Mann-Kendall for autocorrelated data

Ken Butler

July 19, 2015

Introduction

The Mann-Kendall test is a commonly-used nonparametric test for time trend. However, the standard P-values obtained from it are based on an assumption of independence between observations (since the theory is that of the Kendall correlation). However, observations in time series are often autocorrelated: knowing that one observation is larger than the mean may tell you that the next observation is larger than the mean also. Simulation studies reveal that a positive autocorrelation makes it too easy to claim a significant trend (ie. more than 5% of the time if there is actually no time trend), and a negative autocorrelation makes it too difficult to find a significant trend. Since most series in practice exhibit positive autocorrelation (if any), this makes it important to check the autocorrelation in a given series and to adjust the test if necessary.

Adjustments

There are two main classes of adjustment:

1. Obtain the true variance of the Mann-Kendall correlation under the autocorrelation structure displayed by the data, or at least obtain a serviceable approximation to it. This is done by Hamed and Rao (1998).
2. Obtain the null distribution of the Mann-Kendall correlation by bootstrap. This is exemplified in the documentation for the `MannKendall` function in the `Kendall` package. See McLeod (2011).

Hamed and Rao

Introduction

The Mann-Kendall correlation is found by counting the number of "concordant observations", where the later-in-time observation has a larger value for the series, and subtracting the number of "discordant observations", where the later-in-time observation has a *smaller* value for the series. This is done for all $\binom{n}{2}$ pairs of observations, where n is the number of observations in the series. The total difference is denoted S. Tied observations in the series contribute nothing to S. To create a correlation with value between 1 and -1, S is divided by $\binom{n}{2}$.

A test for trend is carried out by dividing S by its standard deviation (under the null hypothesis of no trend) and comparing the result with a normal table. (Under the null hypothesis, S has mean zero whether or not autocorrelation is present.)

When the observations are independent, the variance of S can be shown to be $n(n-1)(2n+5)/18$. When there is autocorrelation, however, the variance of S depends on the true autocorrelation structure, which is unknown in practice. Hamed and Rao (1998) obtain an approximation. First, the Theil-Sen slope is calculated and predicted values subtracted off the data (this is an attempt to make the series stationary). Then, the sample autocorrelations of the ranks of the resulting data are calculated. These are then used in calculating an approximate variance of S, which is larger if the series is positively autocorrelated (and thus makes it more difficult to obtain a significance result by chance than if the unadjusted variance is used).

Specifically, an "effective sample size" n_S^* is obtained that allows for the autocorrelation. When the data are positively autocorrelated, n/n_S^* is greater than 1, and when the data are negatively autocorrelated, n/n_S^* is less than 1. (If there is no autocorrelation, the ratio is equal to 1, and no adjustment is made.) The variance of S is multiplied by the factor n/n_S^* in the test for significance of S.

The `fume` package

This adjustment is incorporated in the `fume` package. Its `mkTrend` function carries out the adjusted and unadjusted Mann-Kendall tests for trend, allowing the user to compare the results. As a bonus, the Theil-Sen slope is also given in the results.

This package is, however, no longer on CRAN, so the standard `install.packages` will not work. First, visit the CRAN archive at https://cran.r-project.org/src/contrib/Archive/fume/. You will see one file `fume_1.0.tar.gz` dating from 2012. Download that file to somewhere you can find it. I stored it in the `Downloads` folder in my account on Linux. If you are running Windows, start with a `c:` and continue with *forward* slashes.

```
install.packages("/home/ken/Downloads/fume_1.0.tar.gz",repos=NULL)
```

```
## Installing package into '/home/ken/R/i686-pc-linux-gnu-library/3.2'
## (as 'lib' is unspecified)
```

The `repos=NULL` means "use the file I give you, rather than trying to download it from anywhere". Now you can use the package `fume` as usual:

```
library(fume)
```

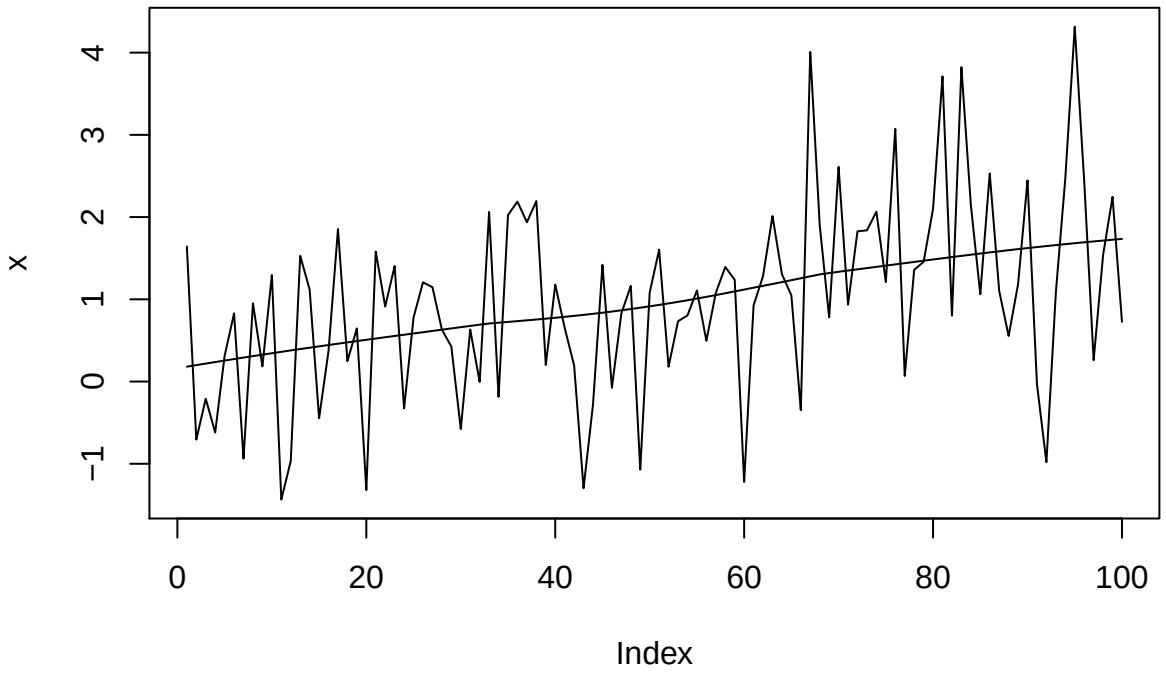
```
## Loading required package: chron
```

