hw2code

Han-Yuan Hsu

2022-09-18

```r
set.seed(111)
```

Reference: lecture code.

1

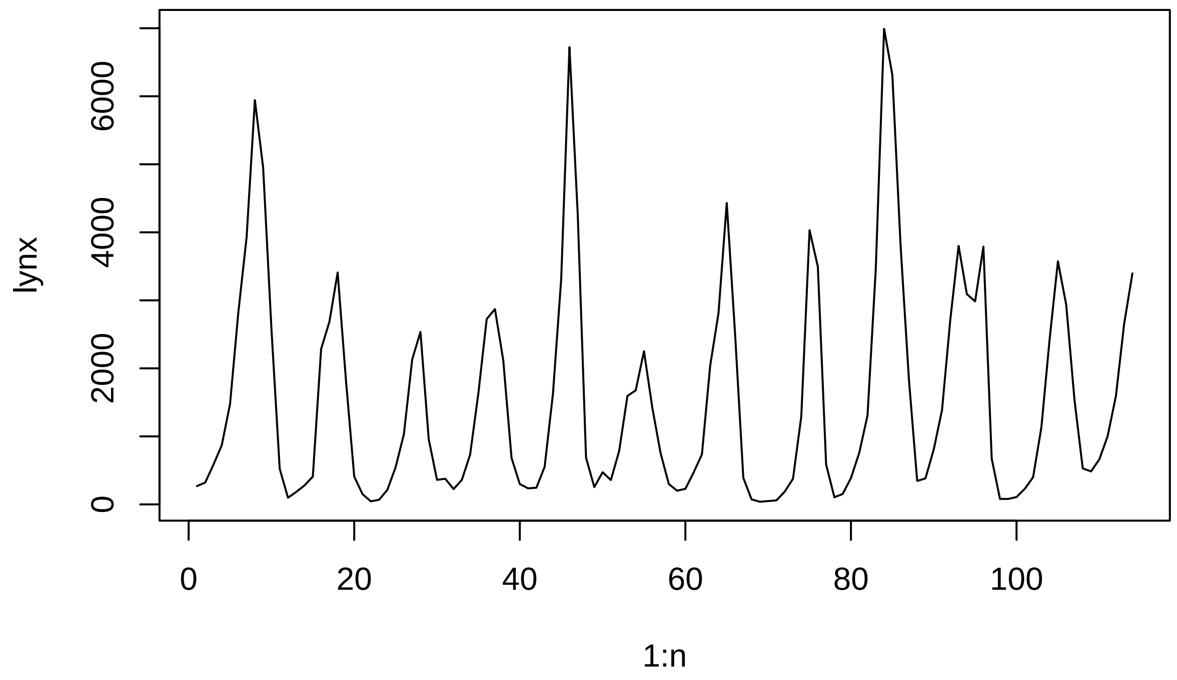
Data set 1 has periodogram B, and Data set 2 has periodogram A. The reason is that data set 2 looks oscillating, which suggests that the time series is a superposition of highly oscillating sine/cosine waves, namely sine/cosine waves with high frequencies. A periodogram tells us the amplitude of a wave with a given frequency, and in periodogram A, large amplitudes are on the high frequency side.

2

Consider the dataset lynx that is available in base R. This gives the annual numbers of lynx trappings for 1821-1934 in Canada. Type help(lynx) to learn more about the dataset.

Here is the plot of lynx time series:

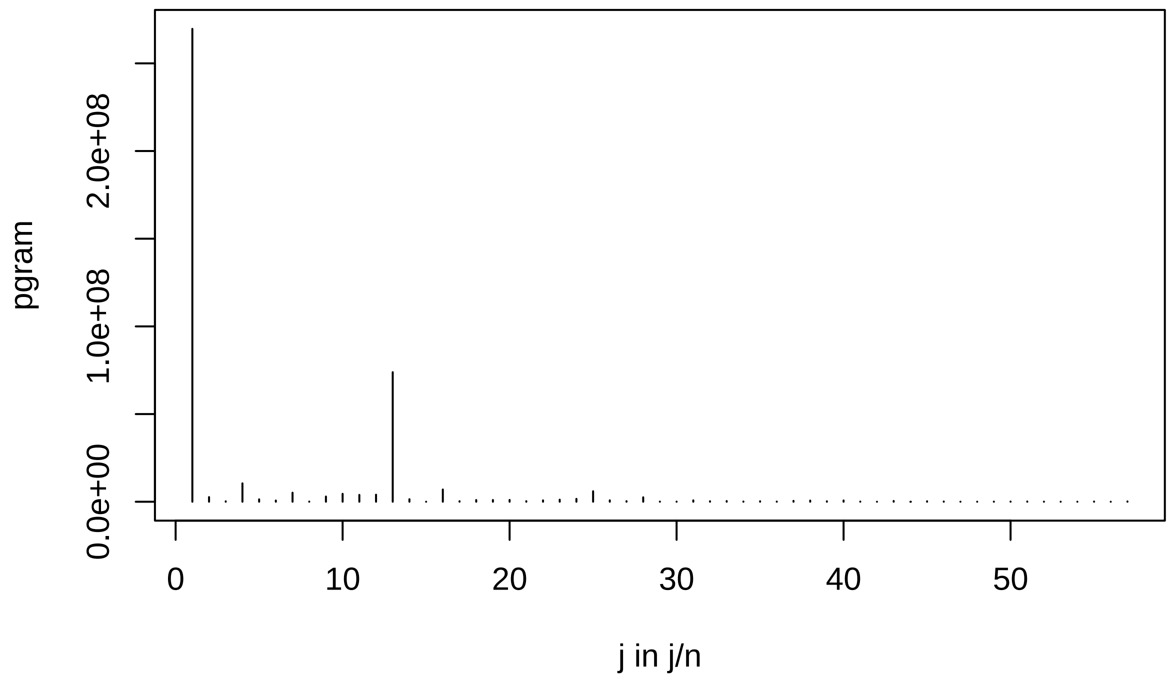
```r
n = length(lynx)
plot(1:n, lynx, type='l')
```



a

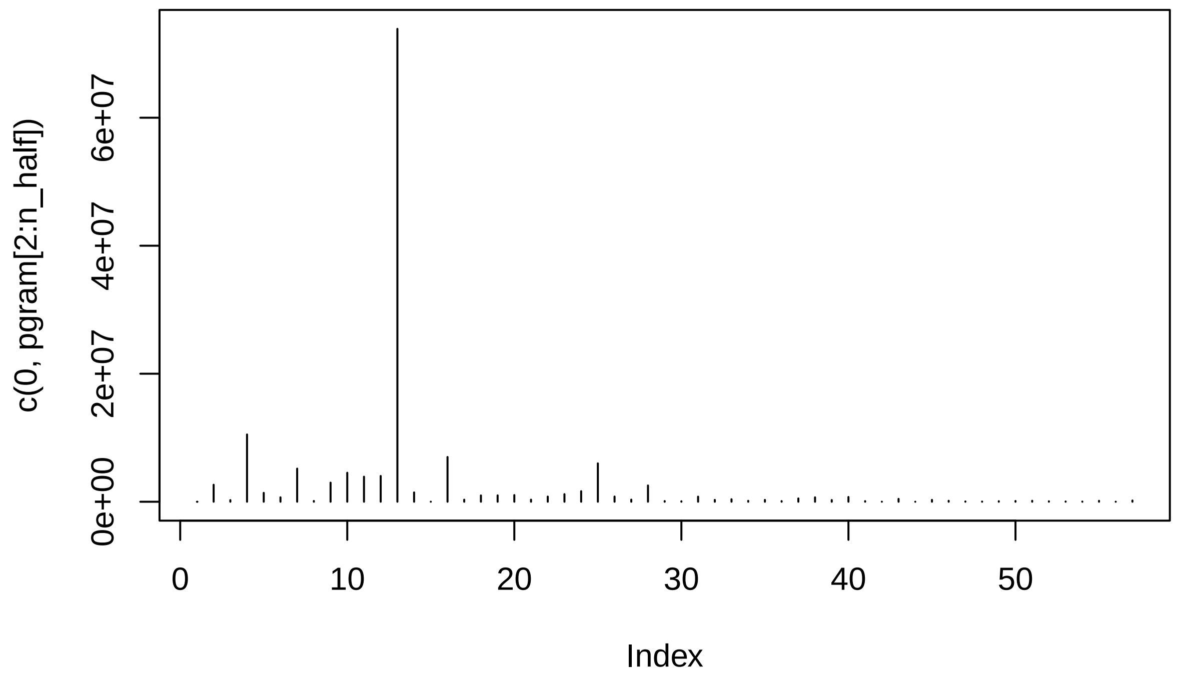
Plot the periodogram of the data and comment on its notable features. (3 points).

```
n_half = floor(n/2)
pgram = abs(fft(lynx)[1:n_half])^2 / n
plot(pgram, type='h', xlab='j in j/n')
```



The largest peak is at j=1, which corresponds to the $e^{2\pi i t \cdot 1/n}$ component. The period of this component is the total length of the time series, so it is not very interesting. Let's focus on the periodogram values when $j \geq 2$:

```
plot(c(0, pgram[2:n_half]), type='h')
```



The peak at 13 suggests that there is a seasonal pattern with period roughly

```r
n / 13
```

```
## [1] 8.769231
```

but the frequency need not be exactly 13/n, as we can see some leakage around the spike at j=13.

b

To this dataset, fit the model:

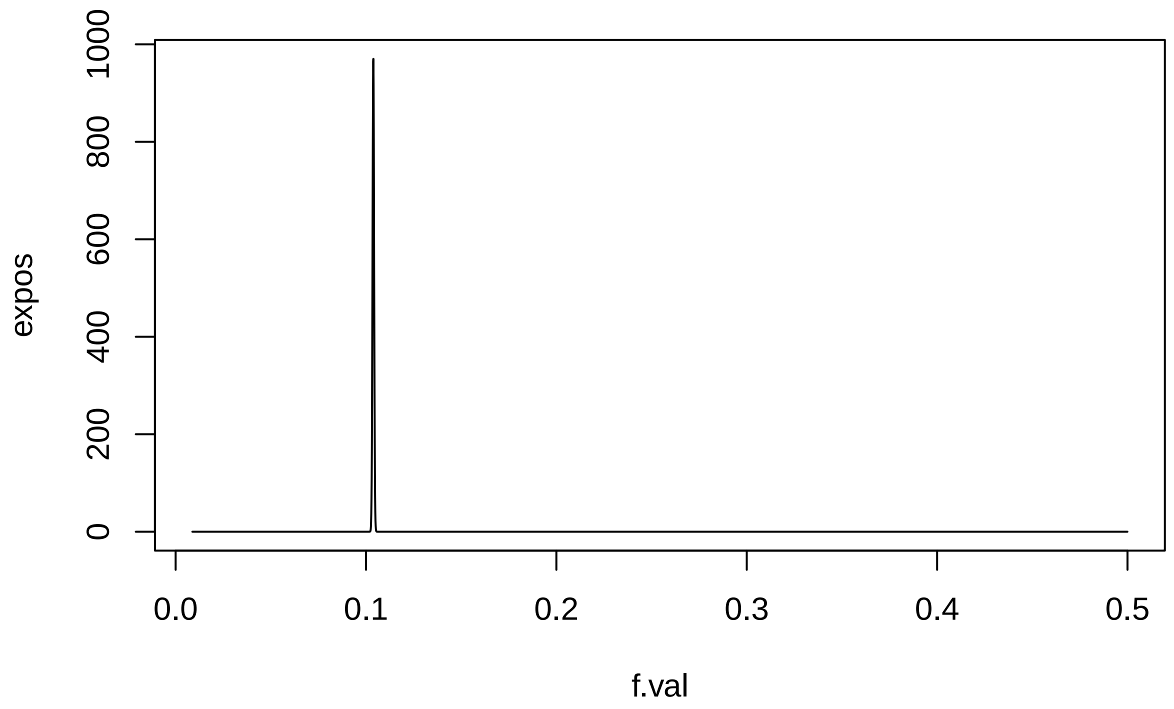
$$Y_t = \beta_0 + \beta_1 \cos(2\pi f t) + \beta_2 \sin(2\pi f t) + Z_t$$

with Z_t iid $\sim N(0, \sigma^2)$. Provide a point estimate and 95% uncertainty interval for the frequency f and the corresponding period of oscillation. (4 points).

