Stats 415: Final Project Code

Andrew Mashhadi

05 June, 2023

Load Libraries

```
library(MASS)
library(astsa)
```

Loading Data

```
# import data
df <- read.csv('dataset/stackexchangecounts_ts.csv')
df <- df[-c(1, 2, nrow(df)), ] # remove incomplete 2008

# first few rows of data
head(df)
```

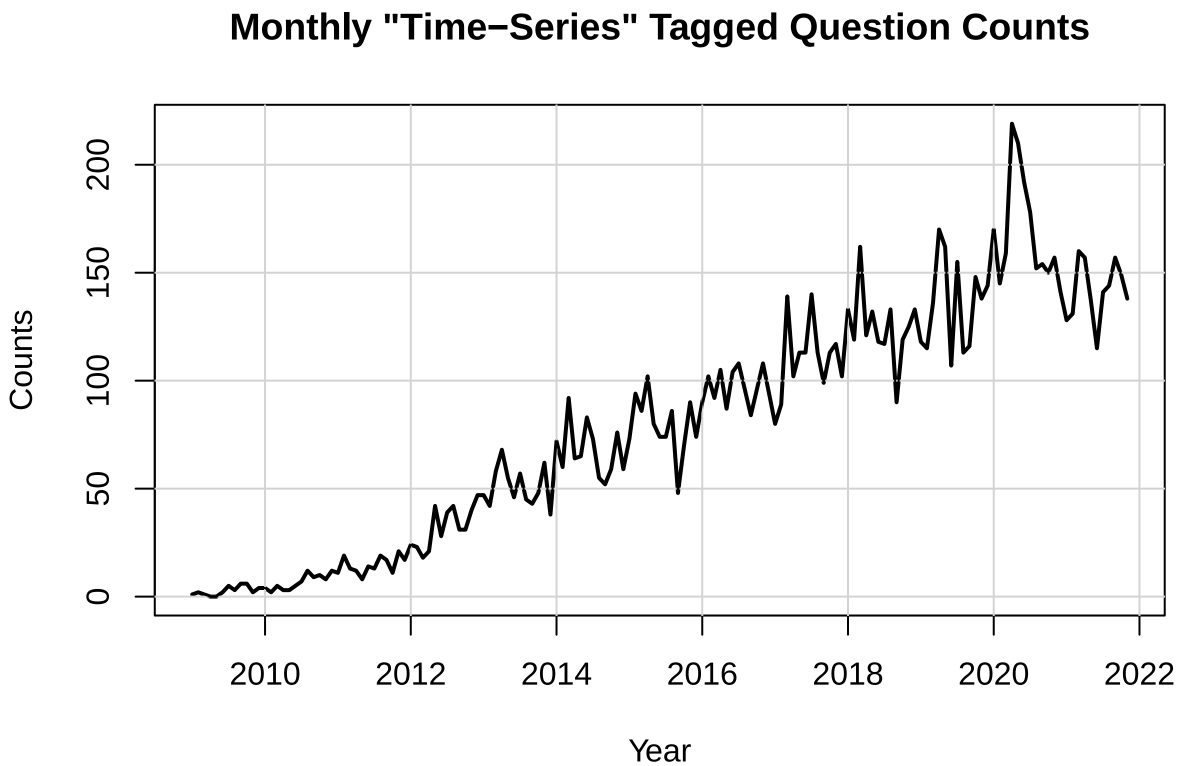
```
##       TagName year month tagcount
## 3 time-series 2009     1        1
## 4 time-series 2009     2        2
## 5 time-series 2009     3        1
## 6 time-series 2009     4        0
## 7 time-series 2009     5        0
## 8 time-series 2009     6        2
```

```
# summary of data
summary(df)
```

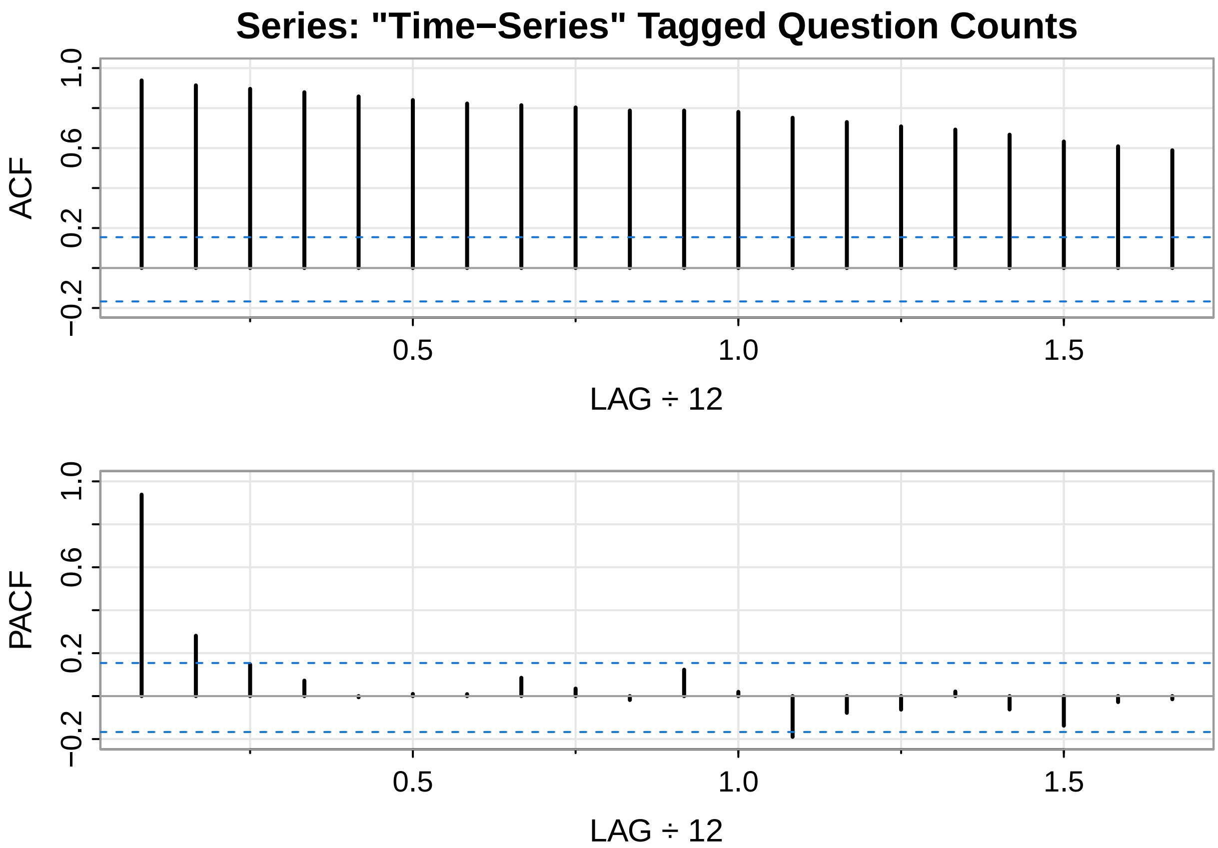
```
##    TagName               year          month           tagcount
##  Length:173         Min.   :2009   Min.   : 1.000   Min.   :  0.00
##  Class :character   1st Qu.:2012   1st Qu.: 3.000   1st Qu.: 38.00
##  Mode  :character   Median :2016   Median : 6.000   Median : 90.00
##                     Mean   :2016   Mean   : 6.399   Mean   : 86.61
##                     3rd Qu.:2019   3rd Qu.: 9.000   3rd Qu.:133.00
##                     Max.   :2023   Max.   :12.000   Max.   :219.00
```

```
# set topic to time-series
count.dat <- df$tagcount

# create time series objects for train/test split
xt <- ts(count.dat, start = c(2009, 1), frequency = 12)

trn <- count.dat[1:floor(0.9*length(count.dat))]
tst <- count.dat[(floor(0.9*length(count.dat))+1):length(count.dat)]
xt_trn <- ts(trn, start = c(2009, 1), frequency = 12)
```

```r
xt_tst <- ts(tst, start = c(2021, 12), frequency = 12)

# create time plot of training data
ts.plot(xt_trn,
        type = "l",
        lwd = 2,
        main = "Monthly \"Time-Series\" Tagged Question Counts",
        xlab = "Year",
        ylab = "Counts")
grid(lty="solid")
lines(xt_trn)
```




```r
# acf and pacf, before trend removed
acf2(xt_trn, max.lag = 20, lwd = 2,
     main="Series: \"Time-Series\" Tagged Question Counts")
```




```
##      [,1] [,2] [,3] [,4]  [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## ACF  0.94 0.91 0.90 0.88  0.86 0.84 0.82 0.81 0.80  0.79  0.79  0.78  0.75
## PACF 0.94 0.28 0.15 0.07 -0.01 0.01 0.01 0.09 0.04 -0.02  0.12  0.02 -0.19
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20]
## ACF   0.73  0.71  0.69  0.67  0.63  0.61  0.59
## PACF -0.08 -0.06  0.02 -0.06 -0.14 -0.03 -0.01
```

Removing trend

```
# fit linear and poly degree 3
xt_trn_df <- data.frame(time=as.vector(time(xt_trn)), count=as.vector(xt_trn))
lm.fit <- lm(count ~ time, data=xt_trn_df) # linear
lm.fit.d3 <- lm(count ~ poly(time, 3), data=xt_trn_df) # poly degree 3

# summarys
summary(lm.fit)
```

```
##
## Call:
## lm(formula = count ~ time, data = xt_trn_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -48.129  -9.258  -0.103   6.479  72.239
##
## Coefficients:
```

```
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.820e+04  7.199e+02  -39.17   <2e-16 ***
## time         1.403e+01  3.572e-01   39.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.58 on 153 degrees of freedom
## Multiple R-squared:  0.9098, Adjusted R-squared:  0.9092
## F-statistic:  1543 on 1 and 153 DF,  p-value: < 2.2e-16
```

```
summary(lm.fit.d3)
```

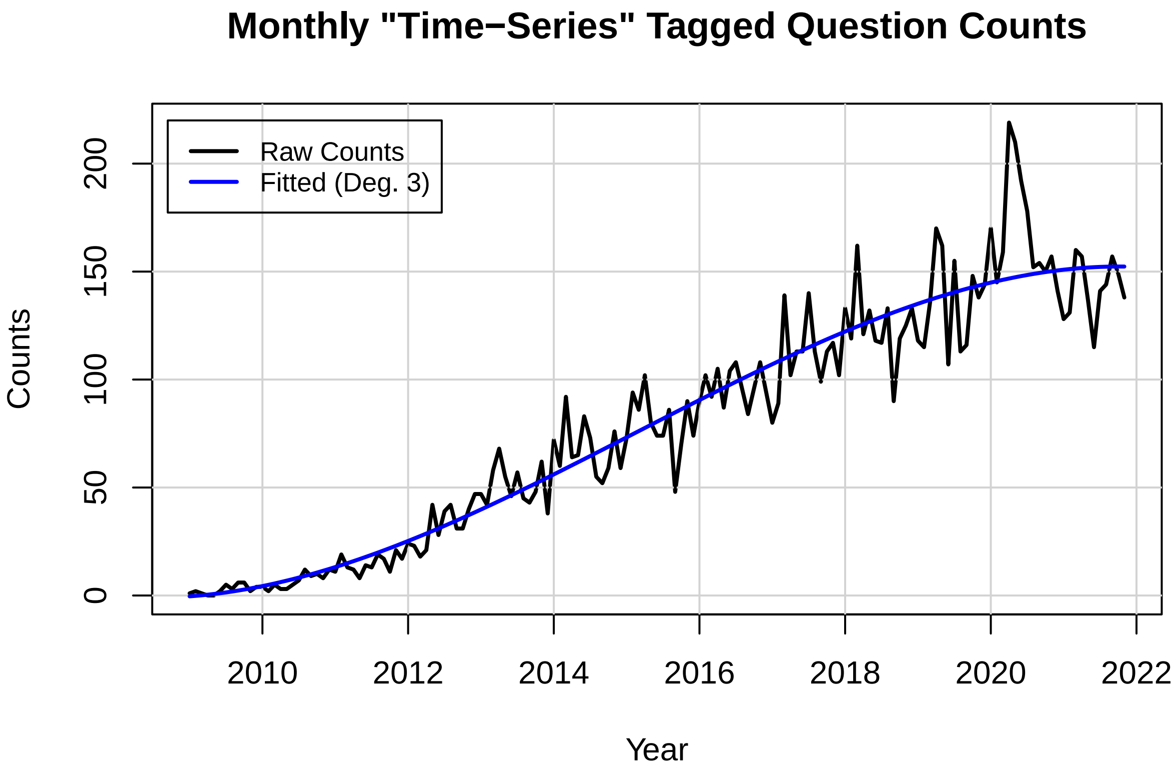
```
##
## Call:
## lm(formula = count ~ poly(time, 3), data = xt_trn_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -41.110  -7.594  -0.692   5.202  72.212
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      78.948      1.263  62.525  < 2e-16 ***
## poly(time, 3)1  651.295     15.720  41.431  < 2e-16 ***
## poly(time, 3)2  -16.894     15.720  -1.075    0.284
## poly(time, 3)3  -66.857     15.720  -4.253 3.68e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.72 on 151 degrees of freedom
## Multiple R-squared:   0.92,  Adjusted R-squared:  0.9184
## F-statistic: 578.6 on 3 and 151 DF,  p-value: < 2.2e-16
```

```
# compare RSS
anova(lm.fit, lm.fit.d3)
```