Examples

I'll generate some random data under a couple of different conditions, so we can predict how it will work. First, some random data, to which I'll add a trend:

Example 1: uncorrelated

```
set.seed(457299)
x=rnorm(100)
x=x+0.02*(1:100)
plot(x,type="l")
lines(lowess(x))
```



This demonstrates an upward trend (though admittedly with a lot of noise). I just generated independent random normals, so the values in `x` are *not* autocorrelated. Thus, the adjusted and unadjusted Mann-Kendall tests should be about the same:

```
mkTrend(x)
```

```
## $Z
## [1] 4.172342
##
## $p.value
## [1] 3.014852e-05
##
## $Zc
## [1] 4.62224
##
## $`Corrected p.value`
## [1] 3.796178e-06
##
## $tau
## [1] 0.2832323
##
## $`N/N*s`
## [1] 0.8148068
##
## $`Sen's Slope`
## [1] 0.01730554
```

The corrected and uncorrected z values are close, and so are their P-values. The correction factor n/n_S^* is fairly close to 1. The correction doesn't make much of a difference.

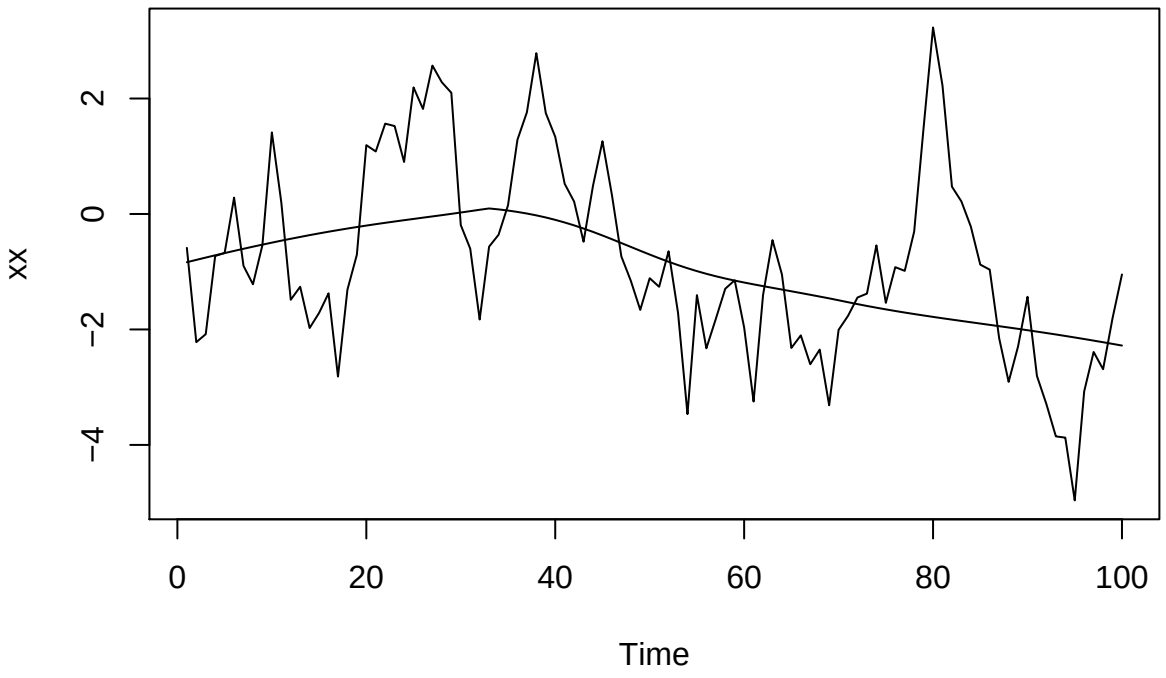
Example 2: autocorrelated

Now, I will generate a random autocorrelated series. This is most easily done with the `forecast` package:

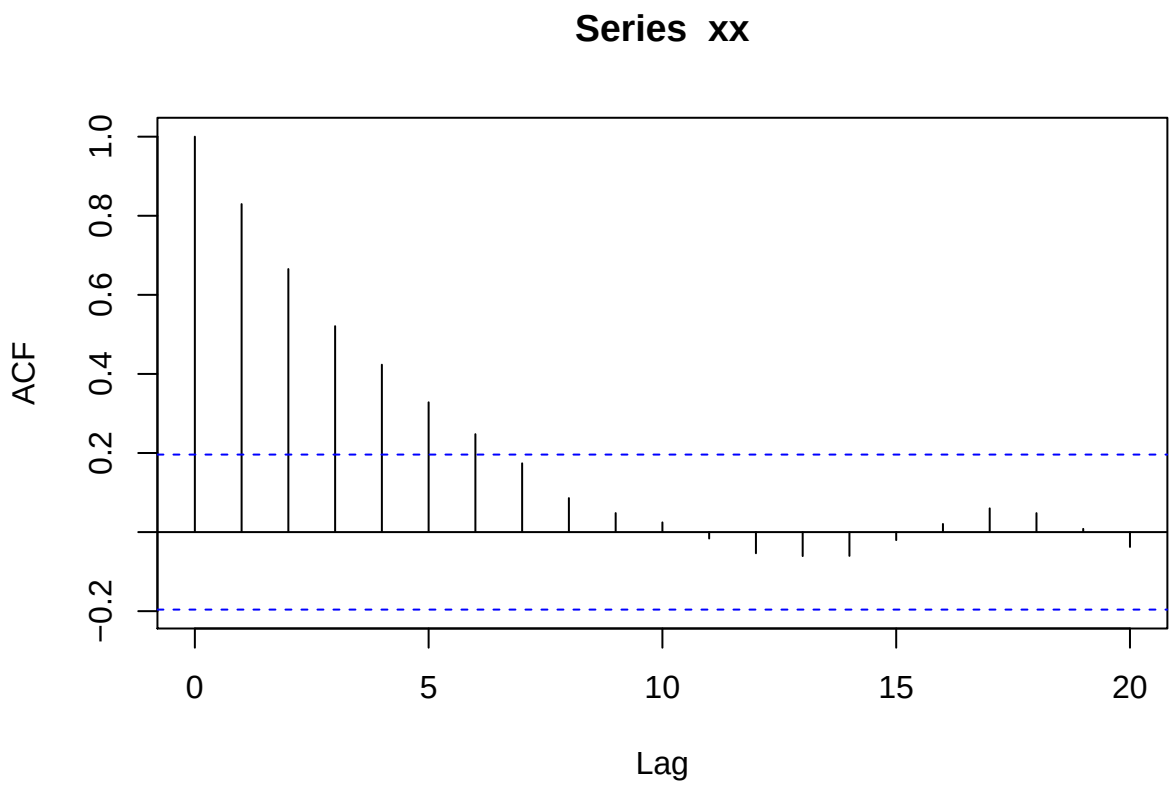
```
library(forecast)
```

```
## Loading required package: zoo
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
## Loading required package: timeDate
## This is forecast 6.1
```

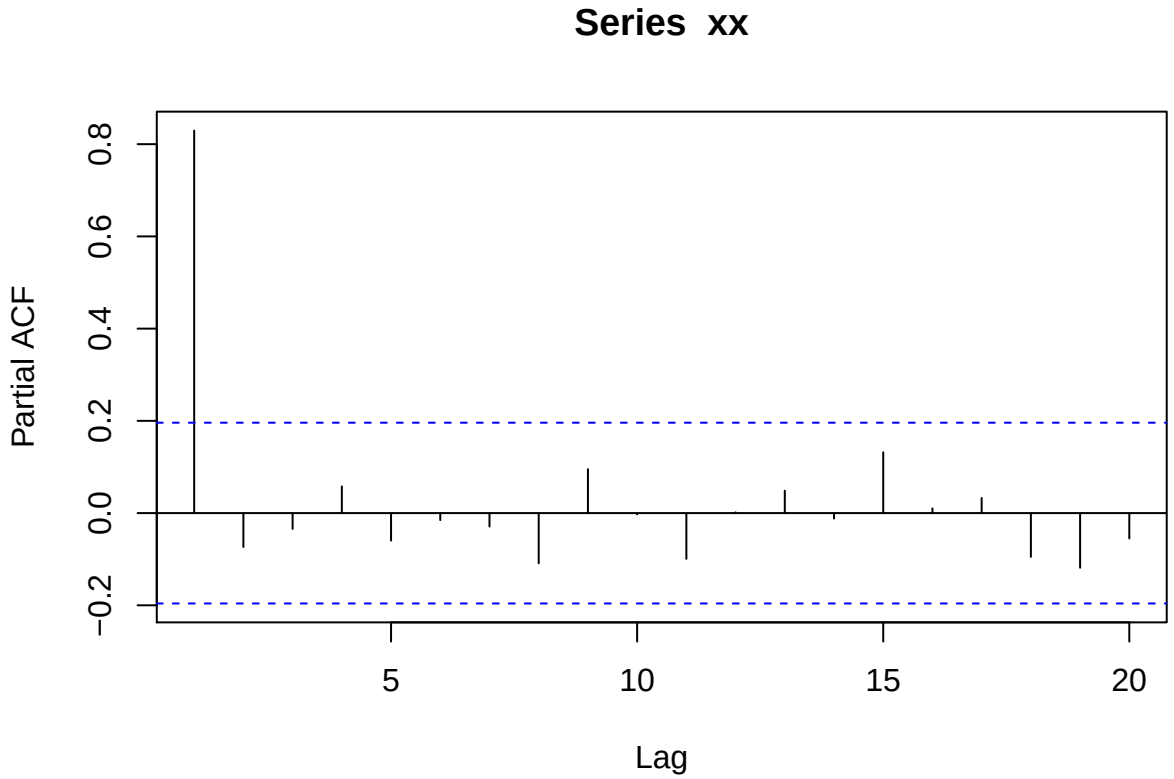
```
set.seed(457298)
xx=arima.sim(list(ar=0.8),100)
plot(xx)
lines(lowess(xx))
```




```
acf(xx)
```


