Series temporales, práctica 1: conjunto de datos meteorológicos de Granada-aeropuerto Chauchina medidos por AEMET.

Carlos Manuel Sequí Sánchez

alumno: Carlos Manuel Sequí Sánchez asignatura: Series temporales y minería de flujos de datos Master: ciencias de datos e ingenieria de computadores Trabajo autónomo I: Series Temporales Fecha entrega: 25-4-2019

TEORÍA

Preprocesamiento

Para el preprocesamiento del conjunto de datos escogido, en el caso del primer problema, simplemente se ha procedido a eliminar los valores perdidos del conjunto, ya que pienso que no influye eliminar unos cuantos valores de temperatura máxima de algunos de los dias de todos los meses estudiados dada la estacionalidad escogida (cada mes puede carecer de la temperatura máxima de alguno de los días). En el segundo de los problemas, por contra, he preferido no eliminar los dias en los que había datos perdidos, ya que la estacionalidad escogida es de 365 y consideraba falseamiento de datos hacerlo. Por tanto, en lugar de eliminar los dias que carecía de valor para la Tmax, he procedido a realizar una imputación de valores mediante interpolación lineal.

Análisis de tendencia y estacionalidad

Estacionareidad

He aplicado transformación logarítmica a los datos con el fin de evitar problemas a la hora de tratar la estacionareidad

Modelado de series temporales

PRÁCTICA

Problema 1. ¿Qué valores de temperatura máxima, a escala mensual, se espera que tengan los meses de Marzo y de Abril de 2018?

Primeramente leemos el conjunto de datos que contiene los siguientes atributos:
- Columna 1 : Identificador Estación
- Columna 2 : Fecha
- Columna 3 : Temperatura Máxima (ºC)
- Columna 4 : Hora Temperatura Máxima
- Columna 5 : Temperatura mínima (ºC)
- Columna 6 : Hora Temperatura mínima
- Columna 7 : Temperatura Media (ºC)
- Columna 8 : Racha máxima de viento (Km/h)
- Columna 9 : Hora de Racha Máxima
- Columna 10 : Velocidad media de Viento (Km/h)
- Columna 11 : Hora de Velocidad Máxima de viento
- Columna 12 : Precipitacion Total diaria (mm)

- Columna 13 : Precipitacion de 0 a 6 horas (mm)
- Columna 14 : Precipitacion de 6 a 12 horas (mm)
- Columna 15 : Precipitacion de 12 a 18 horas (mm)
- Columna 16 : Precipitacion de 18 a 24 horas (mm)

Librerías...

```
library(tseries)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

Leemos el dataset y, como solo nos interesa la fecha y la temperatura máxima nos quedamos con tan solo esos datos.

```
datos = read.csv("5530E.csv", header = TRUE, sep=";")
datos = datos[,c("Fecha","Tmax")]
datos$Fecha = as.Date(datos$Fecha)
```

Veamos los valores NA...

```
apply(datos, 2, function(atributo){sum(is.na(atributo))})
```

```
## Fecha  Tmax
##     0   124
```

Eliminamos las instancias(días) donde hay al menos algún valor NA de temperatura máxima

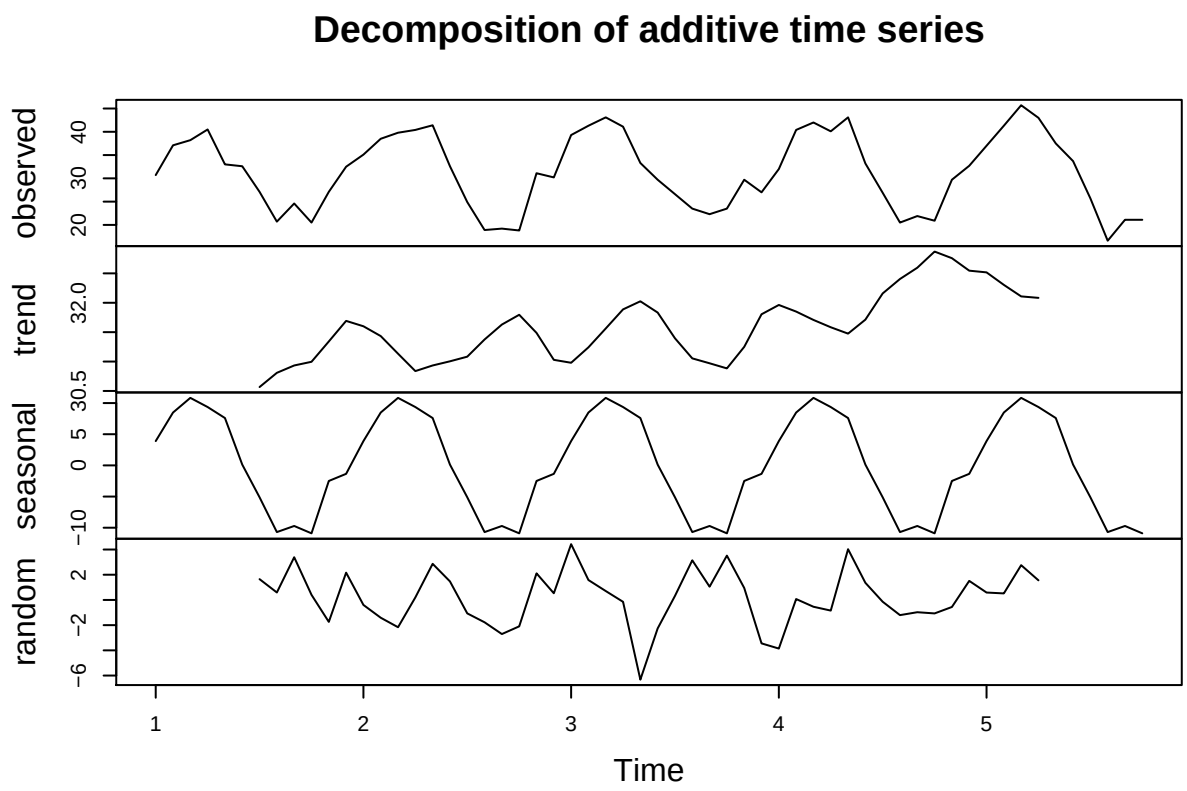
```
datos = datos[complete.cases(datos),]
```

La serie que nos interesa es la temperatura máxima de cada uno de los meses, es decir, obtenemos la temperatura máxima de cada mes de cada año:

```
# agrupamos por año y mes y tomamos el maximo de cada uno
datos = datos %>%
mutate(month = format(Fecha, "%m"), year = format(Fecha, "%Y")) %>%
group_by(year, month) %>%
summarise(total = max(Tmax))

serie = datos$total # aqui estan las temperaturas máximas de cada mes de todos los años
```

Obtenemos ahora la cantidad de datos a predecir y la serie temporal en sí

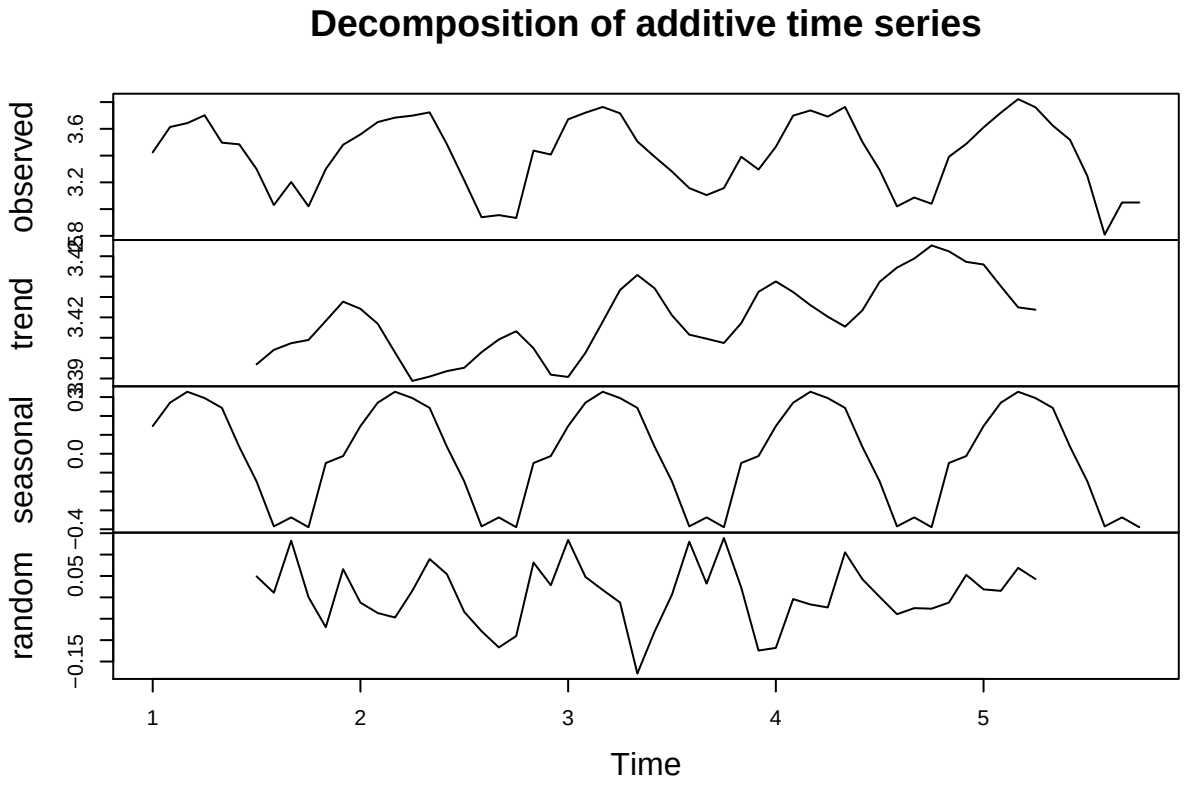
```r
Npred = 2 # cantidad de datos a predecir (temperaturas máximas de marzo y abril)
serie.ts = ts(serie, frequency = 12) # frequency set to 12 to set stationality each 12 months
plot(decompose(serie.ts))
```



Observamos en la gráfica:
-los valores de la serie
-la tendencia calculada mediante filtros
-la estacionalidad repetida cada 12 instantes de tiempo -lo que queda de la serie al eliminar tendencia y estacionalidad

Como vemos, es posible que exista una alta varianza en los datos sin estacionalidad y tendencia, por lo que probamos a aplicar una transformación logarítmica para evitar problemas a la hora de tener en cuenta la estacionariedad, ya que sabemos que una serie con estacionariedad no varía ni en media ni en varianza.

```r
serie.ts.log = log(serie.ts)
serie.log = log(serie)
plot(decompose(serie.ts.log))
```



De forma ligera, la varianza disminuye a lo largo del tiempo, lo que producirá menores problemas a la hora de calcular la estacionariedad. Por simplicidad usare la variable serie como serie.log (con la transformación hecha)

```
serie = serie.log
```

Observamos como el atributo seasonal que nos da la función decompose consta de los mismos valores para cada año de todos los meses, lo cual nos dará la capacidad de calcular la estacionalidad para restársela a la serie.

```
decompose(serie.ts.log)$seasonal
```

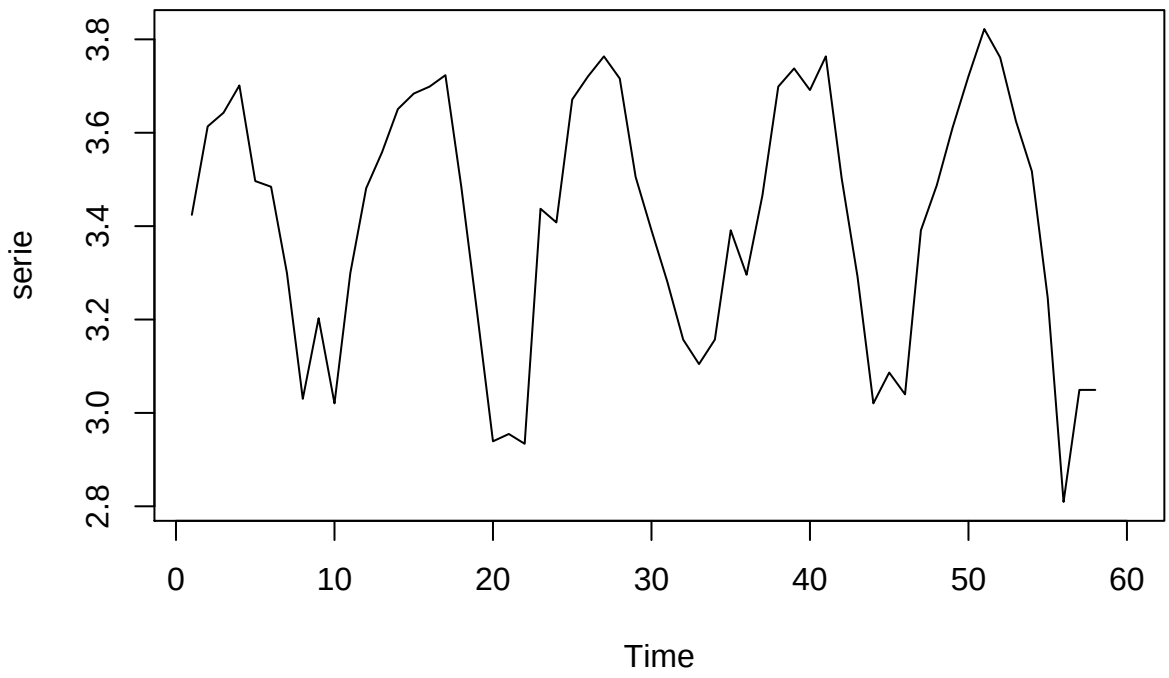
```
##           Jan         Feb         Mar         Apr         May         Jun
## 1  0.14624562  0.27038082  0.32826498  0.29462730  0.24274200  0.03660037
## 2  0.14624562  0.27038082  0.32826498  0.29462730  0.24274200  0.03660037
## 3  0.14624562  0.27038082  0.32826498  0.29462730  0.24274200  0.03660037
## 4  0.14624562  0.27038082  0.32826498  0.29462730  0.24274200  0.03660037
## 5  0.14624562  0.27038082  0.32826498  0.29462730  0.24274200  0.03660037
##           Jul         Aug         Sep         Oct         Nov         Dec
## 1 -0.14645133 -0.38469232 -0.33717450 -0.38909044 -0.04904490 -0.01240760
## 2 -0.14645133 -0.38469232 -0.33717450 -0.38909044 -0.04904490 -0.01240760
## 3 -0.14645133 -0.38469232 -0.33717450 -0.38909044 -0.04904490 -0.01240760
## 4 -0.14645133 -0.38469232 -0.33717450 -0.38909044 -0.04904490 -0.01240760
## 5 -0.14645133 -0.38469232 -0.33717450 -0.38909044
```

Calculamos los instantes de tiempo de la serie para train y para test

