Lab: Week 10

36-350 – Statistical Computing

Week 10 – Fall 2020

Name: Kimberly Zhang

Andrew ID: kyz

You must submit **your own** lab as a PDF file on Gradescope.

```
suppressWarnings(library(tidyverse))
```

```
## -- Attaching packages -------------------------------------------------------------------------------
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## -- Conflicts ---------------------------------------------------------------------------------- t
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Below we define a series of dates:

```
date = rep("",5)
date[1] = "March 8, 2014"
date[2] = "19 Feb 2017"
date[3] = "6-29-2014"
date[4] = "5/17/1986"
date[5] = "12.25.88"       # 1988
```

Question 1

(10 points)

Notes 10A (2,4-5)

Convert each of the dates above to `Date` class format. Save the output for each. You cannot simply declare a vector of type `Date`, so I would suggest the following workaround: initialize a character vector of length 5, then for each date, cast the output of `as.Date()` back to character. When you are done with all five conversions, cast the full character vector back to `Date`. Kludgy, but it works. (There are other workarounds as well, but this is intuitively straightforward.)

```
res = rep("", 5)
res[1] = as.character(as.Date(date[1], format = "%B %d, %Y"))
res[2] = as.character(as.Date(date[2], format = "%d %b %Y"))
res[3] = as.character(as.Date(date[3], format = "%m-%d-%Y"))
```

```r
res[4] = as.character(as.Date(date[4], format = "%m/%d/%Y"))
res[5] = as.character(as.Date(date[5], format = "%m.%d.%y"))
res = as.Date(res)
res
```

```
## [1] "2014-03-08" "2017-02-19" "2014-06-29" "1986-05-17" "1988-12-25"
```

Question 2

(5 points)

Notes 10A (7)

Determine the weekday for each of the dates above. What day of the week is it 100 days after the first date? (Display the result via code, don't just figure this out by hand!) What about 158 days before the second date?

```r
weekdays(res)
```

```
## [1] "Saturday" "Sunday"   "Sunday"   "Saturday" "Sunday"
```

```r
weekdays(res[1] + 100)
```

```
## [1] "Monday"
```

```r
weekdays(res[2] - 158)
```

```
## [1] "Wednesday"
```

Question 3

(5 points)

Notes 10A (7-8)

Write a codelet below that displays the current day of the week, one that works no matter when it is run.

```r
weekdays(Sys.Date())
```

```
## [1] "Tuesday"
```

Question 4

(10 points)

Notes 10A (6)

Create a sorted sequence of five uniformly distributed random dates that lie between January 1, 1990, and January 1, 2000. Display your result.

```r
set.seed(101)
date.seq= sort(as.Date("1990-01-01") +
               as.integer(runif(n=5, min=0, max=3650)))
date.seq
```

```
## [1] "1990-06-09" "1992-06-30" "1993-09-20" "1996-07-28" "1997-02-03"
```

Question 5

(5 points)

Notes 10A (6)

Use `dist()` to display the pairwise distances between each date, i.e., display a matrix-like structure that shows the number of days between each of your randomly selected dates. If the numbers show up as decimals (e.g., 754.667), then you didn't do Q4 optimally; think about how you could amend Q4 so that the output here consists of integer differences.

```
dist(date.seq)
```

```
##      1    2    3    4
## 2  752
## 3 1199  447
## 4 2241 1489 1042
## 5 2431 1679 1232  190
```

Below we define a vector of times:

```
time = rep("",5)
time[1] = "4:32 PM"
time[2] = "16:45:33"
time[3] = "7:30:00 AM"
time[4] = "9:20"
time[5] = "12:00 PM"
```

Question 6

(5 points)

Review

Append each of the times above to each of the dates defined above Q1. (Recall: `paste()`.) Display your output.

```
datetime = paste(res, time)
datetime
```

```
## [1] "2014-03-08 4:32 PM"    "2017-02-19 16:45:33"   "2014-06-29 7:30:00 AM"
## [4] "1986-05-17 9:20"       "1988-12-25 12:00 PM"
```

Question 7

(5 points)

Notes 10A (4-5)

Similarly to how you answered Q1, convert each of the character strings output in Q6 to objects of the `POSIXlt` class.