Series xx
ACF
1.0
0.8
0.6
0.4
0.2
−0.2
0
5
10
15
20
Lag

```
pacf(xx)
```


Series xx
Partial ACF
0.8
0.6
0.4
0.2
0.0
−0.2
5
10
15
20
Lag

```
y=as.vector(xx)
mkTrend(y)
```

```
## $Z
## [1] -3.975786
##
## $p.value
## [1] 7.014721e-05
##
## $Zc
## [1] -2.131444
##
## $`Corrected p.value`
## [1] 0.03305259
##
## $tau
## [1] -0.269899
##
## $`N/N*s`
## [1] 3.47935
##
## $`Sen's Slope`
## [1] -0.02347097
```

The output from `arima.sim` is an R time series, which can be plotted. The time plot shows a tendency to stay high or stay low, which is the positive autocorrelation (a series of independent values would cross the lowess curve more). This being an AR(1) time series, its autocorrelation function decays and its partial autocorrelation function dies off after lag 1, as shown. The function `mkTrend` requires an actual vector of numbers rather than an R time series, so here we have to create what we need (usually we won't have to).

Looking at the Mann-Kendall test, it appears to be strongly significant before adjustment, but the corrected P-value is much less dramatic, only about 0.03. The adjusted sample size is about 3.5 times smaller than the actual sample size, indicating that the test is considerably weakened by the autocorrelation. The data were generated to have no trend at all, so the apparent trend seen here is in fact *all* due to the autocorrelation. This goes to show that we can be easily deceived.

Bootstrapping

Introduction

When we are unsure about the assumptions underlying a test, bootstrapping provides a way of obtaining a sampling distribution for a statistic that is based on "sampling from the data". The simplest form of bootstrap samples without replacement from the *sample*. As long as the sample is representative of the population (that is, as long as we have a probability sample that is reasonably large), this will give a fair impression of how the statistic might vary in the population.

To illustrate ideas, let's investigate the bootstrapped sampling distribution of the sample mean based on this sample:

```
my_sample=c(1:8,100)
my_sample
```

```
## [1]   1   2   3   4   5   6   7   8 100
```

```
xbar=mean(my_sample)
n=length(my_sample)
se=sd(my_sample)/sqrt(n)
se
```

```
## [1] 10.63856
```

which is admittedly not exactly a "reasonably large" sample, but we proceed regardless.

The usual way of proceeding is to invoke the Central Limit Theorem: as long as the sample is reasonably large (ahem) and the population is not too non-normal (ahem) with mean μ and SD σ, the sampling distribution will be approximately normal with mean μ, and SD $\sigma/\sqrt{n}$. We then use that to obtain a confidence interval for the population mean, or test whether the mean is equal to a specific value. A 95% CI uses a t^* value from the t distribution, since the population SD is not known:

```
t_star=qt(0.975,n-1)
t_star
```

```
## [1] 2.306004
```

```
xbar+c(-1,1)*t_star*se
```

```
## [1] -9.421458 39.643680
```

Testing whether the mean is 5 against a two-sided alternative goes this way:

```
test_stat=(xbar-5)/se
p_value=2*(1-pt(test_stat,n-1))
p_value
```

```
## [1] 0.369724
```

The details are that the observed sample mean is in the upper tail (greater than the null mean of 5), so we find the probability of being above the test statistic (if the null is true) and then double it, because the test is two-sided.

Obtaining a bootstrap sampling distribution

We don't really trust these because of the outlier in the data. One alternative would be to use the median and obtain our test and confidence interval from the sign test. Another would be to stick with the mean, but obtain a rather more realistic sampling distribution from a bootstrap. To do *that*, we use the function `boot` from package `boot`.