```r
# para train
tiempo = 1:length(serie)
# para test
tiempoTs = (tiempo[length(tiempo)]+1):(tiempo[length(tiempo)]+Npred)
```

Representamos a continuación la serie que tenemos en este momento:

```r
plot.ts(serie,  xlim=c(1, tiempoTs[length(tiempoTs)]))
```



Modelamos la tendencia, la cual podemos aparentemente ajustarla de forma lineal, sabiendo que:
serie = parametroA * tiempo + parametroB
Con ayuda de la función lm calculamos dichos parámetros:

```r
parametros.H1 = lm(serie ~tiempo)
parametros.H1
```

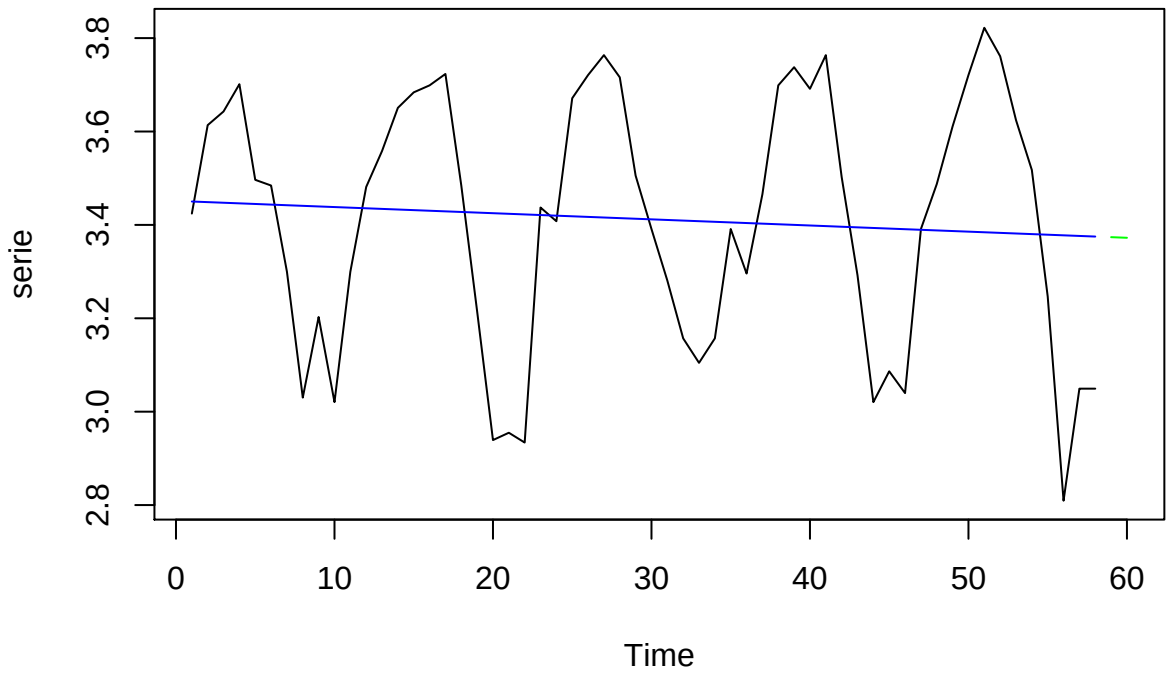
```
##
## Call:
## lm(formula = serie ~ tiempo)
##
## Coefficients:
## (Intercept)       tiempo
##    3.451367    -0.001317
```

Tomamos Intercept como termino independiente (parametroB) y el otro como el que multiplica al tiempo para poder calcular la serie (parametroA). Para modelar la tendencia usamos la fórmula descrita antes:

```r
# tendencia estimada para los datos de train
tendEstimadaTr =  parametros.H1$coefficients[1] + tiempo*parametros.H1$coefficients[2]
# tendencia estimada para las predicciones de test
tendEstimadaTs =  parametros.H1$coefficients[1] + tiempoTs*parametros.H1$coefficients[2]
```

Comprobamos visualmente si la tendencia se ajusta al modelo que tenemos de la serie temporal:

```r
plot.ts(serie, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempo, tendEstimadaTr, col = "blue")
lines(tiempoTs, tendEstimadaTs, col = "green")
```



Validamos de forma estadística que el modelo lineal sea correcto, comprobando que los errores a lo largo de la serie se distribuyen mediante una distribución normal. Para ello aplicamos el test de Jarque-Bera:

```r
JB = jarque.bera.test(parametros.H1$residuals)
JB
```

```
##
##  Jarque Bera Test
##
## data:  parametros.H1$residuals
## X-squared = 3.6086, df = 2, p-value = 0.1646
```

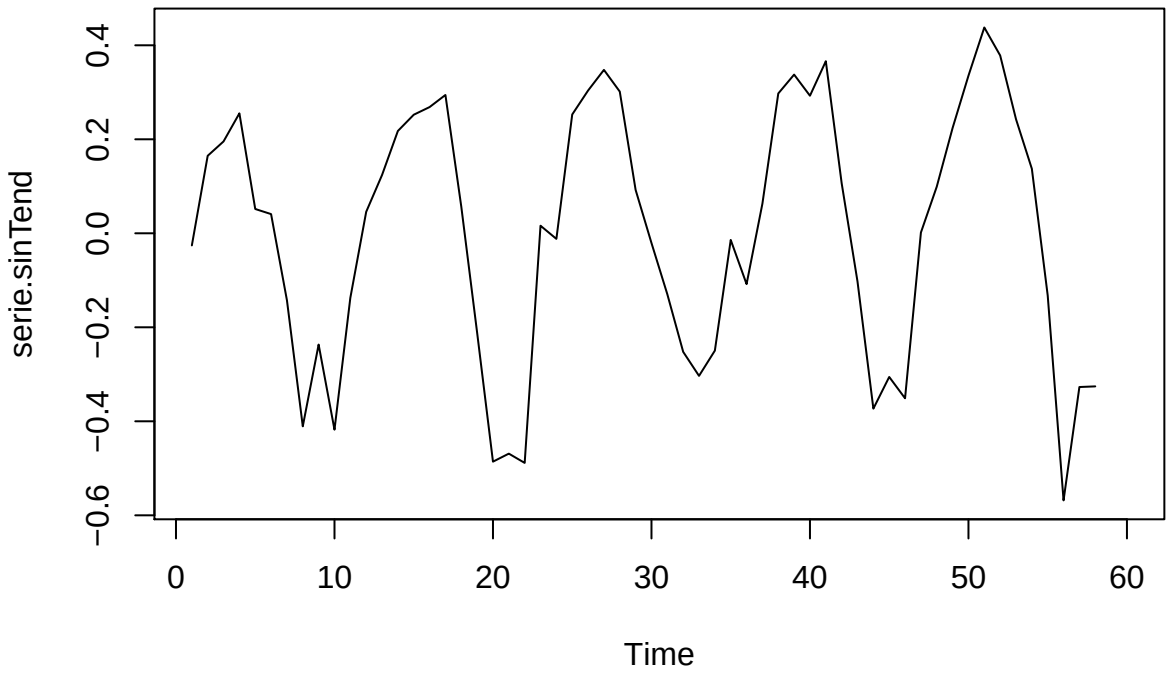
Podemos asumir, con una confianza del 95%, que no hay diferencia significativa de los datos de error con respecto a los de una distribución normal, debido a que el p-value es mayor que 0.05.

Procedemos por tanto a eliminar la tendencia de la serie actual:

```
serie.sinTend = serie - tendEstimadaTr
```

Comprobamos como queda la serie sin tendencia:

```
plot.ts(serie.sinTend, xlim=c(1, tiempoTs[length(tiempoTs)]))
```



Como vemos, la serie sin tendencia prácticamente es la misma que con ella, ya que la pendiente del modelo lineal creado para el cálculo de ésta era ínfima.

Procedemos, una vez eliminada la tendencia, a deshacernos de la estacionalidad con ayuda de la función decompose.

```
k = 12 # aquí guardamos los datos estacionales
estacionalidad = decompose(serie.ts.log)$seasonal[1:k]
```

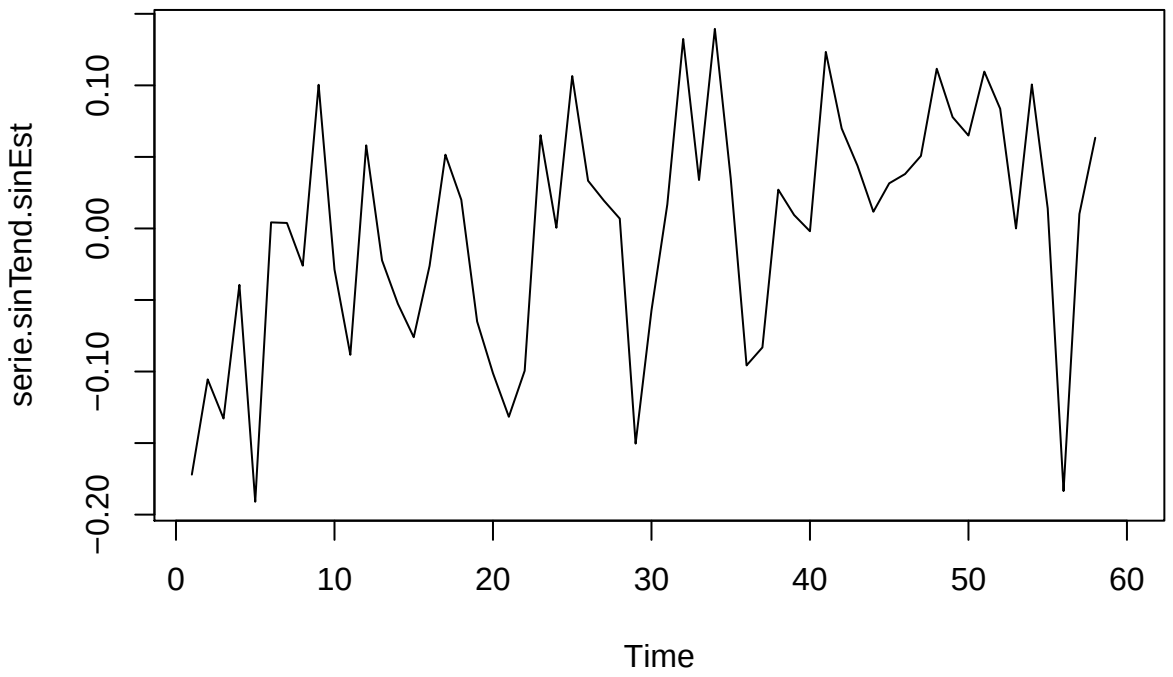
Eliminamos dicha estacionalidad a la serie.

