Series Temporales: Trabajo Guiado

Juanjo Sierra

19 de abril de 2019

Paquetes a cargar

Importamos los paquetes que necesitamos para resolver los problemas planteados para la práctica.

```
library(tseries)
```

Problema

Dadas las cifras de pasajeros que vuelan con una aerolínea por año (en miles, anotadas mensualmente) entre los años 1949 y 1959, se pide estimar la cantidad de pasajeros que esa misma aerolínea tendrá cada mes durante el año 1960.

Se leen los datos de los pasajeros que nos proporciona la aerolínea.

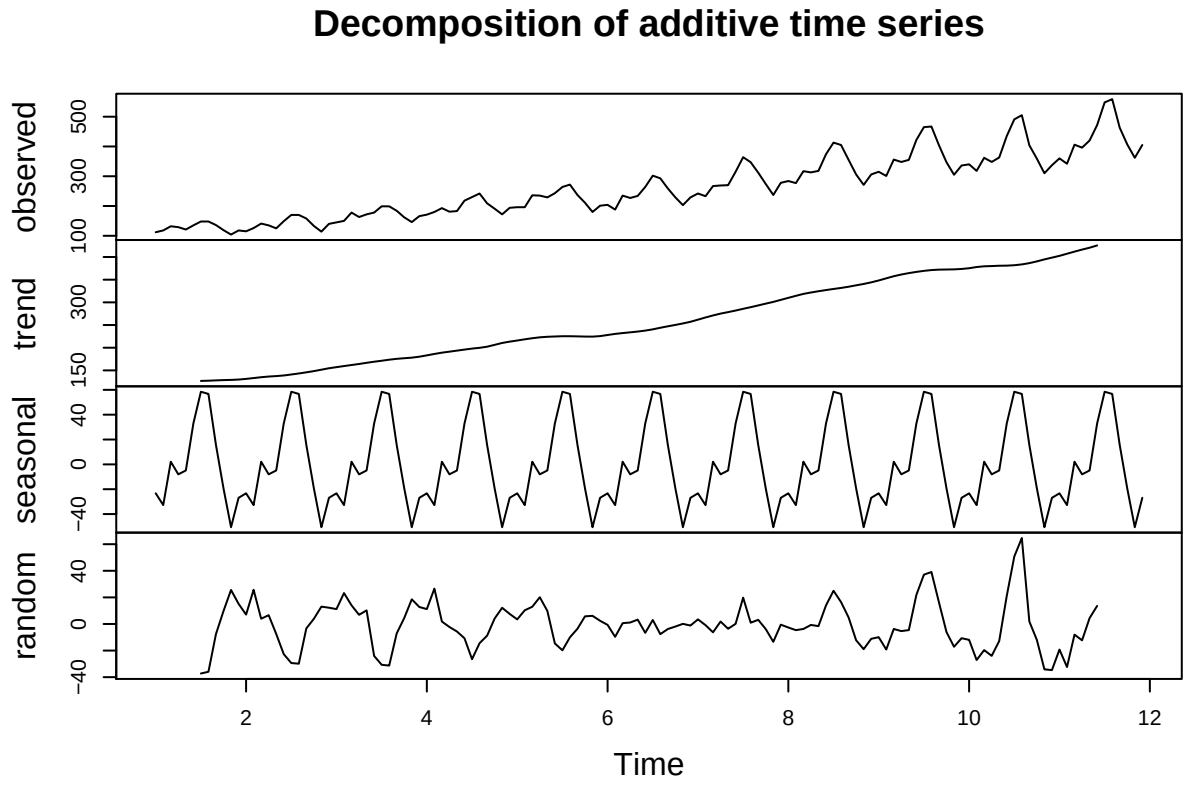
```
serie = scan("Datos/pasajeros_1949_1959.dat")
```

Defino el número de datos que quiero usar para test y para predecir.

```
nTest = 12
nPred = 12
```

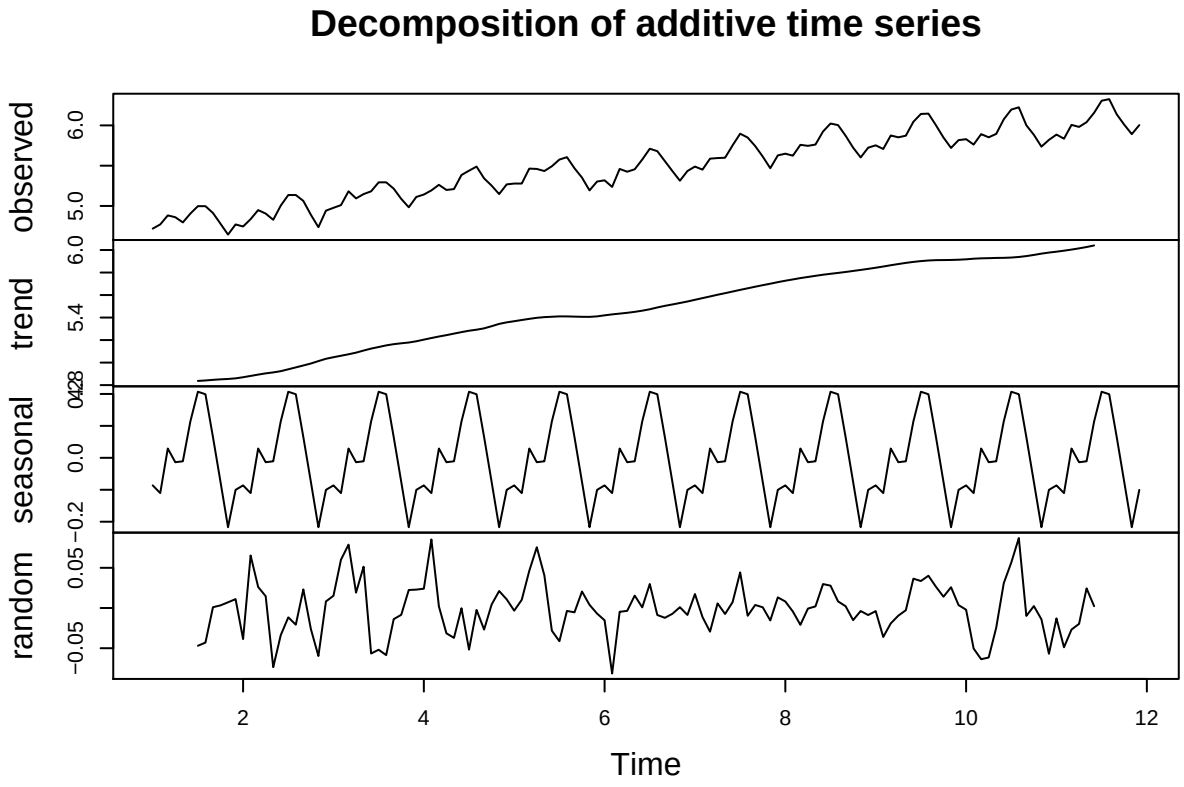
Creo el objeto "Serie temporal" con la librería `tseries` y su función `ts`. La frecuencia será 12 porque suponemos que la estacionalidad es anual. Utilizando la función `plot` junto a `descompose` se pueden ver la tendencia, la estacionalidad y lo que queda de la serie una vez eliminadas la tendencia y la estacionalidad.

```
serie.ts = ts(serie, frequency=12)
plot(decompose(serie.ts))
```