The first thing we have to set up is the statistic that we're going to bootstrap. We have to write a function that takes two things: the data to calculate from, and which elements of the data to calculate it from. (This is because the bootstrap repeatedly samples from the data by randomly selecting elements of it.) This is easier than it sounds:

```
bmean=function(x,v) {
  mean(x[v])
}
```

Here, `x` is the vector of data, and `v` is a vector of elements to use. Let's verify that it works:

```
bmean(c(10,11,13),1:2)
```

```
## [1] 10.5
```

```
bmean(c(10,11,13),c(1,3))
```

```
## [1] 11.5
```

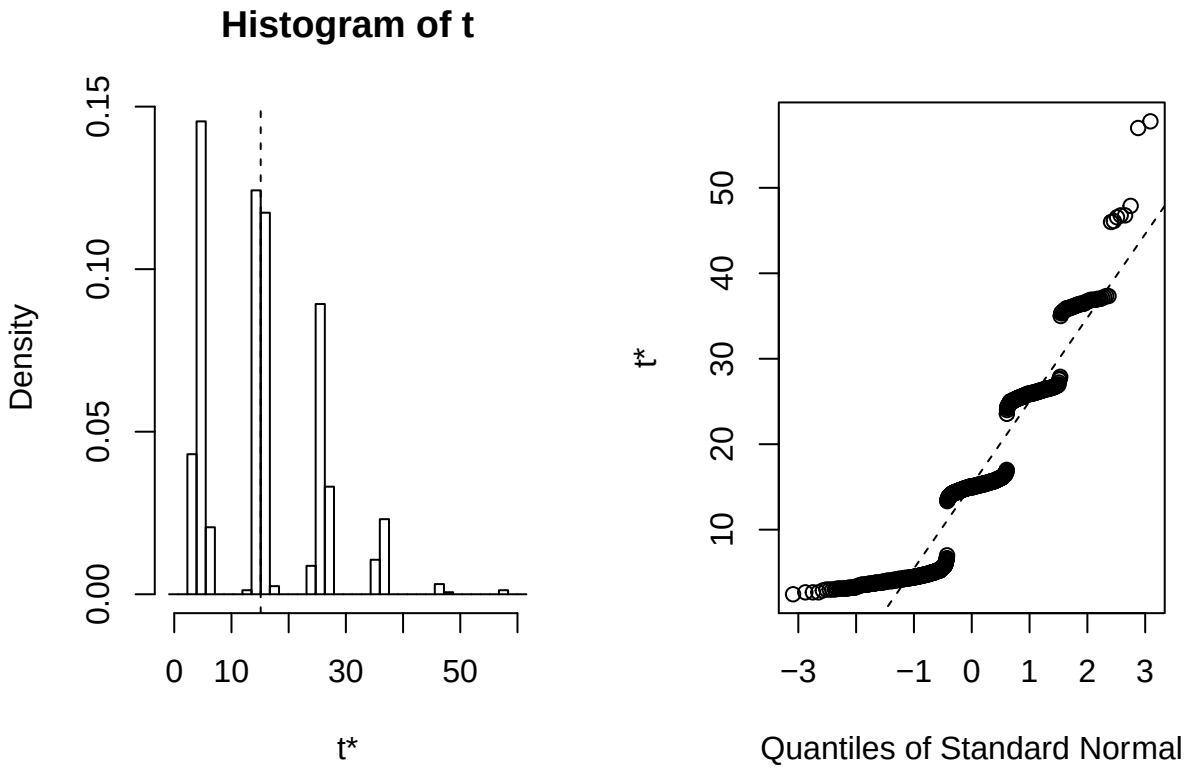
In this case, the data are 10, 11 and 13, and we are first finding the mean of the first two elements, and then the mean of the 1st and 3rd. This checks out.

Now we can run the bootstrap. The function `boot` needs by default three things: the data, a function to calculate the statistic (like my `bmean`), and the number of bootstrap simulations to run:

```
set.seed(457299)
library(boot)
my_boot=boot(my_sample,bmean,1000)
my_boot
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = my_sample, statistic = bmean, R = 1000)
##
##
## Bootstrap Statistics :
##     original    bias    std. error
## t1* 15.11111 0.1218889    9.785295
```

```
samp_dist=my_boot$t
plot(my_boot)
```



According to the theory, this should be "approximately normal". It is far from it. The normal quantile plot suggests a great deal of discreteness (the near-horizontal patches). The bootstrap itself, though, has run all right: the bias is small, and the observed sample mean, shown on the histogram by a dashed line, is in the middle of the bootstrapped sampling distribution.

What is happening is that the sample mean depends critically on the number of times that very large value 100 appears in the bootstrap sample. There were 9 values in the original sample, so the bootstrapped sample contains a number of 100s and the rest of the values are about 5. Thus if k 100s appear in a bootstrap sample and $9-k$ "about-5s", the mean should be about $(100k+5(9-k))/9 = 10.5k+5$: that is, the larger bootstrap means will be about 15.5, 26, 36.5, 47 and so on. This is what is causing the patterns seen above.

Confidence intervals

The easiest way to obtain a bootstrap confidence interval is to take the middle 95% (or whatever) of the bootstrapped sampling distribution:

```
quantile(samp_dist,probs=c(0.025,0.975))
```

```
##      2.5%     97.5%
##  3.444444 36.447222
```

This is the same as the "percentile" interval below (except that R does some interpolation to get more exact percentiles). Another way to get a confidence interval is called the "basic bootstrap"; see Shalizi (2011), page 8, for derivation. Note that the quantiles are *the other way around*:

```
2*my_boot$t0-quantile(samp_dist,probs=c(0.975,0.025))
```

```
##    97.5%     2.5%
## -6.22500 26.77778
```

