

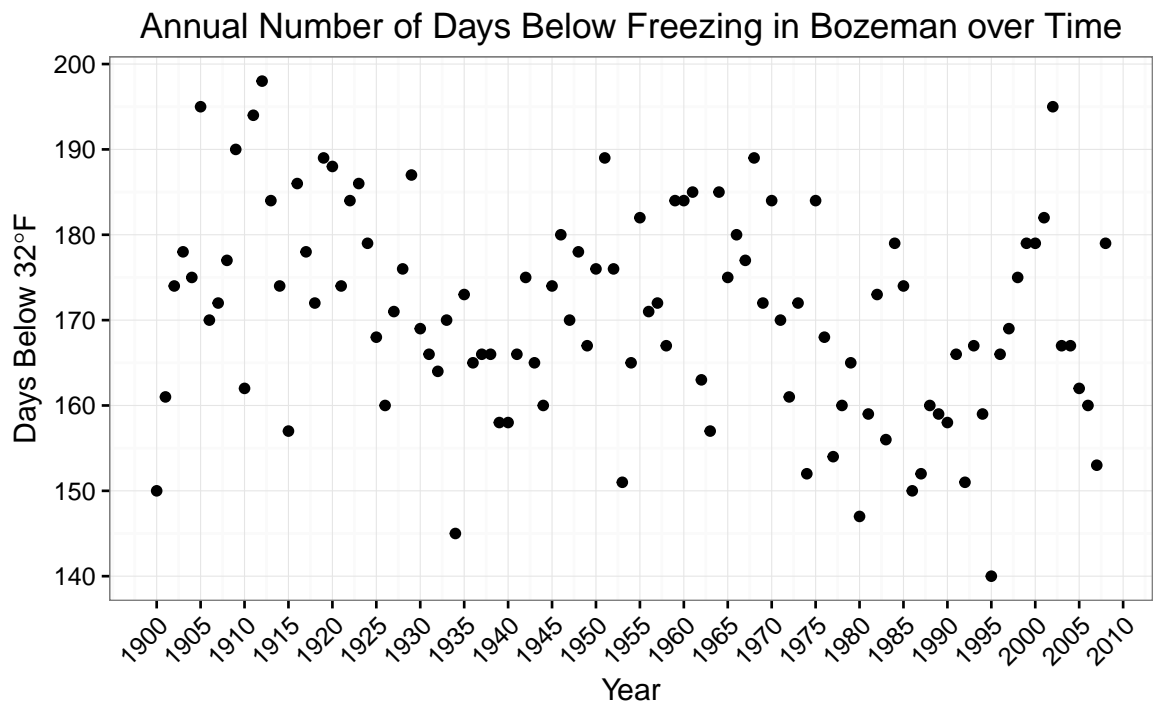
Time Series HW 6

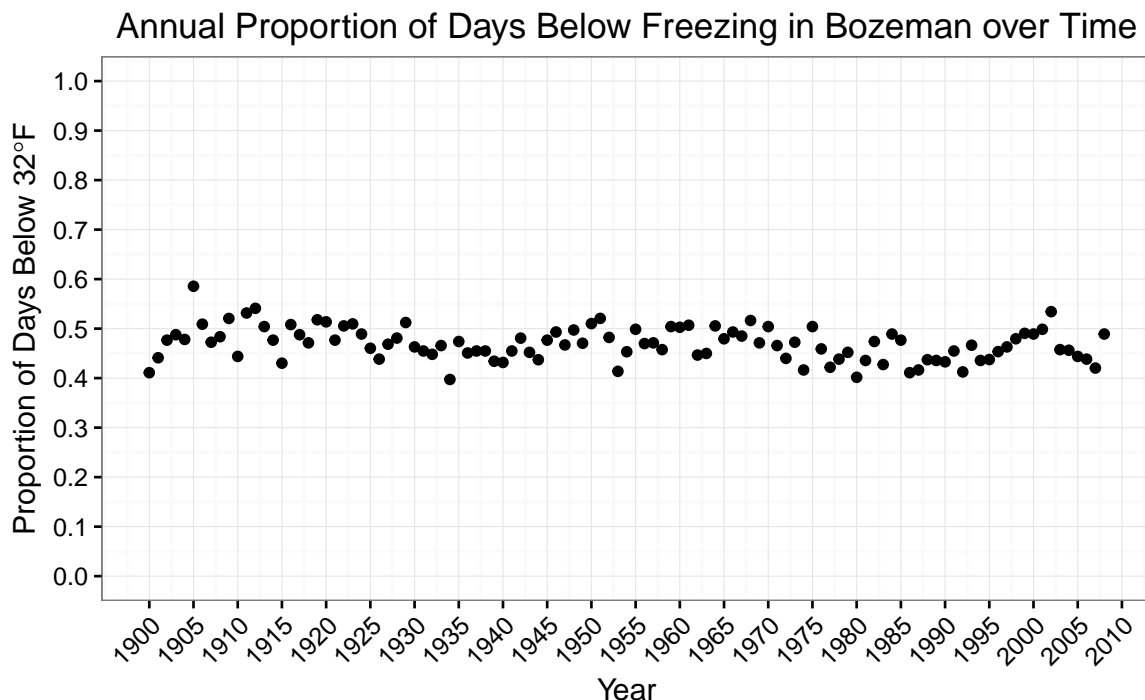
Kenny Flagg
Andrea Mack

October 18, 2016

We will explore one series like those you found in the Vincent and Meki's paper, but for Bozeman. The following code will count the days in Bozeman where the minimum temperature was measured to be below 32 degrees F (0 degrees C) and the number of days where information was available in "Data1".

1. *Make nice looking and labeled time series plots of the number of days below freezing and the proportion of measured days below freezing.*





2. Estimate a linear trend model for the proportion of measured days below freezing and report the parametric (*t*-test) linear trend test results in a sentence. Also discuss scope of inference for this test in a sentence or two (random sampling and random assignment and their implications).

Table 1: Estimated OLS linear trend model, residual SE = 0.032269

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	1.0785	0.1920	5.62	0.0000