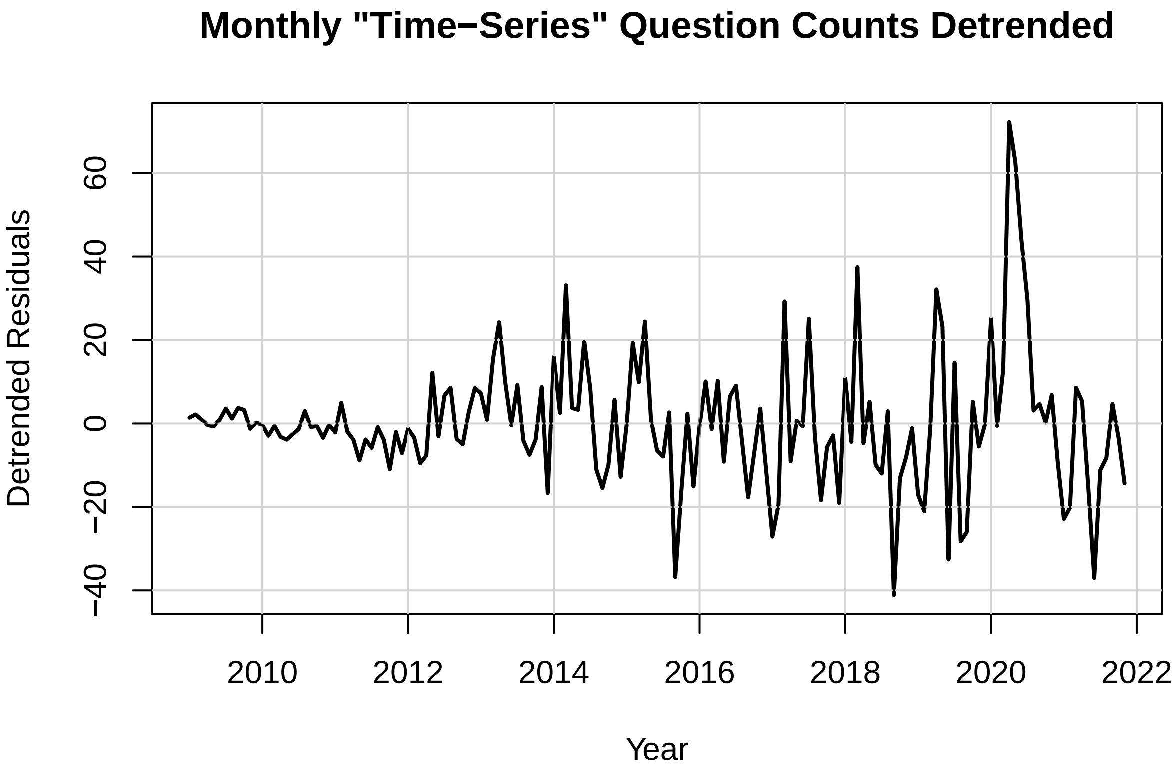
```
## Analysis of Variance Table
##
## Model 1: count ~ time
## Model 2: count ~ poly(time, 3)
##   Res.Df   RSS Df Sum of Sq      F    Pr(>F)
## 1    153 42070
## 2    151 37315  2    4755.2 9.6213 0.0001167 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
ts.plot(xt_trn,
        type = "l",
        main = "Monthly \"Time-Series\" Tagged Question Counts",
        xlab = "Year",
        ylab = "Counts",
        lwd=2)
grid(lty="solid")
lines(xt_trn)
lines(list(x=time(xt_trn), y=lm.fit.d3$fitted.values),
      lwd=2,
```

```
       col="blue")
legend(2008.7, 220,
        legend=c("Raw Counts", "Fitted (Deg. 3)"),
        col=c("black", "blue"),
        lwd=2,
        cex=0.8)
```

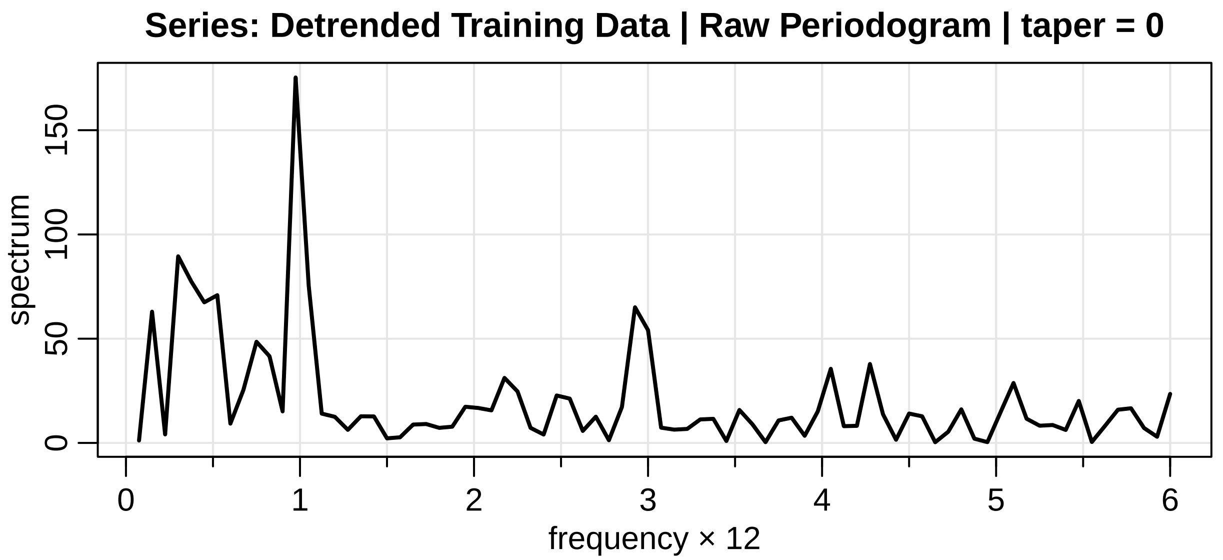


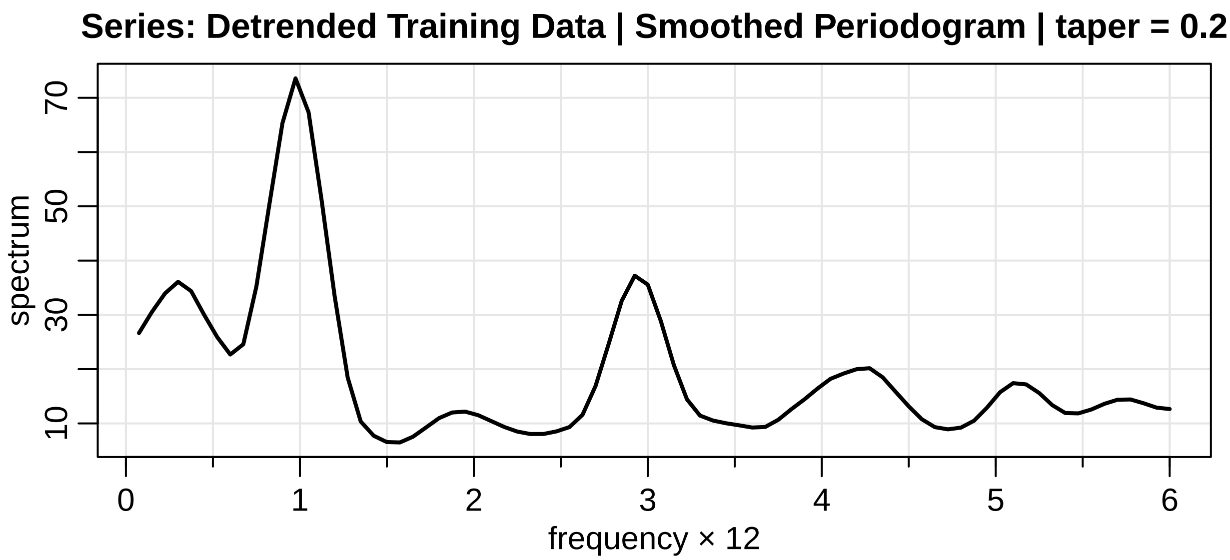

```
# remove trend relationship, show plot
detrended_xt_trn <- xt_trn - lm.fit.d3$fitted.values
ts.plot(detrended_xt_trn,
         main = "Monthly \"Time-Series\" Question Counts Detrended",
         xlab = "Year",
         ylab = "Detrended Residuals",
         type = "l",
         lwd=2)
grid(lty="solid")
lines(detrended_xt_trn)
```



Spectral Analysis

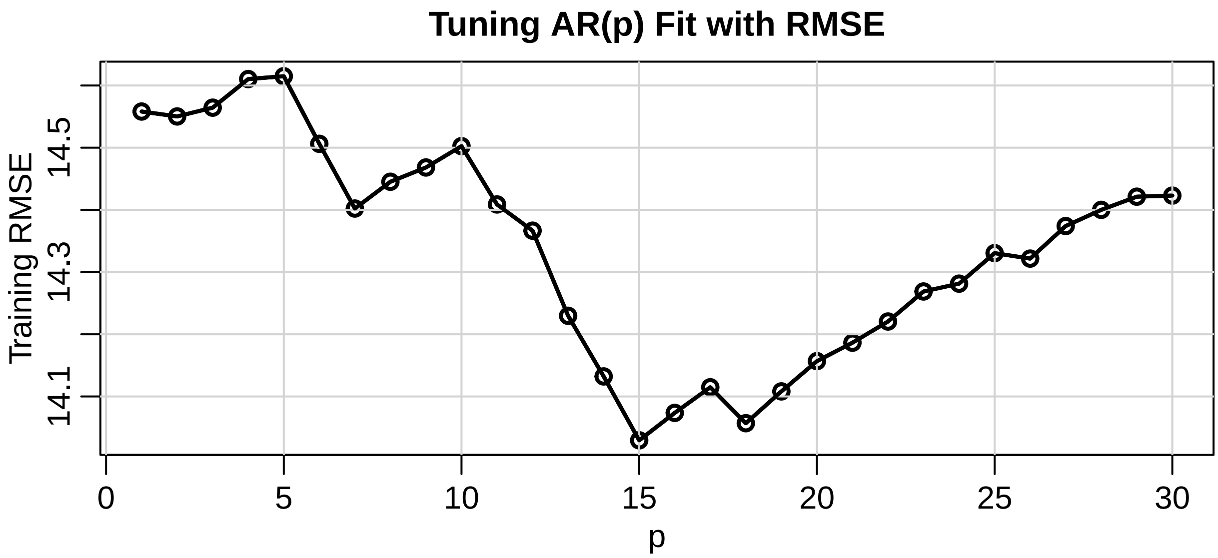
```
par(mfrow=c(2, 1))

# investigate trend/cycles
dat.spec <- mvspec(detrended_xt_trn, log="no", taper=0,
                   main="Series: Detrended Training Data | Raw Periodogram | taper = 0",
                   lwd=2)

# non-parametric
dat.spec.sm.np <- mvspec(detrended_xt_trn,
                         kernel("modified.daniell", c(2, 2)), # not a large set of data points
                         taper=0.2,
                         log="no",
                         main="Series: Detrended Training Data | Smoothed Periodogram | taper = 0.2",
                         lwd=2)
```

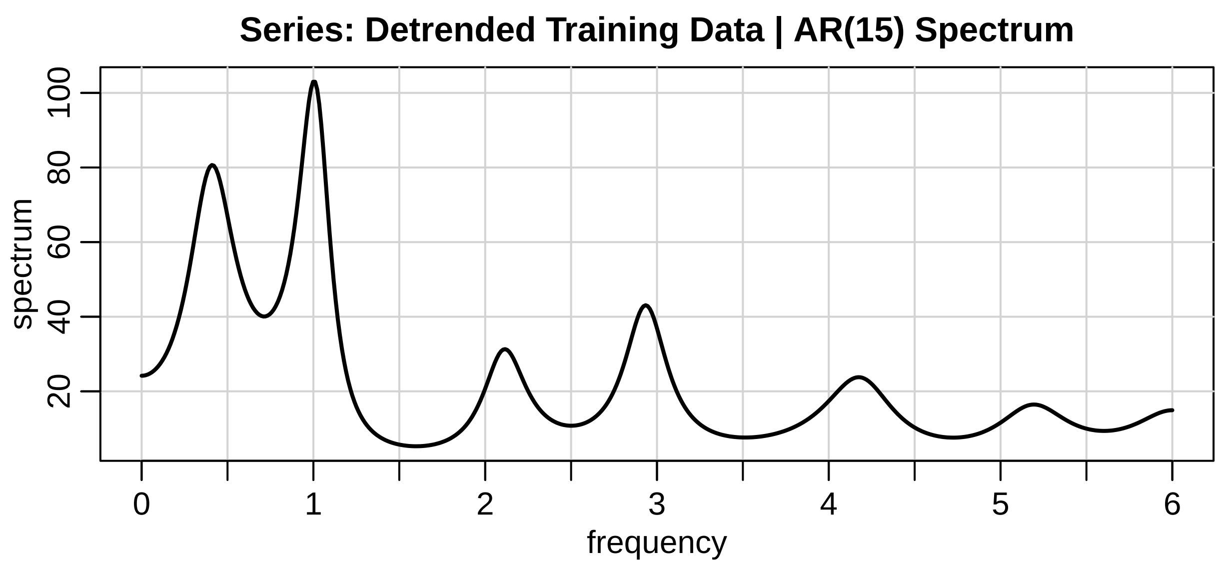





```r
# find min ar order that minimizes the training RMSE
n <- length(detrended_xt_trn)
RMSE <- rep(0, 30)
for (k in 1:30){

  resid <- ar(detrended_xt_trn, order=k, aic=FALSE)$resid
  RMSE[k] <- sqrt(mean(resid^2, na.rm=TRUE))

}

plot(RMSE, type="o", xlab="p", ylab="Training RMSE",
     main="Tuning AR(p) Fit with RMSE",
     lwd=2)
grid(lty="solid")
lines(RMSE, lwd=2)

# parametric
dat.spec.sm.p <- spec.ar(detrended_xt_trn, order=15, log="no",
```

```
                          main="Series: Detrended Training Data | AR(15) Spectrum")
axis(1, at = seq(0, 6, by=0.5), tck = 1, col = "lightgrey", labels=FALSE, lty = "solid")
axis(2, tck = 1, col = "lightgrey", labels = FALSE, lty = "solid", tick=T)
axis(1, at = seq(0, 6, by=0.5), tick = TRUE, labels=FALSE)
axis(2, tick = TRUE, labels=FALSE)
lines(dat.spec.sm.p$freq, dat.spec.sm.p$spec, lwd=2)
```






```
# two-sided confidence intervals
cat("\nTWO-SIDED CONFIDENCE INTERVALS\n")
```

```
##
## TWO-SIDED CONFIDENCE INTERVALS
```

```
df <- dat.spec.sm.np$df
U <- qchisq(.025, df)
L <- qchisq(.975, df)
cat("For peak at freq =", dat.spec.sm.np$freq[4],
    "has the approximate two-sided CI: [", df*dat.spec.sm.np$spec[4]/L, ",",
    df*dat.spec.sm.np$spec[4]/U, "]\n")
```

```
## For peak at freq = 0.3 has the approximate two-sided CI: [ 18.67765 , 96.91799 ]
```

```
cat("For peak at freq =", dat.spec.sm.np$freq[13],
    "has the approximate two-sided CI: [", df*dat.spec.sm.np$spec[13]/L, ",",
    df*dat.spec.sm.np$spec[13]/U, "]\n")
```

```
## For peak at freq = 0.975 has the approximate two-sided CI: [ 38.08241 , 197.609 ]
```

```
cat("For peak at freq =", dat.spec.sm.np$freq[39],
    "has the approximate two-sided CI: [", df*dat.spec.sm.np$spec[39]/L, ",",
    df*dat.spec.sm.np$spec[39]/U, "]\n")
```

```
## For peak at freq = 2.925 has the approximate two-sided CI: [ 19.26234 , 99.95193 ]
```

```
# one-sided confidence intervals
cat("\nONE-SIDED CONFIDENCE INTERVALS\n")
```

```
##
## ONE-SIDED CONFIDENCE INTERVALS
```

```
L <- qchisq(.95, df)
cat("For peak at freq =", dat.spec.sm.np$freq[4],
    "has the approximate one-sided CI: [", df*dat.spec.sm.np$spec[4]/L,
    ", inf ]\n")
```

```
## For peak at freq = 0.3 has the approximate one-sided CI: [ 20.70943 , inf ]
```

```
cat("For peak at freq =", dat.spec.sm.np$freq[13],
    "has the approximate one-sided CI: [", df*dat.spec.sm.np$spec[13]/L,
    ", inf ]\n")
```

```
## For peak at freq = 0.975 has the approximate one-sided CI: [ 42.22507 , inf ]
```