The function `boot.ci` produces confidence intervals. There are five commonly-used ways of getting a confidence interval from a bootstrap sample; see ("Bootstrapping (Statistics) - Wikipedia, the Free Encyclopedia"), "Methods for bootstrap confidence intervals". By default `boot.ci` produces all of them (except for the "studentized interval" in this case). The input to this is the output from `boot`:

```
library(boot)
boot.ci(my_boot)
```

```
## Warning in boot.ci(my_boot): bootstrap variances needed for studentized
## intervals

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = my_boot)
##
## Intervals :
## Level      Normal              Basic
## 95%   (-4.19, 34.17 )   (-6.33, 26.78 )
##
## Level     Percentile            BCa
## 95%   ( 3.44, 36.55 )   ( 3.89, 50.20 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

This gives us the four remaining intervals. They agree with each other only to a certain extent: they extend from somewhere near zero to somewhere roughly about 40. The interval from the t-distribution went down to -9, and none of these go down as far as that.

Hypothesis testing

Bootstrapping most easily gives us confidence intervals. But there is a link between a confidence interval and a hypothesis test: if the confidence interval for a parameter does not contain the null-hypothesis value for the parameter, that value is rejected at the corresponding α, on a two-sided test. If the interval *does* contain the null-hypothesis value, the null hypothesis is *not* rejected. The argument is "does the confidence interval say that the null-hypothesis value is plausible?"

For example, for our data, suppose we were interested in testing the null hypothesis that the population mean was 5, against the alternative that it was not 5 (two-sided). All of the confidence intervals contain 5, so a mean of 5 should not be rejected (at the 0.05 level, since these were 95% confidence intervals). To get a better handle on the P-value, we could try 90% CI's:

```
boot.ci(my_boot,conf=0.90)
```

```
## Warning in boot.ci(my_boot, conf = 0.9): bootstrap variances needed for
## studentized intervals

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
```

```
## boot.ci(boot.out = my_boot, conf = 0.9)
##
## Intervals :
## Level      Normal              Basic
## 90%   (-1.11, 31.08 )   (-5.67, 26.44 )
##
## Level     Percentile            BCa
## 90%   ( 3.78, 35.89 )   ( 4.13, 37.00 )
## Calculations and Intervals on Original Scale
```

5 is within all of these, so the P-value is greater than 0.10.

Or try 60% ones, just to see what is happening:

```
boot.ci(my_boot,conf=0.60)
```

```
## Warning in boot.ci(my_boot, conf = 0.6): bootstrap variances needed for
## studentized intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = my_boot, conf = 0.6)
##
## Intervals :
## Level      Normal              Basic
## 60%   ( 6.75, 23.22 )   ( 4.78, 25.56 )
##
## Level     Percentile            BCa
## 60%   ( 4.67, 25.44 )   ( 5.00, 25.98 )
## Calculations and Intervals on Original Scale
```

Now we are getting to the point where 5 is on the edge of these intervals, so the P-value is somewhere around 0.40. This is surprisingly similar to the P-value from the t-test, 0.37.

In practice, you would find something like 95% and 99% intervals, and from those determine whether the P-value was greater than 0.05, less than 0.01 or somewhere in between.

Bootstrapping time series

When we have independent observations, we can take random samples of them without causing any problems. But if we have a time series of possibly dependent observations, taking a random sample of them will *destroy* any autocorrelation that exists. This is a problem, since the bootstrapped sample has to show the same kind of dependence as the original data.

A workaround for this is the "block bootstrap". This randomly samples *blocks* of l observations, where you have to choose how big l is. In McLeod (2011), a block size of 5 was chosen. I have no sense of what a good block size might be. Enough blocks are chosen to produce a bootstrap time series that is about as long as the original time series. The tradeoff is that you want the blocks to be long enough that you reproduce the autocorrelation structure of the original series, but not so long that a bootstrap sample of your time series is basically the whole series. In the latter case, you would have no sense of how the Mann-Kendall correlation might vary, which is the point of the exercise.

The workhorse function for bootstrapping time series is called `tsboot`. This is rather like `boot` in that it requires some data, a function to calculate the statistic of interest (the Mann-Kendall correlation) and a number of bootstrap samples. In addition, we need to specify that we want a fixed block size and how big that block size should be. (Other possibilities include using a random block size with a mean that we specify.)

The function to calculate the statistic of interest is simpler than before, because this time the input is just a vector of data. We can piggyback on the functions in `Kendall`: `MannKendall` calculates several things, of which one, `tau`, is the actual correlation. Borrowing heavily from McLeod (2011):

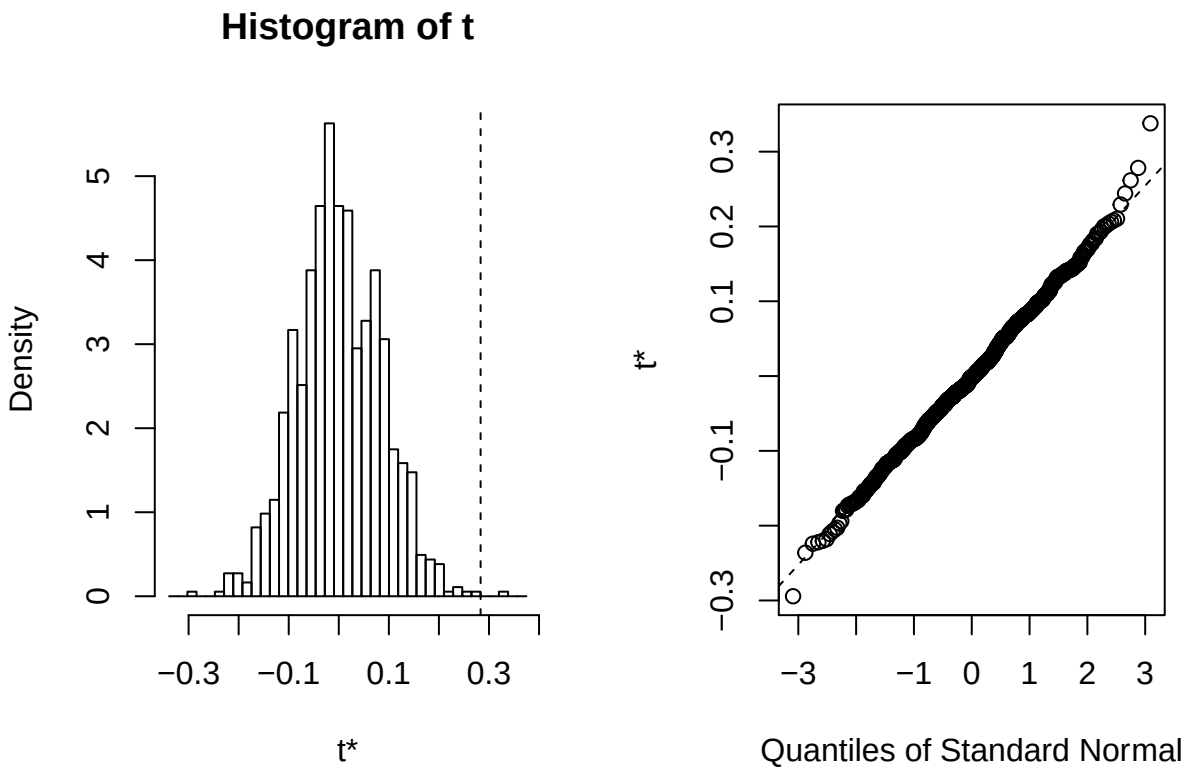
```
library(Kendall)
MKtau=function(x) {
  MannKendall(x)$tau
}
```

Now we can do the actual bootstrap. For data, we'll use the uncorrelated series `x` with trend, and the AR(1) series `y` that was generated with no trend. `x` first:

```
x_boot=tsboot(x,MKtau,1000,sim="fixed",l=5)
x_boot
```

```
##
## BLOCK BOOTSTRAP FOR TIME SERIES
##
## Fixed Block Length of 5
##
## Call:
## tsboot(tseries = x, statistic = MKtau, R = 1000, l = 5, sim = "fixed")
##
##
## Bootstrap Statistics :
##      original     bias    std. error
## t1* 0.2832323 -0.2821183  0.08446833
```

```
plot(x_boot)
```




```
boot.ci(x_boot)
```

```
## Warning in boot.ci(x_boot): bootstrap variances needed for studentized
## intervals

## Warning in boot.ci(x_boot): BCa intervals not defined for time series
## bootstraps

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = x_boot)
##
## Intervals :
## Level      Normal              Basic              Percentile
## 95%   ( 0.3998,  0.7309 )   ( 0.4003,  0.7321 )   (-0.1656,  0.1662 )
## Calculations and Intervals on Original Scale
```

This is worrying. The bias of the bootstrap is almost the same size as the Mann-Kendall correlation calculated from the data. This is because the mean Mann-Kendall correlation of the bootstrap samples was basically zero (seen in the histogram), when we would have expected about 0.28. This is why the confidence intervals (at the bottom) come out so different. A small consolation is that the sampling distribution is very close to normal.

I think the problem is that the series is not stationary, because the mean changes over time. The block bootstrap allows the autocorrelation to show up in the bootstrapped series, but it doesn't allow for a change in mean over time. We should probably try to make our series stationary by subtracting off the Theil-Sen-estimated mean, and *then* try to bootstrap it. The help for `tsboot` shows examples of bootstrapping residuals

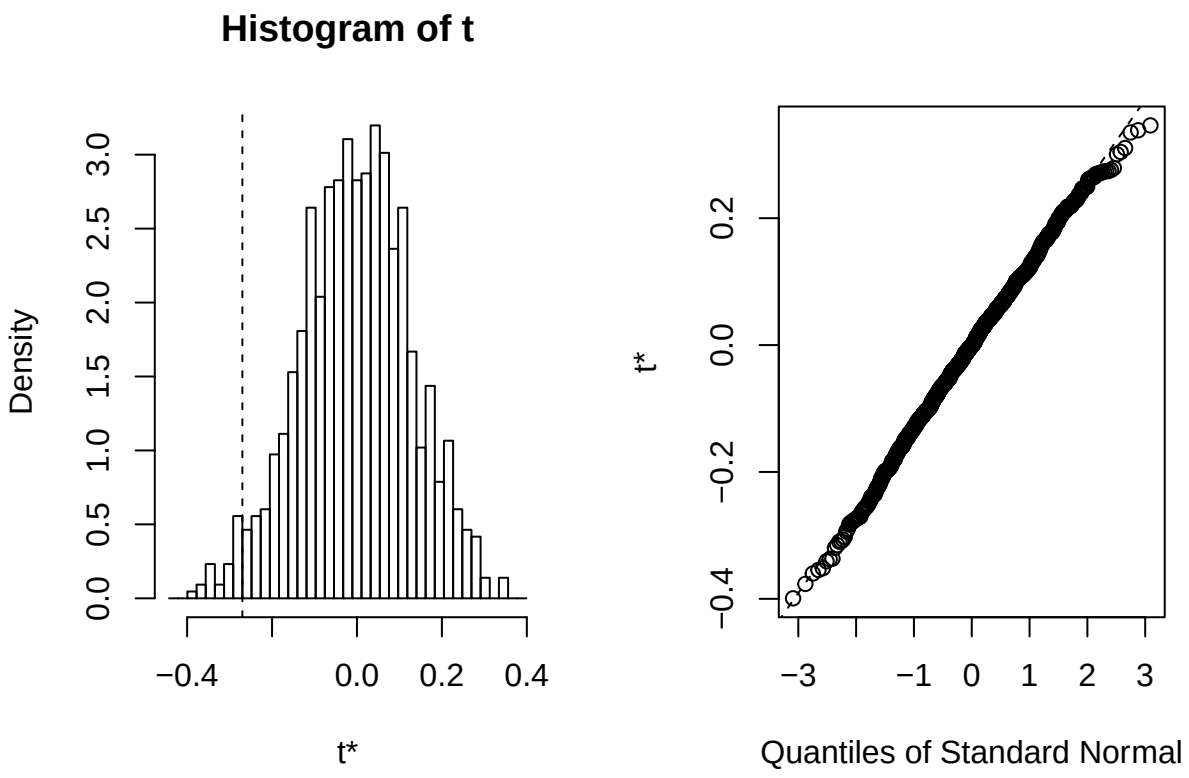
from a time series, and also of "model-based" resampling, whereby we fit an ARIMA model to the time series, bootstrap the residuals (this is called "pre-whitening") and then use the original series to figure out what a version of it with the bootstrapped residuals would look like (this is sometimes called "post-blackening"). This requires a good deal more thought than most of us are prepared to devote.

The same thing happens with the autocorrelated data `y`:

```
y_boot=tsboot(y,MKtau,1000,sim="fixed",l=5)
y_boot
```

```
##
## BLOCK BOOTSTRAP FOR TIME SERIES
##
## Fixed Block Length of 5
##
## Call:
## tsboot(tseries = y, statistic = MKtau, R = 1000, l = 5, sim = "fixed")
##
##
## Bootstrap Statistics :
##      original    bias    std. error
## t1* -0.269899 0.2677913   0.1295901
```