```
serie.sinTend.sinEst = serie.sinTend - estacionalidad
```

```
## Warning in serie.sinTend - estacionalidad: longitud de objeto mayor no es
## múltiplo de la longitud de uno menor
```

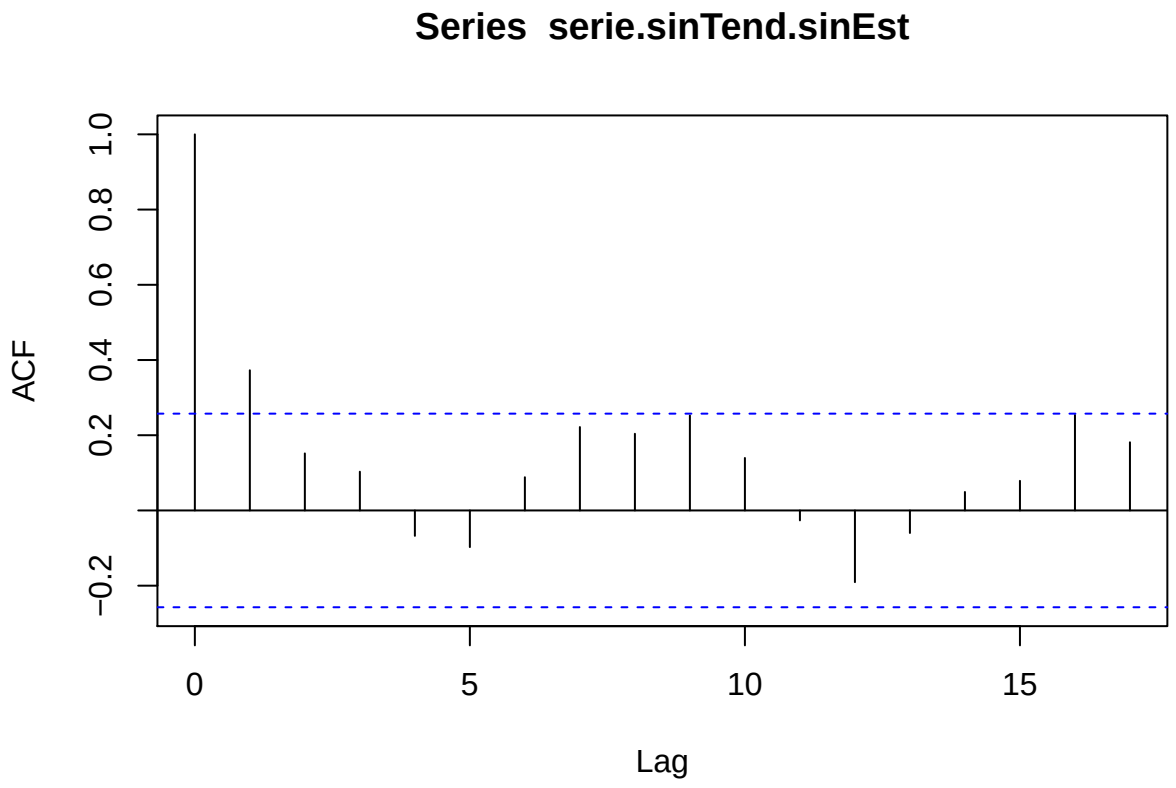
Comprobamos como se queda la serie sin dicha estacionalidad

```
plot.ts(serie.sinTend.sinEst, xlim=c(1, tiempoTs[length(tiempoTs)]))
```



Ahora que tan solo nos queda la componente irregular tras haber eliminado las componentes de tendencia y estacionalidad, procedemos a comprobar si la serie es estacionaria o no mediante el test ACF.

```
acf(serie.sinTend.sinEst)
```



A priori parece ser, debido a la rapidez de bajada entre instantes de tiempo y a la estabilización en forma de ''olas", que existe una clara estacionariedad. Aun así, probaré aplicar el test de Dickey-Fuller aumentado para asegurarnos de ello.

```
adf.test(serie.sinTend.sinEst)
```

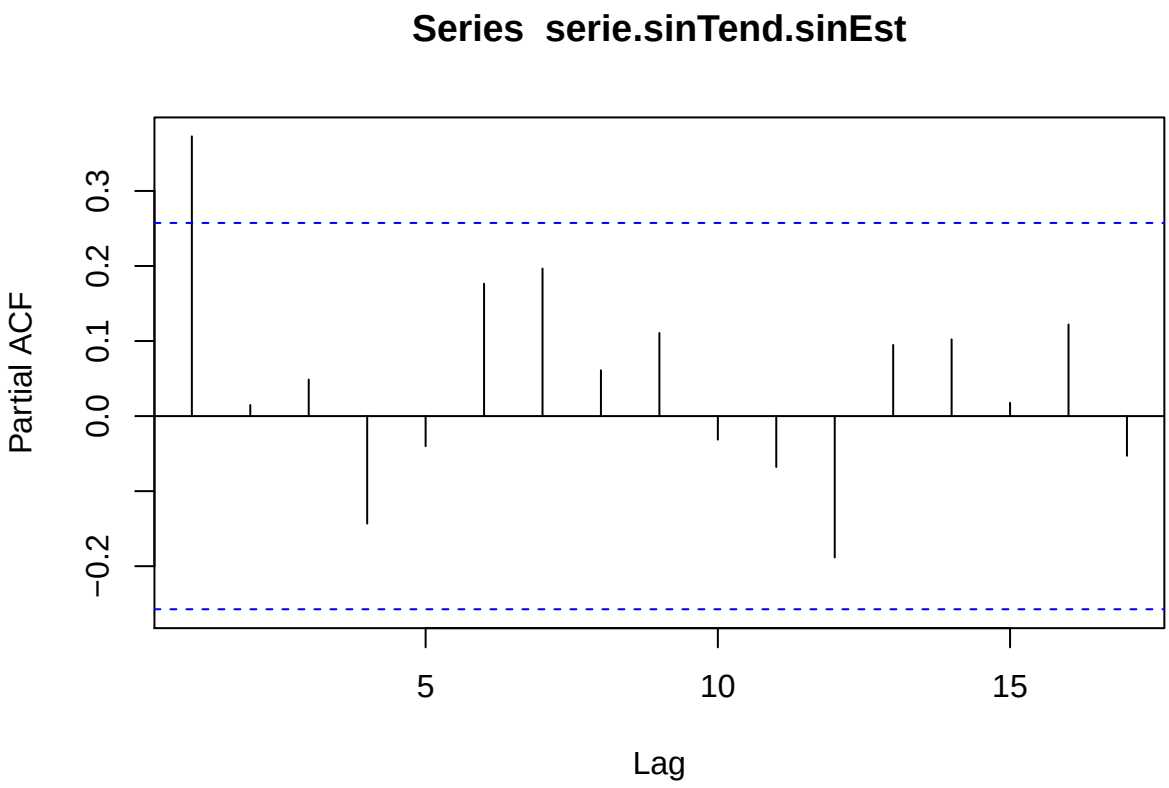
```
## Warning in adf.test(serie.sinTend.sinEst): p-value smaller than printed p-
## value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  serie.sinTend.sinEst
## Dickey-Fuller = -4.9184, Lag order = 3, p-value = 0.01
## alternative hypothesis: stationary
```

Al tomar como hipótesis nula que la serie es estacionaria y al ser el p-value inferior a 0.05, podemos asegurar con un 95% de probabilidad que la serie es estacionaria, por tanto la tomamos así. Debido a esto no será necesario realizar diferenciación para convertir la serie en estacionaria.

Observamos el acf parcial para ver como influyen de forma individual cada uno de los instantes de tiempo sobre el instante actual.

```
pacf(serie.sinTend.sinEst)
```



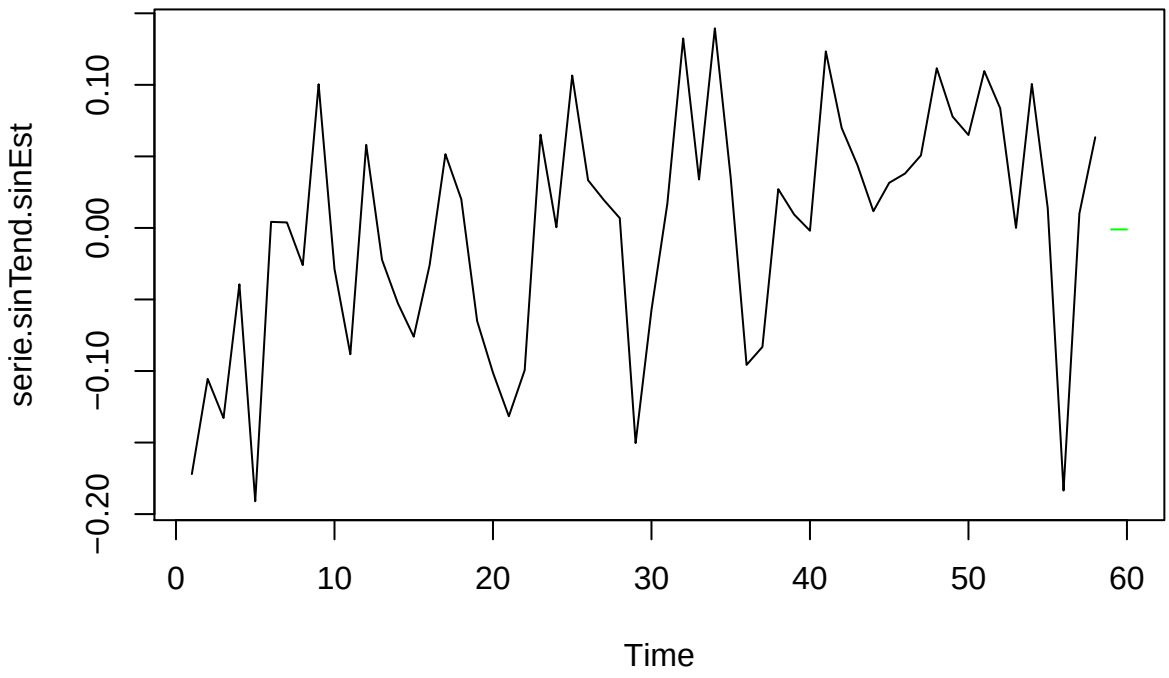
A la vista de las gráficas ofrecidas por ACF y PACF, me resulta de mayor claridad decidirme por un modelo autoregresivo en lugar de uno de medias móviles debido a que en ACF los valores de cada instante son más altos y sobrepasan los límites más que los de PACF, además de que se ve de forma mucho más clara esa estabilización en forma de ''olas" que he dicho anteriormente en la gráfica del ACF. Por ello mismo, deberíamos poder modelar la componente irregular de la serie con un modelo autoregresivo de orden 0 (fijándonos en la gráfica PACF), utilizando un modelo ARIMA con cero diferenciaciones. Entrenamos dicho modelo a continuación:

```
# coeficiente de autoregresión = 0, diferenciaciones = 0, coeficiente de medias moviles = 0
modelo = arima(serie.sinTend.sinEst, order = c(0,0,0))

# calculamos las predicciones
predicciones = predict(modelo, n.ahead = Npred)
valoresPredichos = predicciones$pred
valoresPredichos
```

```
## Time Series:
## Start = 59
## End = 60
## Frequency = 1
## [1] -0.001059526 -0.001059526
```

Observamos ahora de forma visual el modelo incluyendo la serie sin tendencia ni estacionalidad

```
plot.ts(serie.sinTend.sinEst, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempoTs,valoresPredichos, col="green")
```



Parece ser que los resultados obtenidos no son buenamente fieles a lo esperado, quizás debido a la cercanía de los valores en los sucesivos instantes de tiempo en las gráficas ACF y PACF. Visto lo visto, puede decirse que nuestra serie es prácticamente ruido blanco debido a los resultados obtenidos como predicciones.
Finalmente probamos la validez de nuestro modelo (sus errores) mediante tests estadísticos. Aplicamos varios:

Test de Box-Pierce: comprobamos que los residuos que nos quedan son aleatorios:

```
Box.test(modelo$residuals)
```

```
##
##  Box-Pierce test
##
## data:  modelo$residuals
## X-squared = 8.0586, df = 1, p-value = 0.004529
```

Tras este test comprobamos que los errores obtenidos no son de forma aleatoria, por lo que el modelo arima creado no es correcto. Debido a esto, probaré el ajuste de un modelo de medias móviles de grado 1 (el grado determinado por la gráfica ACF) para comprobar su validez:

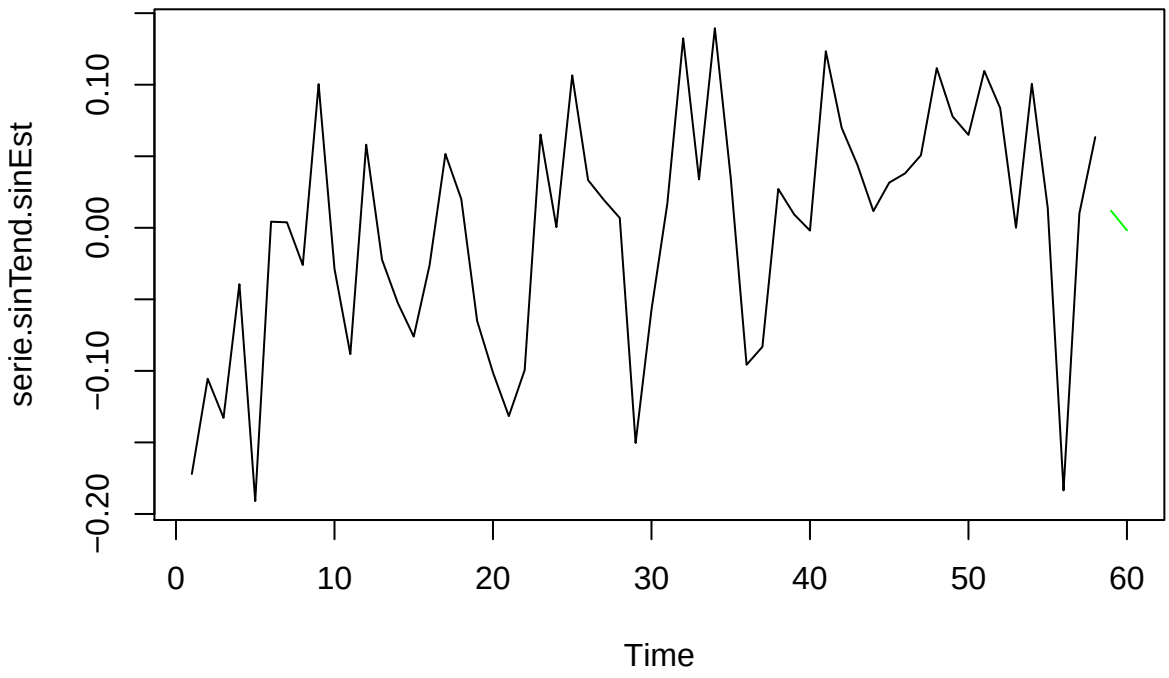
```
# coeficiente de autoregresión = 0, diferenciaciones = 0, coeficiente de medias moviles = 1
modelo = arima(serie.sinTend.sinEst, order = c(0,0,1))