Year	-0.0003	0.0001	-3.17	0.0020

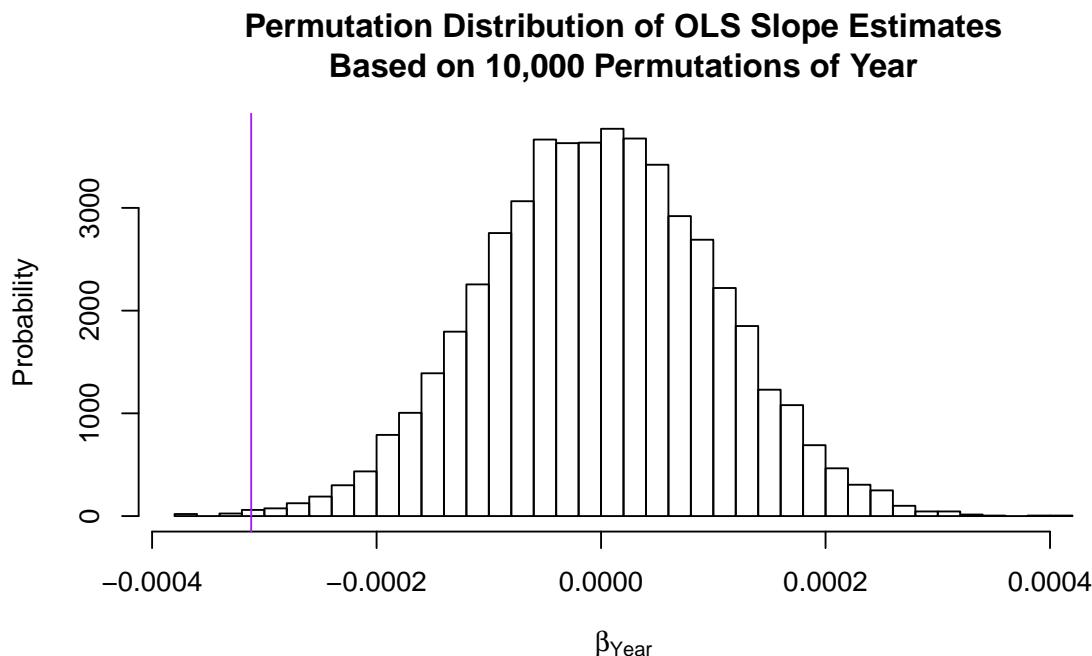
There is very strong evidence ($t = -3.17$, $p\text{-value} = 0.001968$) that year is associated with a linear trend in proportion of days below freezing. The estimated change is a decrease of 0.0312 days per 100 years in these years recorded at the spots in Bozeman where data were collected.

Data were taken through an observational study and years were not randomly chosen, so as we tell the 216 students, we can only infer that time is *associated* with a trend, not that the passage of time *caused* the trend, and we can only infer that an association exists for these specific observed locations in the observed years, not for any other location in Bozeman or elsewhere, or for any other time period.

3. Discuss this proportion response versus using the count of days below zero per year, specific to this example and in general. What issues does using one or the other present?