Below, `expos` is the posterior distribution of the frequency f (f is discretized here):

```r
grid.res = 0.0001
f.val = seq(1/n, 0.5, grid.res)
X = matrix(1, nrow = n, ncol = 3)
#expos = rep(-1, length(f.val)) #exact marginal posterior for omega
log.values = rep(-1, length(f.val))
#log.det.term = rep(-1, length(f.val))
for(i in 1:length(f.val))
```

```
{
    X[,2] = cos(2*pi*f.val[i]*(1:n))
    X[,3] = sin(2*pi*f.val[i]*(1:n))
    mod = lm(lynx ~ -1 + X)
    log.values[i] = ((ncol(X) - n)/2)*(log(sum(mod$residuals^2))) - (0.5*(log(det(t(X) %*% X))))
}

log.values = log.values - max(log.values) #scaling to remove large values
expos = exp(log.values) # exponentiated posterior
expos = expos/sum(expos) / grid.res
plot(f.val, expos, type = "l")
```



From the plot, we see that the posterior distribution is quite peaked. We can use the mean of this distribution as a point estimate for f, which is `pm`. Since the posterior is quite peaked, it approximates a normal distribution, so we can get a rough confidence interval using the variance of the distribution:

```
f.hat = (sum(f.val*expos))*grid.res
f.hat
```

```
## [1] 0.1038321
```

```
#Posterior standard deviation:
psd = sqrt((sum(((f.val - f.hat)^2)*expos))*grid.res)
# rough 95% CI for f
c(f.hat - 1.96*psd, f.hat + 1.96*psd)
```

```
## [1] 0.1030213 0.1046430
```

The period is the reciprocal of frequency, so we easily get the period's point estimate and CI as follows:

```
1 / f.hat
```

```
## [1] 9.630931
```

```
c(1 / (f.hat + 2*psd), 1 / (f.hat - 2*psd))
```

```
## [1] 9.554791 9.708293
```

c

Provide point estimates and 95% marginal uncertainty intervals for β_0, β_1, β_2, and σ.

In the following code, `post.samples[ ,1:5]` stores samples of $f, \vec{\beta}$ from the joint posterior distribution $f, \vec{\beta} \mid \vec{y}$, and `post.samples[ ,c(1,6)]` stores samples of f, σ from the joint posterior distribution $f, \sigma \mid \vec{y}$. More explanation of the code is in 3b.

```
library(mvtnorm)
N = 2000 #number of posterior samples

# in post.samples, column 1 is for f sampled from f | y
# columns 2:5 is for samples from beta0, beta1, beta2 | f, y
# column 6 is for samples from sigma | f, y
post.samples = matrix(-1, N, 5)
post.samples[,1] = sample(f.val, N, replace = T, prob = expos) # iid samples of posterior f

# make columns 2:5
p = 3
X = matrix(1, nrow = n, ncol = p)
for(i in 1:N)
{
    f = post.samples[i, 1]
    X[,2] = cos(2*pi*f*(1:n))
    X[,3] = sin(2*pi*f*(1:n))
    lin.model = lm(lynx ~ -1 + X)
    bhat = lin.model$coefficients
    RSS = sum(lin.model$residuals^2)
    sighat = sqrt( RSS/(n-p) )
    Sigma.mat = (sighat^2)*solve(t(X) %*% X)
    chiran = (rchisq(1, df = n-p))
    beta.samples = bhat + (rmvnorm(1, sigma = Sigma.mat))/(sqrt(chiran/(n-p)))
    sig.sample = sqrt( RSS/chiran )

    post.samples[i,2:4] = beta.samples
    post.samples[i,5] = sig.sample
}
```

Below is a point estimate for β_0 (we use mean) and a 95% confidence interval:

```
b0 = mean(post.samples[,2])
b0
```

```
## [1] 1548.272
```

```
quantile(post.samples[,2], c(.025, .975))
```

```
##     2.5%    97.5%
## 1361.806 1744.905
```

Below is a point estimate for β_1 and a 95% confidence interval:

```
b1 = mean(post.samples[,3])
b1
```

```
## [1] 420.0992
```

```
quantile(post.samples[,3], c(.025, .975))
```

```
##      2.5%     97.5%
## -117.1379  921.8342
```

Below is a point estimate for β_2 and a 95% confidence interval:

```
b2 = mean(post.samples[,4])
b2
```

```
## [1] -1599.354
```

```
quantile(post.samples[,4], c(.025, .975))
```

```
##      2.5%     97.5%
## -1898.013 -1274.668
```

Below is a point estimate for σ and a 95% confidence interval:

```
b3 = mean(post.samples[,5])
b3
```

```
## [1] 1064.978
```

```
quantile(post.samples[,5], c(.025, .975))
```

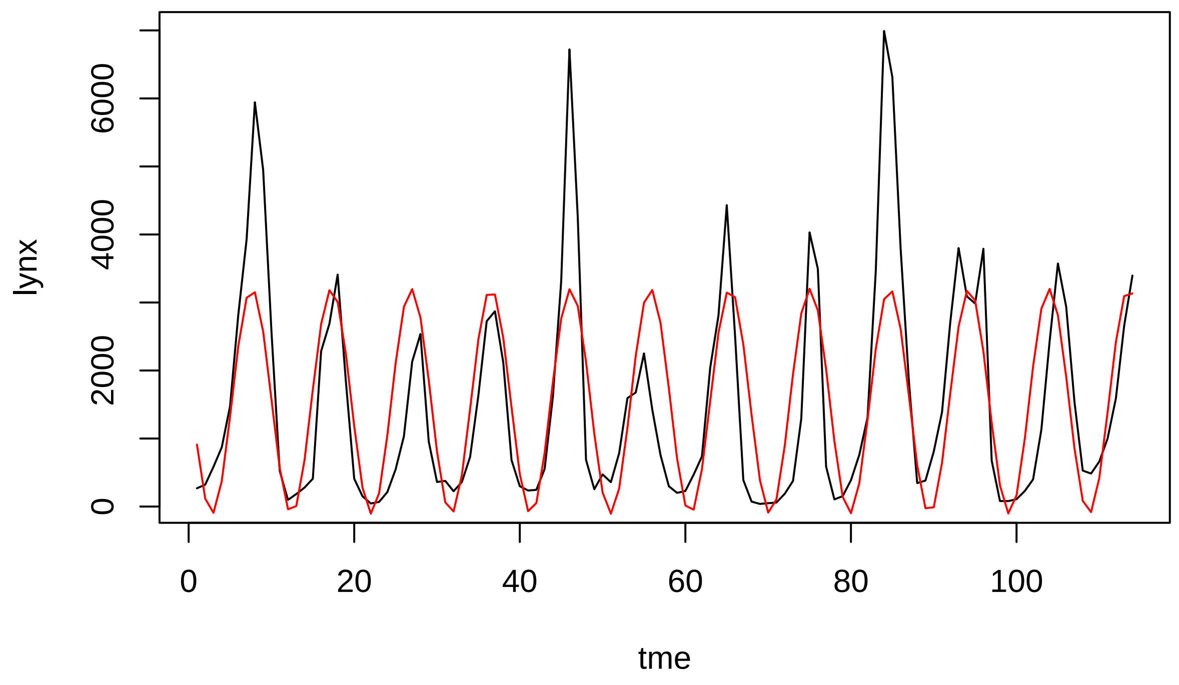
```
##      2.5%     97.5%
##  933.1694 1215.2318
```

d

Comment on whether this is a good model for this dataset. (2 points)

Plot the model output (in red) using the point estimates derived above and compare the model output with the actual time series:

```
tme = 1:n
model.output = rep(b0, n) + b1*cos(2*pi*f.hat*tme) + b2*sin(2*pi*f.hat*tme)
plot(tme, lynx, type='l')
lines(model.output, col='red')
```



The fit is reasonable. f.hat does capture the small-term cycles of the time series, and the bottom of the model output aligns with the bottom of the actual data. The biggest drawback, though, is that in the actual data, the peaks are noticeably bigger for every 4 cycles, and our model cannot reflect this pattern in the data because by design, the amplitude of the model is constant.

3

Download the Google Trends Data (for the United States) for the query mask. This should be a monthly time series dataset that indicates the search popularity of this query from January 2004 to September 2022. To this data, fit the single change point model:

$$Y_t = \beta_0 + \beta_1 I\{t > c\} + Z_t,$$

where Z_t are iid following $N(0, \sigma^2)$.

```
mask <- read.csv('mask.csv', skip=1, header=T)
head(mask)
```

```
##     Month mask...United.States.
## 1 2004-01                     4
## 2 2004-02                        4
## 3 2004-03                        4
## 4 2004-04                        4
## 5 2004-05                        4
## 6 2004-06                        4
```

```
n <- nrow(mask) # time series length, 225
t <- 1:n
y <- mask[,2] # values of the time series
```

a

Provide a point estimate and 95% uncertainty interval for the changepoint parameter c. Explain whether your answers make intuitive sense in the context of this dataset (4 points).

For the prior of our model, $\beta_0, \beta_1, \log(\sigma), c$ are independent; $\beta_0, \beta_1, \log(\sigma)$ follow $\text{Unif}([-C, C])$, where C is large; and the changepoint c is a discrete random variable and follows the uniform distribution on $\{3, 4, 5, ..., n-3\}$.