```
res.datetime = rep("", 5)
res.datetime[1]= as.character(as.POSIXlt(datetime[1],
                                         format="%Y-%m-%d %I:%M %p"))
res.datetime[2]= as.character(as.POSIXlt(datetime[2],
                                         format="%Y-%m-%d %H:%M:%S"))
res.datetime[3]= as.character(as.POSIXlt(datetime[3],
                                         format="%Y-%m-%d %I:%M:%S %p"))
res.datetime[4]= as.character(as.POSIXlt(datetime[4],
                                         format="%Y-%m-%d %H:%M"))
res.datetime[5]= as.character(as.POSIXlt(datetime[5],
```

```
                                          format="%Y-%m-%d %I:%M %p"))
res.datetime = as.POSIXlt(res.datetime)
res.datetime
```

```
## [1] "2014-03-08 16:32:00 EST" "2017-02-19 16:45:33 EST"
## [3] "2014-06-29 07:30:00 EDT" "1986-05-17 09:20:00 EDT"
## [5] "1988-12-25 12:00:00 EST"
```

Question 8

(5 points)

Notes 10A (6)

How many minutes elapse between the second time and the third time? Use `as.numeric()` and `cat()` to have your final output be "Time difference of minutes".

```
minutes= as.numeric(res.datetime[2] - res.datetime[3])*24*60
cat(c("Time difference of", minutes, "minutes"))
```

```
## Time difference of 1391655.55 minutes
```

Question 9

(5 points)

Notes 10A (7)

Google "Modified Julian Day". Then use the origin defined for the MJD scale to compute the modified Julian dates for each element of your time vector. Use `as.POSIXlt()` with midnight specified as "00:00:00". (We will ignore the issue of time zones here.) The final output should include 47520.50.

```
julian(res.datetime, origin =
         as.POSIXlt("November 17, 1858 00:00:00",
                    format = "%B %d, %Y %H:%M:%S"))
```

```
## Time differences in days
## [1] 56724.69 57803.70 56837.27 46567.35 47520.50
## attr(,"origin")
## [1] "1858-11-17 LMT"
```

Question 10

(5 points)

Notes 10A (8)

How long does it take your computer to add the first one million integers together via the use of a `for` loop? Determine this via the use of `system.time()`. How about ten million? One hundred million? Have the number of integers to add together be an input to your function, so that you do not repeat defining the function over and over. (This would be bad form.) If you look at the `user` field in the output, you should see that this additive operation scales approximately linearly with time, i.e., it scales as $\mathcal{O}(n)$, where n is the number of integers. ($\mathcal{O}$ represents "big O notation" [feel free to Google it]. In life, you prefer operations that scale as $\mathcal{O}(n)$ or $\mathcal{O}(n \log n)$, while being leery of operations that are $\mathcal{O}(n^2)$ or slower. I'm looking at you, Support Vector Machine.)

```
f = function(n){
  num.sum = 0
  for (i in 1:n){
```

```
    num.sum = num.sum + i
  }
  return(num.sum)
}
system.time({f(1000000)})
```

```
##    user  system elapsed
##    0.03    0.00    0.03
```

```
system.time({f(10000000)})
```

```
##    user  system elapsed
##    0.35    0.00    0.35
```

```
system.time({f(100000000)})
```

```
##    user  system elapsed
##    3.36    0.00    3.36
```

Here we load in the `R` variable `air.passengers`:

```
load(url("http://www.stat.cmu.edu/~pfreeman/AirPassengers.Rdata"))
```

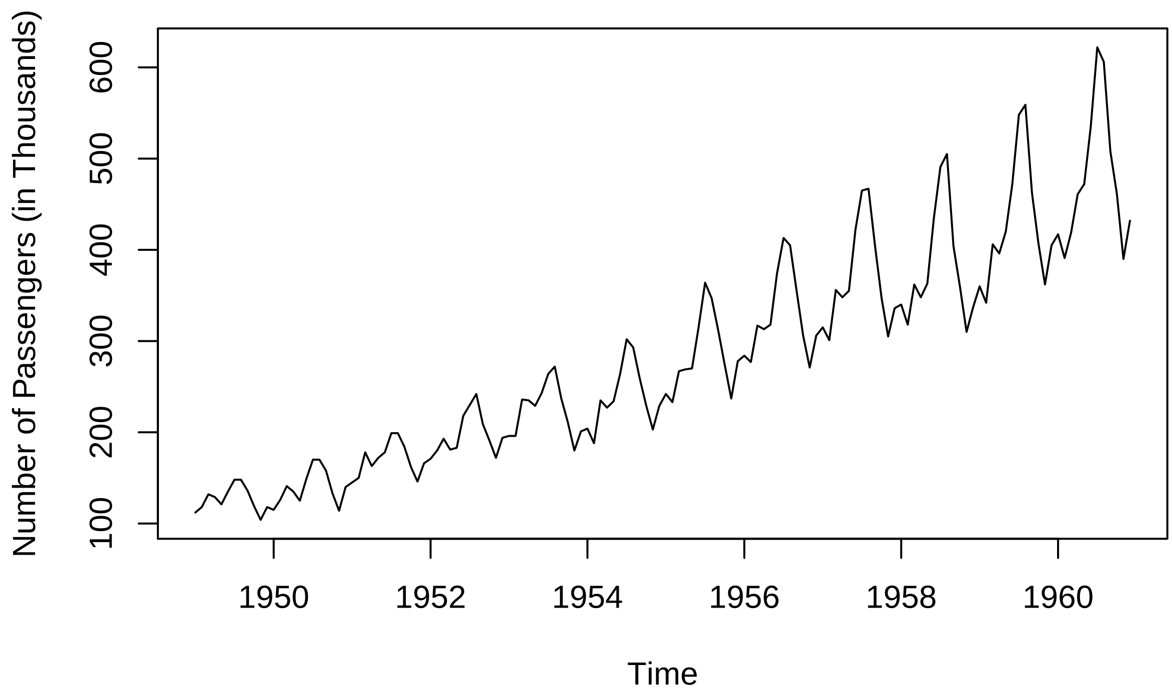
Question 11

(5 points)

Notes 10B (3-4)

Convert the numeric vector `air.passengers` to a time-series object. The data are collected monthly and the first datum is from January 1949. Plot your time-series object via `plot()`. Change the y-axis label to say "Number of Passengers (in Thousands)".