Vamos a construir la serie logarítmica (tanto el objeto ts como la serie original) para tratar de reducir la varianza en la serie resultante de restar la tendencia y estacionalidad. Buscamos que la serie sea estacionaria, y eso implica que no varíe ni en media ni en varianza.

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



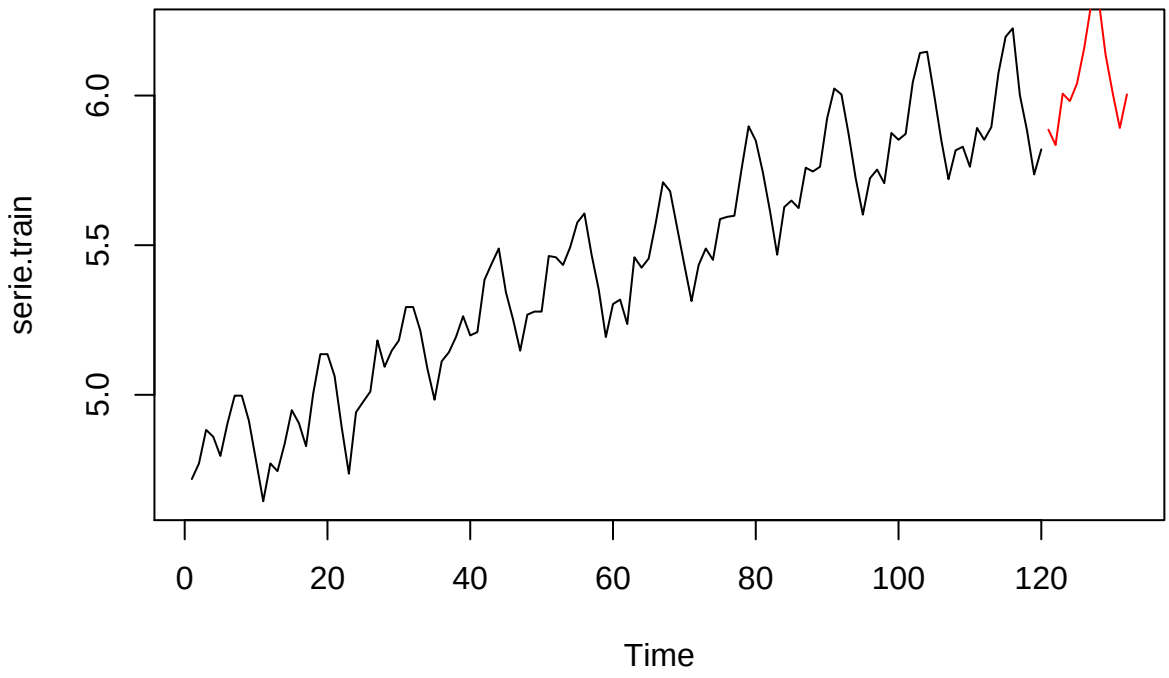
Vamos a seleccionar los datos de train y test del problema, entendiendo test como validación previa a la predicción del año 1960.

```
serie.train = serie.log[1:(length(serie.log) - nTest)]
tiempo.train = 1:length(serie.train)

serie.test = serie.log[(length(serie.log) - nTest+1):length(serie.log)]
tiempo.test = (length(serie.train)+1):length(serie.log)
```

A continuación los mostramos en una gráfica, resaltando en rojo los datos que se están tomando como test.

```
plot.ts(serie.train, xlim=c(1, tiempo.test[length(tiempo.test)]))
lines(tiempo.test, serie.test, col="red")
```