The following function, `log.post.c`, calculates the log of the posterior PMF of c (not normalized).

```
log.post.c <- function(c) {
  X = matrix(1, nrow = n, ncol = 2)
  X[,2] = (t > c)
  #bhat = solve(t(X) %*% X) %*% t(X) %*% y
  mod = lm(y ~ 1 + X[,2])
  # log of posterior pdf of c
  log.post = (ncol(X) - n)/2*log(sum(mod$residuals^2)) - 0.5*log(det(t(X) %*% X))
  log.post
}
```

Below, `values` is the vector of the posterior distribution of c, again not normalized:

```
c.vals <- c(3:(n-3))
log.values <- as.numeric(lapply(c.vals, FUN=log.post.c))
log.values <- log.values - max(log.values)
values <- exp(log.values)
```

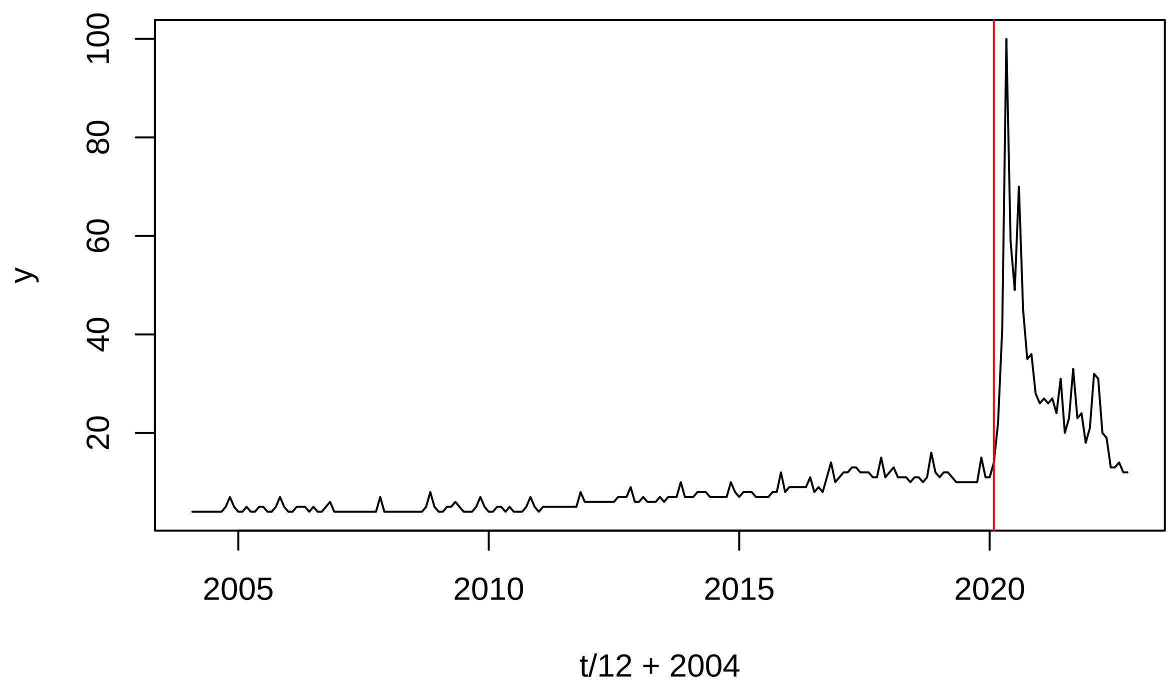
Below, `c.est` is the point estimate of c (maximum a posteriori estimator):

```
ind.max = which.max(log.values)
c.est <- c.vals[ind.max]
print(c.est)
```

```
## [1] 193
```

We can plot where `c.est` is (the red line):

```
plot(t/12 + 2004, y, type='l')
abline(v=c.est/12 + 2004, col='red')
```



The estimated changepoint is located right before the spike when Covid broke out and drastically increased the number of searches for "mask", so the estimated changepoint does make sense.

The code below calculates a 95% confidence interval centered at `c.est`:

```
total = sum(values)
d = 1
conf.total = values[c.est]
while(conf.total <= .95 * total) {
  conf.total <- conf.total + values[c.est-d] + values[c.est+d]
  d <- d + 1
}
c(c.est-d, c.est+d)
```

```
## [1] 189 197
```

b

Provide point estimates and 95% marginal uncertainty intervals for the prechangepoint mean level $\mu_0 := \beta_0$ and the post-changepoint mean level $\mu_1 := \beta_0 + \beta_1$ (4 points).

Let's first normalize `values` so that below, `post.pmf` is really the vector of the posterior distribution of c:

```
post.pmf = values / sum(values)
#plot(c.vals, post.pmf, type='l')
```

We will create N=2000 iid samples of the posterior $(\vec{\beta}, c)$. We achieve that by the posterior distribution of c and the distribution of $\vec{\beta}$ conditioned on the data $\vec{y}$ and c. We already know the posterior distribution of c from part a. We can also derive the distribution of $\vec{\beta} \,|\, \vec{y}, c$ as follows: the model becomes linear in $\vec{\beta}$ when c is fixed, and moreover, the prior distribution of $\vec{\beta}, \log \sigma$ conditioned on c is still uniform because c is assumed to be independent of $\vec{\beta}, \log \sigma$. Hence we can directly apply the results for linear models we learned in previous lectures and conclude that $\vec{\beta} \,|\, \vec{y}, c$ follows the multivariate t distribution

$$t_{n-2}\left(\hat{\beta},\ \hat{\sigma}^2 (X'X)^{-1}\right).$$

Note the parameters $\hat{\beta}$ and $\hat{\sigma}^2 (X'X)^{-1}$ now depend on c.

Thus, if a random variable C follows the distribution of the posterior c and a random vector $\vec{B}$ follows the multivariate t distribution above (which depends on C), then the joint, $(\vec{B}, C)$, will follow the distribution of the desired posterior $(\vec{\beta}, c)$ by the product rule of probability distributions.

The following code implements these ideas. `post.samples` is a matrix of the samples of (c, μ_0, μ_1), where $\mu_0 = \beta_0$ and $\mu_1 = \beta_0 + \beta_1$.

```
library(mvtnorm)
N = 2000 #number of posterior samples
post.samples = matrix(-1, N, 3)
post.samples[,1] = sample(c.vals, N, replace = T, prob = post.pmf) # iid samples of posterior c
for(i in 1:N)
{
    cp = post.samples[i,1] # changepoint

    # below, we sample one mu_0 and one mu_1 from the appropriate t distribution.
    # mu_samples is c(mu_0, mu_1)
    X = matrix(1, nrow = n, ncol = 2)
    X[,2] = (t > cp)
    lin.model = lm(y ~ 1 + X[,2])
    bhat = lin.model$coefficients
    sighat = sqrt((sum((lin.model$residuals)^2))/(n-2)) #this is also denoted by the Residual Standard
    Sigma.mat = (sighat^2)*solve(t(X) %*% X)
    chiran = (rchisq(1, df = n-2))
    mu.samples = bhat + (rmvnorm(1, sigma = Sigma.mat))/(sqrt(chiran/(n-2)))
    mu.samples[2] = mu.samples[1] + mu.samples[2]

    post.samples[i,2:3] = mu.samples
}
```

The estimate of μ_0 (using mean) and its 95% confidence interval are given below:

```
mean(post.samples[,2])
```

```
## [1] 6.943118
```

```
quantile(post.samples[, 2], c(.025, .975))
```

```
##     2.5%    97.5%
## 5.907497 7.994743
```

The estimate of μ_1 (using mean) and its 95% confidence interval are given below:

```r
mean(post.samples[,3])
```

```
## [1] 30.43881
```

```r
quantile(post.samples[, 3], c(.025, .975))
```

```
##     2.5%    97.5% 
## 27.77809 32.98934
```

c

Comment on whether (2) is a good model for this dataset. (2 points)

The changepoint model is not a good. After the huge spike in 2020, the values don't stay near a hypothetical mean but rather decrease. In other words, the model assumes the queries for "mask" will stay high forever since 2020, but in reality, we know that the spike was caused by Covid and the epidemic is getting alleviated, which will bring the interest for masks down. A skewed Gaussian model with the mode being an unknown parameter is a better model.

4

Download the Google Trends Data (for the United States) for the query golf. This should be a monthly time series dataset that indicates the search popularity of this query from January 2004 to September 2022. To this data, fit the model:

$$Y_t = \beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 \cos(2\pi f t) + \beta_4 \sin(2\pi f t) + Z_t$$

with Z_t iid following $N(0, \sigma^2)$.

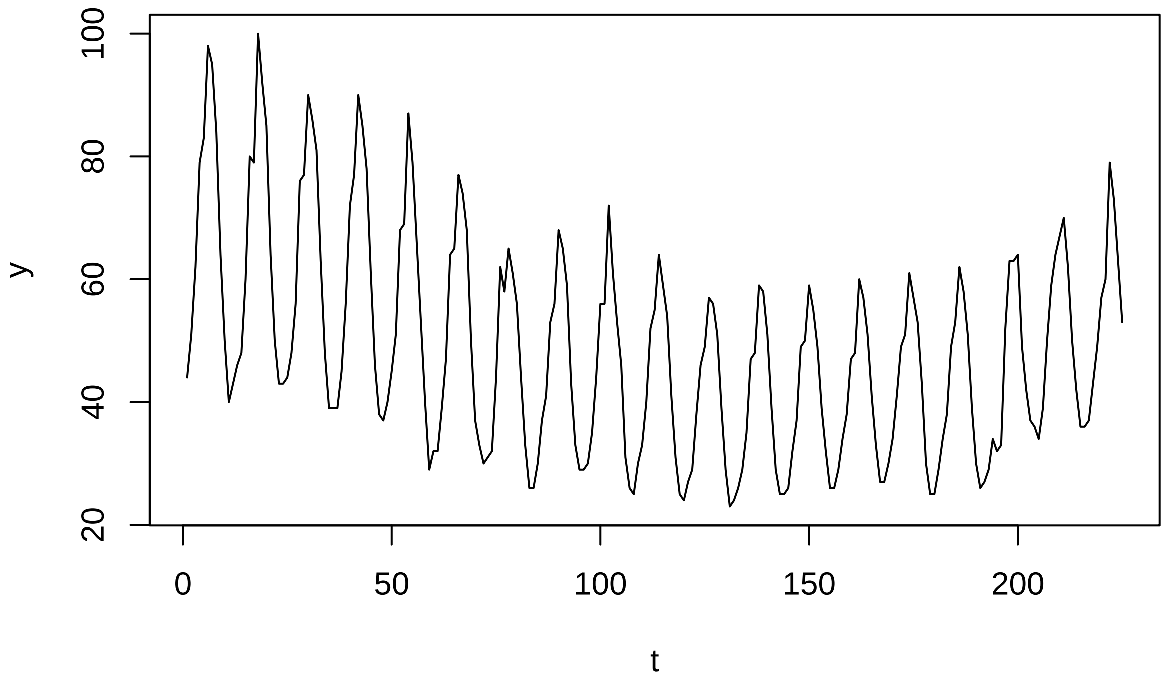
```r
golf <- read.csv('golf.csv', skip=1, header=T)
head(golf)
```

```
##     Month golf...United.States.
## 1 2004-01                    44
## 2 2004-02                    51
## 3 2004-03                    62
## 4 2004-04                    79
## 5 2004-05                    83
## 6 2004-06                    98
```

```r
y = golf[,2]
n = length(y)
t = 1:n
```