```
plot(y_boot)
```




```
boot.ci(y_boot)
```

```
## Warning in boot.ci(y_boot): bootstrap variances needed for studentized
## intervals
```

```
## Warning in boot.ci(y_boot): BCa intervals not defined for time series
## bootstraps

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = y_boot)
##
## Intervals :
## Level      Normal              Basic              Percentile
## 95%   (-0.7917, -0.2837 )   (-0.7885, -0.2688 )   (-0.2710,  0.2487 )
## Calculations and Intervals on Original Scale
```

Again, we have problems. The mean bootstrapped Mann-Kendall correlation is close to zero, not to the observed value of -0.27. Thus the confidence intervals disagree wildly.

I also need to consult Canty (2002), which seems to be a clear description of what's going on.

Recommendation

As I see it, the most straightforward procedure is to use the adjustment of Hamed and Rao as implemented in the `fume` package. This works directly with the data series, and allows the user to determine whether any autocorrelation has a clear effect on the significance of the Mann-Kendall correlation.

The bootstrap procedure requires considerable care in the (common) presence of a non-stationary time series. I think the most promising of the bootstrap approaches is the model-based idea of fitting a time series model and bootstrapping the residuals from it, then re-constructing a bootstrapped time series based on the bootstrapped residuals and calculating Mann-Kendall from that. This, however, requires considerable thought along the lines of the examples in the help for `tsboot`. I have to investigate this some more.

References

"Bootstrapping (Statistics) - Wikipedia, the Free Encyclopedia." *Wikipedia*. https://en.wikipedia.org/wiki/Bootstrapping_(statistics). https://en.wikipedia.org/wiki/Bootstrapping_(statistics).

Canty, Angelo J. 2002. "Resampling Methods in R: The Boot Package." *R News* 2 (3): 2–7. https://cran.r-project.org/doc/Rnews/Rnews_2002-3.pdf.

Hamed, Khaled H., and A. Ramachandra Rao. 1998. "A Modified Mann-Kendall Trend Test for Autocorrelated Data." *Journal of Hydrology* 204 (1-4). Elsevier BV: 182–96. doi:10.1016/s0022-1694(97)00125-x.

McLeod, A.I. 2011. *Kendall: Kendall Rank Correlation and Mann-Kendall Trend Test*. http://CRAN.R-project.org/package=Kendall.

Shalizi, C. 2011. "The Bootstrap." http://www.stat.cmu.edu/~cshalizi/402/lectures/08-bootstrap/lecture-08.pdf.