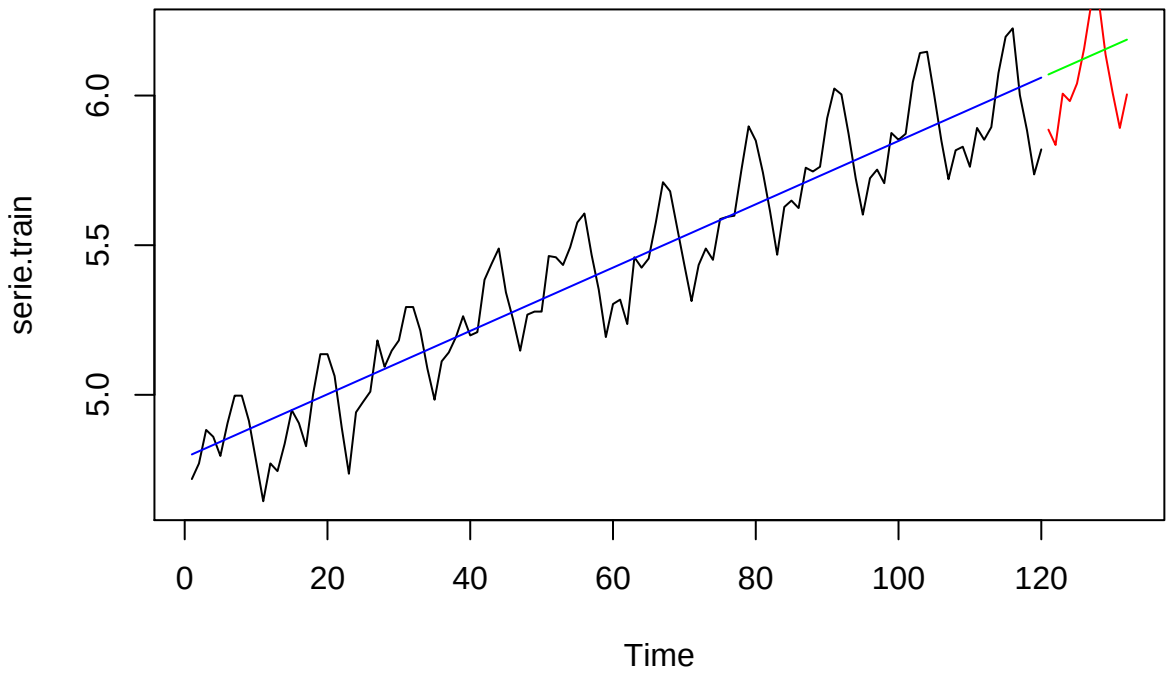
Lo siguiente que se va a hacer es estimar la tendencia. Se va a utilizar un modelo lineal en principio dado que por el aspecto de la serie temporal da la sensación de que puede generalizar bien.

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

tendencia.train.H1 = parametros.H1$coefficients[1]+tiempo.train*parametros.H1$coefficients[2]
tendencia.test.H1 = parametros.H1$coefficients[1]+tiempo.test*parametros.H1$coefficients[2]
```

Y a continuación, mostramos la tendencia estimada en la misma gráfica que la serie temporal, para comprobar cómo se ajusta.

```
plot.ts(serie.train, xlim=c(1, tiempo.test[length(tiempo.test)]))
lines(tiempo.test, serie.test, col="red")
lines(tiempo.train, tendencia.train.H1, col="blue")
lines(tiempo.test, tendencia.test.H1, col="green")
```



Vamos a validar que el modelo es correcto, para apoyarnos en algo más que una hipótesis. Utilizaremos el test de Jarque Bera sobre los residuos que han quedado de generar el modelo lineal y los que quedan en el test.

```
JB.train = jarque.bera.test(parametros.H1$residuals)
JB.test = jarque.bera.test(tendencia.test.H1-serie.test)

JB.train
```

```
##
##  Jarque Bera Test
##
## data:  parametros.H1$residuals
## X-squared = 1.755, df = 2, p-value = 0.4158
```

```
JB.test
```

```
##
##  Jarque Bera Test
##
## data:  tendencia.test.H1 - serie.test
## X-squared = 0.87957, df = 2, p-value = 0.6442
```

Como el p-value resultante del test de Jarque Bera es superior a 0.05 no se puede afirmar con suficiente confianza que los residuos no sigan una distribución normal, por lo que consideramos normales los errores del modelo lineal para train y para test.

A continuación vamos a comparar las medias del error, para comprobar si el error que se produce en la parte de train es similar al que ocurre en la parte de test. Para ello utilizamos el test de Student.

```
TT = t.test(c(parametros.H1$residuals, tendencia.test.H1-serie.test))
TT
```

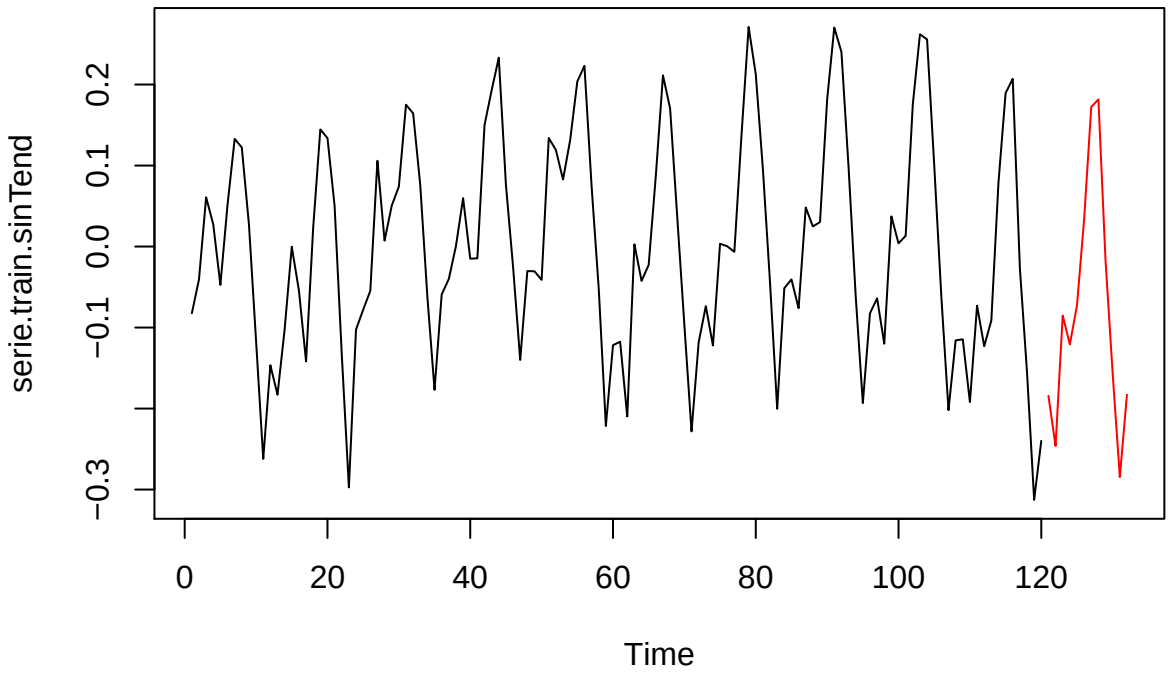
```
##
##  One Sample t-test
##
## data:  c(parametros.H1$residuals, tendencia.test.H1 - serie.test)
## t = 0.61219, df = 131, p-value = 0.5415
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.01628561  0.03088222
## sample estimates:
##   mean of x
## 0.007298304
```

El test de Student indica que la media de los errores es casi 0 (0.0073), y el p-value por encima de 0.05 indica que no hay una desviación significativa con respecto a esta media. Por esto, se puede considerar que no hay una diferencia estructural que invalide el modelo lineal en este caso.

El siguiente paso es eliminar la tendencia de la serie inicial, tanto en entrenamiento como en test. Y comprobamos en una gráfica qué aspecto tiene la serie sin esta tendencia.