Below is the plot of the time series data:

```r
plot(t, y, type='l')
```

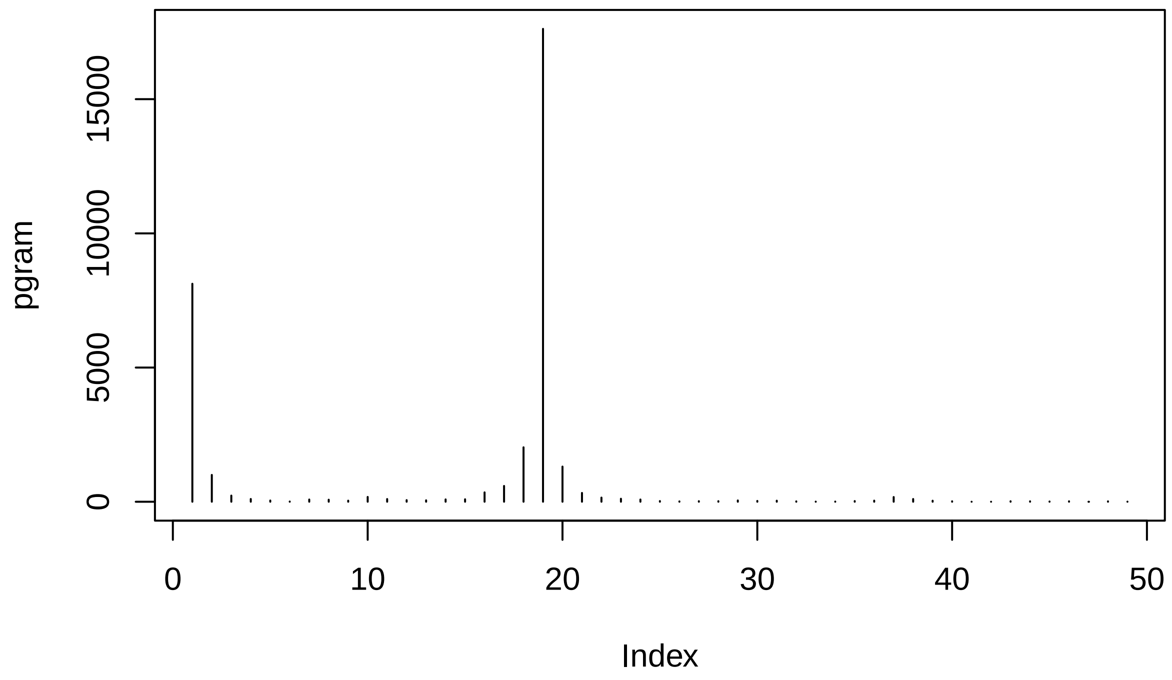


a

Provide a point estimate and a 95% uncertainty interval for the unknown frequency parameter f. (4 points)

To narrow down the search for the estimate for f, let's guess f through the periodogram of the data.

```
n_half = as.integer(n/2)
pgram = abs(fft(y)[2:50])^2 / n
plot(pgram, type='h') # peaks at 19
```



```r
225/19 # roughly 12. As expected, we have yearly trend
```

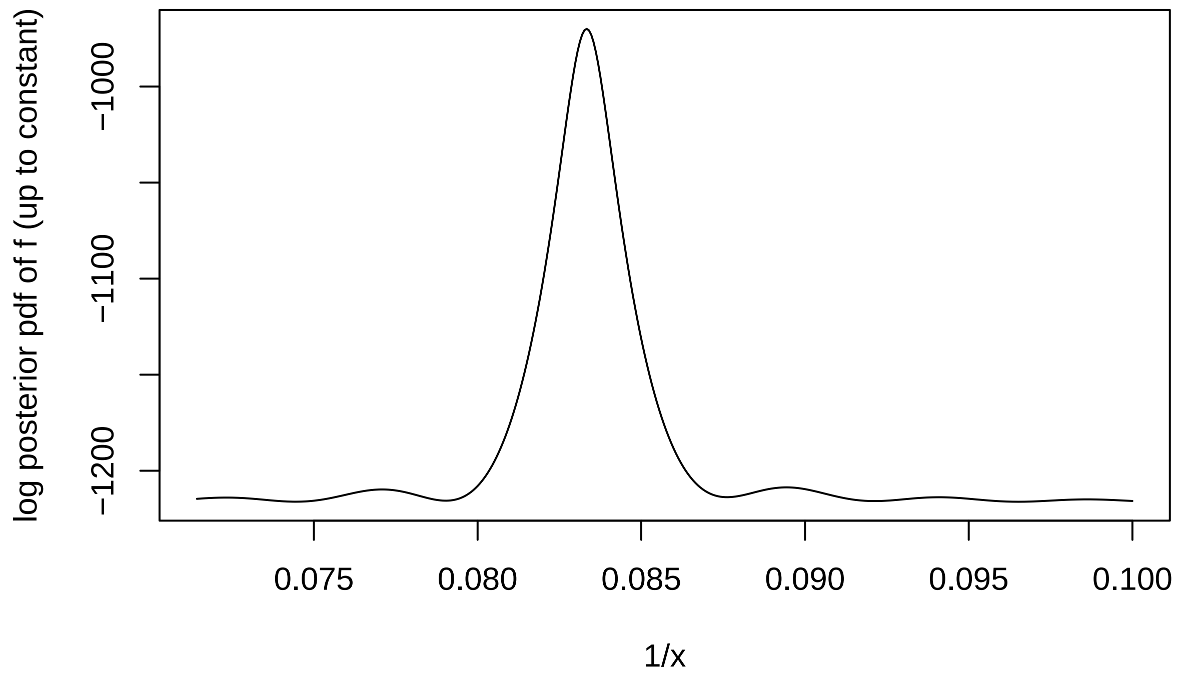
```
## [1] 11.84211
```

We conclude that f should be near 1/12.

```r
log.post.f = function(f) {
  X = matrix(1, nrow=n, ncol=5)
  X[,2] = t
  X[,3] = t^2
  X[,4] = cos(2*pi*f*t)
  X[,5] = sin(2*pi*f*t)

  lin.mod = lm(y ~ 1 + X[,2] + X[,3] + X[,4] + X[,5])

  -log(det(t(X) %*% X))/2 - log(sum((lin.mod$residuals)^2))*(n-5)/2
}
```

Plot the graph of log.post on the interval $(1/14, 1/12)$, which is a small neighborhood of our guess for f, 1/12:

```r
x = seq(10, 14, by=.01)
plot(1/x, sapply(1/x, FUN=log.post.f), type='l', ylab='log posterior pdf of f (up to constant)')
```



Our estimate for f, `f.hat`, will be the posterior maximizer:

```
opt = optimize(log.post.f, interval=c(.080, .085), maximum = T)
f.hat = opt$maximum
log.post.f.max = opt$objective
```

Note the reciprocal of f.hat is close to 12:

```
1/f.hat # very close to 12
```

```
## [1] 11.99865
```

Next, we calculate the posterior pdf of f from `log.post.f`. But we need the normalizing constant, which requires integrating over the range on which f is defined. We will grid f and calculate the Riemann sum to approximate such integration, and to achieve a finer grid resolution, we only integrate over the range (f.hat-.0005, f.hat+.0005) because as the code cell below shows, the posterior pdf of f is negligible outside this range:

```
log.post.f(f.hat-.0005) - log.post.f.max
```

```
## [1] -32.55008
```

```
log.post.f(f.hat+.0005) - log.post.f.max
```

```
## [1] -35.14118
```

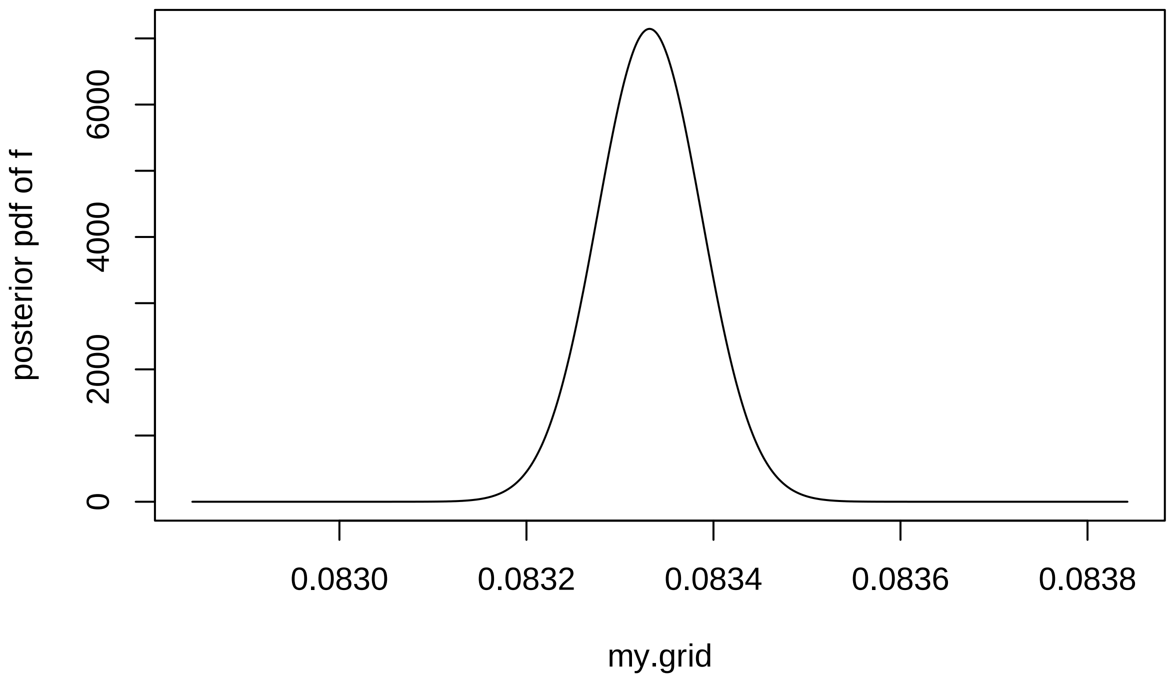
```
exp(-32.55008)
```

```
## [1] 7.306003e-15
```

```
post.f.0 = function(f) { # posterior f but not normalized
  exp(log.post.f(f) - log.post.f.max)
}

radius = .0005
res = radius/500 # resolution
my.grid = seq(f.hat-radius, f.hat+radius, by=res) # the posterior pdf of f is negligible outside this r
total.integral = 0

for (x in my.grid) {
  value = post.f.0(x)
  total.integral = total.integral + res*value
}

post.f = function(f) {
  post.f.0(f) / total.integral
}
```

Plot of posterior pdf of f:

```
post.f.grid = sapply(my.grid, FUN=post.f)
plot(my.grid, post.f.grid, type='l', ylab='posterior pdf of f')
```



95% confidence interval for f:

```
d = res
ci.integral = res*post.f(f.hat)
while (ci.integral <= .95) {
  ci.integral = ci.integral + res*post.f(f.hat-d) + res*post.f(f.hat+d)
  d = d + res
}
c(f.hat-d, f.hat+d)
```

```
## [1] 0.08322971 0.08345571
```

b

On a scatter plot of the data, plot your best estimate of the fitted function:

$$t \mapsto \beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 \cos(2\pi f t) + \beta_4 \sin(2\pi f t)$$

along with appropriate uncertainty quantification. (5 points).