Specific to these data, temperature was not recorded every day of each year. To report the number of days below $32^{\circ}F$ may be misleading if there is a large discrepancy in the total number of days temperature was recorded each year. The proportion of days may still be misleading depending on which days of the year temperature was recorded, but it puts each year on the same scale.

4. Generate a permutation test for the trend with the proportion response. I performed one in the syllabus (page 6) using the “shuffle” function from the “mosaic” package. Report a plot of the permutation distribution, the test statistic you used, and a p-value. Generally randomization based tests are more robust to violations of the normality assumption as long as the distribution (shape and variability) is the same for all observations except for differences in the center or mean. Why would that be advantageous with this response?

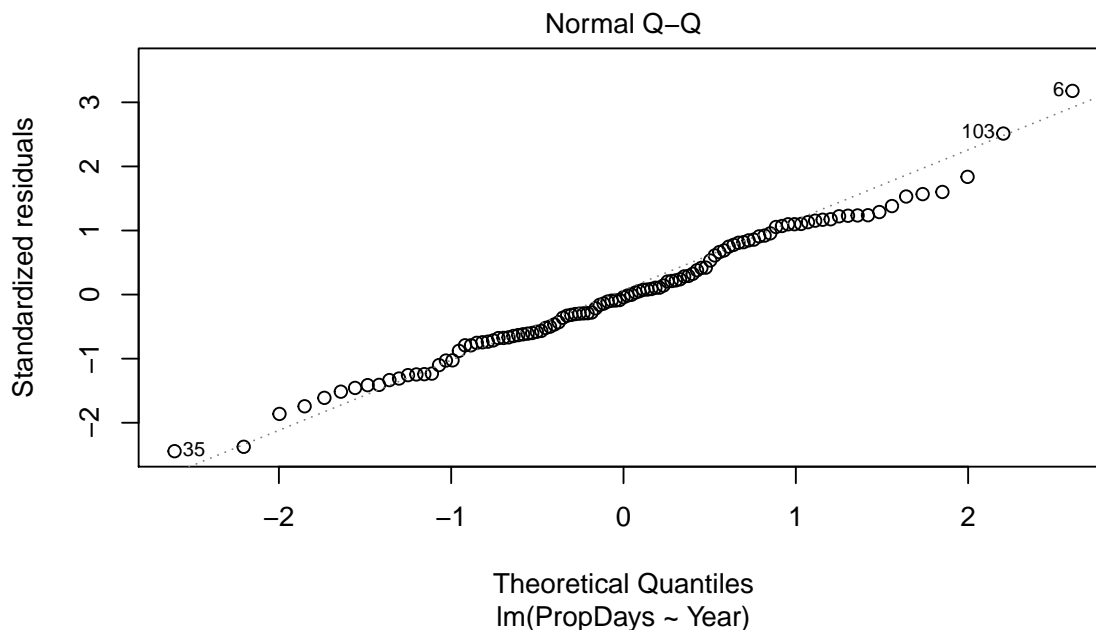


The plot above shows the location of the observed result as a purple vertical line.

$$H_o : \beta_{\text{Year}} = 0$$

$$H_a : \beta_{\text{Year}} \neq 0$$

Based on 10,000 permutations, the test statistic of $\beta_{\text{Year}} = -0.000312$ has a p-value of 0.0018, which provides very strong evidence of a linear association between the year and the annual proportion of days below $32^{\circ}F$.



The Normal Q-Q plot shows some slight deviations from normality in the tails, but they do not look serious. In this case, the residuals are approximately normally distributed so t -procedures are appropriate; however, we should be cautious because the response is a proportion.

In general, if the proportions tend to be close to the bounds of zero or one, the distribution could be skewed and so the t -test would not achieve its nominal type I error rate. The permutation test would be a better option in that situation because it does not require any particular distribution.

5. The Sen estimator or, more commonly, Theil-Sen is based on a single median of all the possible pairwise generated slopes. Its standard version is available in the “mblm” (median based linear models) R package developed by Lukasz Komsta. The package description provides more details (<https://cran.r-project.org/web/packages/mblm/mblm.pdf>). Note that with ‘mblm’, you need to use “repeated=FALSE” to get the Theil-Sen estimator and not the better estimator developed by Siegel. The package has a “summary” function that provides a test based on the nonparametric Wilcoxon test but it had terrible Type I error rates when I explored it. Without further explorations, I would recommend avoiding its use. Fortunately, our permutation approach can be used to develop a test based on the Theil-Sen slope coefficient. First, compare the estimated slope provided by “mblm” to what you found from the linear model and its permutation test. Then develop a permutation test based on the slope coefficient from ‘mblm’ - note that “mblm” conveniently has the same output structure as “lm”. The confidence interval that runs on “mblm” seems to perform well enough to study, so we can make 95% confidence intervals and check whether 0 is in the interval or not as the following code suggests to use it to perform our 5% significance level hypothesis test.