```
passenger.ts = ts(air.passengers, start = c(1949, 1), frequency = 12)
plot(passenger.ts, ylab = "Number of Passengers (in Thousands)")
```



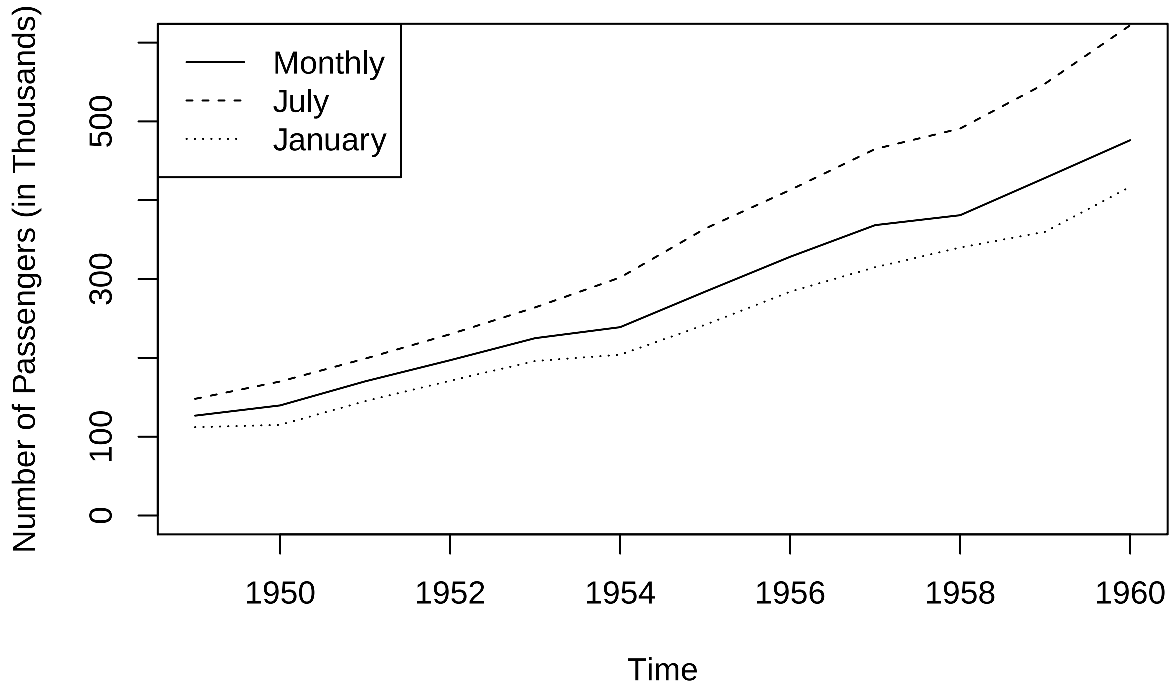
Question 12

(5 points)

Notes 10B (5-6)

Determine the average number of air passengers per month as a function of year. Then determine the average number of air passengers traveling in January and July, also as a function of year. Plot all three quantities on the same panel. (This involves using `plot()` followed by `lines()` followed by `lines()`; each call plots a separate quantity.) Set the y-axes limits to be 0 to 600, set the line type for July to be 2 (`lty=2`) and for January to be 3 (`lty=3`). Lastly, add a legend to the plot panel the indicates which line is which. See the examples in the `legend()` documentation, or mine Google for examples, as you would in real life. Note: the July data will appear "offset" compared to the other two sets of data; don't worry about this.