The following code generates `N` samples of f and $\vec{\beta}$ from the joint posterior distribution $f, \vec{\beta} \mid \vec{y}$. The reasoning is similar to 3b.

```
library(mvtnorm)
N = 2000 #number of posterior samples
```

```r
# in post.samples, column 1 is for f sampled from f | y
# columns 2:6 is for samples from beta0, ..., beta4 | f, y
post.samples = matrix(-1, N, 6)
post.samples[,1] = sample(my.grid, N, replace = T, prob = post.f.grid)

# make columns 2:6
p = 5
X = matrix(1, nrow = n, ncol = p)
for(i in 1:N)
{
    f = post.samples[i, 1]

    X[,2] = 1:n
    X[,3] = (1:n)^2
    X[,4] = cos(2*pi*f*(1:n))
    X[,5] = sin(2*pi*f*(1:n))
    lin.model = lm(y ~ -1 + X)
    bhat = lin.model$coefficients
    RSS = sum(lin.model$residuals^2)
    sighat = sqrt( RSS/(n-p) )
    Sigma.mat = (sighat^2)*solve(t(X) %*% X)
    chiran = (rchisq(1, df = n-p))
    beta.samples = bhat + (rmvnorm(1, sigma = Sigma.mat))/(sqrt(chiran/(n-p)))

    post.samples[i, 2:(1+p)] = beta.samples
}
```

Below is a point estimate for β_0 (we use mean) and a 95% confidence interval:

```r
b0 = mean(post.samples[,2])
b0
```

```
## [1] 73.58729
```

```r
quantile(post.samples[,2], c(.025, .975))
```

```
##     2.5%    97.5%
## 71.54429 75.81616
```

Below is a point estimate for β_1 and a 95% confidence interval:

```r
b1 = mean(post.samples[,3])
b1
```

```
## [1] -0.4647624
```

```r
quantile(post.samples[,3], c(.025, .975))
```

```
##       2.5%      97.5%
## -0.5091333 -0.4225799
```

Below is a point estimate for β_2 and a 95% confidence interval:

```
b2 = mean(post.samples[,4])
b2
```

```
## [1] 0.001629898
```

```
quantile(post.samples[,4], c(.025, .975))
```

```
##        2.5%       97.5%
## 0.001452690 0.001810604
```

Below is a point estimate for β_3 and a 95% confidence interval:

```
b3 = mean(post.samples[,5])
b3
```

```
## [1] -19.93291
```

```
quantile(post.samples[,5], c(.025, .975))
```

```
##      2.5%     97.5%
## -20.86454 -19.02254
```

Below is a point estimate for β_4 and a 95% confidence interval:

```
b4 = mean(post.samples[,6])
b4
```

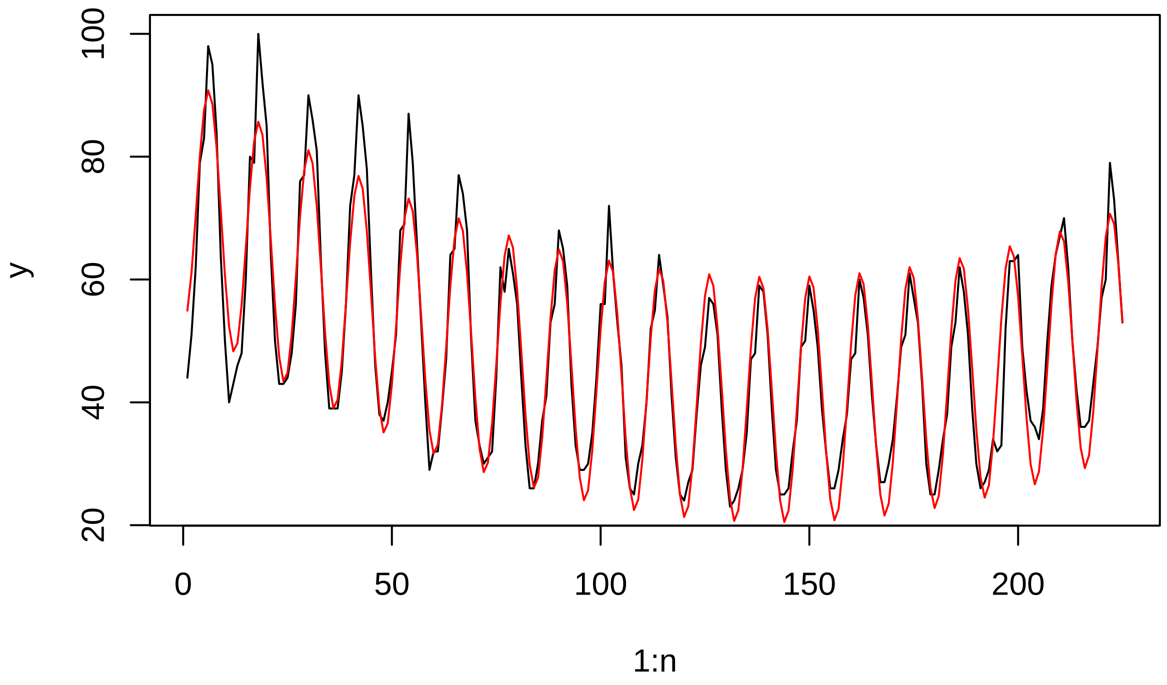
```
## [1] -1.870974
```

```
quantile(post.samples[,6], c(.025, .975))
```

```
##       2.5%      97.5%
## -3.6270520 -0.2051877
```

Plot of my best estimate of the fitted function (in red):

```
model.output = b0*rep(1,n) + b1*(1:n) + b2*(1:n)^2 + b3*cos(2*pi*f.hat*(1:n)) + b4*sin(2*pi*f.hat*(1:n)
plot(1:n, y, type='l')
lines(model.output, col='red')
```



c

Comment on whether model (3) is appropriate for this dataset. (2 points).

The fit looks good from the plot, but again I don't think the model is appropriate for forecasting because of the quadratic part. Since the time series should stay bounded for all times, the curvy, convex long-term trend of the data should just be part of a sine wave with a larger period instead of a quadratic curve.

5

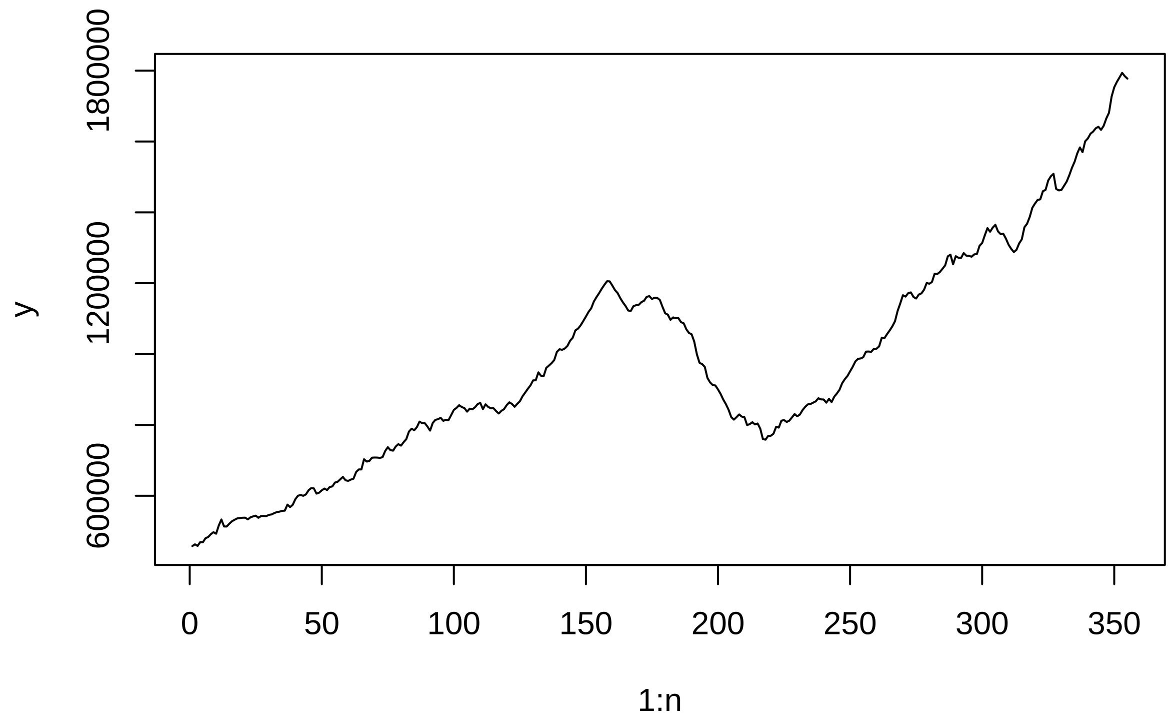
Download the FRED dataset on Total Construction Spending in the United States from https://fred.stlouisfed.org/series/TTLCONS. This gives monthly seasonally adjusted data on total construction spending in the United States in millions of dollars from January 1993 to July 2022. To this dataset, fit the model:

$$Y_t = \beta_0 + \beta_1 t + \beta_2 (t - s_1)_+ + \beta_3 (t - s_2)_+ + Z_t$$

with Z_t i.i.d $\sim N(0, \sigma^2)$.

```
df = read.csv('TTLCONS.csv')
y = df$TTLCONS
n = length(y)
```

```r
plot(1:n, y, type='l')
```



a

Provide point estimates and 95% uncertainty intervals for the change of slope parameters s_1 and s_2. (5 points)