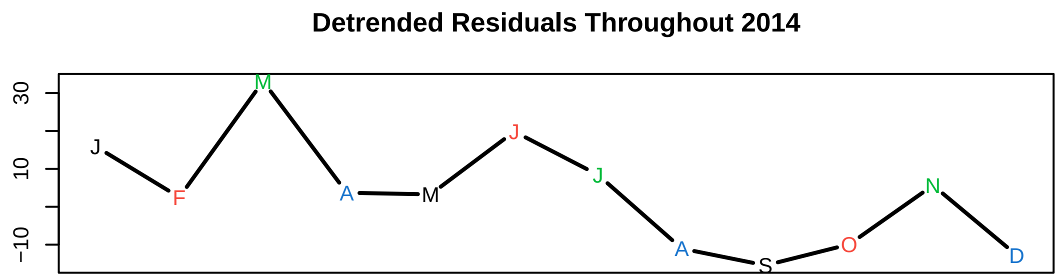
```
cat("For peak at freq =", dat.spec.sm.np$freq[39],
    "has the approximate one-sided CI: [", df*dat.spec.sm.np$spec[39]/L,
    ", inf ]\n")
```

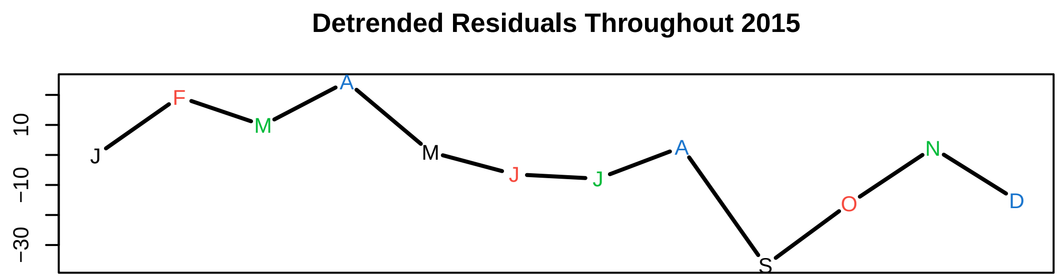
```
## For peak at freq = 2.925 has the approximate one-sided CI: [ 21.35772 , inf ]
```

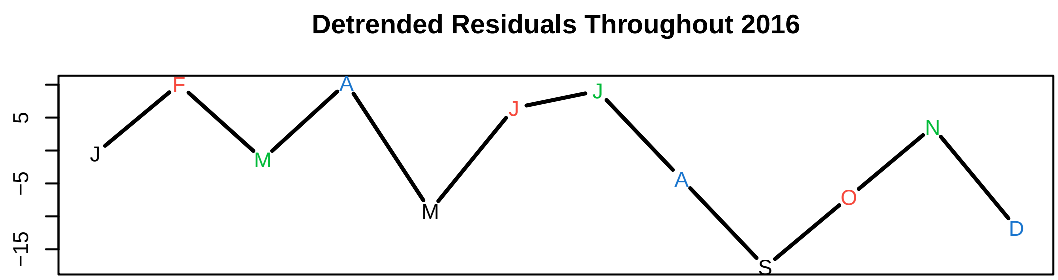
We now visualize the annual cycle with a few example years.

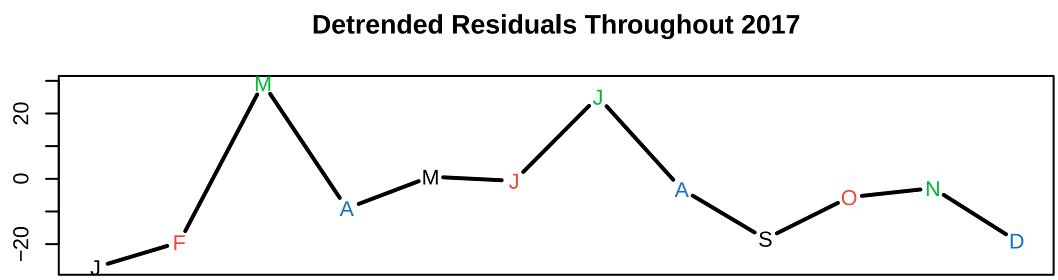
```
# set up 4x1 block of plots
months <- c("J", "F", "M", "A", "M", "J", "J", "A", "S", "O", "N", "D")
par(mfrow=c(4, 1))


# setup plot
years.f2009 <- 5
t1 <- 12*years.f2009 + 1
t2 <- t1 + 11

plot(time(detrended_xt_trn)[t1:t2], detrended_xt_trn[t1:t2],
     type = "c", xlab = "", ylab = "", xaxt = 'n', lwd=2,
     main = "Detrended Residuals Throughout 2014")
points(time(detrended_xt_trn)[t1:t2], detrended_xt_trn[t1:t2], pch = months, col = 1:4)

# setup plot
years.f2009 <- 6
```

```r
t1 <- 12*years.f2009 + 1
t2 <- t1 + 11

plot(time(detrended_xt_trn)[t1:t2], detrended_xt_trn[t1:t2],
     type = "c", xlab = "", ylab = "", xaxt = 'n', lwd=2,
     main = "Detrended Residuals Throughout 2015")
points(time(detrended_xt_trn)[t1:t2], detrended_xt_trn[t1:t2], pch = months, col = 1:4)


# setup plot
years.f2009 <- 7
t1 <- 12*years.f2009 + 1
t2 <- t1 + 11

plot(time(detrended_xt_trn)[t1:t2], detrended_xt_trn[t1:t2],
     type = "c", xlab = "", ylab = "", xaxt = 'n', lwd=2,
     main = "Detrended Residuals Throughout 2016")
points(time(detrended_xt_trn)[t1:t2], detrended_xt_trn[t1:t2], pch = months, col = 1:4)

# setup plot
years.f2009 <- 8
t1 <- 12*years.f2009 + 1
t2 <- t1 + 11

plot(time(detrended_xt_trn)[t1:t2], detrended_xt_trn[t1:t2],
     type = "c", xlab = "", ylab = "", xaxt = 'n', lwd=2,
     main = "Detrended Residuals Throughout 2017")
points(time(detrended_xt_trn)[t1:t2], detrended_xt_trn[t1:t2], pch = months, col = 1:4)
```





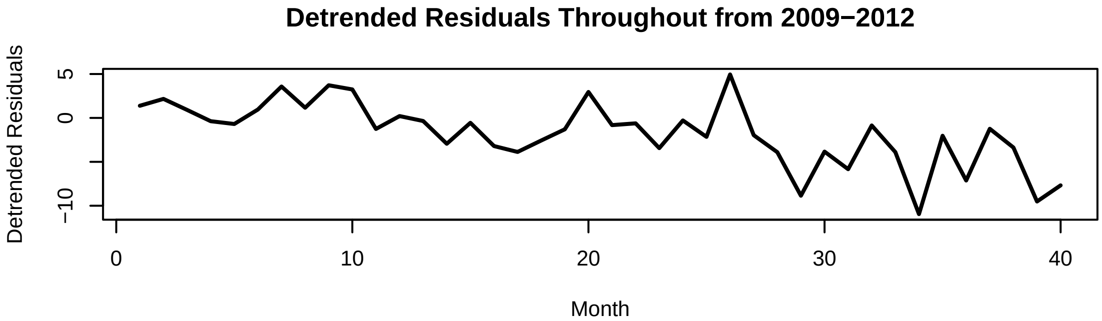


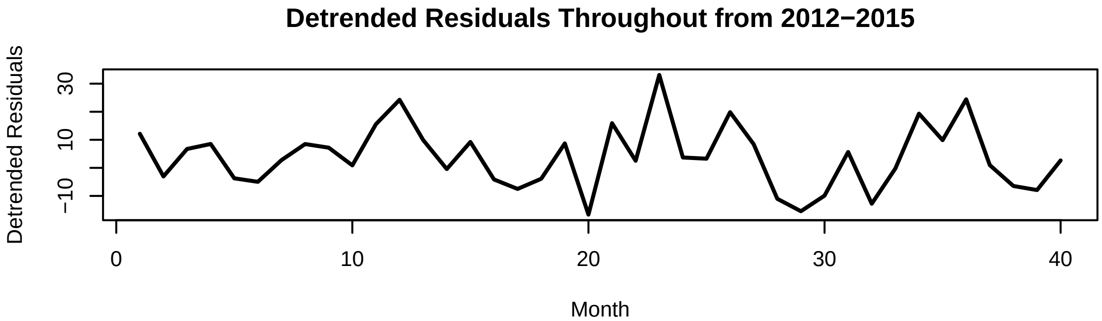


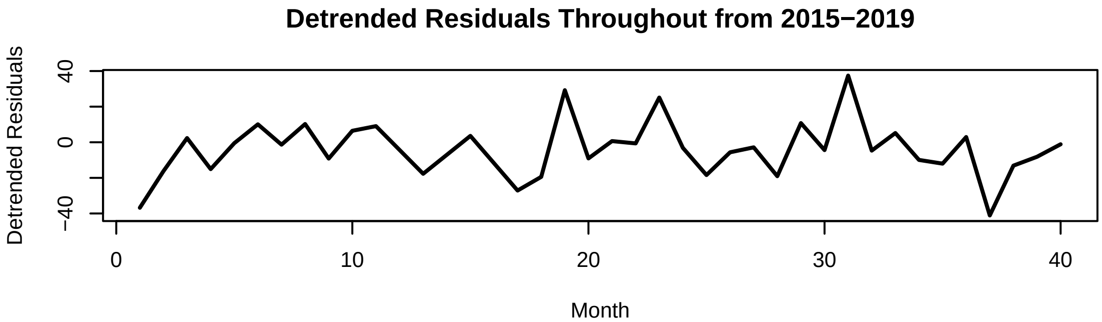
We now visualize the potential 40 months cycle.

```
# set up 3x1 block of plots
par(mfrow=c(3, 1))
```

```r
ts.plot(detrended_xt_trn[1:40],
        type = "l", xlab = "Month", ylab="Detrended Residuals",
        lwd = 2,
        main = "Detrended Residuals Throughout from 2009-2012")

ts.plot(detrended_xt_trn[41:80],
        type = "l", xlab = "Month", ylab="Detrended Residuals",
        lwd = 2,
        main = "Detrended Residuals Throughout from 2012-2015")

ts.plot(detrended_xt_trn[81:120],
        type = "l", xlab = "Month", ylab="Detrended Residuals",
        lwd = 2,
        main = "Detrended Residuals Throughout from 2015-2019")
```