```
passenger.ts.annual = aggregate(passenger.ts, FUN = mean)
agg.res = aggregate(passenger.ts, nfreq = 12, FUN = mean)
agg.mat = t(matrix(agg.res, ncol=12))
july.ts = agg.mat[, 7]
jan.ts = agg.mat[, 1]
plot(passenger.ts.annual, ylab = "Number of Passengers (in Thousands)",
     ylim = c(0, 600))
lines(1949:1960, july.ts, lty= 2)
lines(1949:1960, jan.ts, lty = 3)
legend("topleft",
  legend = c("Monthly", "July", "January"),
  lty = 1:3)
```



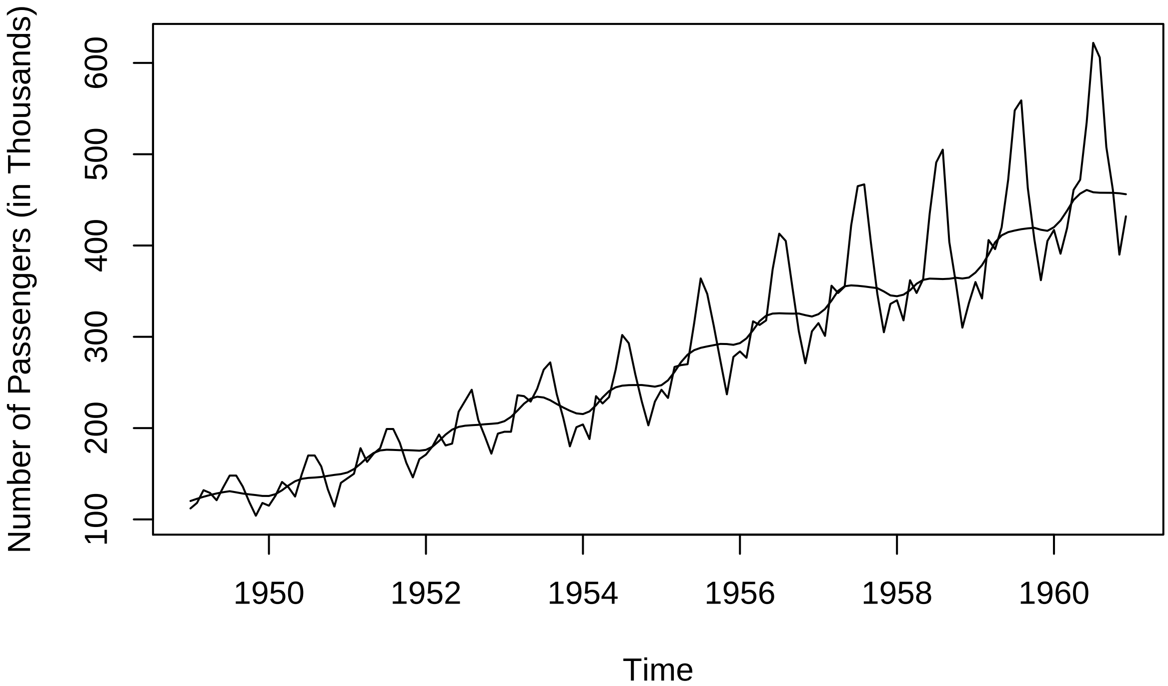
Question 13

(5 points)

Not in Notes

The plot that you made in Q12 indicates that there is definitely an overall trend in the data: the number of passengers is growing larger year-by-year. In Q12, you made an estimate of the trend by using the sample mean in each year. Another way to do this is with `lowess()`, which you have used previously. Below, plot the time series for `air.passengers`, and overlay an estimate of the trend using `lowess()`. Change the argument `f` to improve the estimate without losing the overall smoothness in the trend.