Table 2: Estimated Sen linear trend model.

	Estimate	MAD	V value	$\Pr(> V)$
(Intercept)	1.0857	0.0332	5995.00	0.0000
Year	-0.0003	0.0015	6858653.50	0.0000

`lm()` gave an estimated slope coefficient of -0.000312 and `mblm()` gave an estimated slope coefficient of -0.000316 .

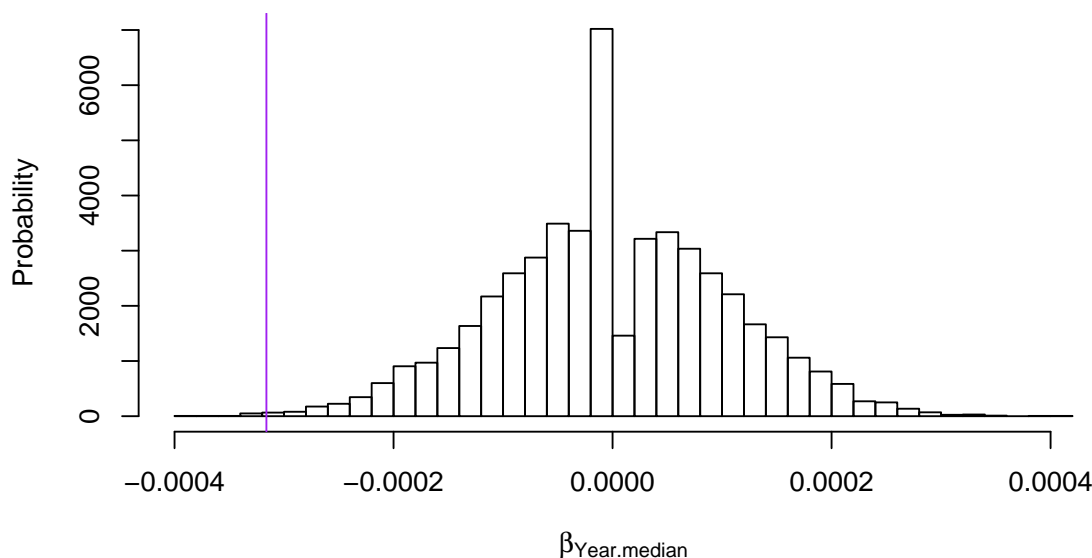
$$H_o : \beta_{Year.median} = 0$$

$$H_a : \beta_{Year.median} \neq 0$$

Using the same hypotheses from 4, but with the `mblm()` function estimating the median slope, and using 10,000 permutations, the test statistic of $\beta_{Year.median} - 0.000316$ led to a two sided p-value of 0.0023. There is strong evidence that year is associated with a linear trend in annual proportion of days below $32^\circ F$.

The median change in annual proportion of days below $32^\circ F$ is estimated to be a decrease of $0.0003161^\circ F$ per year with an associated 95% confidence interval for the decrease of $0.0003698^\circ F$ to $0.0002405^\circ F$ each subsequent year.

Permutation Distribution of Sen Slope Estimates Based on 10,000 Permutations of Year



- Use the residual error variance estimate from your linear model for the proportion responses to simulate a series with no trend (constant mean and you can leave it at 0) and normal white noise with that same variance. Use that simulation code to perform a simulation study of the Type I error rate for the parametric t-test for the slope coefficient, the test using the confidence interval from “mblm”, and your permutation test (use 500 permutations and do 250 simulations to keep the run time somewhat manageable). Report the simulation-based

Type I error rates when using a 5% significance level test for the three procedures with the same sample size as the original data set.

- *For the parametric test, the p-value can be extracted from the “lm” model “summary”’s using “summary(model1)\$coef[2,4]”.*
- *It is best and easiest if you do one loop for the simulations and then for each simulated data set in each loop generate the three test results, extracting the p-values that each produces. If you struggle to set this up, please send me an email or stop by with an attempt at your code for some feedback.*
- *This will be computationally intensive. To avoid needing to re-run results in R-markdown, you can try the “cache=T” option for any of the permutation or simulation code chunks. Or for this section, you can just report the three error rates and comment out the code you used.*

See page 10 for our simulation code. Based on 10,000 simulations for the t -test and the Sen confidence interval procedure, and 1,000 simulations for the permutation test with 1,000 permutations, we found the following type I error rates.