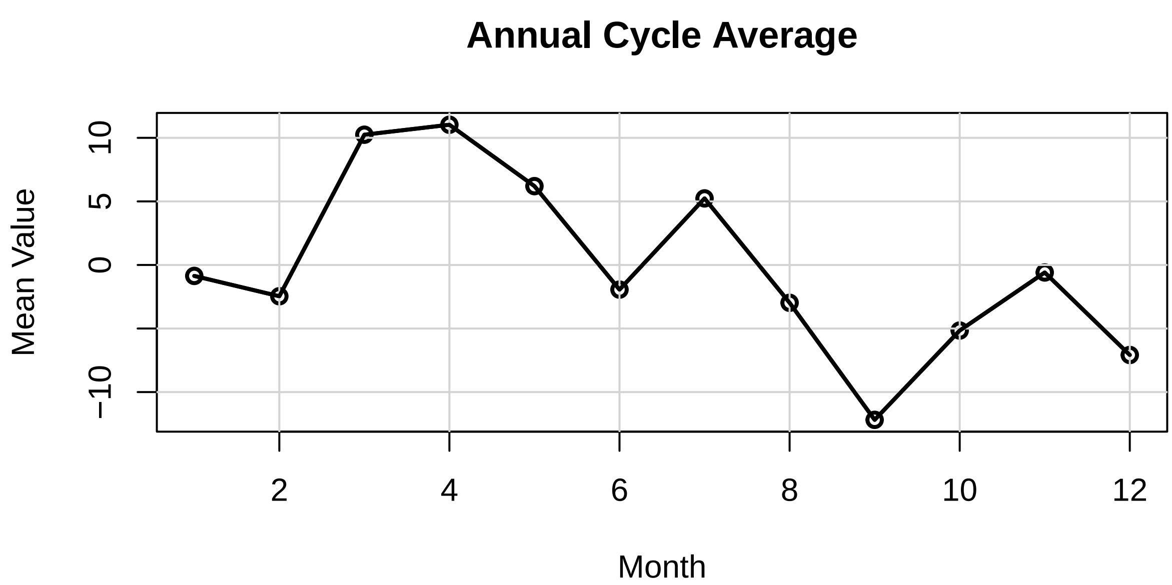


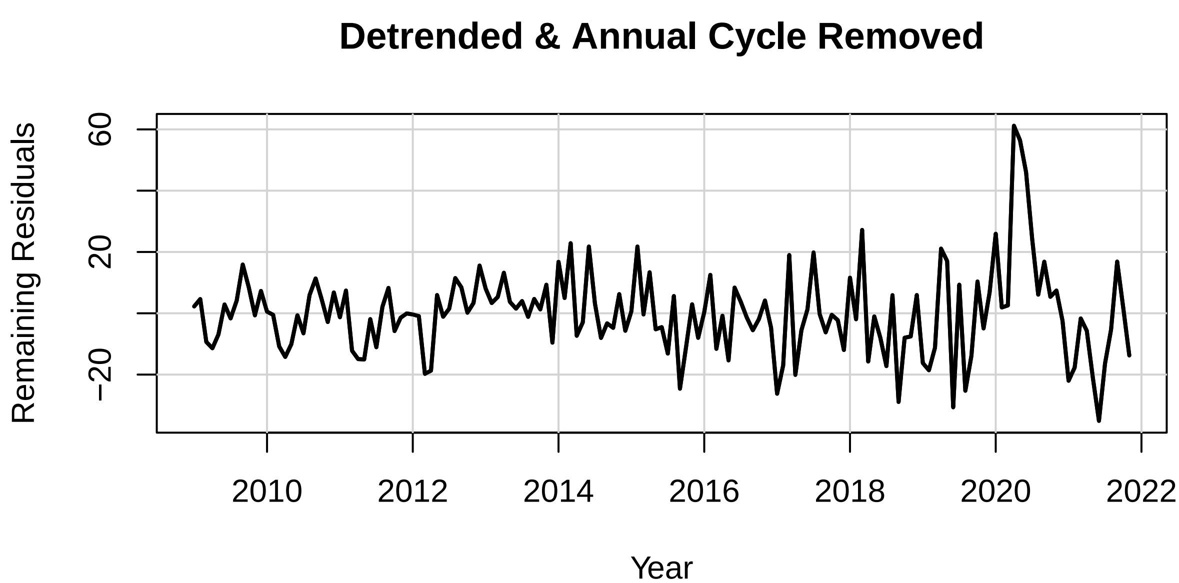


Removing Cycles

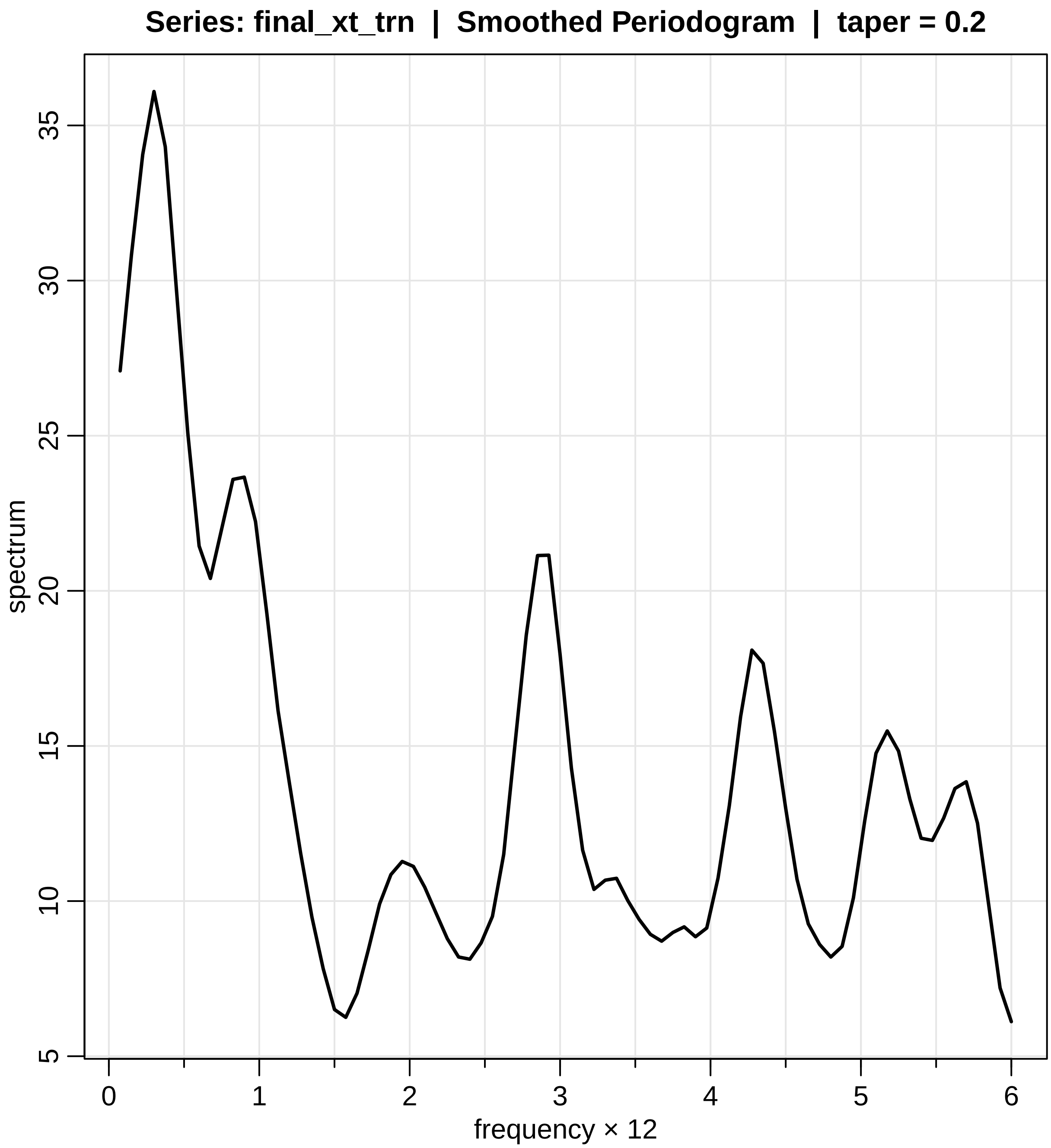
Now we display the average annual cycle, and attempt to remove it.

```r
par(mfrow=c(2, 1))
ann.matrix <- matrix(detrended_xt_trn, byrow = TRUE, ncol = 12)
ann.matrix[13, 12] <- NA # incomplete year in training cutoff

monthly_avg <- colMeans(ann.matrix, na.rm = TRUE)
plot(1:12, monthly_avg, xlab = "Month", ylab = "Mean Value",
     type = "o",
     lwd=2,
     main="Annual Cycle Average")
grid(lty="solid")
lines(1:12, monthly_avg, lwd = 2)

## Now remove this cycle and show the residuals.
final_xt_trn <- detrended_xt_trn - monthly_avg


ts.plot(final_xt_trn, xlab = "Year",
        ylab = "Remaining Residuals",
        main = "Detrended & Annual Cycle Removed",
        type = "l",
        lwd=2)
grid(lty="solid")
lines(final_xt_trn, lwd = 2)
```





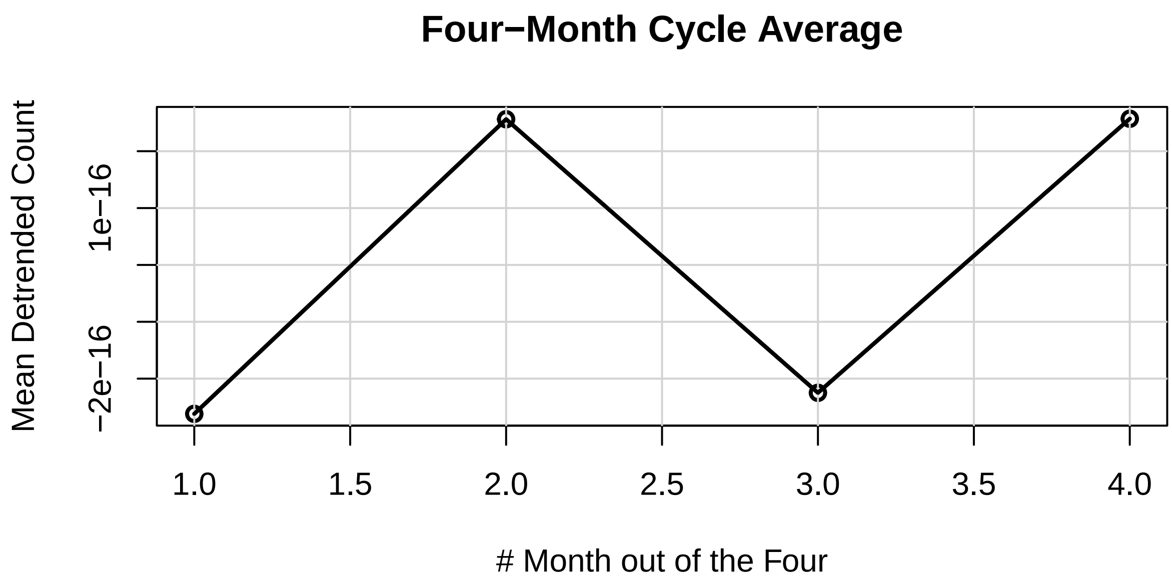

```
par(mfrow=c(1, 1))
# assess new spectrum
mvspec(final_xt_trn,
      kernel("modified.daniell", c(2, 2)), # not a large set of data points
      taper=0.2,
      log="no",
      lwd = 2)
```

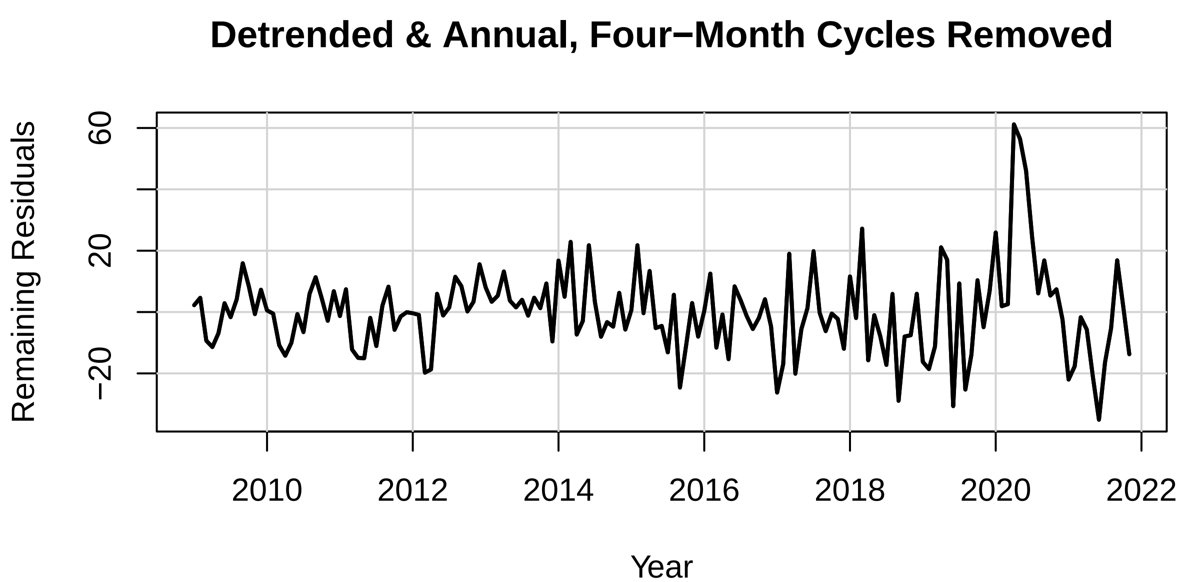


Now we display the average 4 month cycle, and attempt to remove it.

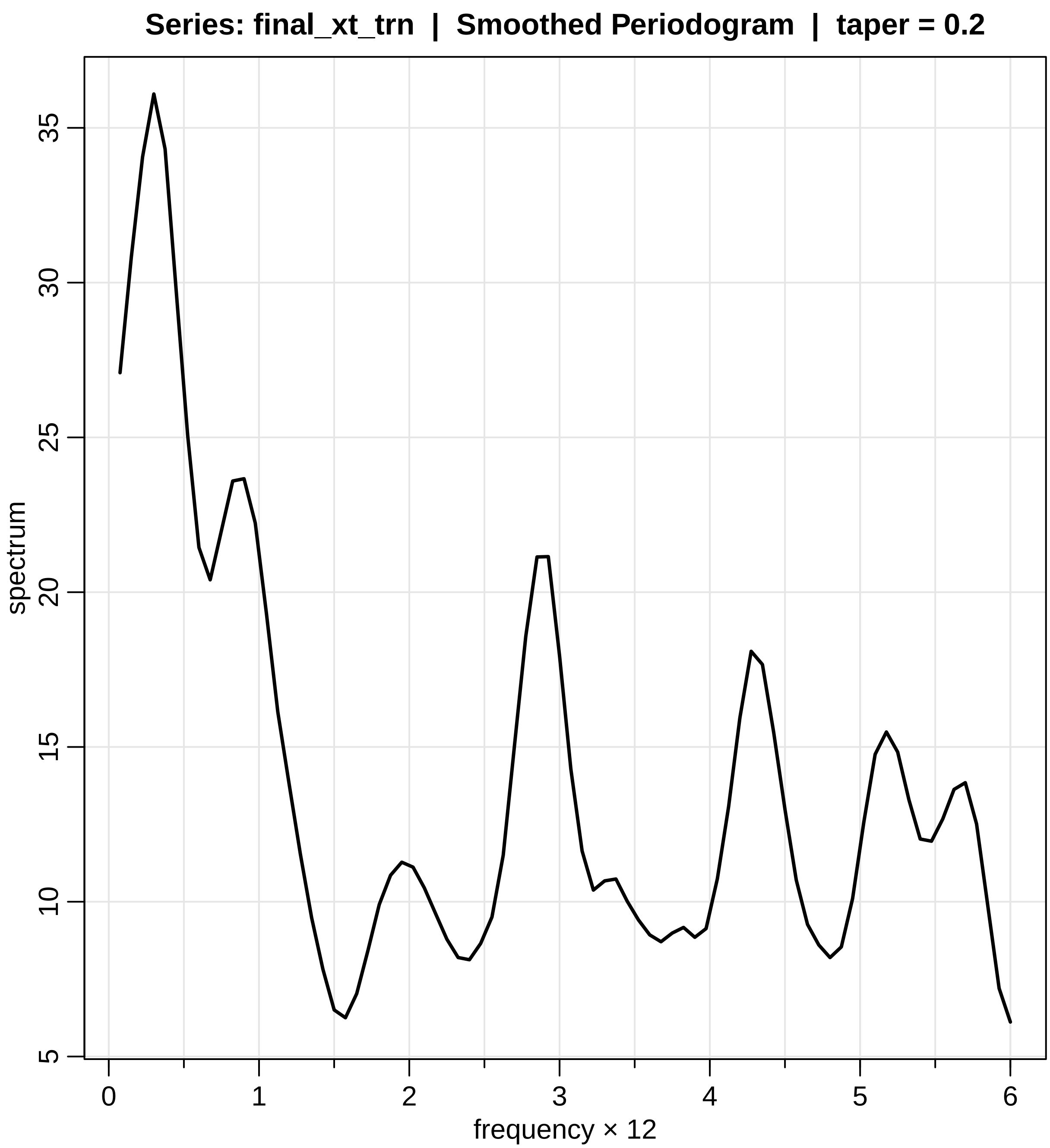
```
par(mfrow=c(2, 1))
four.month.matrix <- matrix(final_xt_trn, byrow = TRUE, ncol = 4)
four.month.matrix[39, 4] <- NA # incomplete year in training cutoff

four.month.avg <- colMeans(four.month.matrix, na.rm = TRUE)
plot(1:4, four.month.avg,
     xlab = "# Month out of the Four",
     ylab = "Mean Detrended Count",
     main = "Four-Month Cycle Average",
     type = "o",
```

```
      lwd=2)
grid(lty="solid")
lines(1:4, four.month.avg, lwd = 2)

## Now remove this cycle and show the residuals.
final_xt_trn <- final_xt_trn - four.month.avg

ts.plot(final_xt_trn, xlab = "Year",
        ylab = "Remaining Residuals",
        main = "Detrended & Annual, Four-Month Cycles Removed",
        type = "l",
        lwd=2)
grid(lty="solid")
lines(final_xt_trn, lwd = 2)
```