```
plot(passenger.ts, ylab = "Number of Passengers (in Thousands)")
lines(lowess(passenger.ts, f= 0.1))
```



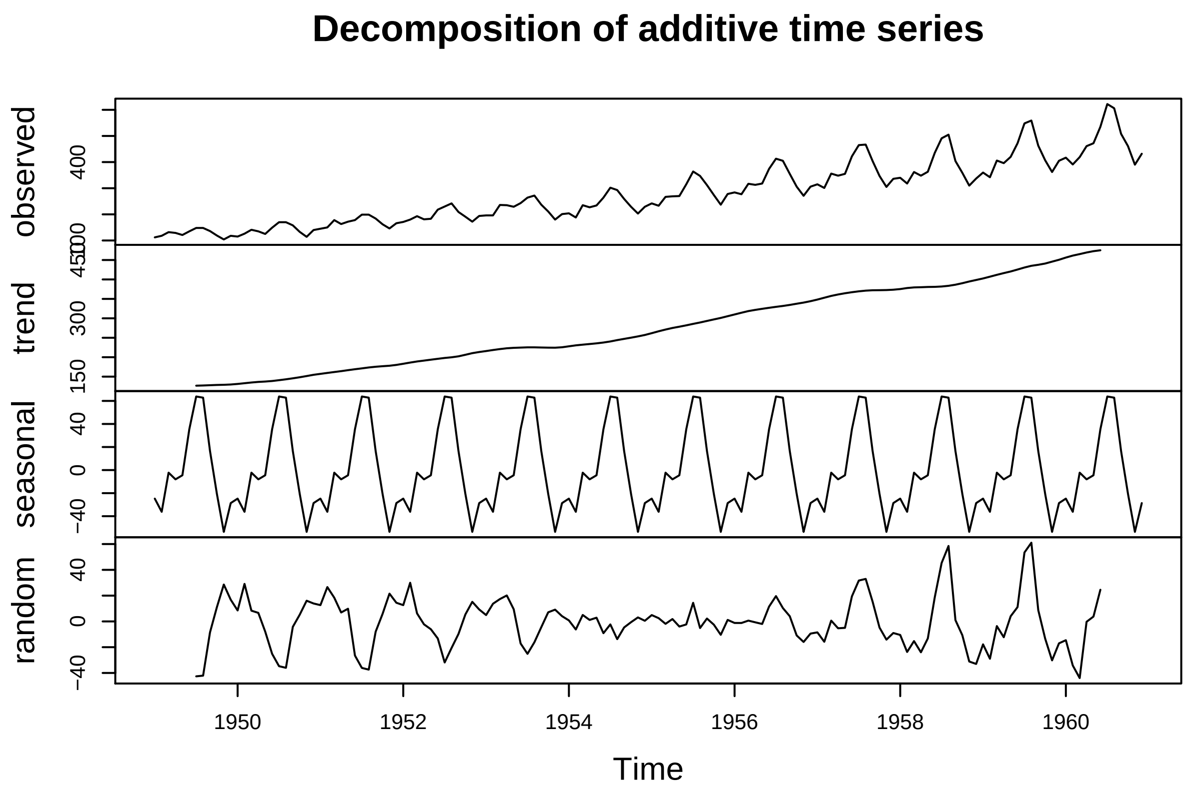
Question 14

(5 points)

Notes 10B (7)

Decompose the `air.passengers` time series using the `decompose()` function. Plot the decomposition.