# calculamos las predicciones
predicciones = predict(modelo, n.ahead = Npred)
valoresPredichos = predicciones$pred
valoresPredichos
```

```
## Time Series:
## Start = 59
```

```
## End = 60
## Frequency = 1
## [1]  0.011869071 -0.001802729
```

Observamos los valores predichos con el modelo de medias móviles con la serie sin tendencia ni estacionalidad:

```
plot.ts(serie.sinTend.sinEst, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempoTs,valoresPredichos, col="green")
```



Aplicamos de nuevo el test de Box-Pierce para la comprobación de la aleatoriedad de los errores:

```
Box.test(modelo$residuals)
```

```
##
##  Box-Pierce test
##
## data:  modelo$residuals
## X-squared = 0.0086579, df = 1, p-value = 0.9259
```

Este valor de p-value nos dice con un 95% de confianza que los errores obtenidos por el modelo creado son aleatorios, por lo que el modelo queda validado por este test.

Continuamos con los tests de normalidad.

Jarque Bera: vemos si los residuos, aunque aleatorios, son normales (media cero y desviación típica la que sea).

```
jarque.bera.test(modelo$residuals)
```

```
##
##  Jarque Bera Test
```

```
##
## data:  modelo$residuals
## X-squared = 1.3191, df = 2, p-value = 0.5171
```

Saphiro-Wilk: mismo cometido que Jarque Bera

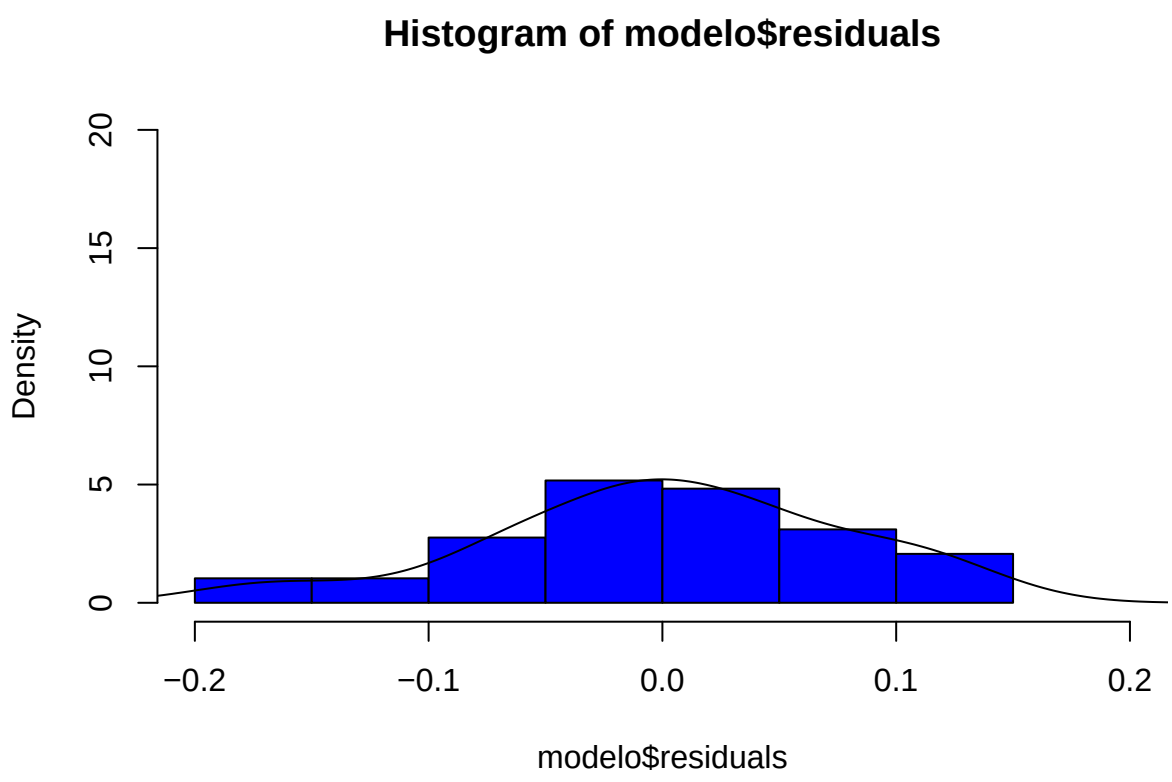
```
shapiro.test(modelo$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  modelo$residuals
## W = 0.9788, p-value = 0.4025
```

Asumimos tras la ejecución de ambos tests que los residuos siguen una distribuición normal debido a los p-values obtenidos, ya que la hipótesis nula propone que los datos no siguen una distribución normal.

A continuación obtenemos un histograma y función de densidad para obtener resultados gráficos de confirmación.

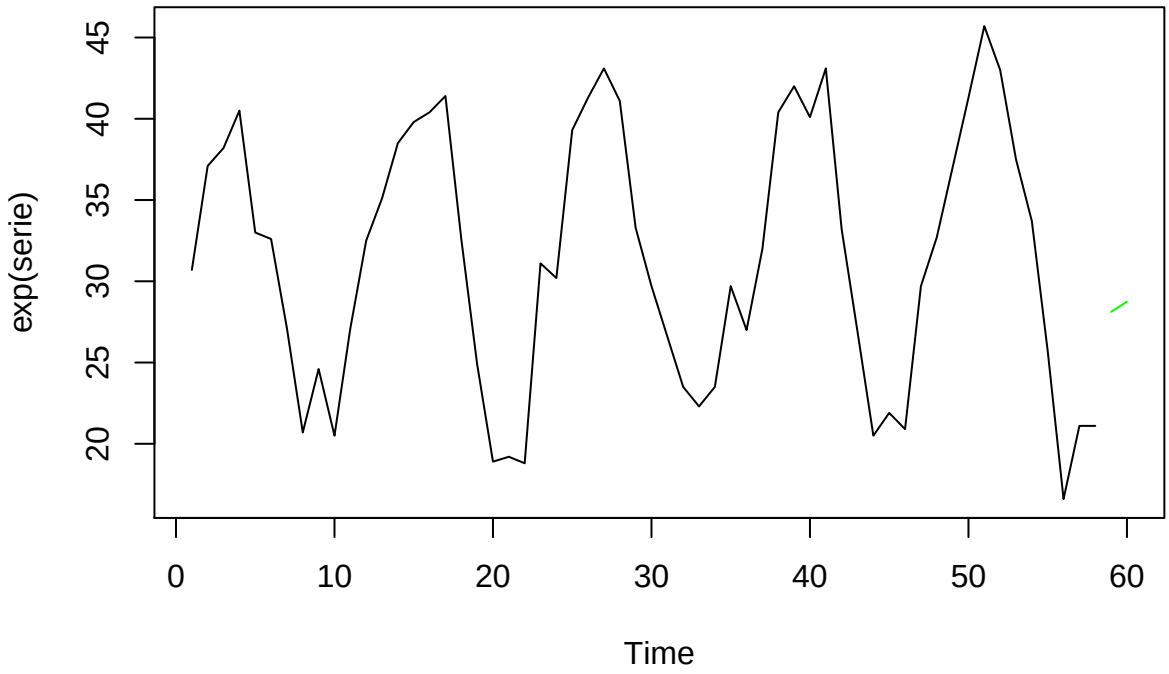
```
hist(modelo$residuals, col="blue", prob=T,ylim=c(0,20), xlim=c(-0.2,0.2))
lines(density(modelo$residuals))
```



Con estas comprobaciones el modelo queda validado, ya que los errores que nos dan son aleatorios que provienen de una distribución normal. A continuación hemos de deshacer los cambios para obtener los predicciones reales:

```
estacionalidades = c(estacionalidad[11], estacionalidad[12])

# Incluimos la estacionalidad
```

```r
valoresPredichos.Est = valoresPredichos + estacionalidades

# Incluimos la tendencia
valoresPredichos.Est.Tend = valoresPredichos.Est + tendEstimadaTs

# Transformación de los logaritmos
valoresPredichos.Est.Tend.exp = exp(valoresPredichos.Est.Tend)

# usamos exp(serie) por haber hecho al principio serie = log(serie)
plot.ts(exp(serie),xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempoTs, valoresPredichos.Est.Tend.exp, col = "green")
```



Finalmente podemos observar en la gráfica los valores de temperatura máxima obtenidos para los meses de Marzo y Abril de 2018, los cuales se corresponden respectivamente con los valores:

```r
valoresPredichos.Est.Tend.exp
```

```
## Time Series:
## Start = 59
## End = 60
## Frequency = 1
## [1] 28.12089 28.73632
```

Problema 2. ¿Qué valores de temperatura máxima, a escala diaria, se espera para la primera semana de Marzo de 2018?

Primeramente leemos el conjunto de datos que contiene los siguientes atributos:
- Columna 1 : Identificador Estación
- Columna 2 : Fecha
- Columna 3 : Temperatura Máxima (ºC)
- Columna 4 : Hora Temperatura Máxima
- Columna 5 : Temperatura mínima (ºC)
- Columna 6 : Hora Temperatura mínima
- Columna 7 : Temperatura Media (ºC)
- Columna 8 : Racha máxima de viento (Km/h)
- Columna 9 : Hora de Racha Máxima
- Columna 10 : Velocidad media de Viento (Km/h)
- Columna 11 : Hora de Velocidad Máxima de viento
- Columna 12 : Precipitacion Total diaria (mm)
- Columna 13 : Precipitacion de 0 a 6 horas (mm)
- Columna 14 : Precipitacion de 6 a 12 horas (mm)
- Columna 15 : Precipitacion de 12 a 18 horas (mm)
- Columna 16 : Precipitacion de 18 a 24 horas (mm)

Librerías...

```
library(tseries)
library(dplyr)
library(lubridate)
library(imputeTS)
```

```
##
## Attaching package: 'imputeTS'
```

```
## The following object is masked from 'package:tseries':
##
##     na.remove
```

Leemos el dataset y, como solo nos interesa la fecha y la temperatura máxima nos quedamos con tan solo esos datos.

```
datos = read.csv("5530E.csv", header = TRUE, sep=";")
datos = datos[,c("Fecha","Tmax")]
datos$Fecha = as.Date(datos$Fecha)
```

Veamos los valores NA...