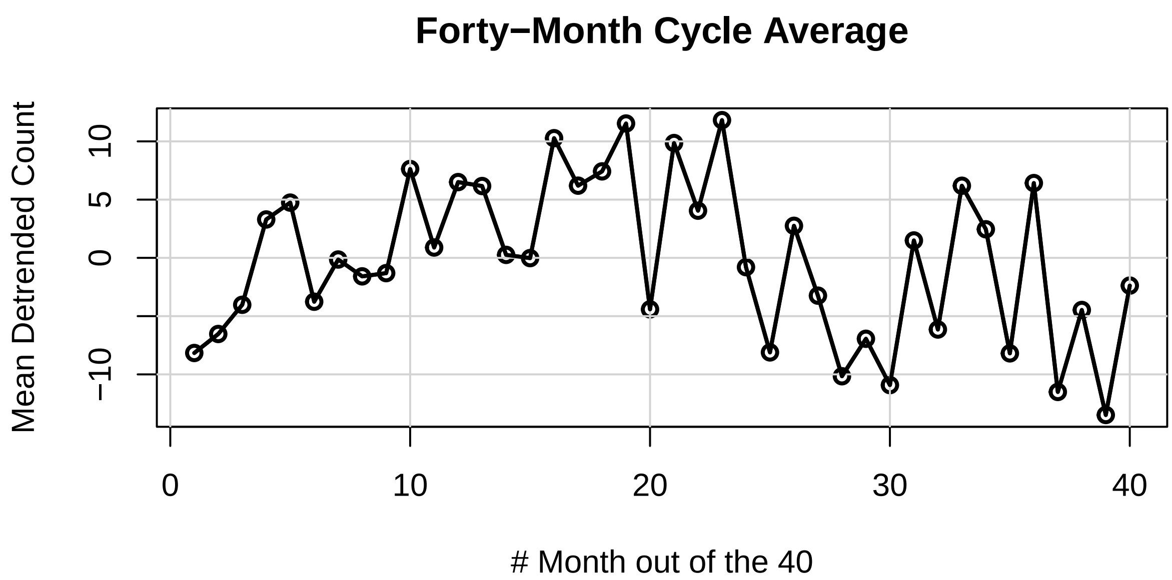
```r
par(mfrow=c(1, 1))
# assess new spectrum
mvspec(final_xt_trn,
      kernel("modified.daniell", c(2, 2)), # not a large set of data points
      taper=0.2,
      log="no",
      lwd = 2)
```

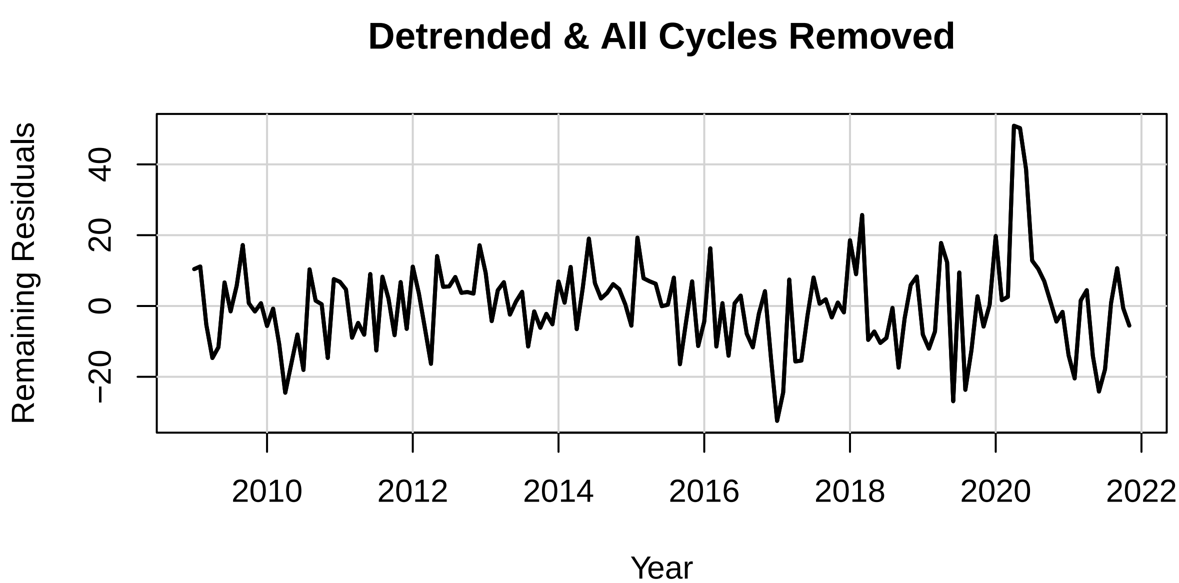


Now we display the average 40 month cycle, and attempt to remove it.

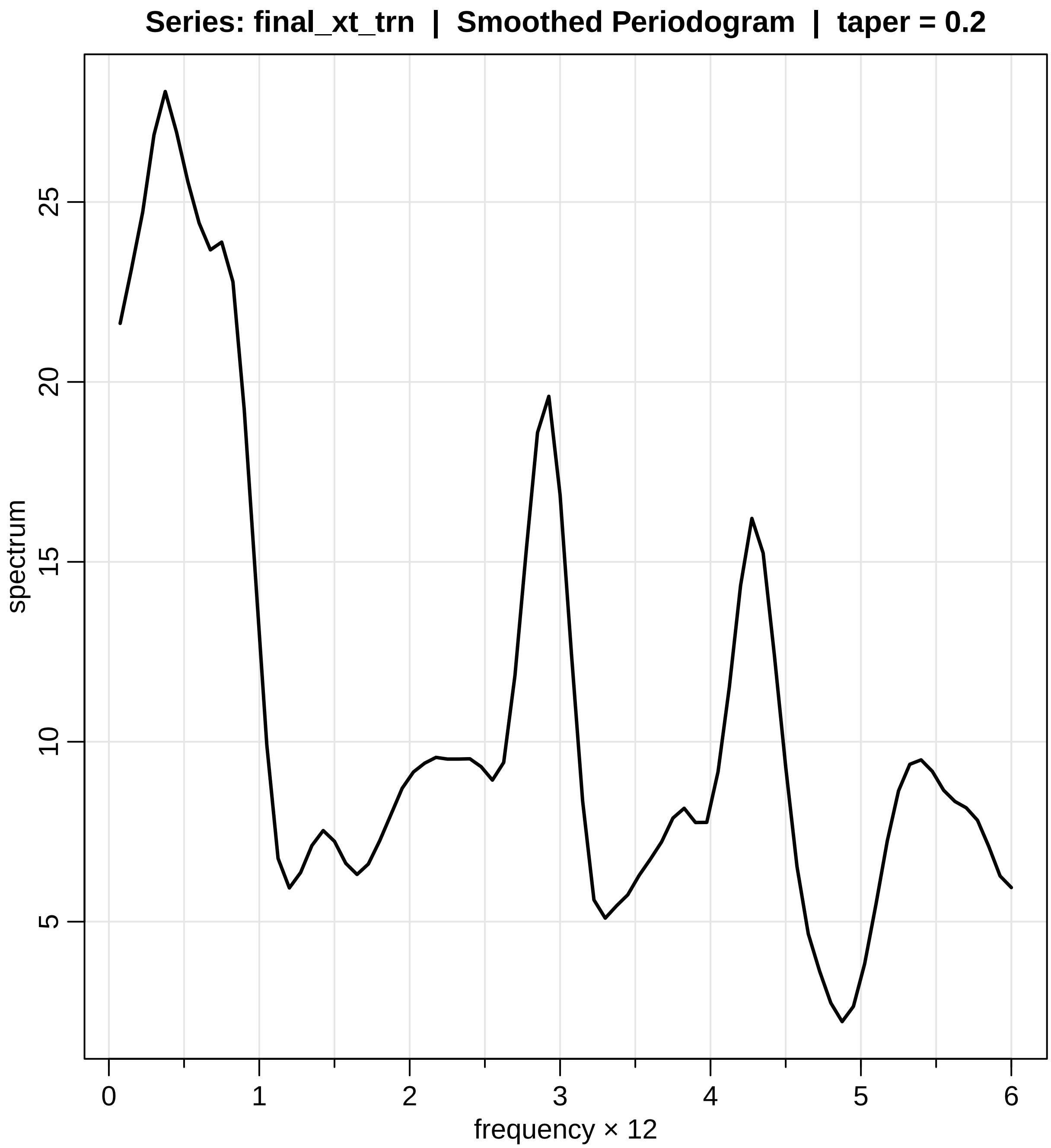
```r
par(mfrow=c(2, 1))
forty.month.matrix <- matrix(final_xt_trn, byrow = TRUE, ncol = 40)
```

```r
forty.month.matrix[4, c(36:40)] <- NA # incomplete year in training cutoff

forty.month.avg <- colMeans(forty.month.matrix, na.rm = TRUE)
plot(1:40, forty.month.avg,
     xlab = "# Month out of the 40",
     ylab = "Mean Detrended Count",
     main = "Forty-Month Cycle Average",
     lwd=2,
     type = "o")
grid(lty="solid")
lines(1:40, forty.month.avg, lwd = 2)

## Now remove this cycle and show the residuals.
final_xt_trn <- final_xt_trn - forty.month.avg


ts.plot(final_xt_trn,
        xlab = "Year",
        ylab = "Remaining Residuals",
        main = "Detrended & All Cycles Removed",
        lwd=2,
        type = "l")
grid(lty="solid")
lines(final_xt_trn, lwd = 2)
```



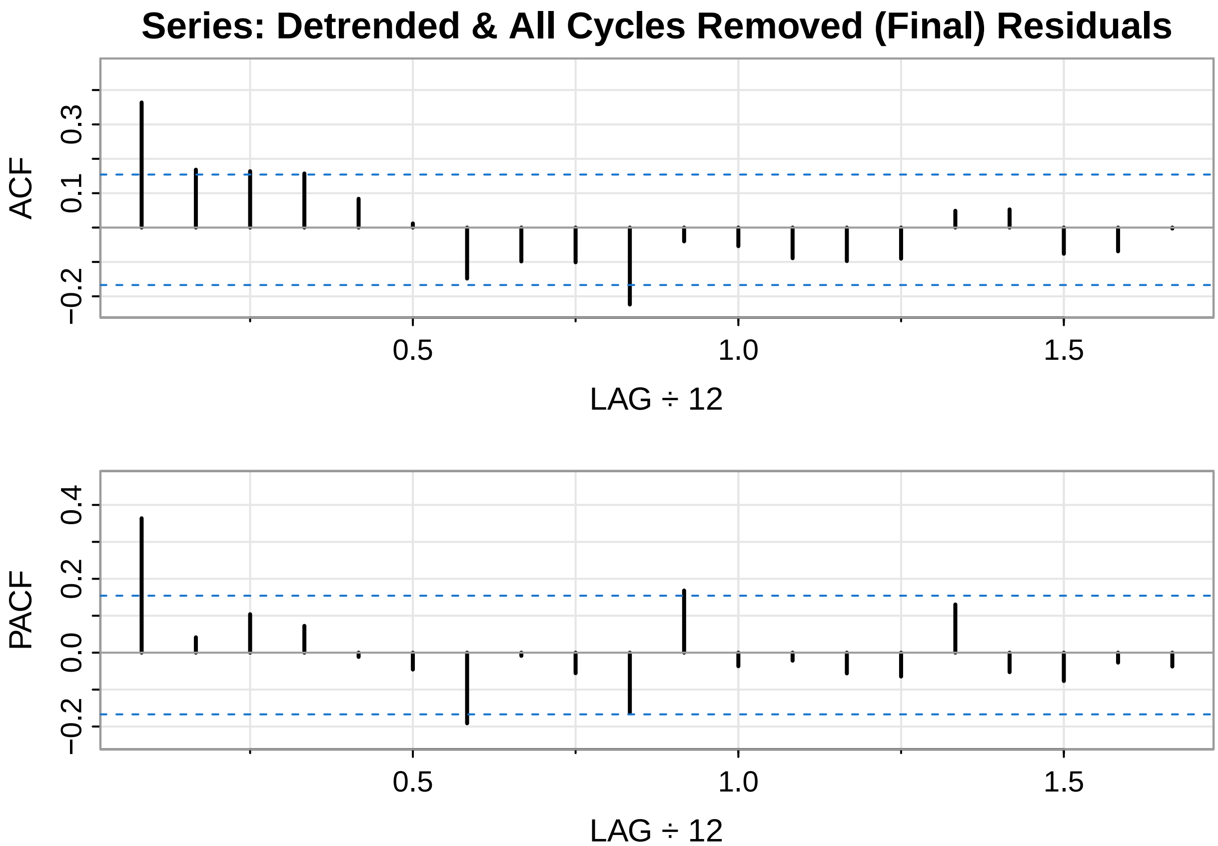



```
par(mfrow=c(1, 1))
# assess new spectrum
mvspec(final_xt_trn,
       kernel("modified.daniell", c(2, 2)), # not a large set of data points
       taper=0.2,
       log="no",
       lwd=2)
```



Estimate New ACF and PACF

```
# acf and pacf, before trend removed
acf2(final_xt_trn, max.lag = 20,
     main="Series: Detrended & All Cycles Removed (Final) Residuals",
     lwd = 2)
```




```
##      [,1] [,2] [,3] [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] [,12] [,13]
## ACF  0.36 0.17 0.16 0.16  0.08  0.01 -0.15 -0.10 -0.10 -0.22 -0.04 -0.05 -0.09
## PACF 0.36 0.04 0.10 0.07 -0.01 -0.05 -0.19 -0.01 -0.06 -0.16  0.17 -0.04 -0.02
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20]
## ACF  -0.10 -0.09  0.05  0.05 -0.08 -0.07  0.00
## PACF -0.06 -0.06  0.13 -0.05 -0.08 -0.03 -0.04
```