For the prior of our model, $\beta_0, \beta_1, \log(\sigma)$ and the random vector $\vec{s} = (s_1, s_2)$ are independent; $\beta_0, \beta_1, \log(\sigma)$ follow Unif$([-C, C])$, where C is large; and $\vec{s}$ is a discrete random vector following the uniform distribution on

$$\{(s_1, s_2) \in \mathbb{N}^2 : \ 3 \leq s_1 < s_2 \leq n - 3\}.$$

The following function, `logpost`, calculates the log of the posterior PMF of $\vec{s}$ (not normalized).

```r
#cps: change of slope points s_1, s_2.
logpost = function(cps)
{
    cps = sort(cps)
    k = length(cps)
    X = matrix(1, nrow = n, ncol = (k+2))
    X[,2] = 1:n
    for(j in 1:k)
    {
        X[,(j+2)] = pmax(c(1:n) - cps[j], 0)
    }
    mod = lm(y ~ -1 + X) # -1 is needed because the default provides an intercept term, but
    # X already contains the intercept term
    log.value = ((ncol(X) - n)/2)*(log(sum(mod$residuals^2))) - (0.5*(log(det(t(X) %*% X))))
    return(log.value)
}
```

Below, `logpost.max` is the max of `logpost`, `s.hat` is where the max of `logpost` (hence the MAP estimator for $\vec{s}$).

```
logpost.matrix = matrix(0, nrow=n-3, ncol=n-3) # saves logpost(c(s1, s2)) so that we don't need
# to compute again

logpost.max = logpost(c(3,4))
s.hat = c(0, 0) # MAP estimator for s
# the nested for loops take 20 seconds
for (s1 in 3:(n-3)) {
  if (s1+1 > n-3) {
    # then no need to go to the second for loop
    # need this if statement, or (s1+1):(n-3) will cause bug when s1+1 > n-3
    break
  }
  for (s2 in (s1+1):(n-3)) {
    m = logpost(c(s1, s2))
    logpost.matrix[s1, s2] = m
    if (m > logpost.max) {
      logpost.max = m
      s.hat[1] = s1
      s.hat[2] = s2
    }
  }
}
```

`s.hat`:

```
s.hat
```

```
## [1] 177 217
```

Below, `post.s` is the distribution of $\vec{s}$ (normalized), `post.s1` is the posterior distribution of s_1 (normalized), and `post.s1` is the posterior distribution of s_2 (normalized).

```
post.s = matrix(0, nrow=n-3, ncol=n-3)
for (s1 in 3:(n-3)) {
  if (s1+1 > n-3) {
    # then no need to go to the second for loop
    # need this if statement, or (s1+1):(n-3) will cause bug when s1+1 > n-3
    break
  }
  for (s2 in (s1+1):(n-3)) {
    post.s[s1, s2] = exp(logpost.matrix[s1, s2] - logpost.max)
  }
}
# normalize
post.s = post.s / sum(post.s)
```

```r
post.s1 = apply(post.s, MARGIN=1, sum)
post.s2 = apply(post.s, MARGIN=2, sum)
# normalize
post.s1 = post.s1 / sum(post.s1)
post.s2 = post.s2 / sum(post.s2)
```

95% CI for s_1:

```r
d = 1
conf.total = post.s1[s.hat[1]]
while(conf.total <= .95) {
  conf.total <- conf.total + post.s1[s.hat[1]-d] + post.s1[s.hat[1]+d]
  d <- d + 1
}
c(s.hat[1]-d, s.hat[1]+d)
```

```
## [1] 174 180
```

95% CI for s_2:

```r
d = 1
conf.total = post.s2[s.hat[2]]
while(conf.total <= .95) {
  conf.total <- conf.total + post.s2[s.hat[2]-d] + post.s2[s.hat[2]+d]
  d <- d + 1
}
c(s.hat[2]-d, s.hat[2]+d)
```

```
## [1] 214 220
```

b

On a scatter plot of the data, plot your best estimate of the fitted function along with appropriate uncertainty quantification. (5 points).

The following code generates `N` samples of $\vec{s}$ and $\vec{\beta}$ from the joint posterior distribution $\vec{s}, \vec{\beta} \mid \vec{y}$. The reasoning is similar to 3b.

```r
N = 2000 #number of posterior samples
# in post.samples, column 1, 2 is for samples from the distribution (s_1, s_2) | y
# column 3, 4, 5, 6 is for samples from (beta_0, ..., beta_3) | s_1, s_2, y
#
# equivalently, column 1 is for samples from s_1 | y
# and column 2 is for samples from s_2 | s_1, y
post.samples = matrix(-1, nrow=N, ncol=6)
post.samples[,1] = sample(3:(n-3), N, replace = T, prob = post.s1[3:(n-3)])

# make column 2
# s2.cond(s1) returns one sample from s_2 | s_1, y
s2.cond = function(s1) {
  s2.range = (s1+1):(n-3)
  cond.prob = post.s[s1, s2.range] / post.s1[s1] # conditional probability distribution represented by
  sample(s2.range, size=1, prob=cond.prob)
}
post.samples[,2] = sapply(post.samples[,1], FUN=s2.cond)

# make columns 3, 4, 5, 6
# column 3, 4, 5, 6 is for samples from (beta_0, ..., beta_3) | s_1, s_2, y, which is multivariate t
for(i in 1:N)
{
    p = 4

    s1 = post.samples[i,1]
    s2 = post.samples[i,2]
    X = matrix(1, nrow = n, ncol = p)
    X[,2] = 1:n
    X[,3] = pmax(c(1:n) - s1, 0)
    X[,4] = pmax(c(1:n) - s2, 0)
    lin.model = lm(y ~ -1 + X)
    bhat = lin.model$coefficients
    sighat = sqrt((sum((lin.model$residuals)^2))/(n-p))
    Sigma.mat = (sighat^2)*solve(t(X) %*% X)
    chiran = rchisq(1, df = n-p)
    beta.samples = bhat + (rmvnorm(1, sigma = Sigma.mat))/(sqrt(chiran/(n-p)))
    post.samples[i, 3:6] = beta.samples
}
```

Our best estimates for posterior $\vec{s}$ and $\vec{\beta}$ are `s.hat` and the mean of the distribution $\vec{\beta} \mid \vec{s}, y$, which is

$$\hat{\beta} = (X^\top X)^{-1} X^\top \vec{y},$$

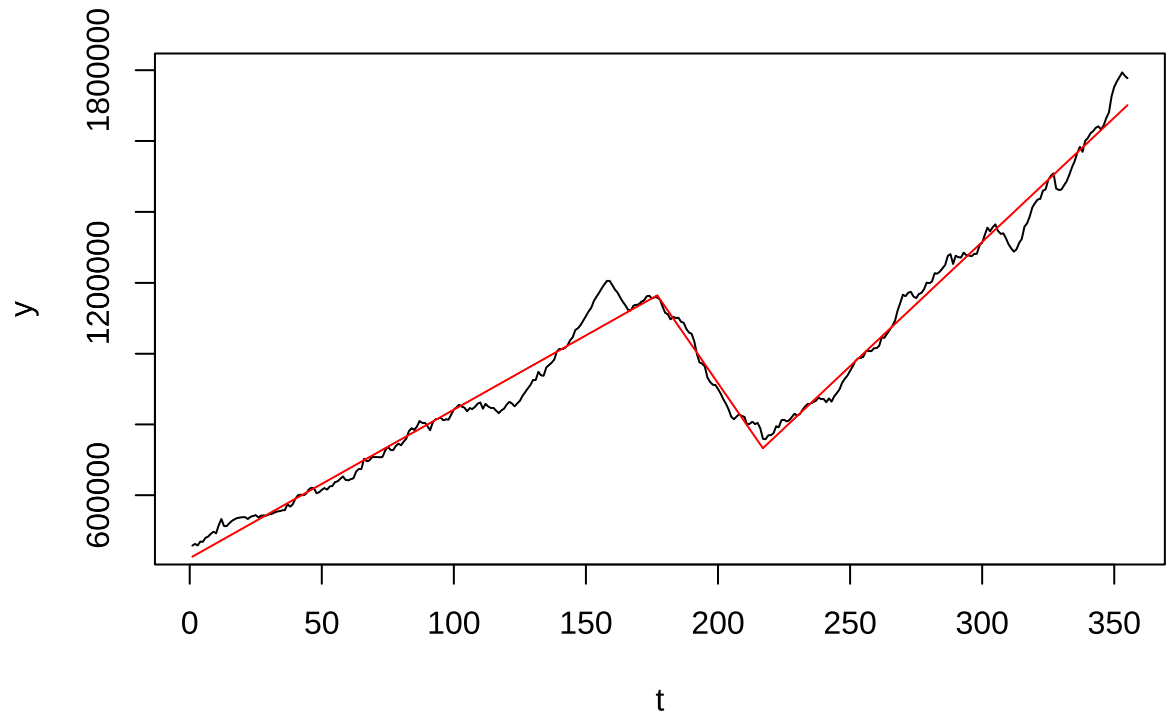
where $X = X(\hat{s})$ depends on `s.hat`.

```r
# get beta hat
X = matrix(1, nrow = n, ncol = p)
X[,2] = 1:n
X[,3] = pmax(c(1:n) - s.hat[1], 0)
X[,4] = pmax(c(1:n) - s.hat[2], 0)
lin.model = lm(y ~ -1 + X)
best.bhat = lin.model$coefficients
best.bhat
```

```
##         X1          X2          X3          X4
## 422638.988   4191.962 -14982.082  17805.898
```

In the following plot, the black graph is the actual values of the time series, and the red graph is the fitted function.

```r
plot(1:n, y, type='l', xlab='t')
lines(lin.model$fitted.values, col='red')
```



We use the beta samples in `post.samples` to estimate 95% confidence intervals for posterior $\beta_0, \beta_1, \beta_2$, and β_3.

95% CI for β_0:

```
quantile(post.samples[,3], c(.025, .975))
```

```
##     2.5%    97.5%
## 410875.2 433174.3
```

95% CI for β_1:

```
quantile(post.samples[,4], c(.025, .975))
```

```
##     2.5%    97.5%
## 4090.977 4316.997
```

95% CI for β_2:

```
quantile(post.samples[,5], c(.025, .975))
```

```
##      2.5%     97.5%
## -15809.25 -13702.87
```

95% CI for β_3:

```r
quantile(post.samples[,6], c(.025, .975))
```

```
##     2.5%    97.5%
## 16517.90 18621.14
```

c

Comment on whether the model is appropriate for this dataset. (2 points).

Visually, the model fits the data well. But in terms of forecasting, this model is bad; its future predictions will all be linear, while in reality, another change-of-slope point may occur.