```
serie.train.sinTend = serie.train - tendencia.train.H1
serie.test.sinTend = serie.test - tendencia.test.H1

plot.ts(serie.train.sinTend, xlim=c(1, tiempo.test[length(tiempo.test)]))
lines(tiempo.test, serie.test.sinTend, col="red")
```



Una vez hecho esto, procedemos a eliminar la estacionalidad de la serie resultante. Inicialmente asumimos una estacionalidad de 12 valores (anual).

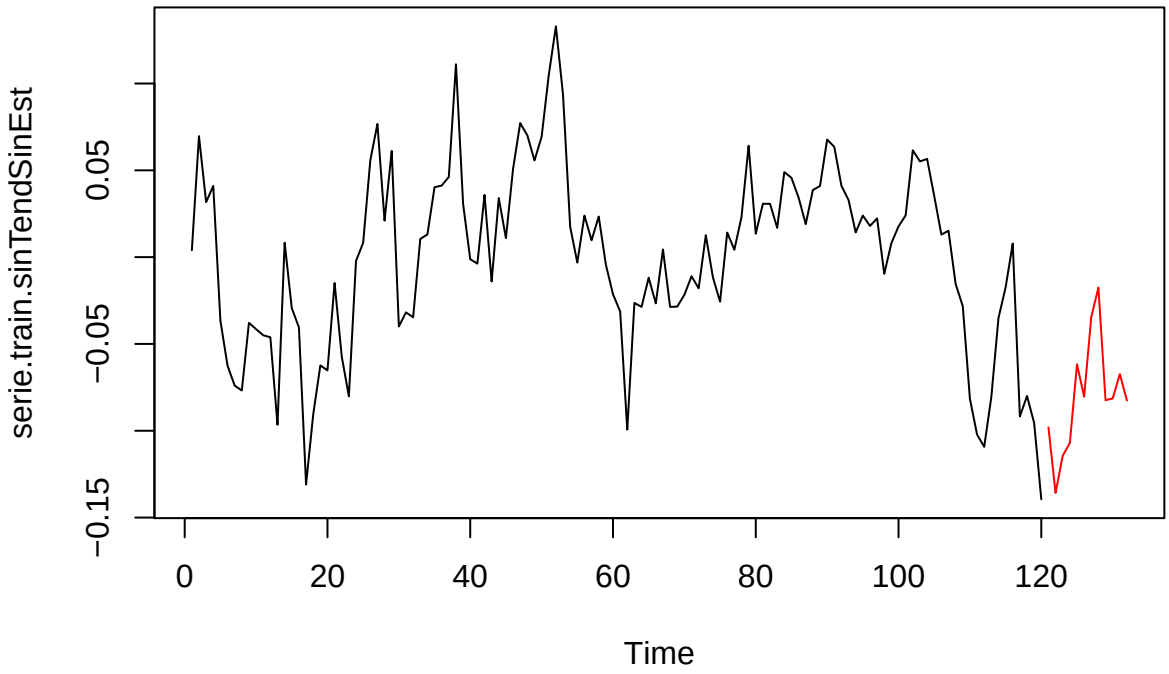
```
k = 12
estacionalidad = decompose(serie.ts.log)$seasonal[1:k]
estacionalidad
```

```
##  [1] -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726
##  [6]  0.11403832  0.20689984  0.19914832  0.06503613 -0.07542627
## [11] -0.21722154 -0.10035385
```

Aquí se pueden observar los 12 valores que definen la estacionalidad. Para eliminar la estacionalidad de nuestra serie hay que ir eliminando estos 12 valores de forma periódica a lo largo de la serie.

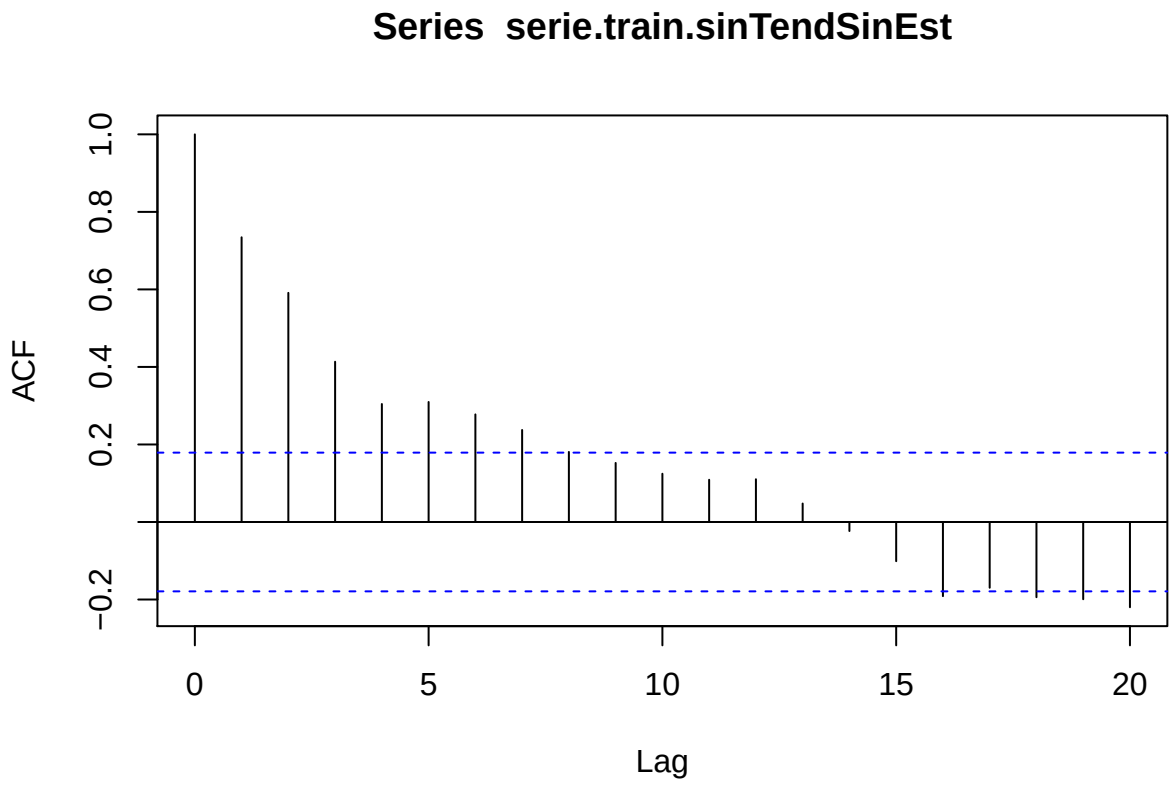
```
# Aprovecho el reciclaje de R para no tener que crear una variable auxiliar
serie.train.sinTendSinEst = serie.train.sinTend - estacionalidad
serie.test.sinTendSinEst = serie.test.sinTend - estacionalidad