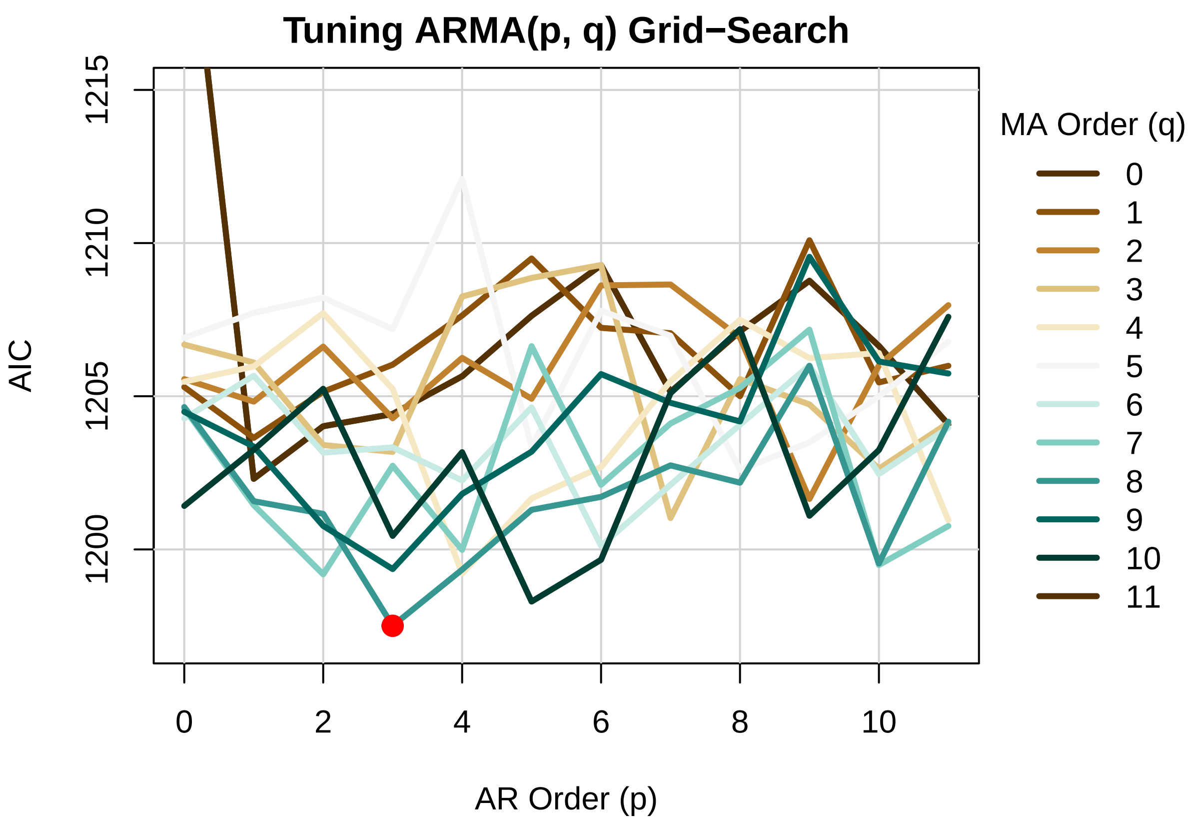
Fitting the ARMA Model to Remaining Residuals

Note, the `arima()` function here uses conditional-sum-of-squares to find starting values, then applies maximum likelihood.

```
# loop through all 12x12 settings of ARMA(p, q) and calculate the AICs
aics <- matrix(0, nrow = 12, ncol = 12)

for(i in 0:11) {
  for(j in 0:11) {

    if (i == 5 & j == 3){

      # issue with non-stationary for this combination of p and q (for CSS)
      t.arma <- arima(final_xt_trn, order = c(i, 0, j), method = "ML")
      aics[i+1,j+1] <- t.arma$aic

    } else {

      t.arma <- arima(final_xt_trn, order = c(i, 0, j))
```

```r
      aics[i+1,j+1] <- t.arma$aic
    }


  }
}
```

```r
# load library for plot colors
library(RColorBrewer)

# setup colors
cols <- brewer.pal(12, "BrBG")


## AIC TUNING PLOT
par(mar=c(5,4,2,7))
plot(0:11, aics[, 1],
     xlab = "AR Order (p)",
     ylab = "AIC",
     main = "Tuning ARMA(p, q) Grid-Search",
     type = "l",
     lwd = 3,
     col = cols[1],
     ylim = c(1197, 1215))
grid(lty="solid")
for (r in 1:12) {

  lines(0:11, aics[, r],
        lwd = 3,
        col = cols[r],
        type = "l")

}
legend("right", inset = c(-0.26,0), legend = 0:11, xpd = NA,
       title = "MA Order (q)", col = cols, lty = 1, bty = "n", lwd=3)
points(3, min(aics), col="red", lwd=4, pch=19)
```



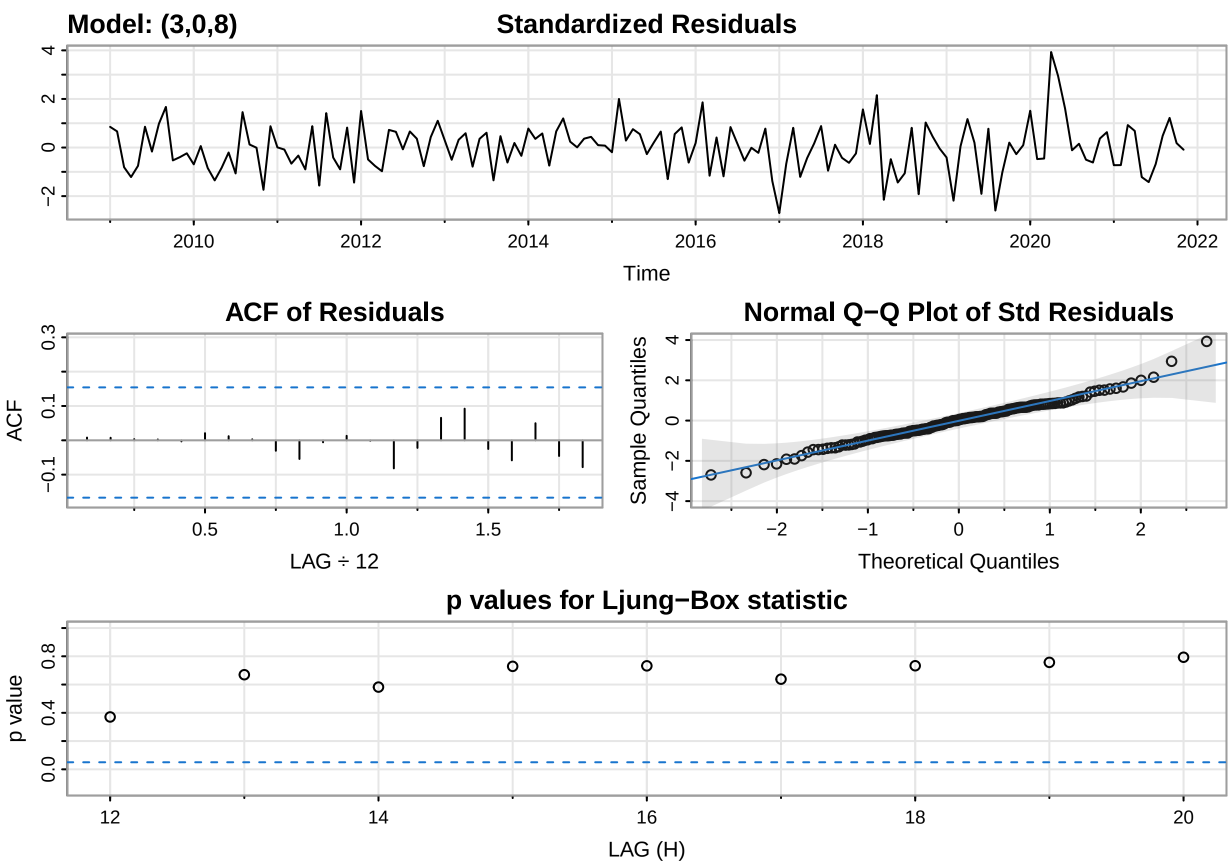
Now we re-fit the optimal ARMA(3, 8) model and display associated diagnostic plots and fitted coefficients (with full summary).

```
# use sarima to get the diagnostics
sarima(final_xt_trn, p = 3, q = 8, d = 0)
```

```
## initial  value 2.515293
## iter   2 value 2.449573
## iter   3 value 2.421300
## iter   4 value 2.412702
## iter   5 value 2.405161
## iter   6 value 2.403368
## iter   7 value 2.402410
## iter   8 value 2.401785
## iter   9 value 2.400926
## iter  10 value 2.399888
## iter  11 value 2.399154
## iter  12 value 2.398614
## iter  13 value 2.398323
## iter  14 value 2.397251
## iter  15 value 2.396813
## iter  16 value 2.395673
## iter  17 value 2.394634
## iter  18 value 2.392993
## iter  19 value 2.392896
## iter  20 value 2.391439
## iter  21 value 2.390901
## iter  22 value 2.389204
## iter  23 value 2.388680
```

```
## iter  24 value 2.387568
## iter  25 value 2.387341
## iter  26 value 2.387087
## iter  27 value 2.386982
## iter  28 value 2.386847
## iter  29 value 2.386764
## iter  30 value 2.386666
## iter  31 value 2.386297
## iter  32 value 2.386173
## iter  33 value 2.385985
## iter  34 value 2.385920
## iter  35 value 2.385890
## iter  36 value 2.385876
## iter  37 value 2.385869
## iter  38 value 2.385865
## iter  39 value 2.385837
## iter  40 value 2.385837
## iter  41 value 2.385821
## iter  42 value 2.385794
## iter  43 value 2.385776
## iter  44 value 2.385755
## iter  45 value 2.385718
## iter  46 value 2.385700
## iter  47 value 2.385662
## iter  48 value 2.385635
## iter  49 value 2.385609
## iter  50 value 2.385591
## iter  51 value 2.385582
## iter  52 value 2.385580
## iter  53 value 2.385575
## iter  54 value 2.385568
## iter  55 value 2.385561
## iter  56 value 2.385556
## iter  57 value 2.385552
## iter  58 value 2.385548
## iter  59 value 2.385543
## iter  60 value 2.385534
## iter  61 value 2.385525
## iter  62 value 2.385518
## iter  63 value 2.385516
## iter  64 value 2.385515
## iter  64 value 2.385515
## final  value 2.385515
## converged
## initial  value 2.383532
## iter   2 value 2.372386
## iter   3 value 2.371875
## iter   4 value 2.370978
## iter   5 value 2.370840
## iter   6 value 2.369100
## iter   7 value 2.368625
## iter   8 value 2.366709
## iter   9 value 2.365107
## iter  10 value 2.364126
```

```
## iter  11 value 2.363675
## iter  12 value 2.362634
## iter  13 value 2.362094
## iter  14 value 2.361346
## iter  15 value 2.361068
## iter  16 value 2.360684
## iter  17 value 2.360460
## iter  18 value 2.360267
## iter  19 value 2.360179
## iter  20 value 2.360133
## iter  21 value 2.360122
## iter  22 value 2.360118
## iter  23 value 2.360117
## iter  24 value 2.360116
## iter  25 value 2.360116
## iter  25 value 2.360116
## iter  25 value 2.360116
## final  value 2.360116
## converged
```




```
## $fit
##
## Call:
## arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D, Q), period = S),
##     xreg = xmean, include.mean = FALSE, transform.pars = trans, fixed = fixed,
##     optim.control = list(trace = trc, REPORT = 1, reltol = tol))
##
```

```
## Coefficients:
##           ar1     ar2     ar3     ma1      ma2      ma3      ma4     ma5
##       -0.4066  0.4695  0.6305  0.7632  -0.2789  -0.7545  -0.0596  0.0302
## s.e.   0.1005  0.0867  0.1025  0.1276   0.1232   0.1435   0.1217  0.1190
##           ma6      ma7      ma8   xmean
##       -0.0555  -0.3342  -0.3108  0.1066
## s.e.   0.1139   0.1111   0.0985  0.3629
##
## sigma^2 estimated as 106.8:  log likelihood = -585.75,  aic = 1197.51
##
## $degrees_of_freedom
## [1] 143
##
## $ttable
##       Estimate     SE t.value p.value
## ar1    -0.4066 0.1005 -4.0467  0.0001
## ar2     0.4695 0.0867  5.4147  0.0000
## ar3     0.6305 0.1025  6.1486  0.0000
## ma1     0.7632 0.1276  5.9831  0.0000
## ma2    -0.2789 0.1232 -2.2632  0.0251
## ma3    -0.7545 0.1435 -5.2562  0.0000
## ma4    -0.0596 0.1217 -0.4898  0.6250
## ma5     0.0302 0.1190  0.2541  0.7998
## ma6    -0.0555 0.1139 -0.4868  0.6272
## ma7    -0.3342 0.1111 -3.0080  0.0031
## ma8    -0.3108 0.0985 -3.1541  0.0020
## xmean   0.1066 0.3629  0.2939  0.7693
##
## $AIC
## [1] 7.725851
##
## $AICc
## [1] 7.740026
##
## $BIC
## [1] 7.981106
```