6

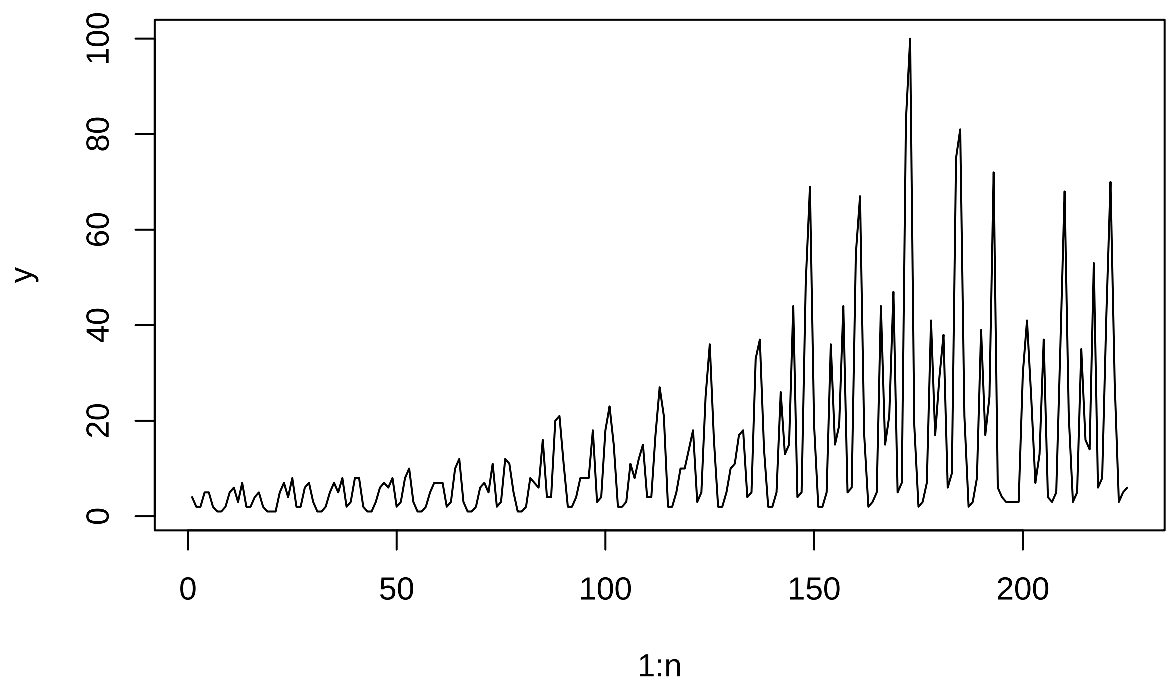
Download the google trends time series dataset for the query playoffs (download the trends for the United States and not worldwide). This should be a monthly time series dataset that indicates the search popularity of this query from January 2004 to September 2022.

```r
dat = read.csv('playoffs.csv', skip=1, header=T)
y = dat[,2]
n = length(y)
```

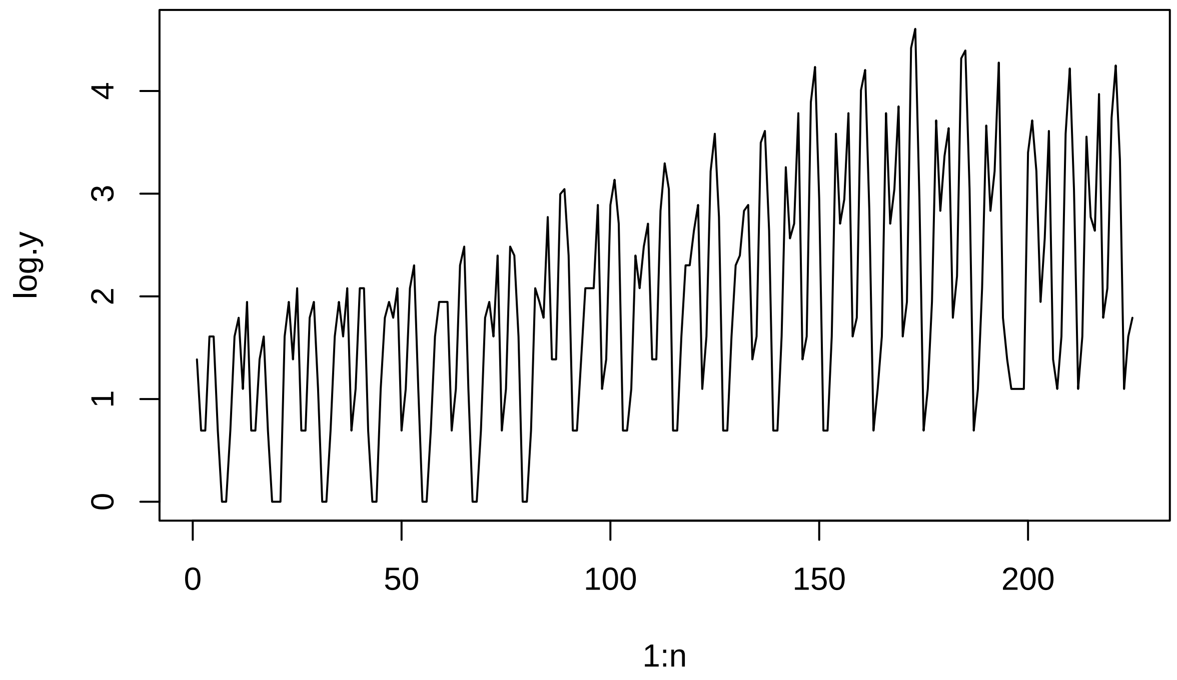
a

Plot the data and observe that the scale of variability increases with time. To fix this, take the logarithm of the data. Plot the logarithm and comment on whether the variability can now be assumed to be constant over time. (2 points)

```r
plot(1:n, y, type='l')
```



```
log.y = log(y)
plot(1:n, log.y, type='l')
```

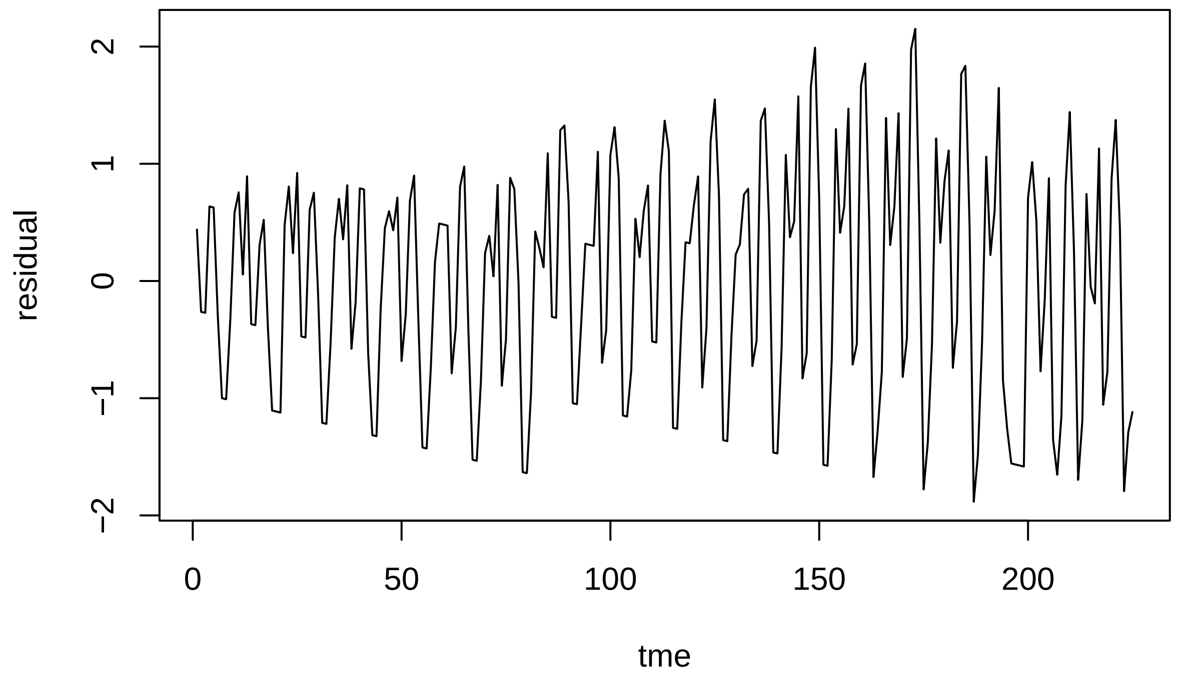


The variability is still increasing a little bit, but it is small. After all, taking logarithm cannot completely wipe off something that is increasing, but it can significantly slow down the increasing rate.

b

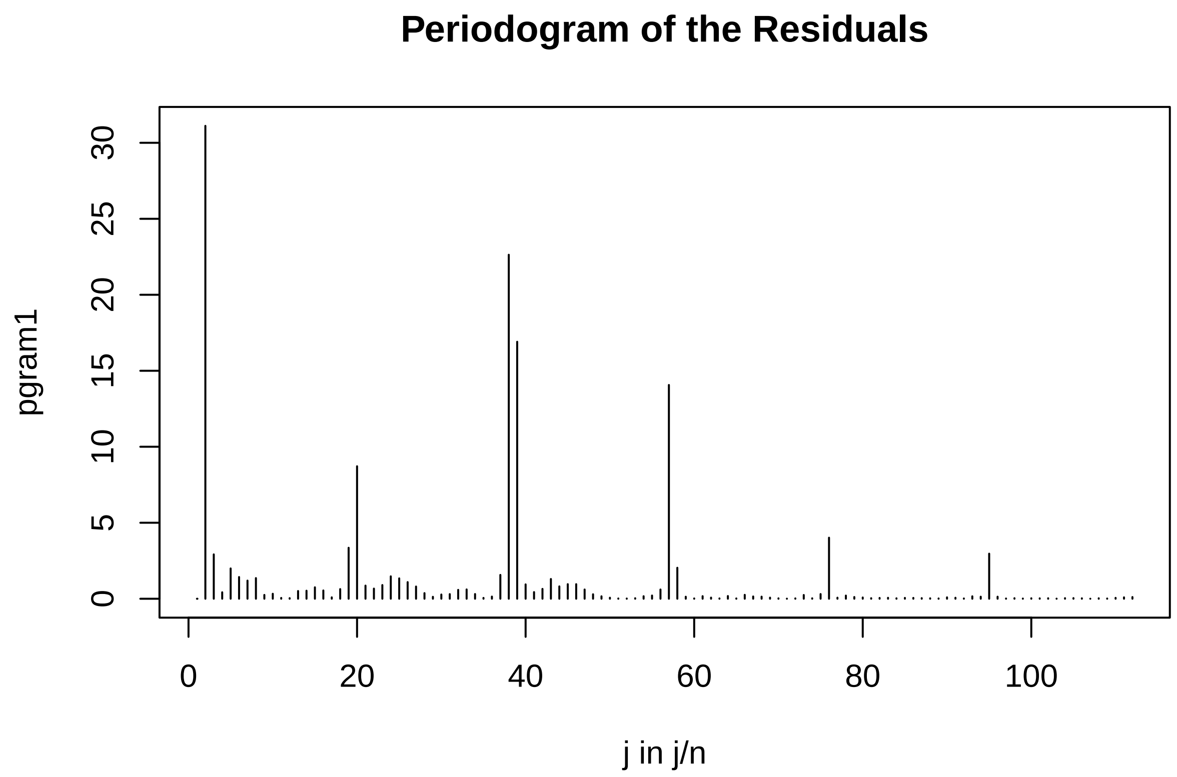
The logarithmed data have an increasing trend and a periodic component superimposed on the trend. To get an idea of the frequencies present in the periodic component, fit a linear trend model to the logarithmed data, compute the residuals and then plot the periodogram of the residuals. Comment on the location and size of the main spikes in the periodogram. (3 points)

```r
tme = 1:n
lin = lm(log.y ~ 1 + tme)
# the residuals are simply given by lin$residuals
plot(tme, lin$residuals, type='l', ylab='residual')
```