plot.ts(serie.train.sinTendSinEst, xlim=c(1, tiempo.test[length(tiempo.test)]))
lines(tiempo.test, serie.test.sinTendSinEst, col="red")
```



Aquí se puede observar ya limpiamente la serie temporal sin la tendencia y sin la estacionalidad. El siguiente paso es comprobar si la serie es o no estacionaria. Podemos comenzar por un test ACF.

```
acf(serie.train.sinTendSinEst)
```



Como no se puede observar claramente que sea estacionaria vamos a hacer el test de Dickey-Fuller aumentado para asegurarnos.

```
adf.serie.train = adf.test(serie.train.sinTendSinEst)
adf.serie.train
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  serie.train.sinTendSinEst
## Dickey-Fuller = -1.8407, Lag order = 4, p-value = 0.6427
## alternative hypothesis: stationary
```

El p-value es 0.64, por lo que al buscar una confianza del 95% no se puede asegurar que la serie sea estacionaria. Lo normal es que la serie no sea estacionaria en media, para hacerla estacionaria vamos a diferenciarla. Cuando lo hayamos hecho volveremos a pasarle el test.

```
serie.train.sinTendSinEstDif = diff(serie.train.sinTendSinEst)
serie.test.sinTendSinEstDif = diff(serie.test.sinTendSinEst)

adf.serie.train.2 = adf.test(serie.train.sinTendSinEstDif)
```

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

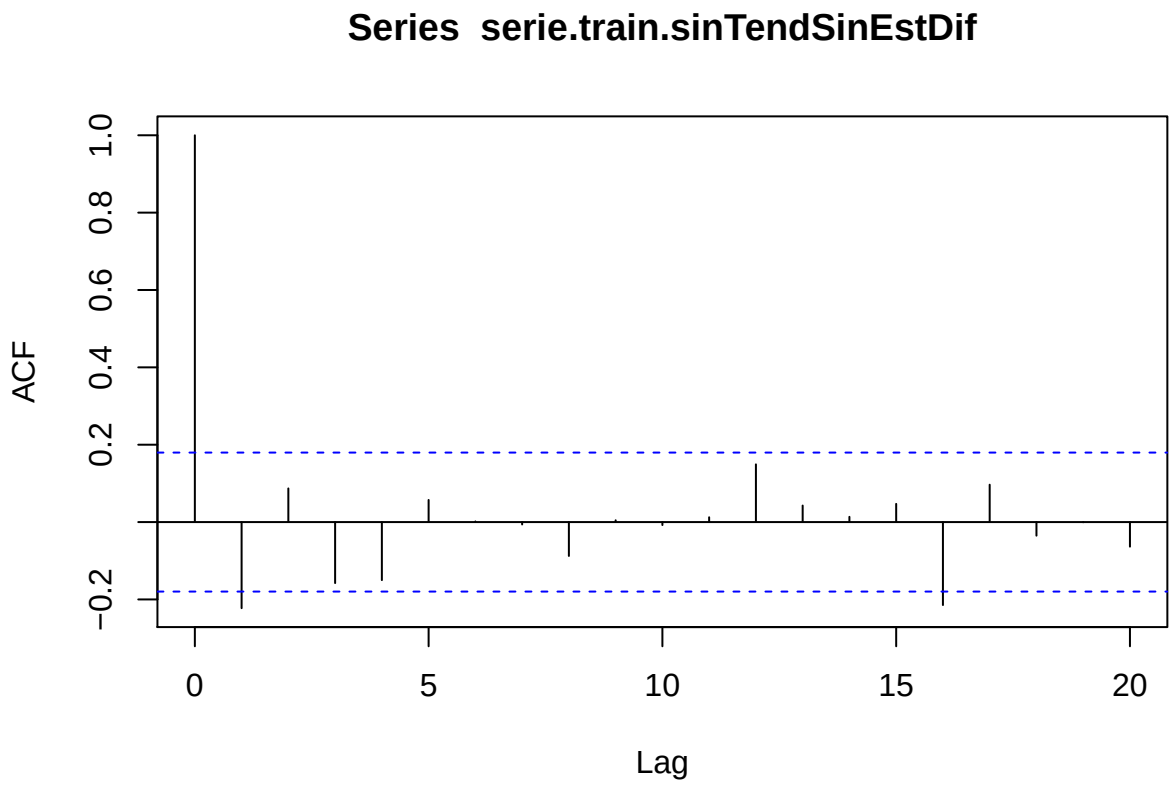
```
adf.serie.train.2
```