Procedure	Type I Error Rate
OLS t -test	0.0516
OLS permutation test	0.0660
Sen CI	0.0557

7. *Instead of white noise errors, we might also be interested in Type I error rates when we have autocorrelation present (again with no trend in the true process). Use the results for an AR(1) process variance (derived in class) to calculate the white noise variance needed to generate a process with the same variance as you used for your previous simulation, but when $\phi=0.3$ and 0.6 . In other words, γ_0 of the AR(1) process needs to match the white noise variance used above and the white noise process driving the AR(1) process needs to be adjusted appropriately.*

- (a) *Show your derivation of the required white noise variances first for $\phi = 0.3$ and $\phi = 0.6$.*

Let ϕ^k = the correlation between observations taken k time points apart (k lags apart), such that for time points 0 units apart, $\phi^0 = 1$.

In class we derived that the variance at a given time point, $\text{Var}(y_t) = \text{Cov}(y_t, y_t) = \frac{\sigma^2}{(1-\phi^2)}$.

In problem 2, we observed a residual variance of 0.0010413, so $0.0010413 = \frac{\sigma^2}{1-\phi^2}$.

Solving for σ^2 , the white noise variance is

$$\sigma^2 = (1 - \phi^2)0.0010413.$$

Now plugging in $\phi = 0.3$ and $\phi = 0.6$, we get

$$\sigma_{0.3}^2 = (1 - 0.3^2)0.0010413 = 0.0009475$$

and

$$\sigma_{0.6}^2 = (1 - 0.6^2)0.0010413 = 0.0006664.$$

- (b) To simulate the process we can use this value in the “*arima.sim*” function in something like “*arima.sim(n=2000,list(ar=c(0.3)),sd=5)*” where “*n=2000*” provides 2000 simulated observations, “*model=list(ar=c(0.3))*” determines that we are using an $AR(1)$ process with parameter of 0.3, and “*sd=5*” controls the SD of the normal white noise used to build the $AR(1)$ process (this is **not** the variance of the $AR(1)$ process). Check that you get about your expected results using something like:

```
sig3 <- sqrt((1-0.3^2)) * resid.se
sig6 <- sqrt((1-0.6^2)) * resid.se

#arima.sim goes backwards, rather than estimating parameters,
#we set parameters and generate responses
ar1sim<-arima.sim(n=2000,model=list(ar=c(0.3)),sd=sig3)
var(ar1sim)

[1] 0.00104929

ar1sim<-arima.sim(n=2000,model=list(ar=c(0.6)),sd=sig6)
var(ar1sim) # about right :)

[1] 0.001104394
```

Using a $\phi = 0.6$, the variance of the simulated time series is 0.0011044. This is very close to the residual variance of 0.0010413 from the `lm()` fit.

8. Repeat your simulation study of the parametric, permutation, and Theil-Sen linear trend test based on the CI. Report the estimated Type I error rates in the presence of $AR(1)$ correlations with a parameter of 0.6 based on your work in the previous question for simulating the response time series. Discuss the impacts of having autocorrelation present on the various procedures.

Our code is on page 12. Based on 10,000 simulations for the t -test and the Sen confidence interval procedure, and 1,000 simulations for the permutation test with 1,000 permutations, these are the type I error rates we found.

Procedure	Type I Error Rate	
	No autocorrelation	Autocorrelation with $\phi = 0.6$
OLS t -test	0.0516	0.3248
OLS permutation test	0.0660	0.0200
Sen CI	0.0557	0.3235

The t -test and Sen type I error rates are inflated, meaning they find a spurious trend too often, and the permutation type I error rate was too small, so it may also have low power and it won't find trends often enough, assuming the nominal 5% type I error rate. Methods that incorporate autocorrelation structure into the model may fix this.