```
plot(decompose(passenger.ts))
```



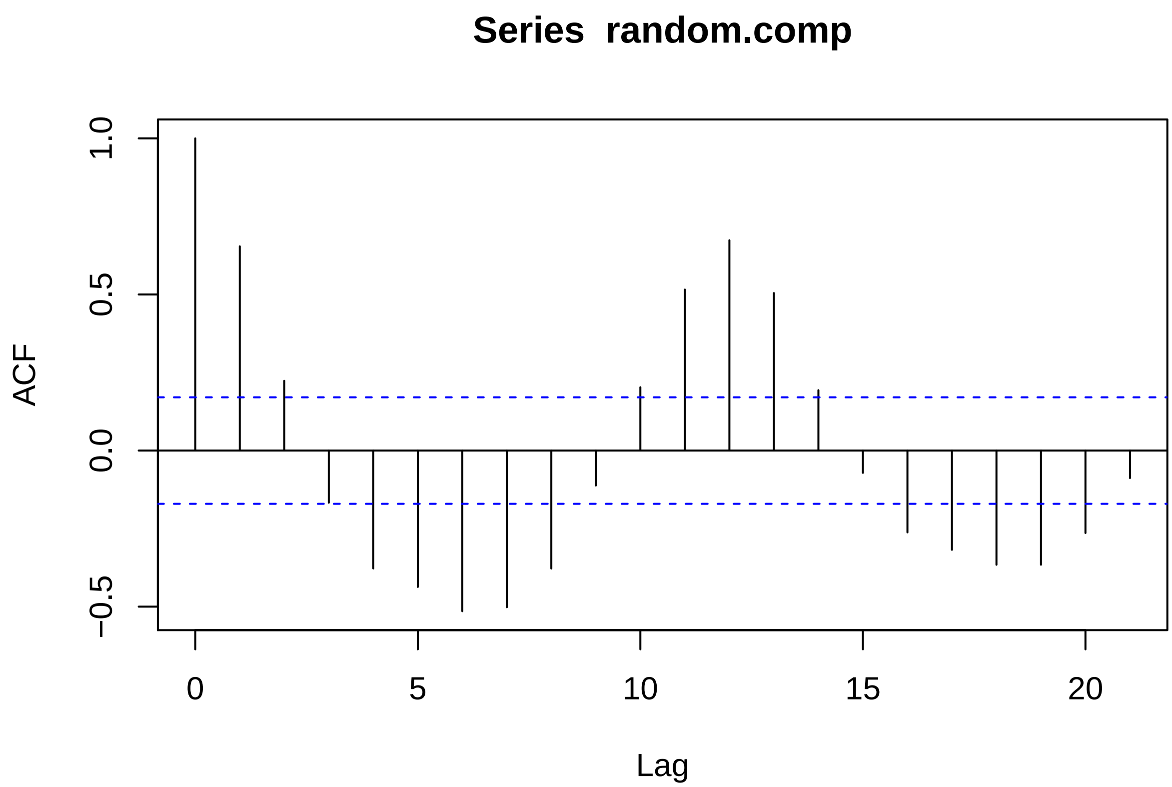
Question 15

(5 points)

Notes 10B (8-9)

Plot the autocorrelation function (acf) for the random component of the `air.passengers` time series. From its appearance, do you think that the random portion of the `air.passengers` time series is well-modeled? (The appearance of the random component in the decomposition plot might offer a clue here.)

```
random.comp = decompose(passenger.ts)$random
random.comp = random.comp[!is.na(random.comp)]
a = acf(random.comp)
plot(a)
```




```
No it doesn't appear to be well-modeled because patterns are apparent in the
ACF plot and the positive/negative correlations are signficant at most
time points.
```

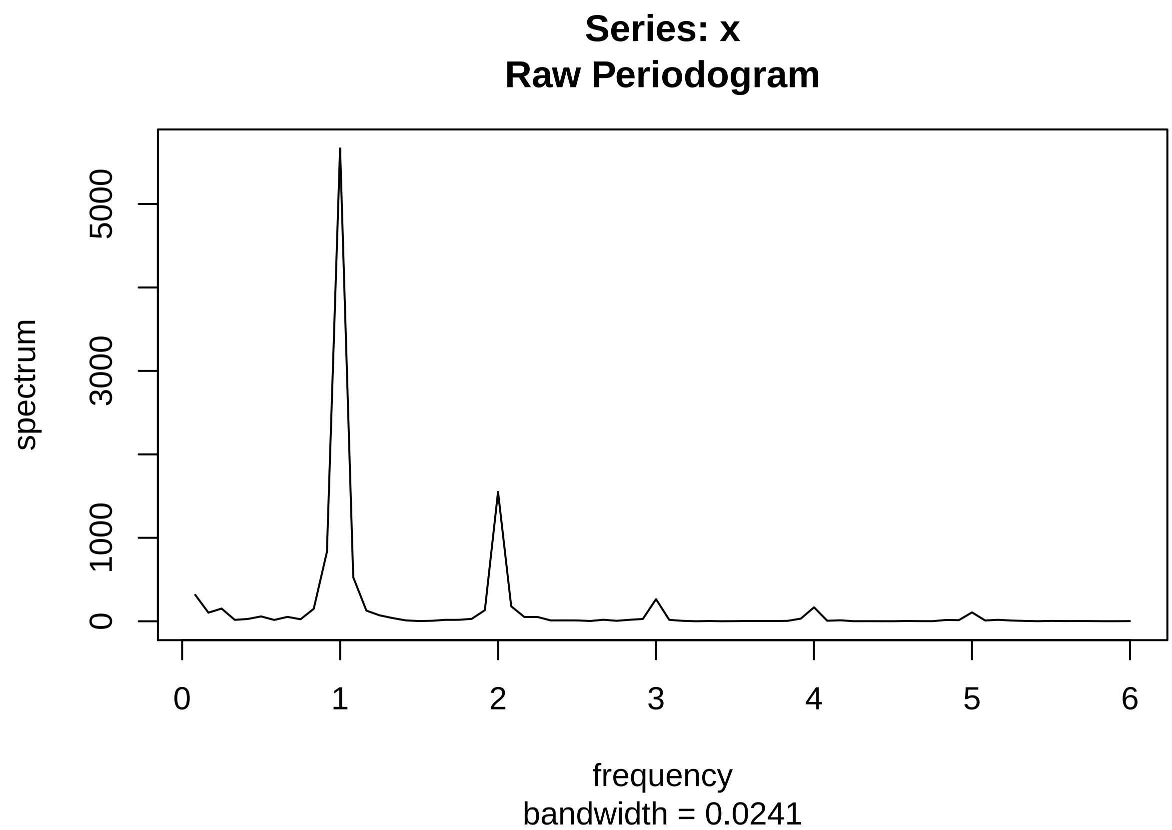
Question 16

(5 points)

Notes 10C (4-6)

As stated in the notes, we don't have to cast an input time series to class `ts` prior to creating a periodogram, but it doesn't necessarily hurt to do so. Plot the periodogram of your `air.passengers` time series. Set `log="no"`. What do you see? Google "harmonics periodogram" and explain what you see. (Note that a linear trend was removed prior to spectral decomposition!)