```
apply(datos, 2, function(atributo){sum(is.na(atributo))})
```

```
## Fecha  Tmax
##     0   124
```

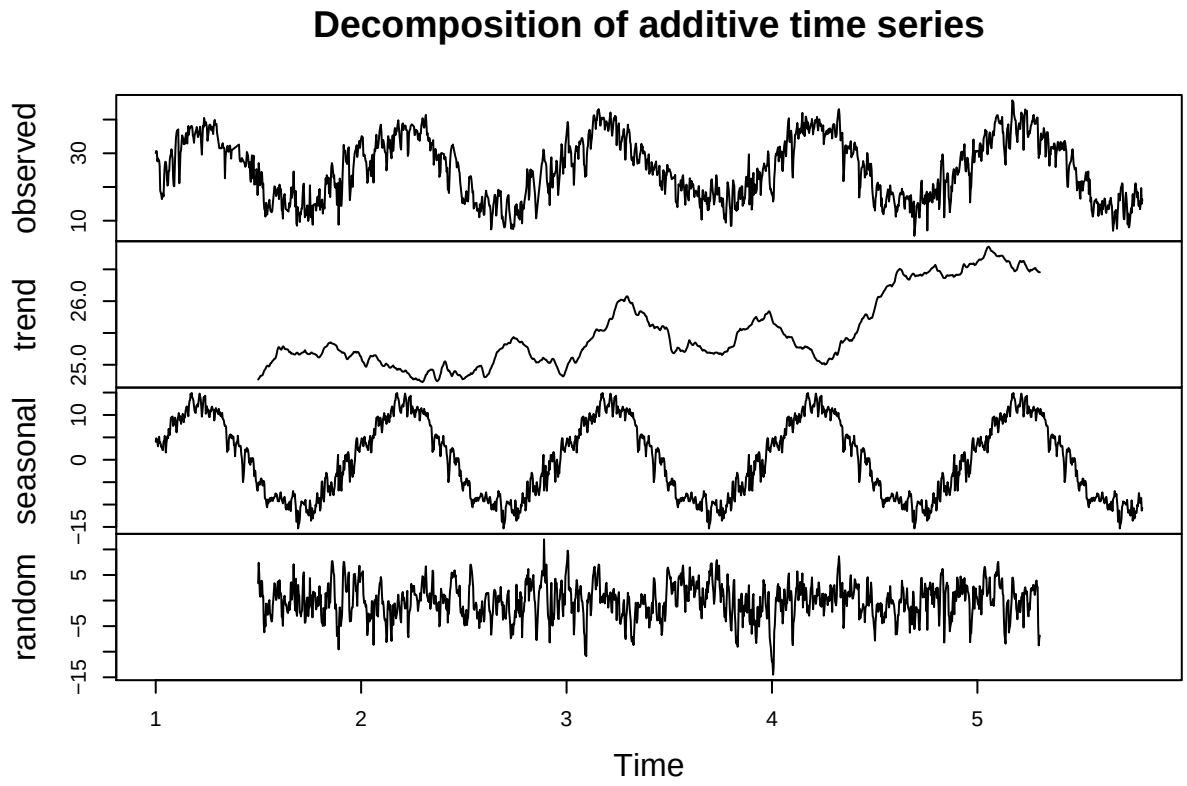
Imputamos valores mediante interpolación en las temperaturas máximas donde existan valores perdidos con el fin de no tergiversar los datos eliminando días del año.

```
datos$Tmax = na.interpolation(datos$Tmax)
```

Obtenemos ahora la cantidad de datos a predecir y la serie temporal en sí

```
Npred = 7 # cantidad de datos a predecir (temperaturas máximas de marzo y abril)
serie = datos$Tmax
```

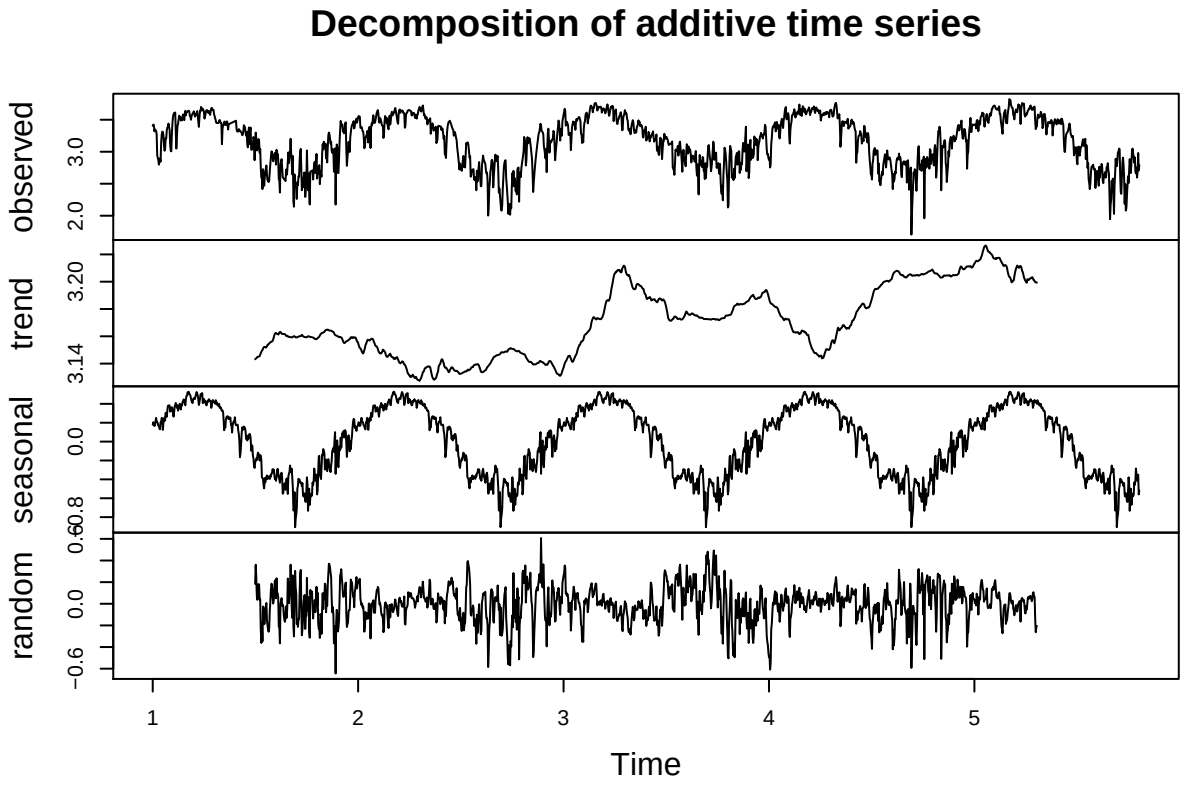
```
serie.ts = ts(serie, frequency = 365) # frequency set to 12 to set stationality each 12 months
plot(decompose(serie.ts))
```



Observamos en la gráfica:
-los valores de la serie
-la tendencia calculada mediante filtros
-la estacionalidad repetida cada 12 instantes de tiempo -lo que queda de la serie al eliminar tendencia y estacionalidad

Probamos, como hemos hecho en experimentos anteriores, a realizar una transformación logarítmica de la serie para reducir la varianza y así evitar problemas con el cálculo de la estacionariedad:

```
serie.ts.log = log(serie.ts)
serie.log = log(serie)
plot(decompose(serie.ts.log))
```



El cambio es relativamente muy pequeño, es posible que tenga mucho que ver la gran cantidad de datos, siendo difícil distinguir la variación entre la gráfica anterior y esta. Aún así, por precaución, tomaremos esta serie por válida.

Observamos los datos de estacionalidad que nos servirán en el futuro para eliminársela a la serie temporal.

```
head(decompose(serie.ts.log)$seasonal)
```

```
## Time Series:
## Start = c(1, 1)
## End = c(1, 6)
## Frequency = 365
## [1] 0.1797105 0.2053683 0.1774451 0.1648794 0.1722653 0.2133311
```

Por simplicidad en el uso de la nomenclatura utilizare la variable serie en lugar de serie.log

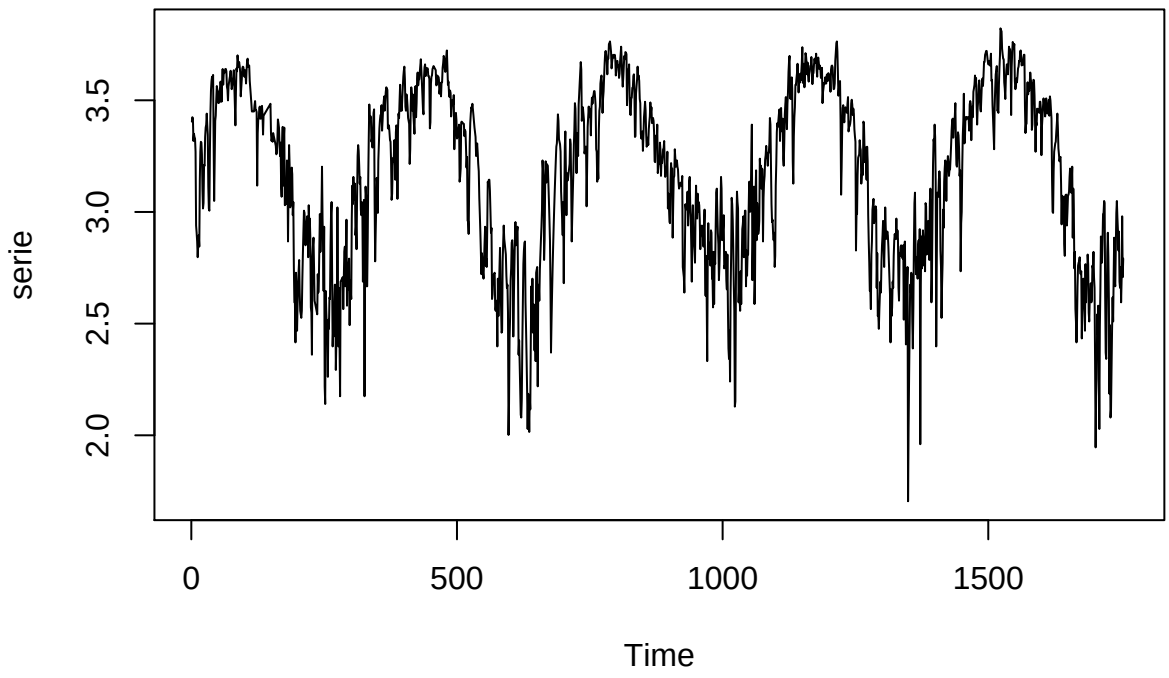
```
serie = serie.log
```

Calculamos los instantes de tiempo de la serie

```
# para train
tiempo = 1:length(serie.ts.log)
# para test
tiempoTs = (tiempo[length(tiempo)]+1):(tiempo[length(tiempo)]+Npred)
```

Representamos la serie

```
plot.ts(serie, xlim=c(1, tiempoTs[length(tiempoTs)]))
```



Modelamos la tendencia, la cual podemos aparentemente ajustarla de forma lineal, sabiendo que:
serie = parametroA * tiempo + parametroB
Con ayuda de la función lm calculamos dichos parámetros:

```
parametros.H1 = lm(serie ~tiempo)
parametros.H1
```

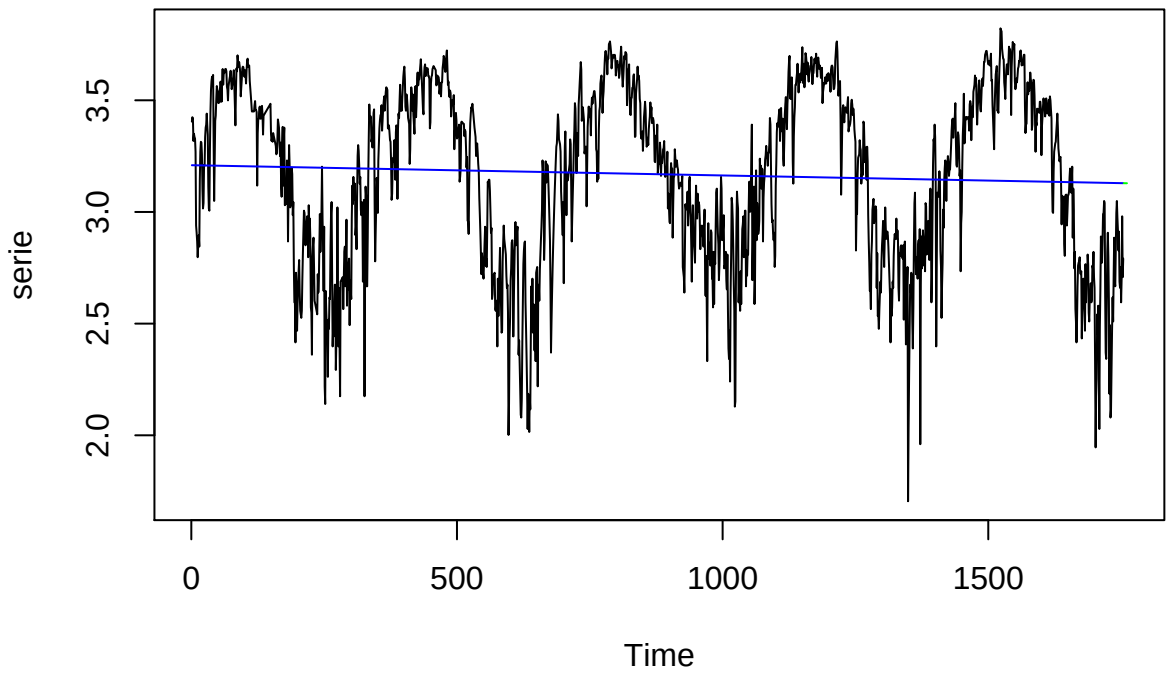
```
##
## Call:
## lm(formula = serie ~ tiempo)
##
## Coefficients:
## (Intercept)       tiempo
##   3.209e+00   -4.589e-05
```

Tomamos Intercept como termino independiente (parametroB) y el otro como el que multiplica al tiempo para poder calcular la serie (parametroA). Para modelar la tendencia usamos la fórmula descrita antes:

```
# tendencia estimada para los datos de train
tendEstimadaTr = parametros.H1$coefficients[1] + tiempo*parametros.H1$coefficients[2]
# tendencia estimada para las predicciones de test
tendEstimadaTs = parametros.H1$coefficients[1] + tiempoTs*parametros.H1$coefficients[2]
```

Comprobamos visualmente si la tendencia se ajusta al modelo que tenemos de la serie temporal:

```
plot.ts(serie, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempo, tendEstimadaTr, col = "blue")
lines(tiempoTs, tendEstimadaTs, col = "green")
```



Podemos observar los ajustes tanto de la tendencia para los datos de train como de la estimada para los datos de test.

A continuación procedemos a validar de forma estadística que el modelo lineal creado sea correcto. Con el test de Jarque Bera comprobamos si los errores siguen una distribución normal a lo largo del tiempo.