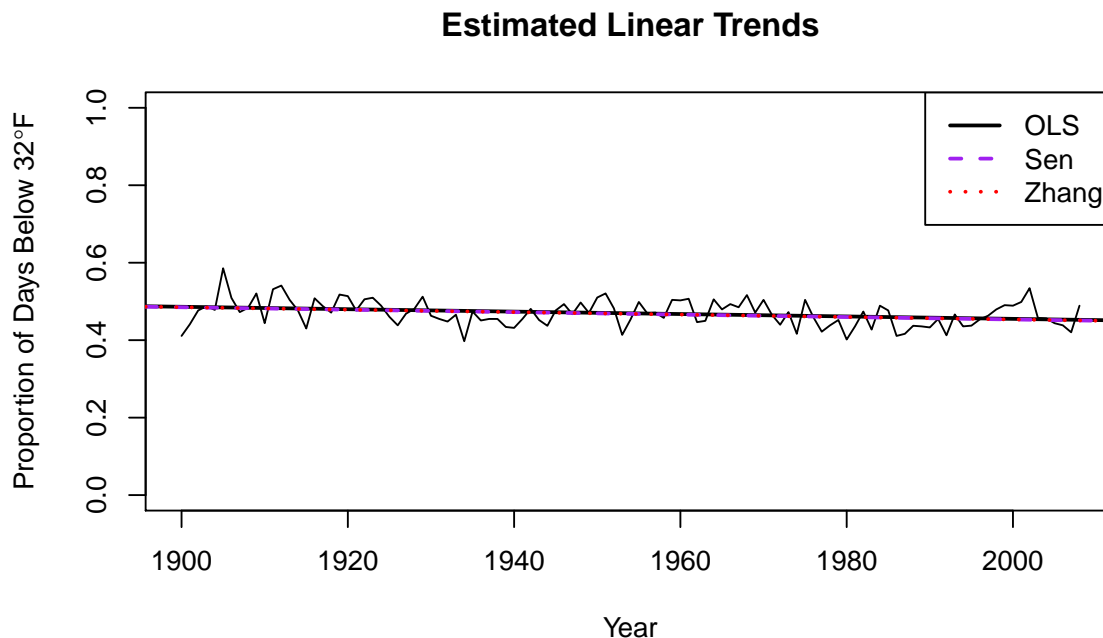
9. The Zhang method you read about is also available in the “zyp” package but it only provides confidence intervals and I am not completely convinced by their discussion of the intervals provided without more exploration. But you can get estimates from “zyp.sen” and confidence intervals using “confint.zyp” on the results from “zyp.sen”. The “confint” function can also be applied to “mblm” results. Find and compare the two confidence intervals for the Sen-estimators for the proportion response time series. No simulation study here - just complete the analysis.

The 95% confidence interval using `confint(mblm.object)` was $(-0.0003698, -0.0002405)$. The 95% confidence interval using `zyp.confint(zyp.sen.object)` was $(-0.0005074, -0.0001099)$.

The widths are nearly the same, but the `confint(mblm())` confidence interval is shifted up slightly compared to the `confint.zyp()` confidence interval. It is interesting that there is any difference as theoretically, they are both creating a 95% confidence interval for the slope coefficient for the Theil-Sen estimate.

10. Make a plot of the original proportion response time series with the parametric linear, Theil-Sen, and Zhang methods/models on the same plot. You may want to use `plot(y~x, type="l")` and then add lines to the plot.

Estimates from all three functions visually are very similar, which may be because we have yearly data and not monthly data. The three functions are all smooth because we did not estimate within-year trends.



R Code

```

1. ggplot(data = Data1, aes(x = Year, y = DaysBelow32)) + geom_point() +
  scale_x_continuous(breaks = c(seq(min(as.numeric(Data1$Year)),
                                     max(as.numeric(Data1$Year))+5, by = 5))) +
  scale_y_continuous(breaks = c(seq(min(as.numeric(Data1$DaysBelow32)),
                                     max(as.numeric(Data1$DaysBelow32))+5, by = 10))) +
  theme_bw() + theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = 'Annual Number of Days Below Freezing in Bozeman over Time') +
  ylab(expression('Days Below 32'*degree*F))

ggplot(data = Data1, aes(x = Year, y = PropDays)) + geom_point() +
  scale_x_continuous(breaks = c(seq(min(as.numeric(Data1$Year)),
                                     max(as.numeric(Data1$Year))+5, by = 5))) +
  scale_y_continuous(limits = c(0,1), breaks = c(seq(0,1, by = 0.1))) +
  theme_bw() + theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = 'Annual Proportion of Days Below Freezing in Bozeman over Time') +
  ylab(expression('Proportion of Days Below 32'*degree*F))

2. lm.Year <- lm(PropDays ~ Year, data = Data1)
   resid.se <- summary(lm.Year)$sigma # Save this for later
   xtable(lm.Year, caption = paste("Estimated OLS linear trend model, residual SE =", signif(resid.se, 5)))

4. # I don't trust knitr's caching so I'm saving things manually.
   if(file.exists("prob4.Rdata")){
     load("prob4.Rdata")
   }else{
     # Run in parallel.
     # Note: Most guidelines say don't use more than half of your CPU cores but whatevs...
     cl <- makeCluster(detectCores())
     clusterExport(cl, c("Data1", "shuffle")) # Load Data1 and mosaic::shuffle() on the nodes
     lm_perm <- parSapply(cl, seq_len(10000), function(i){ # seq_len(10000) is the same as 1:10000
       return(coef(lm(Data1$PropDays ~ shuffle(Data1$Year)))[2])
     })
     stopCluster(cl) # Stop this cluster and start with a clean environment next time

     save(lm_perm, file = "prob4.Rdata")
   }

hist(lm_perm, main = "Permutation Distribution of OLS Slope Estimates
Based on 10,000 Permutations of Year",
      xlab = expression(beta[Year]), ylab = "Probability", freq = FALSE, breaks = 50)
abline(v = coef(lm.Year)[2], col = "purple")

perm_pvalue <- (length(which(lm_perm<=coef(lm.Year)[2])) + length(which(lm_perm>=-coef(lm.Year)[2]))) /
  length(lm_perm)

par(mfrow=c(1,1))
plot(lm.Year, which = 2)