```r
spectrum(passenger.ts, log = "no")
```




```
The series is not random because otherwise all
the sinusoidals would be of equal importance. The series has a strong
nonsinusoidal signal at frequency = 1 (fundamental frequency), and peaks of
decreasing height at 2, 3, 4, 5 (harmonics). The dataset has an excessive of
low frequency; the time series is probably smooth.
```

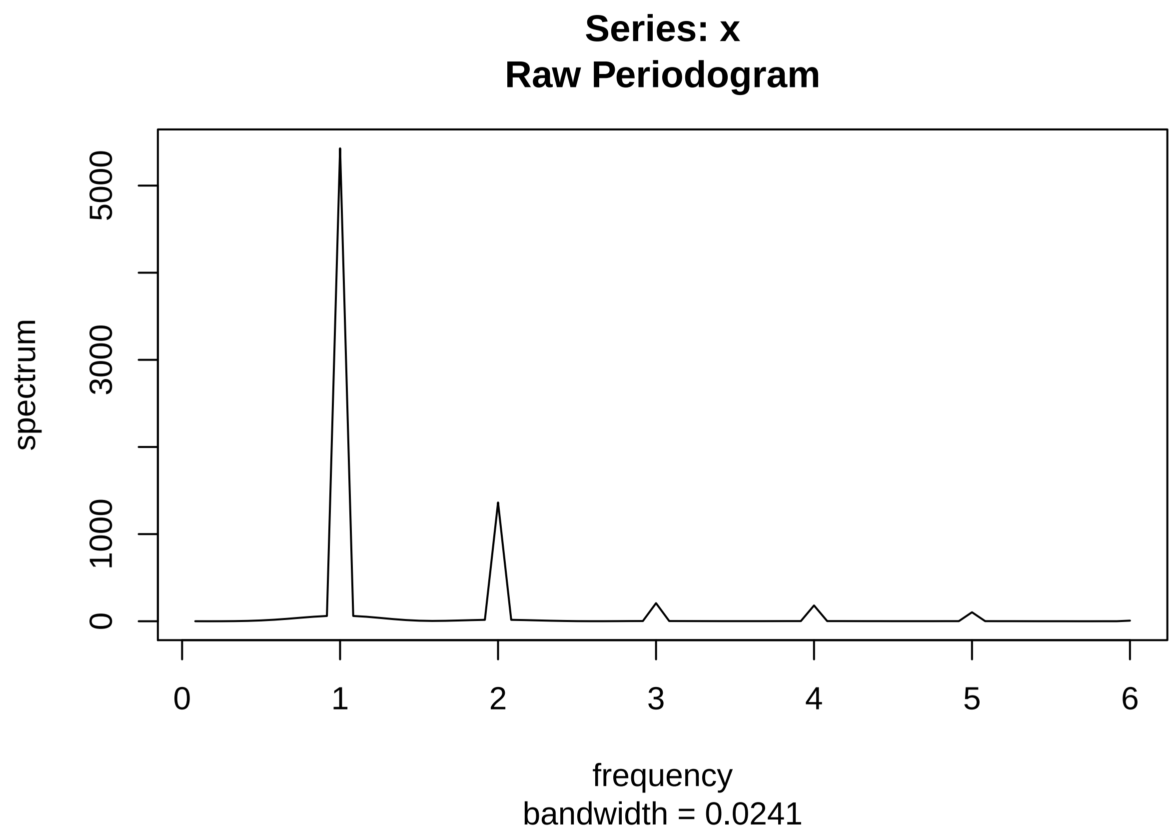
Question 17

(5 points)

Notes 10C (4-6)

Plot the periodogram for the seasonal component of your decomposition of the `air.passengers` time series. You should observe something similar to what you observed in Q16.