Periodogram for residuals:

```r
n_half = floor(n/2)
pgram = abs(fft(log.y)[1:n_half])^2 / n
# if you plot pgram, the first entry (j=1) is so huge that other spikes are hardly noticeable;
# hence, we set the first entry to be 0 in pgram1 and plot pgram1 instead
#plot(pgram, type='h', xlab='j in j/n')
pgram1 = pgram; pgram1[1] = 0
plot(pgram1, type='h', xlab='j in j/n', main='Periodogram of the Residuals')
```



The main spikes occur at j=20, 38, 57, 76, 95, which are multiples of 19 (except 20, but almost). This suggests the main period of the data is n/19:

```
n/19
```

```
## [1] 11.84211
```

The magnitude of the spike (excluding those near j=1) is largest when j=38.

c

To the logarithmed data, fit the model

$$y_i = g(t_i) + Z_i$$

where Z_i are iid $\sim N(0, \sigma^2)$ and

$$g(t) = \beta_0 + \beta_1 t + \beta_2 \cos(2\pi f_1 t) + \beta_3 \sin(2\pi f_1 t) + \beta_4 \cos(2\pi f_2 t) + \beta_5 \sin(2\pi f_2 t) + \beta_6 \cos(2\pi f_3 t) + \beta_7 \sin(2\pi f_3 t).$$

Treat the betas, f's, and sigma as unknown parameters and estimate them from data. Plot your estimate of g (and appropriate uncertainty indicators) along with the actual data. (6 points)

Reference: lecture 7 code.

```
tme = 1:n
p = 8
X = matrix(1, nrow = n, ncol = p)
```

```
logpost = function(fs) # fs: a vector of f_1, f_2, f_3
{
    f_1 = fs[1]
    f_2 = fs[2]
    f_3 = fs[3]
    X[,2] = tme
    X[,3] = cos(2*pi*f_1*tme)
    X[,4] = sin(2*pi*f_1*tme)
    X[,5] = cos(2*pi*f_2*tme)
    X[,6] = sin(2*pi*f_2*tme)
    X[,7] = cos(2*pi*f_3*tme)
    X[,8] = sin(2*pi*f_3*tme)
    mod = lm(log.y ~ -1 + X)
    log.value = ((p - n)/2)*(log(sum(mod$residuals^2))) - (0.5*(log(det(t(X) %*% X))))
    return(log.value)
}
```

We need to discretize the frequencies f_1, f_2, f_3 to do Bayesian inference. From the periodogram, we see that a good range of the frequencies to consider is from 10/n to 60/n. Because my computer is not fast, the grid used here is quite crude.

```
f.cg = combn(seq(10/n, 60/n, by=2/n), m=3) # cg means crude grid
lp.cg = apply(f.cg, MARGIN=2, FUN=logpost) # log posterior for crude grid
```

Estimate the frequencies by posterior maximizer:

```
ind.max.cg = which.max(lp.cg)
f.hat.cg = f.cg[ ,ind.max.cg]
f.hat.cg
```

```
## [1] 0.0800000 0.1688889 0.2488889
```

Make samples of posterior f1, f2, f3, stored in `post.f.sample`:

```
N = 2000
post.f = exp(lp.cg - max(lp.cg))
post.f = post.f / sum(post.f)
post.f.ind.sample = sample(1:ncol(f.cg), N, replace=T, prob=post.f)
post.f.sample = f.cg[ ,post.f.ind.sample]
```

95% CI for f_1, f_2, and f_3:

```
quantile(post.f.sample[1,], c(.025, .975))
```

```
##       2.5%      97.5% 
## 0.06222222 0.16888889
```

```
quantile(post.f.sample[2,], c(.025, .975))
```

```
##      2.5%     97.5% 
## 0.1688889 0.2488889
```

```
quantile(post.f.sample[3,], c(.025, .975))
```

```
##      2.5%     97.5%
## 0.2488889 0.2577778
```

```
sd(post.f.sample[1,])
```

```
## [1] 0.03889233
```

```
sd(post.f.sample[2,])
```

```
## [1] 0.01828346
```

```
sd(post.f.sample[3,])
```

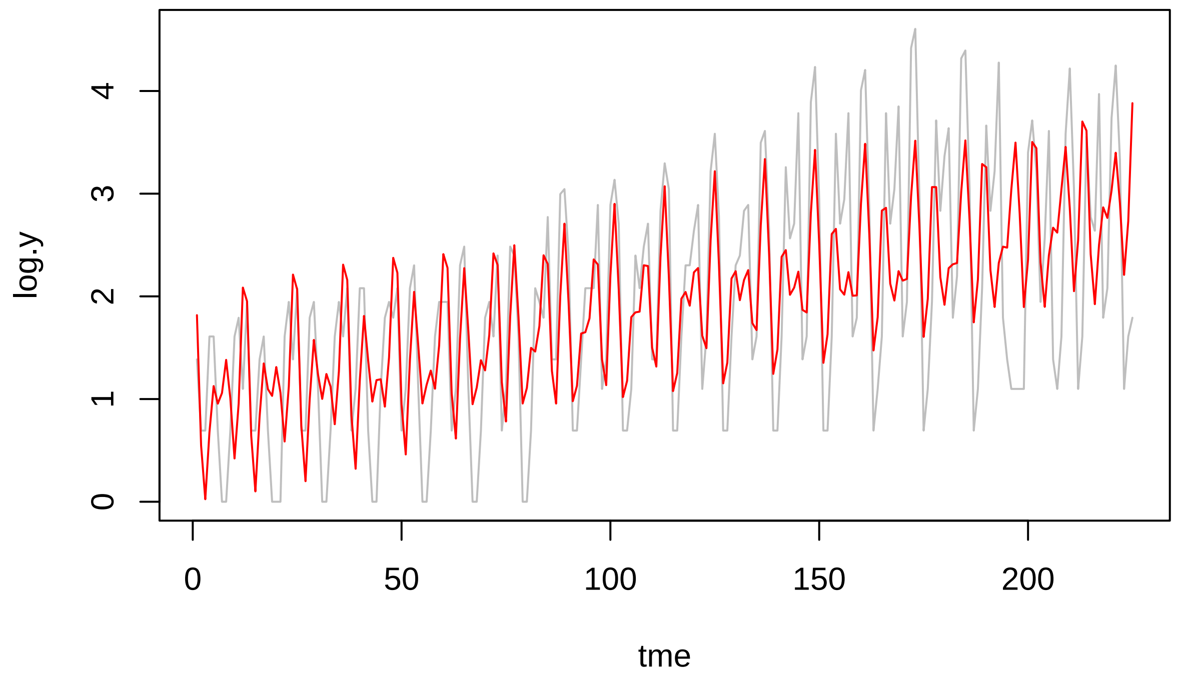
```
## [1] 0.002261451
```

Use `f.hat.cg`, the estimated frequency vector, to plot the model function g. The model output is in red, while the actual output is in gray.

```
f_1 = f.hat.cg[1]
f_2 = f.hat.cg[2]
f_3 = f.hat.cg[3]
#p = 8
X = matrix(1, nrow = n, ncol = p)
X[,2] = tme
X[,3] = cos(2*pi*f_1*tme)
X[,4] = sin(2*pi*f_1*tme)
X[,5] = cos(2*pi*f_2*tme)
X[,6] = sin(2*pi*f_2*tme)
X[,7] = cos(2*pi*f_3*tme)
X[,8] = sin(2*pi*f_3*tme)
lin.model = lm(log.y ~ -1 + X)

# estimated betas:
lin.model$coefficients
```

```
##           X1           X2           X3           X4           X5           X6
##  0.948825429  0.008668736  0.142182939 -0.158254345  0.459041159  0.300865123
##           X7           X8
##  0.381066387  0.322034503
```

```
plot(tme, log.y, type='l', col='gray')
lines(lin.model$fitted.values, col='red')
```



Uncertainty quantification for the unknown parameters:

```
library(mvtnorm)
# columns 1:8 is for samples from beta0, ..., beta7 | f, y
# column 9 is for samples from sigma | f, y
post.samples = matrix(-1, N, 9)

#p = 8
X = matrix(1, nrow = n, ncol = p)
for(i in 1:N)
{
    f_1 = post.f.sample[1,i]
    f_2 = post.f.sample[2,i]
    f_3 = post.f.sample[3,i]
    X[,2] = tme
    X[,3] = cos(2*pi*f_1*tme)
    X[,4] = sin(2*pi*f_1*tme)
    X[,5] = cos(2*pi*f_2*tme)
    X[,6] = sin(2*pi*f_2*tme)
    X[,7] = cos(2*pi*f_3*tme)
    X[,8] = sin(2*pi*f_3*tme)
    lin.model = lm(log.y ~ -1 + X)
    bhat = lin.model$coefficients
    RSS = sum(lin.model$residuals^2)
    sighat = sqrt( RSS/(n-p) )
    Sigma.mat = (sighat^2)*solve(t(X) %*% X)
    chiran = (rchisq(1, df = n-p))
    beta.samples = bhat + (rmvnorm(1, sigma = Sigma.mat))/(sqrt(chiran/(n-p)))
    sig.sample = sqrt( RSS/chiran )

    post.samples[i,1:8] = beta.samples
    post.samples[i,9] = sig.sample
}
```

Below are the 95% CIs for the betas:

```
for (j in 1:8) {
  print(quantile(post.samples[,j], c(.025, .975)))
}
```

```
##      2.5%     97.5%
## 0.7317975 1.1509191
##        2.5%       97.5%
## 0.007137423 0.010386486
##       2.5%      97.5%
## -0.2255888  0.5601758
##       2.5%      97.5%
## -0.2816482  0.4016932
##        2.5%       97.5%
## -0.09789474  0.59959552
##        2.5%       97.5%
## -0.01813564  0.44893638
##       2.5%      97.5%
## 0.07694496 0.52891948
##      2.5%     97.5%
## 0.0617800 0.4811667
```

Below is the 95% CI for sigma:

```
quantile(post.samples[,9], c(.025, .975))
```

```
##      2.5%     97.5%
## 0.7526950 0.9096953
```