```
##
##  Augmented Dickey-Fuller Test
```

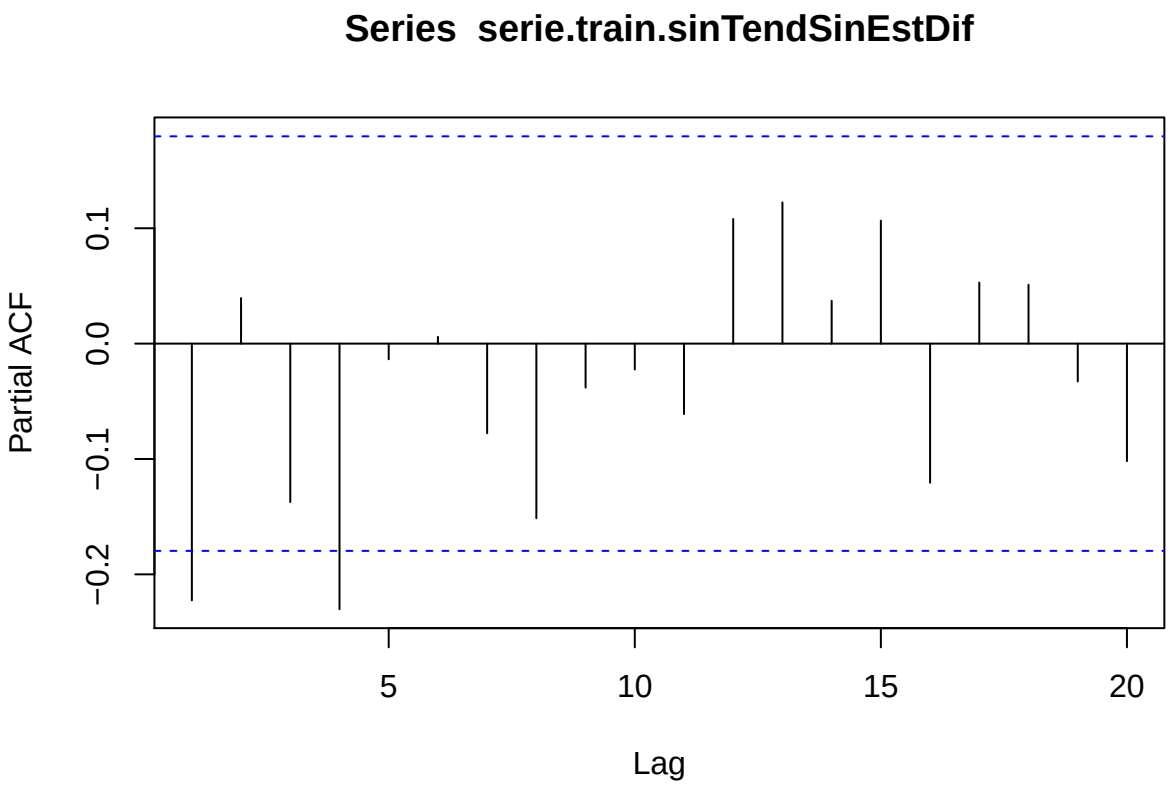
```
##
## data:  serie.train.sinTendSinEstDif
## Dickey-Fuller = -6.2153, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

Ahora que el p-value se encuentra por debajo de 0.05 sí que se puede asumir que la serie es estacionaria. Si comprobamos con un nuevo test ACF y un test PACF las gráficas resultantes de esta serie obtenemos algo que encaja mucho más con una serie estacionaria.

```
acf(serie.train.sinTendSinEstDif)
```




```
pacf(serie.train.sinTendSinEstDif)
```



Las gráficas ACF y PACF parecen típicas de un modelo autorregresivo de grado 4 (el valor más alto del PACF que queda fuera del umbral).

Podemos asumir un modelo AR(4), y como hemos diferenciado solamente un instante de tiempo (solamente hemos aplicado la función `diff` una vez) podemos aplicar un modelo ARIMA(4,1,0) sobre la serie sin tendencia ni estacionalidad (la diferenciación se aplica dentro del modelo).

```
modelo = arima(serie.train.sinTendSinEst, order = c(4,1,0))
```

Como es un modelo autorregresivo, los valores ajustados del modelo salen como los residuos del modelo que acabamos de ajustar más la serie.

```
valoresReconstruidos = serie.train.sinTendSinEst + modelo$residuals
```

Para hacer una predicción llamamos a la función `predict` e indicamos cuántos valores queremos predecir. Primero predeciremos la parte del test y comprobaremos que funciona bien con los valores que conocemos (forman parte de la serie inicial que se nos dio).

```
predicciones = predict(modelo, n.ahead = nPred)
valoresPredichos = predicciones$pred
valoresPredichos
```

```
## Time Series:
## Start = 121
## End = 132
## Frequency = 1
##  [1] -0.10507935 -0.11427537 -0.09925247 -0.09825939 -0.10548787
##  [6] -0.10413396 -0.10862673 -0.10637456 -0.10537488 -0.10509949
## [11] -0.10442190 -0.10536040
```

Podemos observar que el modelo ha predicho los 12 datos que le hemos pedido (los que equivaldrían al conjunto de test que hemos extraído de la serie inicial). Ahora toca comprobar cómo de bien se ajustan esos datos a los reales de test, calculando el error cuadrático acumulado.

```
error.train = sum(modelo$residuals^2)
error.train
```