```r
spectrum(decompose(passenger.ts)$seasonal, log = "no")
```



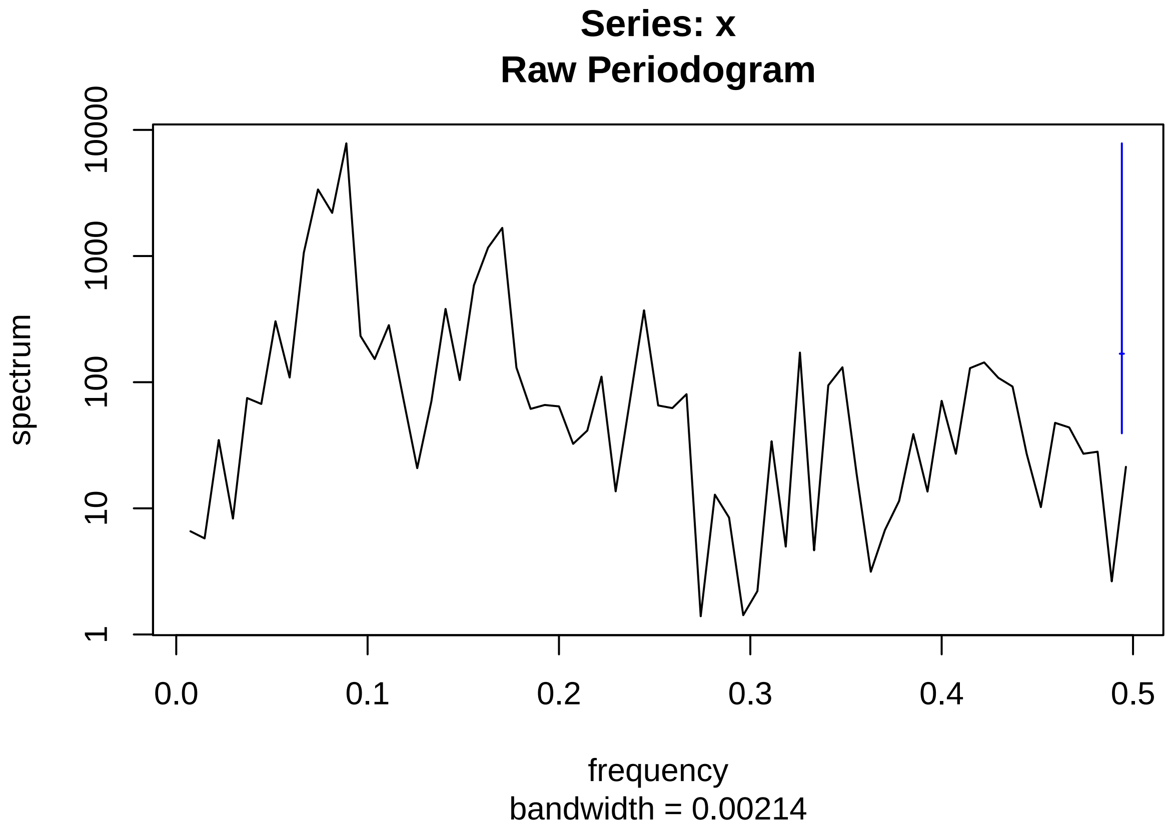
Question 18

(5 points)

Notes 10C (6)

Now plot the periodogram for the random component of your decomposition of the `air.passengers` time series. When you get rid of the `NA`s in the `random` vector, the time series element is lost. So here, after you get rid of the `NA`s, cast `random` back to a `ts`-class object, with the `start` being July 1949. Then pass `random` on... What do you observe? Compare the amplitudes of the peaks to the amplitudes you see above in the plot for Q17. Is there any visual evidence for a non-seasonal periodic component to the data?

```
random.comp = decompose(passenger.ts)$random
random.comp = random.comp[!is.na(random.comp)]
random.ts = ts(random.comp, start = c(1949, 7))
spectrum(random.ts)
```



The frequency is 10x smaller, the periodogram values have a much larger range. The peaks are shifted to the left by half, probably due to the different start month. The dominant peak happens around frequency = 0.1, which corresponds to a timescale of 10 months. There is evidence for non-seasonal periodic component to the data.