```
JB = jarque.bera.test(parametros.H1$residuals)
JB
```

```
##
##  Jarque Bera Test
##
## data:  parametros.H1$residuals
## X-squared = 102.63, df = 2, p-value < 2.2e-16
```

Como el valor del p-value es inferior a 0.05, no podemos asegurar que los residuos no guarden una diferencia significativa con los datos de una distribución normal, por lo que procedemos a intentar ajustar la tendencia con otro modelo, esta vez cuadrático.

```
# calculo de parámetros
parametros.H1 = lm(serie ~tiempo + I(tiempo^2))
parametros.H1
```

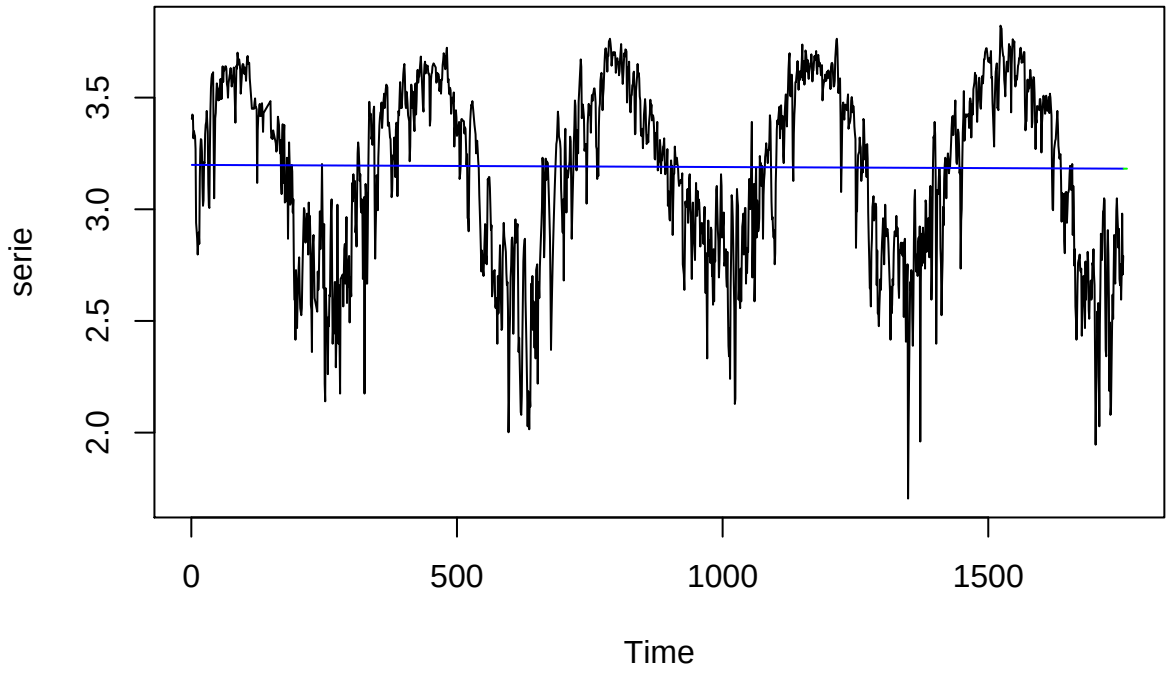
```
##
## Call:
## lm(formula = serie ~ tiempo + I(tiempo^2))
##
## Coefficients:
## (Intercept)       tiempo  I(tiempo^2)
```

```
##   3.199e+00   -9.575e-06   -2.069e-08
```

```
# tendencia estimada para los datos de train
tendEstimadaTr = parametros.H1$coefficients[1] + tiempo*parametros.H1$coefficients[2]  + tiempo*parameti
# tendencia estimada para las predicciones de test
tendEstimadaTs = parametros.H1$coefficients[1] + tiempoTs*parametros.H1$coefficients[2]  + tiempoTs*para
```

Comprobamos visualmente si la tendencia se ajusta al modelo que tenemos de la serie temporal:

```
plot.ts(serie, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempo, tendEstimadaTr, col = "blue")
lines(tiempoTs, tendEstimadaTs, col = "green")
```



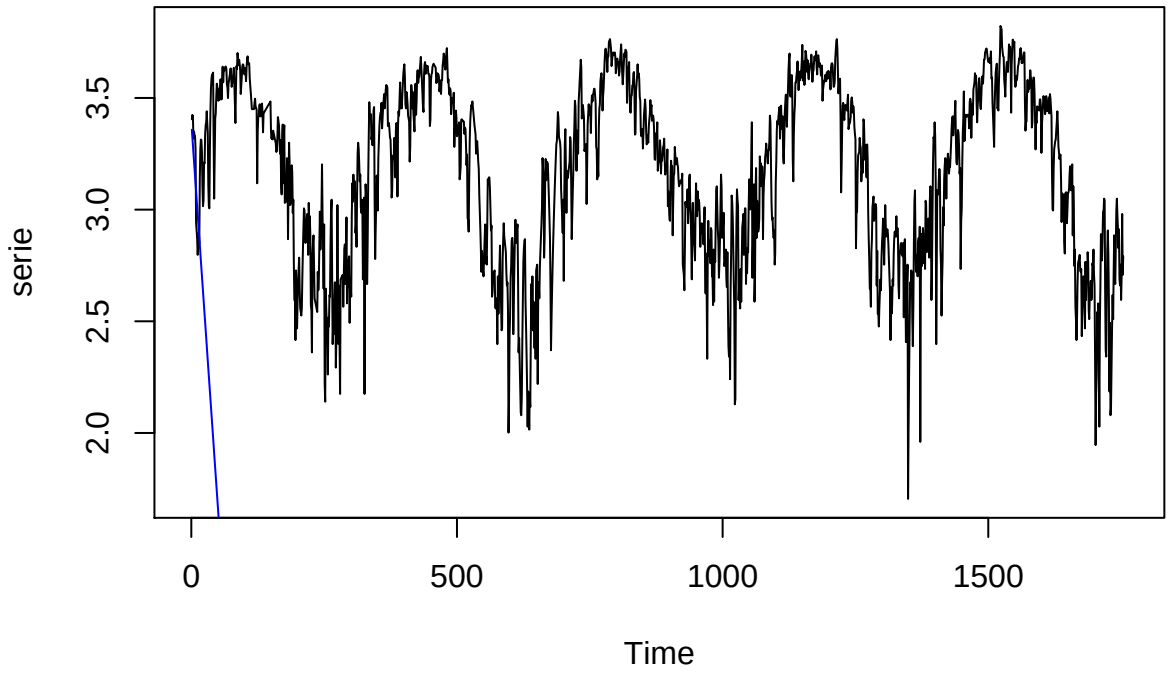
Validamos de forma estadística de nuevo con el test de Jarque Bera, con el que comprobamos si los errores siguen una distribución normal a lo largo del tiempo.

```
JB = jarque.bera.test(parametros.H1$residuals)
JB
```

```
##
##  Jarque Bera Test
##
## data:  parametros.H1$residuals
## X-squared = 102.26, df = 2, p-value < 2.2e-16
```

Una vez más, no logramos modelar la tendencia de manera correcta, veamos como se comporta con una transformación logarítmica:

```r
# calculo de parametros
parametros.H1 = lm(serie ~ log(tiempo))

# tendencia estimada para los datos de train
tendEstimadaTr = parametros.H1$coefficients[1] + tiempo*parametros.H1$coefficients[2]
# tendencia estimada para las predicciones de test
tendEstimadaTs = parametros.H1$coefficients[1] + tiempoTs*parametros.H1$coefficients[2]

# representacion de la tendencia sobre la serie
plot.ts(serie, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempo, tendEstimadaTr, col = "blue")
lines(tiempoTs, tendEstimadaTs, col = "green")
```

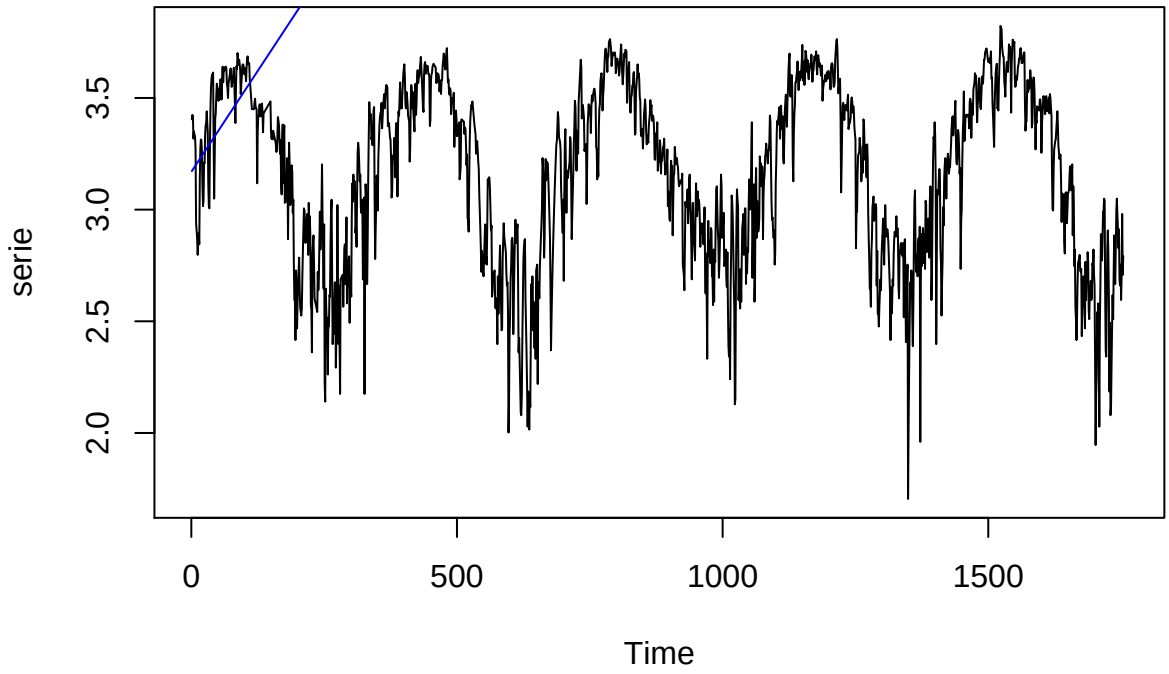



```r
# test de normalizacion Jarque Bera
JB = jarque.bera.test(parametros.H1$residuals)
JB
```

```
##
##  Jarque Bera Test
##
## data:  parametros.H1$residuals
## X-squared = 99.71, df = 2, p-value < 2.2e-16
```

Como última prueba de transformaciones matemáticas aplicadas para modelar la tendencia de la serie, procedo a intentarlo con una sinusoidal:

```r
# calculo de parametros
parametros.H1 = lm(serie ~ sin(tiempo))

# tendencia estimada para los datos de train
tendEstimadaTr = parametros.H1$coefficients[1] + tiempo*parametros.H1$coefficients[2]
# tendencia estimada para las predicciones de test
tendEstimadaTs = parametros.H1$coefficients[1] + tiempoTs*parametros.H1$coefficients[2]

# representacion de la tendencia sobre la serie
plot.ts(serie, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempo, tendEstimadaTr, col = "blue")
lines(tiempoTs, tendEstimadaTs, col = "green")
```



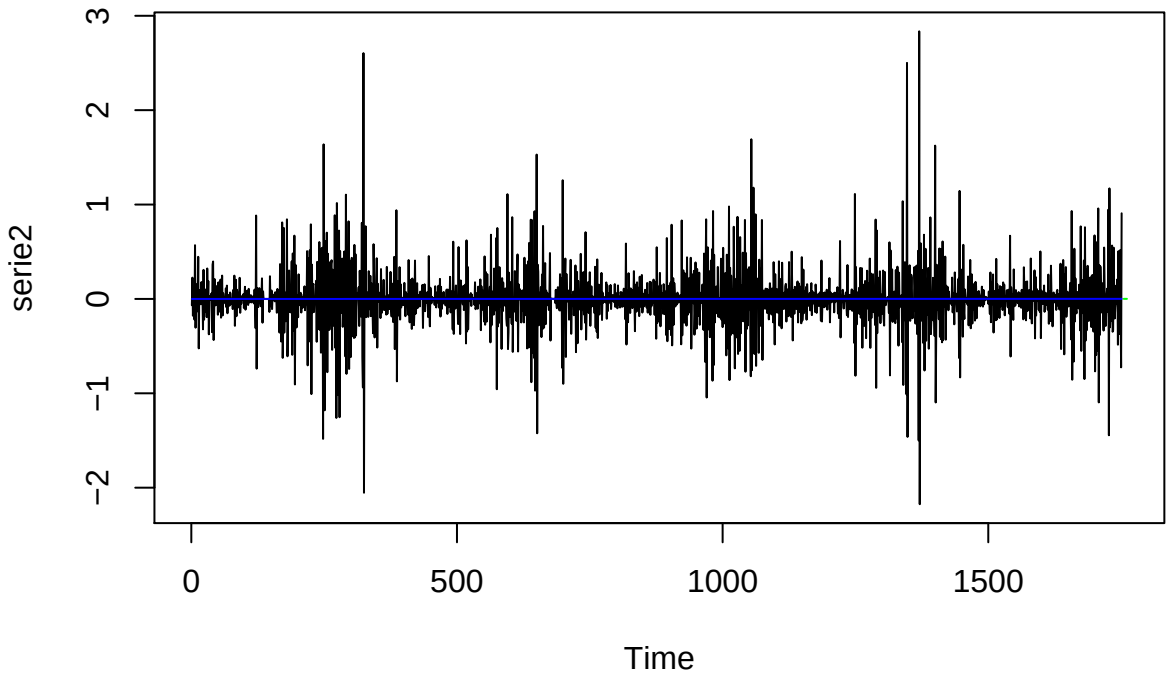

```r
# test de normalizacion Jarque Bera
JB = jarque.bera.test(parametros.H1$residuals)
JB
```

```
##
##  Jarque Bera Test
##
## data:  parametros.H1$residuals
## X-squared = 106.92, df = 2, p-value < 2.2e-16
```

Tras probar con estos distintos modelos obtenidos mediante diferentes transformaciones matemáticas, me decido por realizar diferenciación tantas veces como sea necesario con tal de suazivar la tendencia:

Probamos de nuevo...