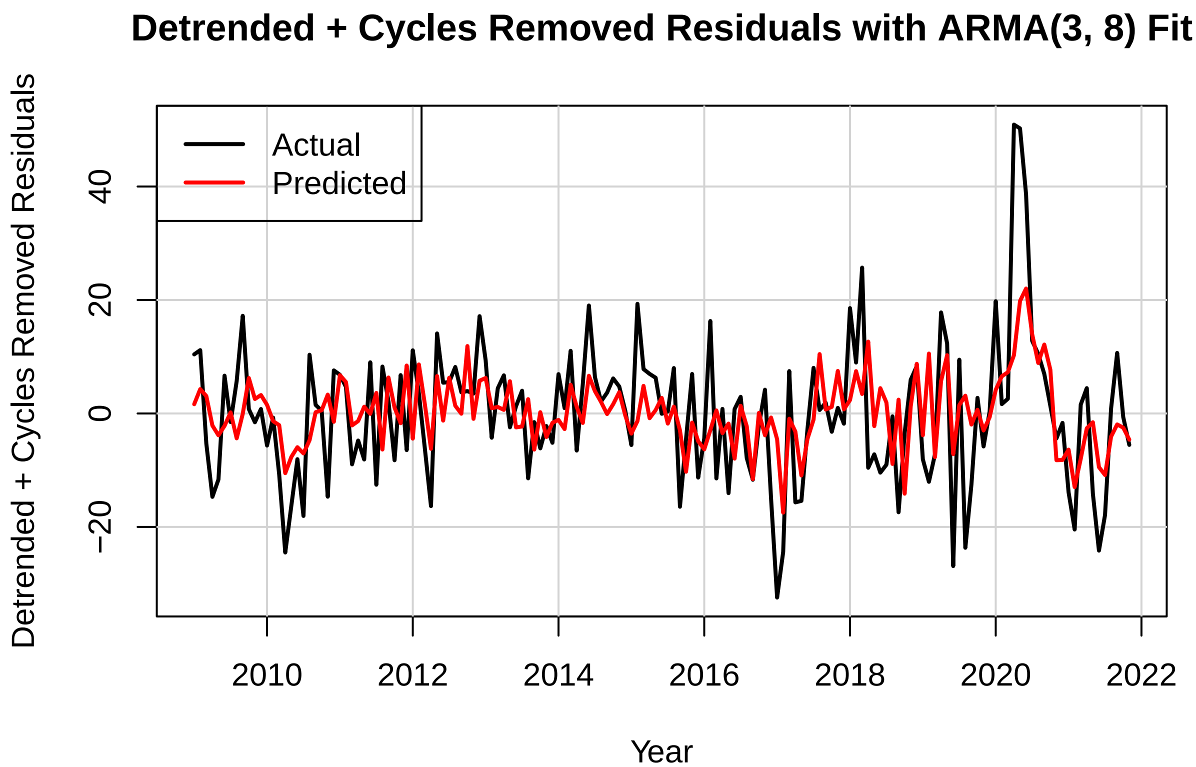
```
# fit arima using regular arima model (same coefficients were found)
arma.mod <- arima(final_xt_trn, order=c(3, 0, 8))

# display coefficients and summary
arma.mod$coef
```

```
##         ar1         ar2         ar3         ma1         ma2         ma3
## -0.40658847  0.46953847  0.63050083  0.76322274 -0.27892884 -0.75445133
##         ma4         ma5         ma6         ma7         ma8   intercept
## -0.05959133  0.03023169 -0.05546342 -0.33422949 -0.31078610  0.10663528
```

We now assess our model's fit to the training data. In addition, we display the associated AIC and training RMSE score.

```
# load library
library(forecast)

# overlay fitted to final residuals training data
ts.plot(final_xt_trn,
```

```r
        type = "l",
        main = "Detrended + Cycles Removed Residuals with ARMA(3, 8) Fit",
        xlab = "Year",
        ylab = "Detrended + Cycles Removed Residuals")
grid(lty="solid")
lines(final_xt_trn, lwd=2)
lines(list(x=time(final_xt_trn), y=fitted(arma.mod)),
      lwd=2,
      col="red")
legend("topleft", cex = 1, c("Actual", "Predicted"),
       lty = 1, col = c("black", "red"), lwd=2)
```




```r
# output AIC and training RMSE for residuals
cat("FOR RESIDUALS\n")
```

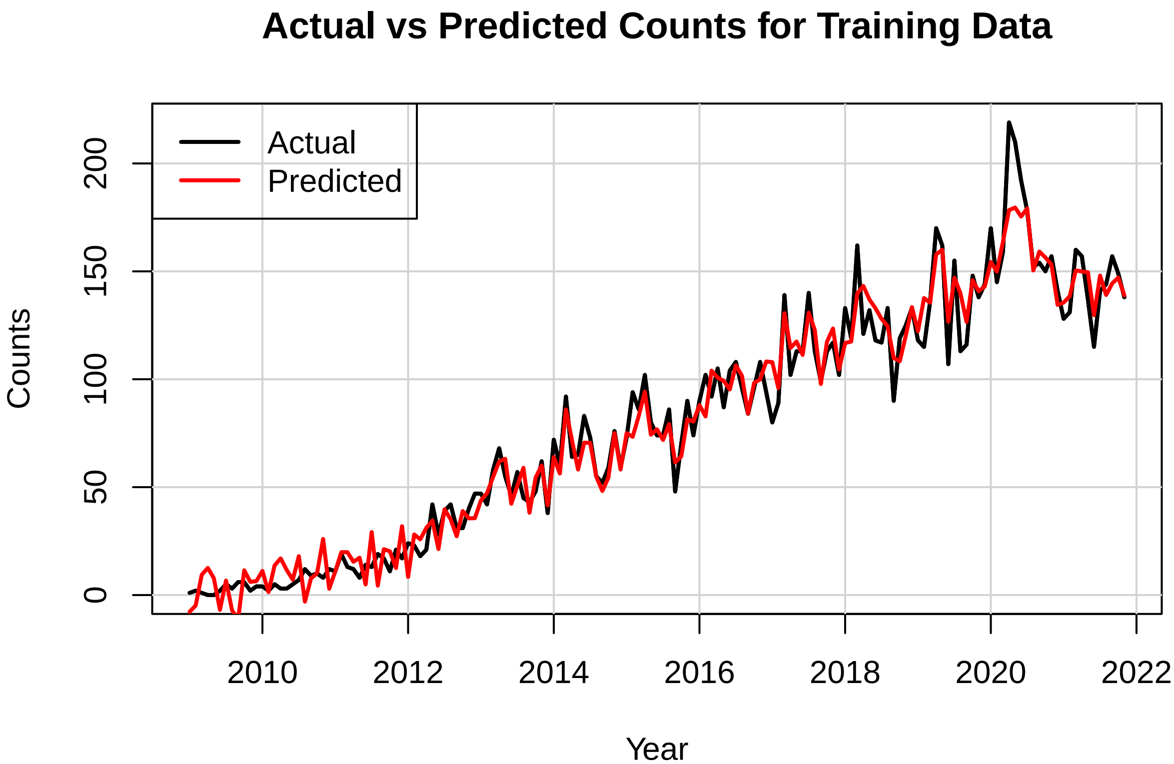
```
## FOR RESIDUALS
```

```r
cat("AIC:", arma.mod$aic, " | RMSE:", sqrt(mean(arma.mod$residuals^2)), "\n\n")
```

```
## AIC: 1197.507  | RMSE: 10.33674
```

```r
# convert back to counts
trn.count.preds <- fitted(arma.mod) + monthly_avg + four.month.avg +
                   forty.month.avg + lm.fit.d3$fitted.values

# overlay fitted to final residuals training data
ts.plot(xt_trn,
        type = "l",
        main = "Actual vs Predicted Counts for Training Data",
        xlab = "Year",
        ylab = "Counts",
```

```r
        lwd=2)
grid(lty="solid")
lines(xt_trn, lwd=2)
lines(list(x=time(xt_trn), y=trn.count.preds),
      lwd=2,
      col="red")
legend("topleft", cex = 1, c("Actual", "Predicted"),
       lty = 1, col = c("black", "red"), lwd=2)
```




```r
# output training RMSE for counts
cat("FOR COUNTS\n")
```

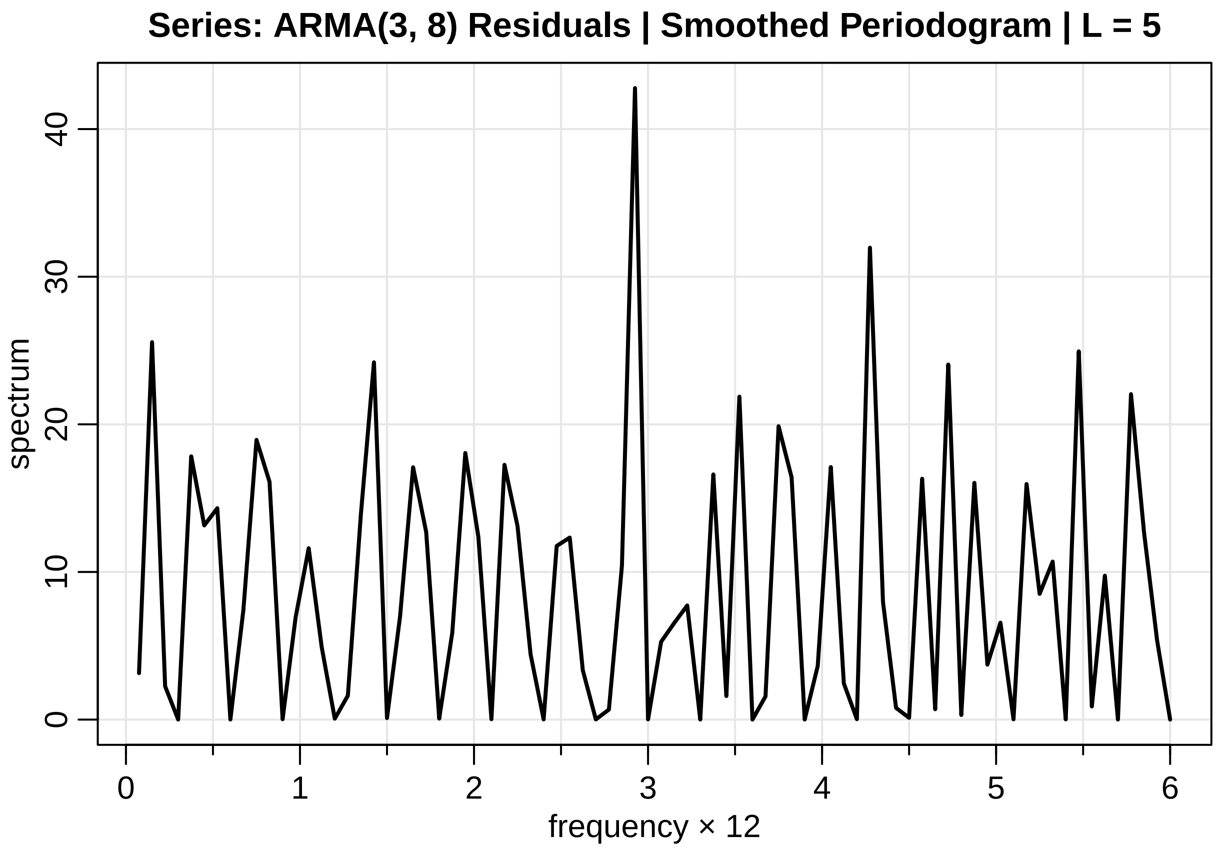
```
## FOR COUNTS
```

```r
cat("RMSE:", sqrt(mean((xt_trn-trn.count.preds)^2)), "\n\n")
```

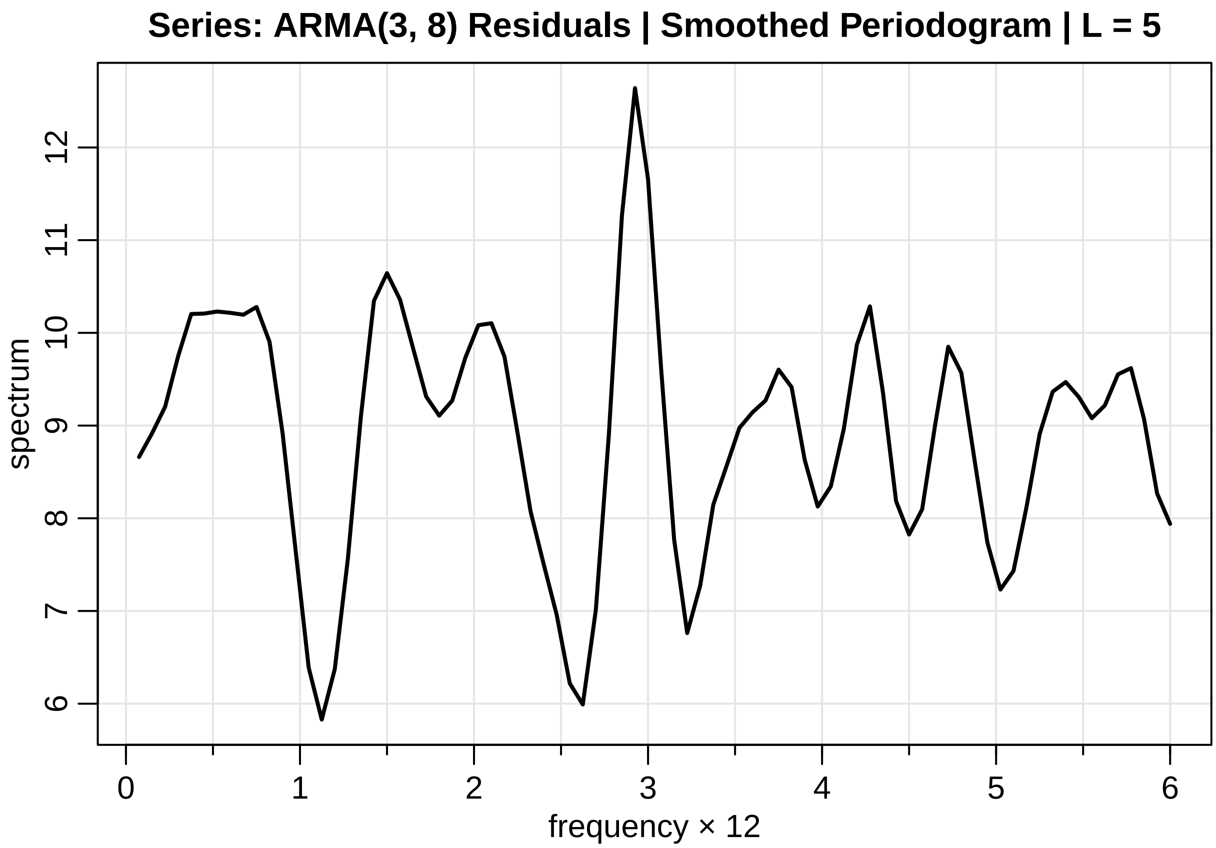
```
## RMSE: 10.33674
```

Spectrum of Residuals from ARMA(3, 8) Model

```r
# assess spectrum residuals from ARMA
arma.res.spec <- mvspec(resid(arma.mod),
                        log="no",
                        main="Series: ARMA(3, 8) Residuals | Smoothed Periodogram | L = 5",
                        lwd=2)
```