```

```

5. model1s<-mblm(PropDays~Year,data=Data1,repeated=F)
model1s.coef <- coef(model1s)[2]

xtable(model1s, caption = "Estimated Sen linear trend model.")

year <- data.frame(replicate(10000, shuffle(Data1$Year)))
shuffled.frame <- data.frame(pdays = Data1$PropDays,
                             suffled.year = rep(as.numeric(NA), 109))

if(file.exists("prob5.Rdata")){
  load("prob5.Rdata")
}else{
  cl <- makeCluster(detectCores())
  clusterExport(cl, c("Data1", "shuffled.frame", "mblm"))
  mblm.mod <- parCapply(cl, year, function(x){
    shuffled.frame$year <- x
    return(mblm(pdays~year, data=shuffled.frame,repeated = F))
  })
  stopCluster(cl)

  # Excercise for Andrea!
  # level of difficulty: 1 (after seeing mblm.mod of course ;)
  mblm.coeff <- sapply(mblm.mod, function(x){coef(x)[2]})

  save(mblm.coeff, file = "prob5.Rdata")
}

pvalue.mblm <- length(c(which(mblm.coeff<=model1s.coef), which(mblm.coeff>=-model1s.coef))) /
length(mblm.coeff)

CI<-confint(model1s)[2,] #Extract CI and check whether 0 is in interval
#(0>CI[1])&(0<CI[2]) #If 0 is in interval, FTR H0

hist(mblm.coeff, main = "Permutation Distribution of Sen Slope Estimates
Based on 10,000 Permutations of Year",
      xlab = expression(beta[Year.median]), ylab = "Probability", freq = FALSE, breaks = 50)
abline(v = model1s.coef, col = "purple")

6. # first is lm

rand.noise <- data.frame(replicate(10000, rnorm(109, 0, resid.se)))
#dim(rand.noise)# 109 rows

if(file.exists("prob6t.Rdata")){
  load("prob6t.Rdata")
}else{
  cl <- makeCluster(detectCores())
  clusterExport(cl, c("Data1"))
  lm.noise <- parCapply(cl, rand.noise, function(x){
    # Get the pvalues!
    return(summary(lm(x ~ Year, data = Data1))$coefficients["Year","Pr(>|t|)"])
  })
}

```

```

    stopCluster(cl)
    save(lm.noise, file = "prob6t.Rdata")
  }

  # Get proportion with small pvalues
  lm.noise_type1 <- mean(lm.noise < 0.05)

  # next is mblm

  noise.frame <- data.frame(Year = Data1$Year, out = rep(as.numeric(NA), 109))

  if(file.exists("prob6m.Rdata")){
    load("prob6m.Rdata")
  }else{
    cl <- makeCluster(detectCores())
    clusterExport(cl, c("Data1", "noise.frame", "mblm"))
    mblm.noise.reject <- parCapply(cl, rand.noise, function(x){
      noise.frame$out <- x
      # Is this where we use the CIs?
      CI<-confint(mblm(out~Year, data=noise.frame,repeated = F))[2,]
      return((0 < CI[1]) | (0 > CI[2])) # Is 0 outside the CI?
    })
    stopCluster(cl)
    save(mblm.noise.reject, file = "prob6m.Rdata")
  }

  # Get proportion with 0 outside the interval
  mblm.noise_type1 <- mean(mblm.noise.reject)

  # finally the permutation test

  if(file.exists("prob6p.Rdata")){
    load("prob6p.Rdata")
  }else{
    cl <- makeCluster(detectCores())
    clusterExport(cl, c("Data1", "shuffle"))
    # Loop for each simulated noise vector
    lm.perm.pvals <- parCapply(cl, rand.noise[,1:1000], function(x){
      # Get the "observed result" from fitting the ith dataset
      lm.slope <- coef(lm(x ~ Year, data = Data1))[2]

      # Now loop to make a permutation distribution using this sim and suffled years
      lm.perms <- replicate(1000, coef(lm(x ~ shuffle(Year), data = Data1))[2])

      # Return pvalue
      return(mean(abs(lm.slope) > abs(lm.perms))))
    })
    stopCluster(cl)
    save(lm.perm.pvals, file = "prob6p.Rdata")
  }

  # Get proportion with small pvalues
  lm.perm_type1 <- mean(lm.perm.pvals < 0.05)