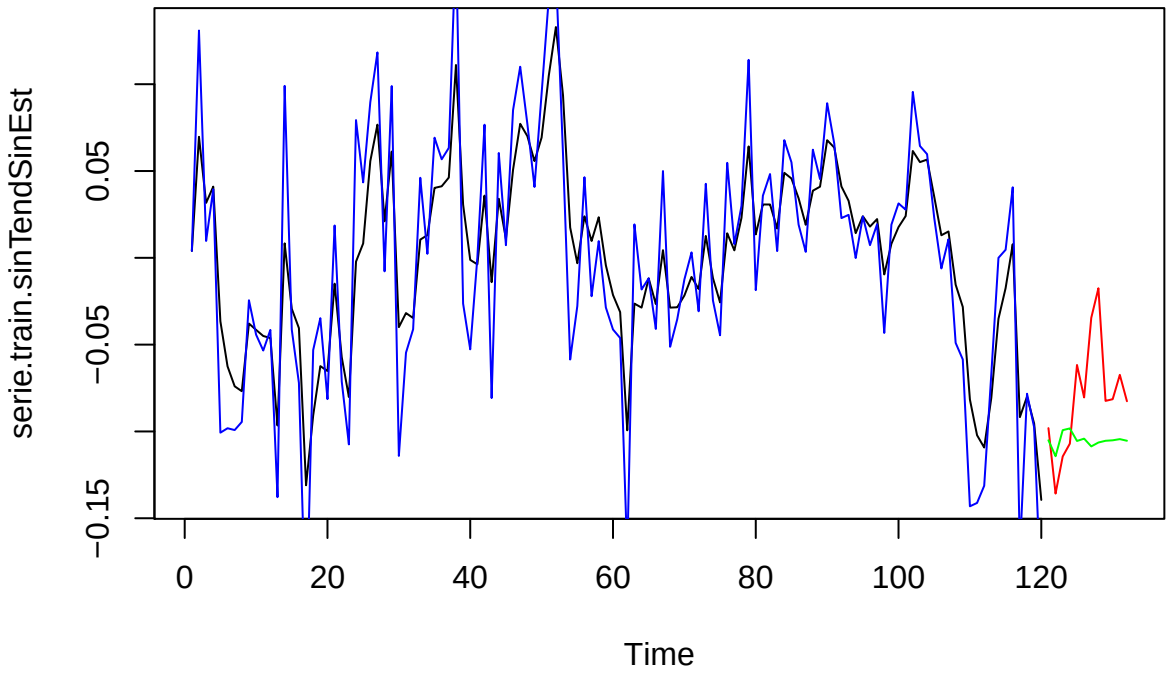
```
## [1] 0.1344656
```

```
error.test = sum((valoresPredichos-serie.test.sinTendSinEst)^2)
error.test
```

```
## [1] 0.01965443
```

Vamos a ilustrar las predicciones en la misma gráfica que los datos reales, para comprobar cómo se ajustan.

```
plot.ts(serie.train.sinTendSinEst, xlim=c(1, tiempo.test[length(tiempo.test)]))
lines(valoresReconstruidos, col="blue")
lines(tiempo.test, serie.test.sinTendSinEst, col="red")
lines(tiempo.test, valoresPredichos, col="green")
```



Como se observa en la gráfica, los valores reconstruidos sí que se ajustan bastante bien a los datos de train, pero sin embargo la predicción de test es demasiado genérica y no se ajusta muy bien.

Si volvemos a fijarnos en las gráficas de ACF y PACF que teníamos anteriormente, se puede observar que los valores que salen del umbral (a excepción del valor 0 de la gráfica ACF) son muy cercanos a estos límites del 0, por lo que podríamos decir que la serie es casi ruido blanco, y es por eso por lo que aparece esta predicción tan genérica.

Para comprobar que el modelo es válido vamos a estudiar los errores de dicho modelo con varios tests

estadísticos. Primero utilizamos el test de Box-Pierce que los residuos que quedan del modelo son aleatorios.

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

```
##
##  Box-Pierce test
##
## data:  modelo$residuals
## X-squared = 0.005349, df = 1, p-value = 0.9417
```

El test de Box-Pierce refleja la confianza en que los residuos no sean aleatorios. En este caso obtenemos una confianza de apenas el 6% así que negamos la hipótesis nula y afirmamos que los residuos del modelo siguen una distribución aleatoria.

El siguiente test será el de Jarque Bera y el de Shapiro-Wilk para valorar la normalidad de los residuos.

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

```
##
##  Jarque Bera Test
##
## data:  modelo$residuals
## X-squared = 0.39988, df = 2, p-value = 0.8188
```

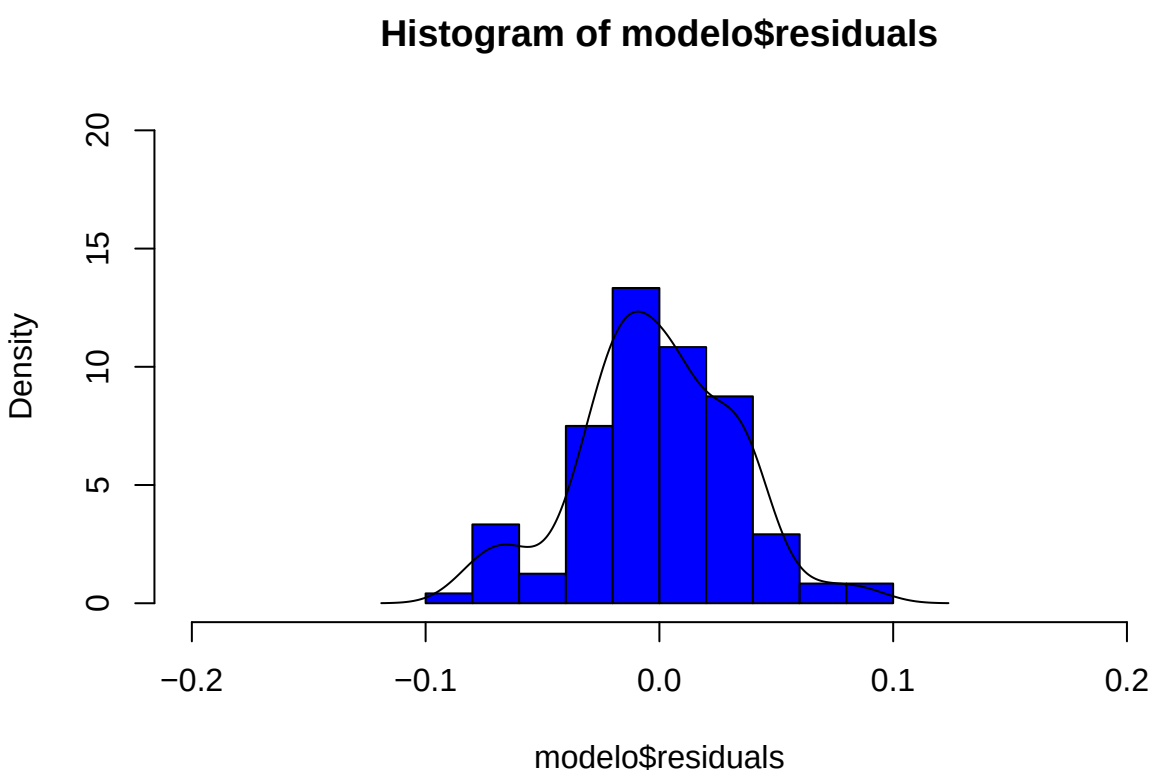
```
shapiroWilk.test = shapiro.test(modelo$residuals)
shapiroWilk.test
```

```
##
##  Shapiro-Wilk normality test
##
## data:  modelo$residuals
## W = 0.9867, p-value = 0.2904
```