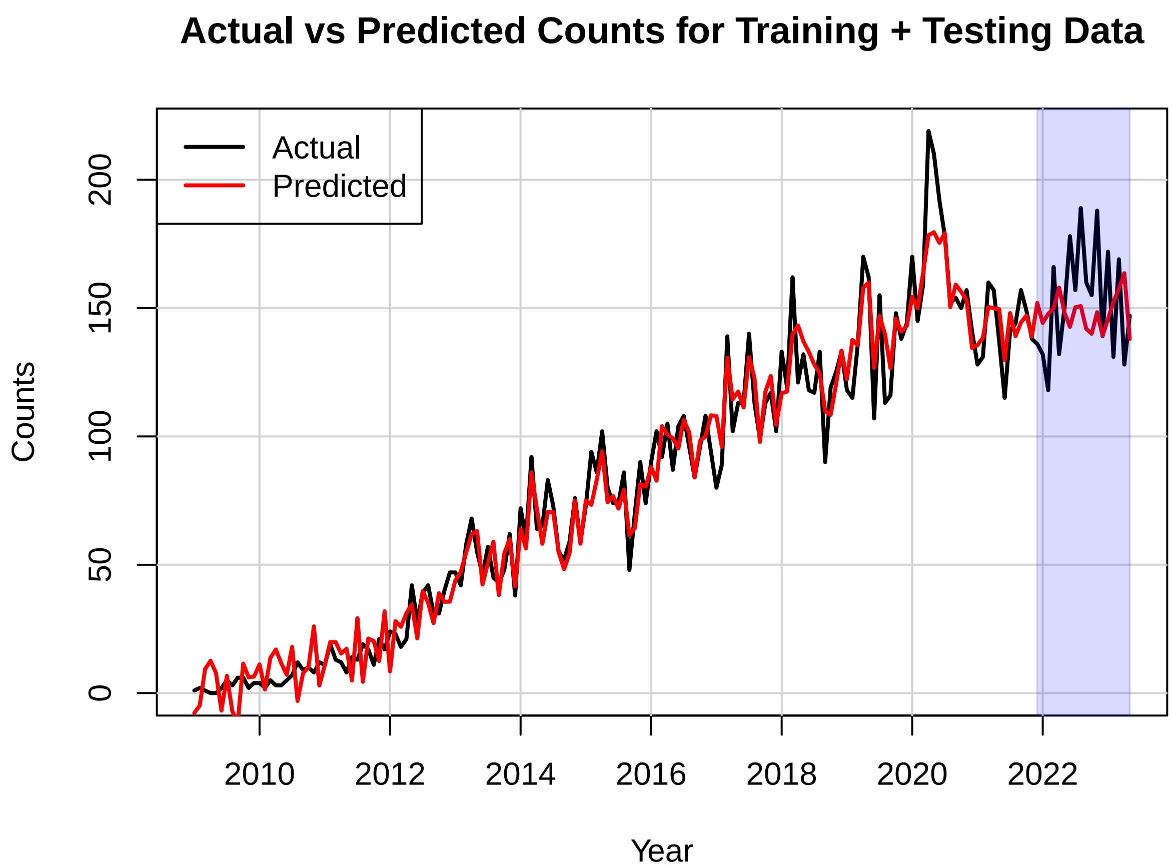



```r
# non-parametric
arma.res.spec.np <- mvspec(resid(arma.mod),
                          kernel("modified.daniell", c(2, 2)),
                          log="no",
                          main="Series: ARMA(3, 8) Residuals | Smoothed Periodogram | L = 5",
                          lwd=2)
```



Testing Performance with trained ARMA(3, 8)

```
## TRAINING + TESTING COUNTS PLOTS

# convert back to counts (+ cycles + trend)
tot.cnt.preds <- c(fitted(arma.mod), predict(arma.mod, 17)$pred) +
                   monthly_avg + four.month.avg + forty.month.avg +
                   predict(lm.fit.d3, data.frame(time=as.vector(time(xt))))

# overlay fitted to final residuals training data
ts.plot(xt,
        type = "l",
        main = "Actual vs Predicted Counts for Training + Testing Data",
        xlab = "Year",
        ylab = "Counts")
grid(lty="solid")
lines(xt, lwd=2)
lines(list(x=time(xt), y=tot.cnt.preds),
      lwd=2,
      col="red")
rect(xleft=head(time(xt_tst),1), xright=tail(time(xt_tst),1),
     ybottom=par("usr")[3], ytop=par("usr")[4],
     density=NA, col = adjustcolor("blue", alpha = 0.15))
legend("topleft", cex = 1, c("Actual", "Predicted"),
       lty = 1, col = c("black", "red"), lwd=2)
```




```
## ONLY TRAINING DATA PERFORMANCE

trnpreds <- tot.cnt.preds[1:(length(tot.cnt.preds)-18)]

# output training RMSE for counts
cat("Testing RMSE:", sqrt(mean((xt_trn - trnpreds)^2)), "\n\n")
```

```
## Testing RMSE: 10.33674
```

```
## ONLY TESTING DATA PERFORMANCE

tstpreds <- tot.cnt.preds[(length(tot.cnt.preds)-17):length(tot.cnt.preds)]

# output testing RMSE for counts
cat("Testing RMSE:", sqrt(mean((xt_tst - tstpreds)^2)), "\n\n")
```

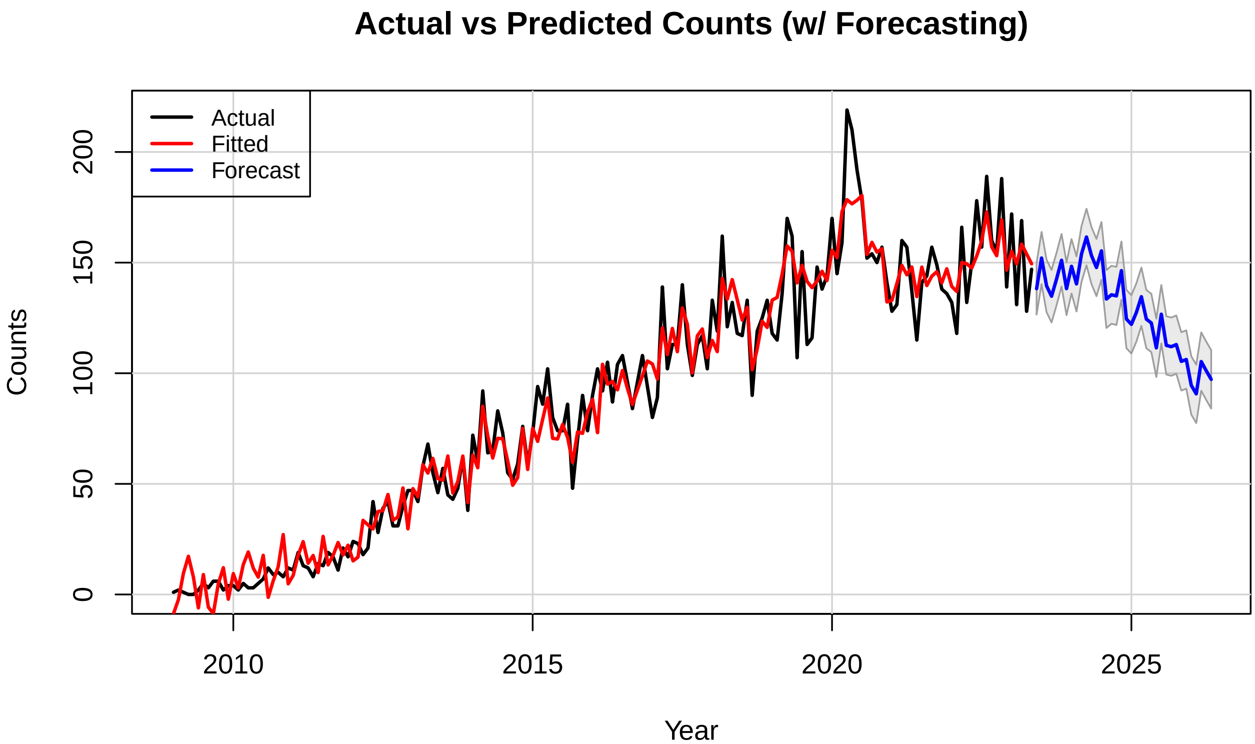
```
## Testing RMSE: 23.25007
```

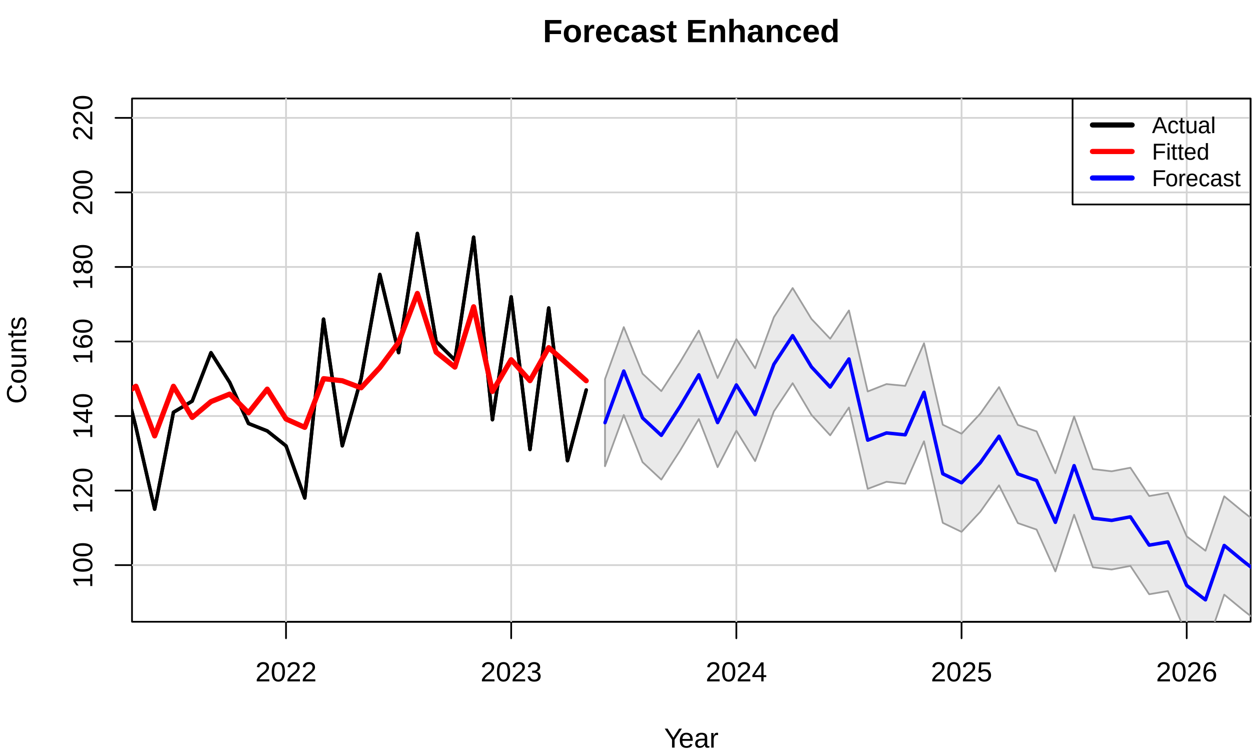
Making Forecasts Beyond Available Data

```
# refit trend, and then remove
xt_df <- data.frame(time=as.vector(time(xt)), count=as.vector(xt))
t.fit <- lm(count ~ poly(time, 3), data=xt_df) # fit
detrended_xt <- xt - t.fit$fitted.values # remove trend

## remove cycles again
```

```r
ann.matrix <- matrix(detrended_xt, byrow = TRUE, ncol = 12)
ann.matrix[15, c(6:12)] <- NA # incomplete year 2023
monthly_avg <- colMeans(ann.matrix, na.rm = TRUE)

# remove annual cycle
final_xt <- detrended_xt - monthly_avg

four.month.matrix <- matrix(final_xt, byrow = TRUE, ncol = 4)
four.month.matrix[44, c(2:4)] <- NA # incomplete year 2023
four.month.avg <- colMeans(four.month.matrix, na.rm = TRUE)

# remove 4 month cycle
final_xt <- final_xt - four.month.avg

forty.month.matrix <- matrix(final_xt, byrow = TRUE, ncol = 40)
forty.month.matrix[5, c(14:40)] <- NA # incomplete year 2023
forty.month.avg <- colMeans(forty.month.matrix, na.rm = TRUE)

# remove 40 month cycle
final_xt <- final_xt - forty.month.avg


# fit arima using regular arima model (same coefficients were found)
full.arma.mod <- arima(final_xt, order=c(3, 0, 8))

## FORECASTING PLOT

# make forecast
fore <- predict(full.arma.mod, 36)

fore.preds <- ts(start = c(2023, 1), end = c(2026, 5), frequency = 12)
fore.preds[-c(1:5)] <- fore$pred

# generate fitted values
tot.fitted <- fitted(full.arma.mod) +
                     monthly_avg + four.month.avg + forty.month.avg +
                     fitted(t.fit)

# convert to counts (have to align starting at Jan to recycle with the
# averages correctly)
fore.preds <- fore.preds +
              monthly_avg + four.month.avg + forty.month.avg +
              predict(t.fit, data.frame(time=as.vector(time(fore.preds))))

# remove NAs
fore.preds <- window(fore.preds, start=c(2023, 6))

# for CI
U <- fore.preds + fore$se
L <- fore.preds - fore$se
xx <- c(time(U), rev(time(U)))
yy <- c(L, rev(U))
```

```r
# overlay fitted to final residuals training data
par(mfrow=c(2, 1))
ts.plot(xt,
        type = "l",
        main = "Actual vs Predicted Counts (w/ Forecasting)",
        xlab = "Year",
        ylab = "Counts",
        xlim=c(2009, 2026.3))
grid(lty="solid")
lines(xt, lwd=2)
lines(list(x=time(xt), y=tot.fitted),
      lwd=2,
      col="red")
polygon(xx, yy, border = 8, col = gray(.6, alpha = .2))
lines(list(x=time(fore.preds), y=fore.preds), lwd=2, col="blue")
legend("topleft", cex = 0.8, c("Actual", "Fitted", "Forecast"),
       lty = 1, col = c("black", "red", "blue"), lwd=2)

ts.plot(xt,
        type = "l",
        xlab = "Year",
        ylab = "Counts",
        main = "Forecast Enhanced",
        lwd = 2,
        xlim=c(2021.5, 2026.1),
        ylim=c(90, 220))
grid(lty="solid")
lines(xt, lwd = 2)
lines(list(x=time(xt), y=tot.fitted),
      lwd=3,
      col="red")
polygon(xx, yy, border = 8, col = gray(.6, alpha = .2))
lines(list(x=time(fore.preds), y=fore.preds), lwd=2, col="blue")
legend("topright", cex = 0.8, c("Actual", "Fitted", "Forecast"),
       lty = 1, col = c("black", "red", "blue"), lwd=3)
```


Actual vs Predicted Counts (w/ Forecasting)
Actual
Fitted
Forecast
Counts
0
50
100
150
200
2010
2015
2020
2025
Year


Forecast Enhanced
Actual
Fitted
Forecast
Counts
100
120
140
160
180
200
220
2022
2023
2024
2025
2026
Year