```

*# Note: We could combine these three simulations and have parSapply return a
matrix of reject/not reject outcomes*

```
7. sig3 <- sqrt((1-0.3^2)) * resid.se
   sig6 <- sqrt((1-0.6^2)) * resid.se
```

*#arima.sim goes backwards, rather than estimating parameters,
#we set parameters and generate responses*

```
ar1sim<-arima.sim(n=2000,model=list(ar=c(0.3)),sd=sig3)
var(ar1sim)
```

```
ar1sim<-arima.sim(n=2000,model=list(ar=c(0.6)),sd=sig6)
var(ar1sim) # about right :)
```

8. *# first is lm*

```
rand.ar1 <- data.frame(replicate(10000, arima.sim(n=109,model=list(ar=c(0.6)),sd=sig6)))
```

```
if(file.exists("prob8t.Rdata")){
  load("prob8t.Rdata")
}else{
  cl <- makeCluster(detectCores())
  clusterExport(cl, c("Data1"))
  lm.ar1 <- parCapply(cl, rand.ar1, function(x){
    # Get the pvalues!
    return(summary(lm(x ~ Year, data = Data1))$coefficients["Year","Pr(>|t|)"])
  })
  stopCluster(cl)
  save(lm.ar1, file = "prob8t.Rdata")
}
```

Get proportion with small pvalues

```
lm.ar1_type1 <- mean(lm.ar1 < 0.05)
```

next is mblm

```
ar1.frame <- data.frame(Year = Data1$Year, out = rep(as.numeric(NA), 109))
```

```
if(file.exists("prob8m.Rdata")){
  load("prob8m.Rdata")
}else{
  cl <- makeCluster(detectCores())
  clusterExport(cl, c("Data1", "ar1.frame", "mblm"))
  mblm.ar1.reject <- parCapply(cl, rand.ar1, function(x){
    ar1.frame$out <- x
    # Is this where we use the CIs?
    CI<-confint(mblm(out~Year, data=ar1.frame,repeated = F))[2,]
    return((0 < CI[1]) | (0 > CI[2])) # Is 0 outside the CI?
  })
  stopCluster(cl)
```

```

    save(mblm.ar1.reject, file = "prob8m.Rdata")
  }

  # Get proportion with 0 outside the interval
  mblm.ar1_type1 <- mean(mblm.ar1.reject)

  # finally the permutation test

  if(file.exists("prob8p.Rdata")){
    load("prob8p.Rdata")
  }else{
    cl <- makeCluster(detectCores())
    clusterExport(cl, c("Data1", "shuffle"))
    # Loop for each simulated noise vector
    lm.ar1.perm <- parCapply(cl, rand.ar1[,1:1000], function(x){
      # Get the "observed result" from fitting the ith dataset
      lm.slope <- coef(lm(x ~ Year, data = Data1))[2]

      # Now loop to make a permutation distribution using this sim and suffled years
      lm.perms <- replicate(1000, coef(lm(x ~ shuffle(Year), data = Data1))[2])

      # Return pvalue
      return(mean(abs(lm.slope) > abs(lm.perms)))
    })
    stopCluster(cl)
    save(lm.ar1.perm, file = "prob8p.Rdata")
  }

  # Get proportion with small pvalues
  lm.ar1.perm_type1 <- mean(lm.ar1.perm < 0.05)

```

```

9. zyp.Year <- zyp.sen(PropDays ~ Year, data = Data1)

```

```

CI.zyp <- c(confint.zyp(zyp.Year)[2,])

```

```

10. plot(PropDays ~ Year, data = Data1, type = "l", ylim = c(0,1),
      main = "Estimated Linear Trends",
      ylab = expression("Proportion of Days Below 32"*degree*F))
abline(lm.Year, lwd = 2)
abline(model1s, col = "purple", lty = 2, lwd = 2)
abline(a = zyp.Year$coefficients[1], b =
      zyp.Year$coefficients[2], col = "red", lty = 3, lwd = 2)
legend("topright", lwd = 2, lty = 1:3, col = c("black", "purple", "red"),
      legend = c("OLS", "Sen", "Zhang"))

```