k-1$ and $(b-1)(k-1)$.

```
In [11]: T2 = ((b-1)*T1)/(b*(k-1)-T1); T2
Out[11]: 1.0

In [12]: stats.f(k-1,(b-1)*(k-1)).sf(T2)
Out[12]: 0.421875
```

An interesting exercise is to work out the exact distribution for these statistics for this case. In general there are $(k!)^b$ different arrangements of ranks possible with no ties; in this case that is $6^4 = 1296$.

Note that, although we use ranks, this is actually the k-sample analogy of the sign test. The rankings within a block correspond to the sign of $y_i - x_i$, which we could rename as $X_{i2} - X_{i1}$, which is equivalent to the ordering of X_{i1} and X_{i2}.

[10]This statistic comes from applying two-way ANOVA to the ranks.

7.3 The Quade Test

To get an analog of the Wilcoxon signed rank statistic[11], we need the equivalent of the magnitude of the difference. The obvious choice is the spread of the values X_{ij} within block i, which we write as

$$M_i = \max_j X_{ij} - \min_j X_{ij} \tag{7.6}$$

The ranks of these are called Q_i, and the equivalent of the signed ranks are then

$$S_{ij} = Q_i \left(R_{ij} - \frac{k+1}{2} \right) \tag{7.7}$$

Q_i are the ranks and the quantity in parentheses is the generalization of the sign of the difference. The statistic is then constructed out of the sums of these,

$$S_j = \sum_{i=1}^{b} S_{ij} \tag{7.8}$$

and the test statistic is

$$T_3 = \frac{(b-1)\frac{1}{b}\sum_{j=1}^{k} S_j^2}{\sum_{i=1}^{b}\sum_{j=1}^{k} S_{ij}^2 - \frac{1}{b}\sum_{j=1}^{k} S_j^2} \tag{7.9}$$

which is again supposed to be $F(k-1,(b-1)(k-1))$-distributed. It seems as though the statistic depends on more than just the $\{S_j\}$ due to the first term in the denominator, but that is only true if there are ties. If there are no ties, $\sum_{i=1}^{b}\sum_{j=1}^{k} S_{ij}^2$ has a fixed, if somewhat complicated, value in terms of b and k.

We can calculate this for the data above:

```
In [13]: Mi = np.max(Xij,axis=-1)-np.min(Xij,axis=-1); Mi
Out[13]: array([ 17.86,   2.29,  26.45,  10.67])

In [14]: Qi = stats.rankdata(Mi); Qi
Out[14]: array([ 3.,  1.,  4.,  2.])

In [15]: Sij = Qi[:,None]*(Rij-0.5*(k+1)); Sij
Out[15]:
array([[-3.,  3.,  0.],
       [-1.,  0.,  1.],
       [-4.,  0.,  4.],
       [ 2., -2.,  0.]])

In [16]: Sj = np.sum(Sij,axis=0); Sj
Out[16]: array([-6.,  1.,  5.])

In [17]: B = np.sum(Sj**2)/b; B
Out[17]: 15.5

In [18]: A2 = np.sum(Sij**2); A2
Out[18]: 60.0

In [19]: b*(b+1)*(2*b+1)*k*(k**2-1)/72
Out[19]: 60.0

In [20]: T3 = (b-1)*B/(A2-B); T3
Out[20]: 1.0449438202247192

In [21]: stats.f(k-1,(b-1)*(k-1)).sf(T3)
Out[21]: 0.40796817129629637
```

[11]This is known as the Quade test: Quade, *Journal of the American Statistical Association*, **74**, 680. https://www.jstor.org/stable/2286991