```r
serie2=diff(serie, differences = 3)
tiempo2= 1:length(serie2)
# calculo de parametros
parametros.H1 = lm(serie2 ~ tiempo2)

# tendencia estimada para los datos de train
tendEstimadaTr = parametros.H1$coefficients[1] + tiempo2*parametros.H1$coefficients[2]
# tendencia estimada para las predicciones de test
tendEstimadaTs = parametros.H1$coefficients[1] + tiempoTs*parametros.H1$coefficients[2]

# representacion de la tendencia sobre la serie
plot.ts(serie2, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempo2, tendEstimadaTr, col = "blue")
lines(tiempoTs, tendEstimadaTs, col = "green")
```




```r
# test de normalizacion Jarque Bera
JB = jarque.bera.test(parametros.H1$residuals)
JB
```

```
##
##  Jarque Bera Test
##
## data:  parametros.H1$residuals
## X-squared = 7374.5, df = 2, p-value < 2.2e-16
```

Como podemos observar, tras las pruebas de modelado de tendencia tanto con diversas transformaciones matemáticas (logarítmica, cuadrática, senoidal) como por diferenciación de la serie con el fin de suavizar la

tendencia, no logramos que los residuos de las diversas tendencias calculadas pasen el test de Jarque Bera, por lo que es posible con alta probabilidad que dichos residuos no sigan una distribución normal. Conociendo este dato, es posible que no haya conseguido modelar la tendencia con ninguno de los modelos obtenidos, por tanto me decido por no restársela a la serie en los siguientes pasos de este ejercicio.

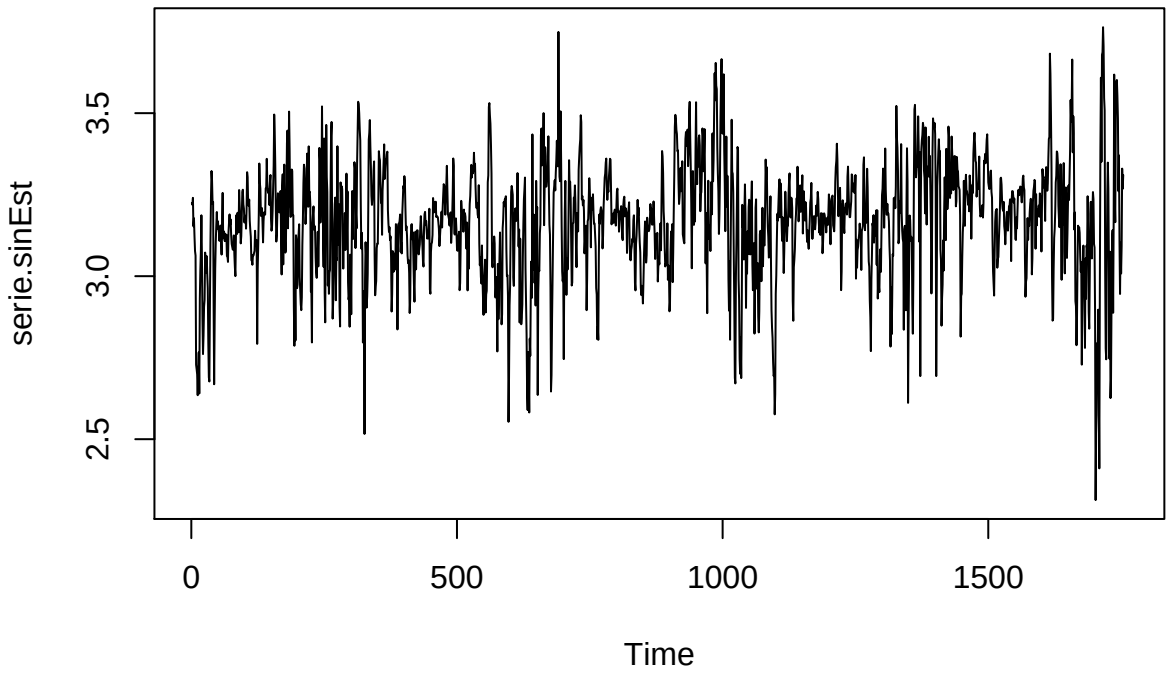
Tras este extenso estudio de la tendencia, procedemos a hayar y eliminar la estacionalidad de la seria:

```
k = 365 # aquí guardamos los datos estacionales
estacionalidad = decompose(serie.ts.log)$seasonal[1:k]
# elminamos la estacionalidad
serie.sinEst = serie - estacionalidad
```

```
## Warning in serie - estacionalidad: longitud de objeto mayor no es múltiplo
## de la longitud de uno menor
```

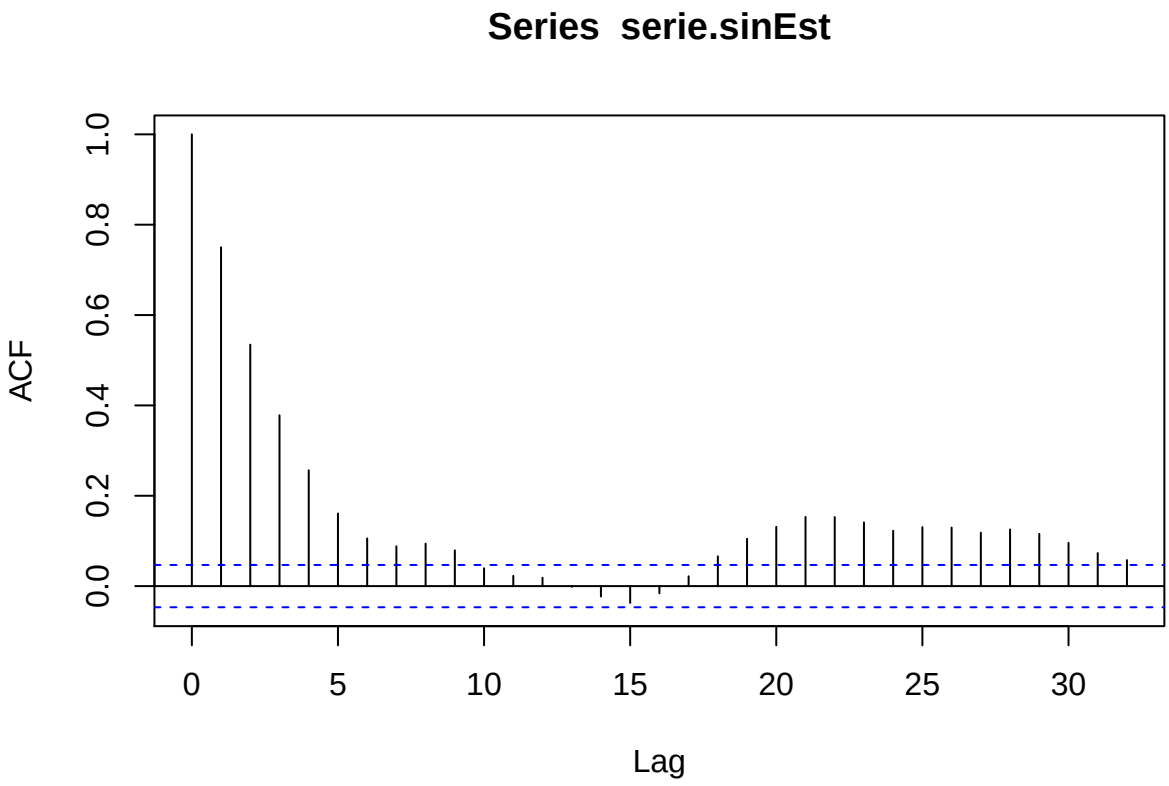
Comprobamos como se queda la serie sin dicha estacionalidad

```
plot.ts(serie.sinEst, xlim=c(1, tiempoTs[length(tiempoTs)]))
```



Ahora que tan solo nos queda la componente irregular tras haber eliminado la componente de estacionalidad, procedemos a comprobar si la serie es estacionaria o no mediante el test ACF.

```
acf(serie.sinEst)
```



Observando este gráfico acf, puede caber la posibilidad de tomar como ''bajada lenta" de los datos con respecto a los instantes sucesivos de tiempo, para asegurarnos de la estacionariedad de la serie, procedemos a realizar el test de Dickey-Fuller aumentado:

```
adf.test(serie.sinEst)
```

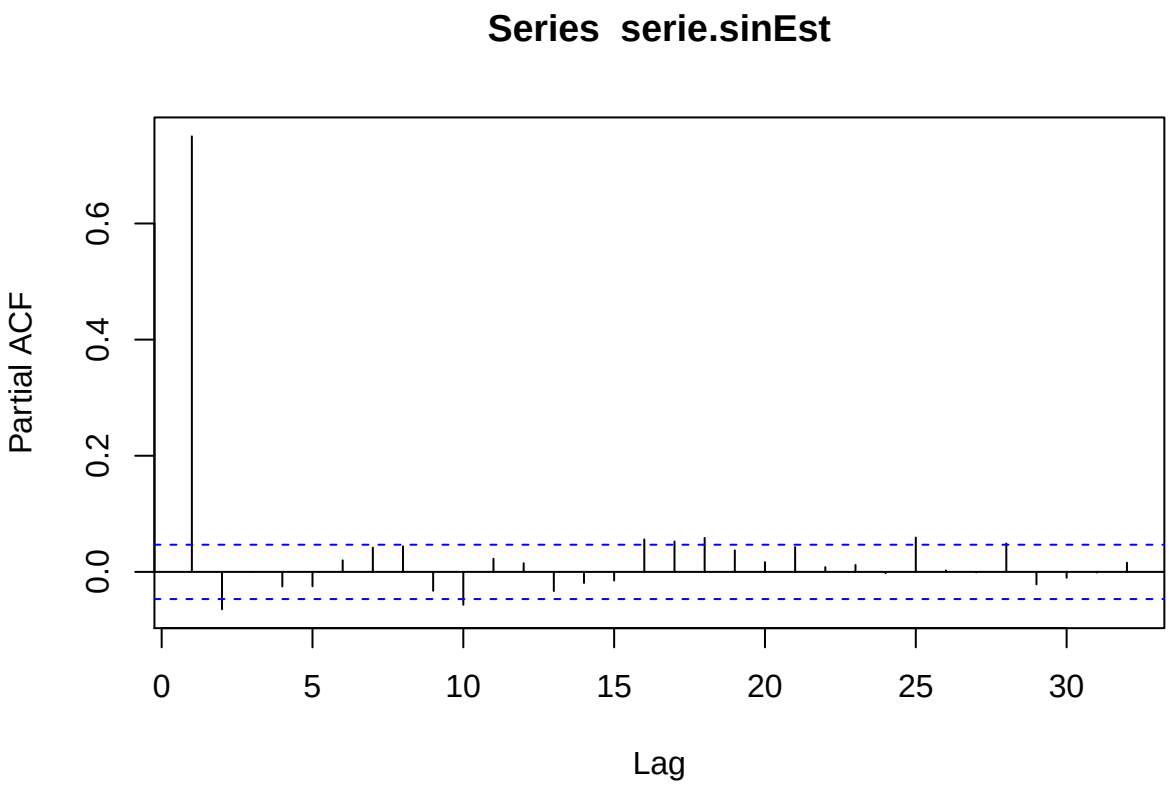
```
## Warning in adf.test(serie.sinEst): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  serie.sinEst
## Dickey-Fuller = -10.378, Lag order = 12, p-value = 0.01
## alternative hypothesis: stationary
```

Al tomar como hipótesis nula que la serie es estacionaria y al ser el p-value inferior a 0.05, podemos asegurar con un 95% de probabilidad que la serie es estacionaria, por tanto la tomamos así. Debido a esto no será necesario realizar diferenciación para convertir la serie en estacionaria.

Observamos el acf parcial para ver como influyen de forma individual cada uno de los instantes de tiempo sobre el instante actual.

```
pacf(serie.sinEst)
```



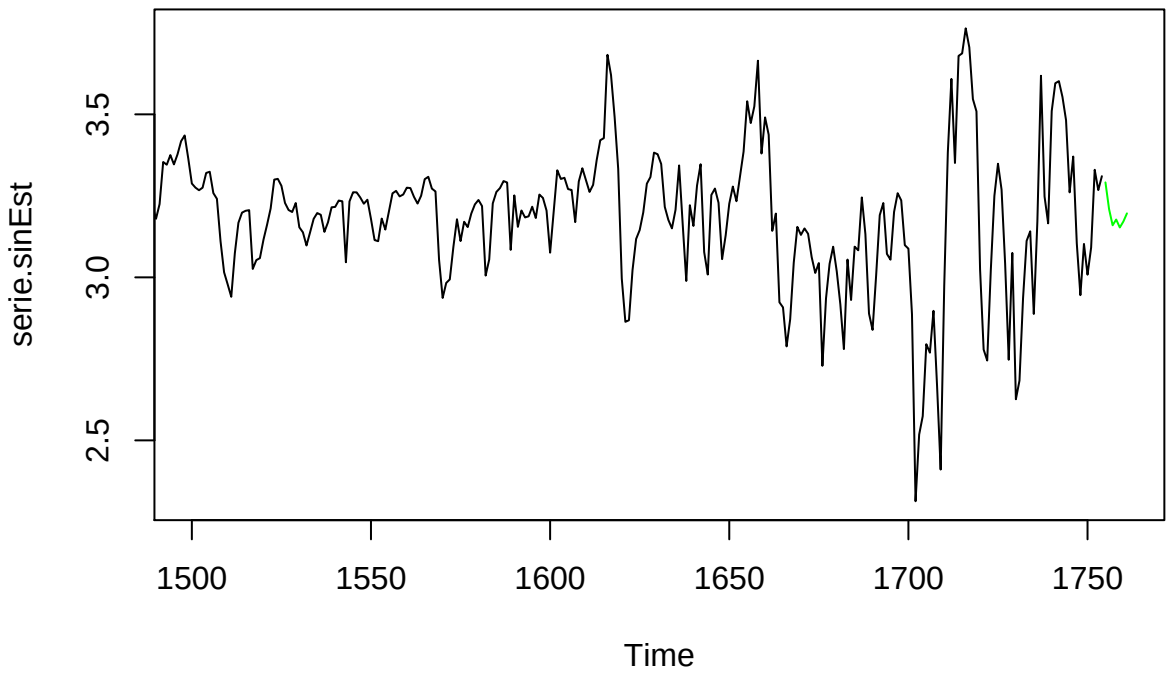
Observando los gráficos ofrecidos por acf y pacf, y sabiendo que la serie es estacionaria gracias al test de Dickey-Fuller, me resulta coherente pensar en un modelo de medias móviles de grado 22 para modelar la serie temporal mediante ARIMA. La elección del modelo es debido a que en pacf existe una bajada en los datos drástica y una estabilización en forma de ''olas", escogiendo el grado del modelo con ayuda del ultimo de los datos que traspasan de forma clara el umbral mínimo en el gráfico acf. Entrenamos el modelo escogido a continuación:

```
# coeficiente de autoregresión = 0, diferenciaciones = 0, coeficiente de medias moviles = 0
modelo = arima(serie.sinEst, order = c(0,0,22))
# calculamos las predicciones
predicciones = predict(modelo, n.ahead = Npred)
valoresPredichos = predicciones$pred
valoresPredichos
```

```
## Time Series:
## Start = 1755
## End = 1761
## Frequency = 1
## [1] 3.291288 3.209668 3.159646 3.177438 3.152966 3.171279 3.196229
```

Observamos ahora de forma visual el modelo incluyendo la serie sin estacionalidad:

```
plot.ts(serie.sinEst, xlim=c(1500, tiempoTs[length(tiempoTs)]))
lines(tiempoTs,valoresPredichos, col="green")
```



Parece ser que las predicciones obtenidas hasta el momento caen dentro de los límites razonables de la serie.

Finalmente probamos la validez de nuestro modelo (sus errores) mediante tests estadísticos.
Aplicamos varios:

Test de Box-Pierce: comprobamos que los residuos que nos quedan son aleatorios:

```
Box.test(modelo$residuals)
```

```
##
##  Box-Pierce test
##
## data:  modelo$residuals
## X-squared = 0.003811, df = 1, p-value = 0.9508
```

Este valor de p-value nos dice con un 95% de confianza que los errores obtenidos por el modelo creado son aleatorios, por lo que el modelo queda validado por este test. Continuamos con los tests de normalidad.

Jarque Bera: vemos si los residuos, aunque aleatorios, son normales (media cero y desviación típica la que sea).

```
jarque.bera.test(modelo$residuals)
```

```
##
##  Jarque Bera Test
##
## data:  modelo$residuals
## X-squared = 910.17, df = 2, p-value < 2.2e-16
```

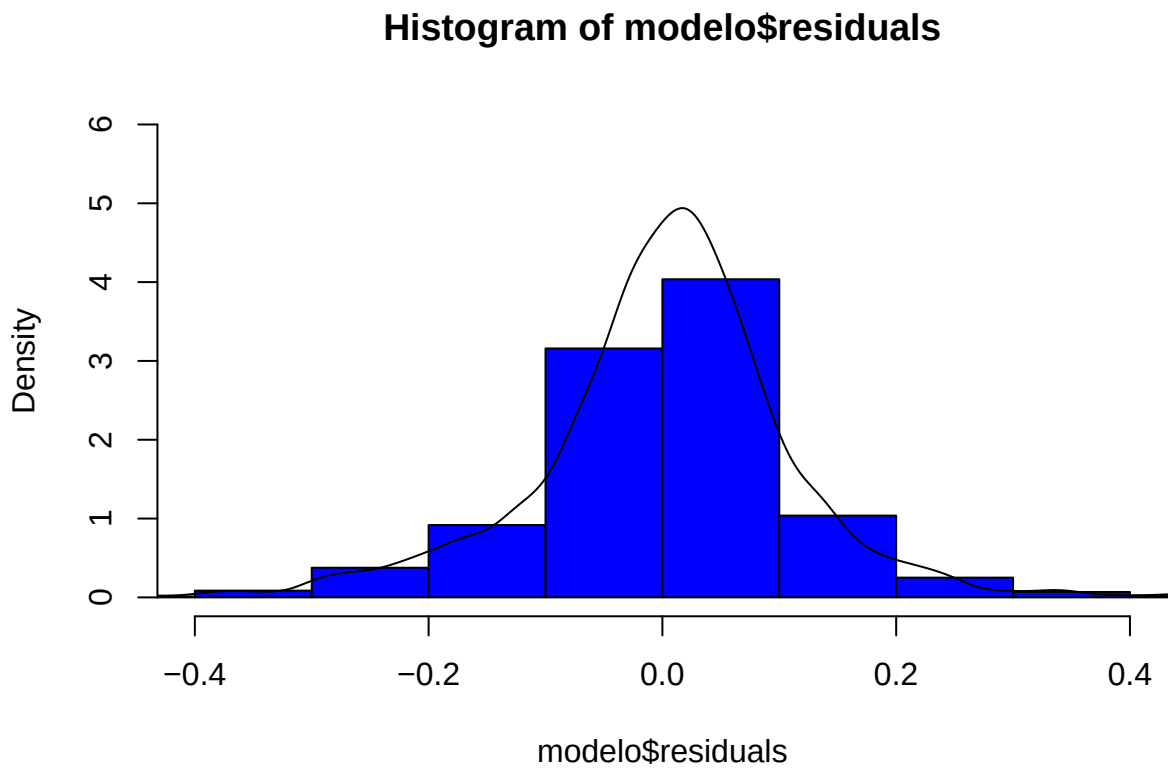
Saphiro-Wilk: mismo cometido que Jarque Bera

```
shapiro.test(modelo$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  modelo$residuals
## W = 0.95311, p-value < 2.2e-16
```

Asumimos, con alta probabilidad, tras la ejecución de ambos tests que los residuos no siguen una distribución normal debido a los p-values obtenidos, ya que la hipótesis nula propone que los datos no siguen una distribución normal. A continuación obtenemos un histograma y función de densidad para obtener resultados gráficos de confirmación.

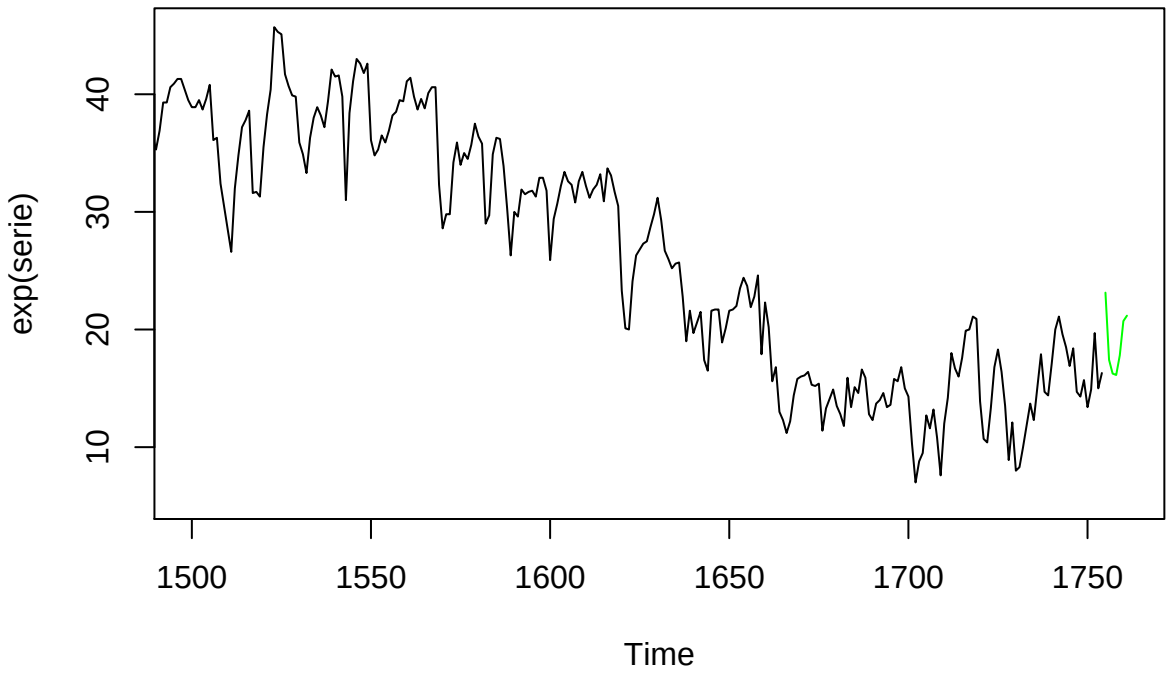
```
hist(modelo$residuals, col="blue", prob=T,ylim=c(0,6), xlim=c(-0.4,0.4))
lines(density(modelo$residuals))
```



Como podemos observar, el histograma representando la distribución de los residuos del modelo nos dice que estos sí que siguen una distribución normal, al contrario que los tests de Sphiro-Wilk y Jarque Bera. Llegados a este punto, nos acogeremos a la validez del teorema del límite central para asumir que, teniendo la cantidad elevada de datos que tenemos, los cuales toman una varianza no nula pero finita, estos se ajustan de buena forma a los de una distribución normal, dando por válida la representación de la distribución de los residuos dados por el histograma y por tanto, el modelo obtenido.

Con estas comprobaciones el modelo queda validado, ya que los errores que nos dan son aleatorios que provienen de una distribución normal. A continuación hemos de deshacer los cambios para obtener los predicciones reales:

```r
# obtenemos la estacionalidad correspondiente restandole a los dias del año al completo (365)
# la cantidad de dias que hay entre el mes de inicio del dataset (7 de mayo) y el comienzo del
# día y el mes que queremos predecir (1 de marzo)
estacionalidadCorrespondiente = 365-30-31-7

# incluimos la estacionalidad
valoresPredichos.Est =
  valoresPredichos + estacionalidad[estacionalidadCorrespondiente : (estacionalidadCorrespondiente+6)]

# Transformación de los logaritmos
valoresPredichos.Est.exp = exp(valoresPredichos.Est)

# usamos exp(serie) por haber hecho al principio serie = log(serie)
plot.ts(exp(serie),xlim=c(1500, tiempoTs[length(tiempoTs)]))
lines(tiempoTs, valoresPredichos.Est.exp, col = "green")
```



Parecen razonables los resultados de predicción obtenidos, los cuales son los siguientes para los 7 primeros días de Marzo de 2018:

```r
valoresPredichos.Est.exp
```

```
## Time Series:
## Start = 1755
## End = 1761
## Frequency = 1
## [1] 23.13305 17.42480 16.25250 16.13258 17.80971 20.71788 21.17717
```