Según ambos tests podemos afirmar que los errores siguen una distribución normal, ya que la hipótesis nula propone que los datos no sigan dicha distribución.

A continuación podemos mostrar un histograma de los residuos de este modelo para apoyarnos en un recurso gráfico.

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



En esta representación se puede apreciar que los residuos siguen una distribución normal con media 0, y que por tanto las afirmaciones que hemos hecho basándonos en los tests estadísticos son ciertos y hemos terminado de validar el modelo ARIMA(4,1,0) que nos ha llevado hasta aquí.

El siguiente paso es deshacer los cambios para predecir los valores del año 1960. Es decir, al modelo resultante (que no tiene estacionalidad ni tendencia) hay que sumarle los valores que le hemos sustraído previamente para que se tengan en cuenta a la hora de predecir.

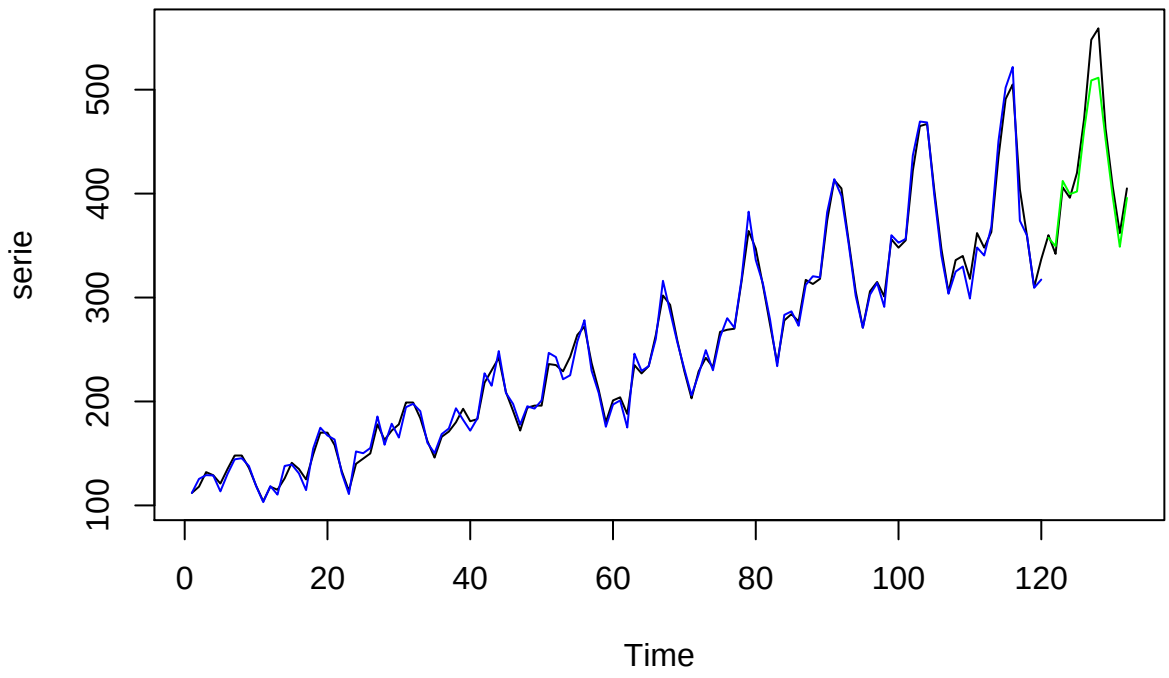
```
# Sumamos estacionalidad
# De nuevo aprovecho el reciclaje de R
valoresReconstruidos.Est = valoresReconstruidos + estacionalidad
valoresPredichos.Est = valoresPredichos + estacionalidad

# Sumamos tendencia
valoresReconstruidos.EstTend = valoresReconstruidos.Est + tendencia.train.H1
valoresPredichos.EstTend = valoresPredichos.Est + tendencia.test.H1

# Deshacemos transformación logarítmica
valoresReconstruidos.EstTendExp = exp(valoresReconstruidos.EstTend)
valoresPredichos.EstTendExp = exp(valoresPredichos.EstTend)
```

Ahora que ya tenemos los valores del modelo con los cambios deshechos, podemos mostrarlos en una nueva gráfica y comprobar cómo de bien se adaptan a los datos reales de la serie.

```
plot.ts(serie)
lines(tiempo.train, valoresReconstruidos.EstTendExp, col="blue")
lines(tiempo.test, valoresPredichos.EstTendExp, col="green")
```



Como vemos, el modelo resulta bastante acertado, y recoge de forma adecuada los altibajos propios de la serie temporal en el año de test.

Ahora toca predecir el año 1960 utilizando el resto de los datos como train. Para esto vamos a utilizar la serie logarítmica `serie.log` como train.

```
serie.train = serie.log
tiempo.train = 1:length(serie.log)
tiempo.test = (length(tiempo.train)+1):(length(tiempo.train)+nPred)
```

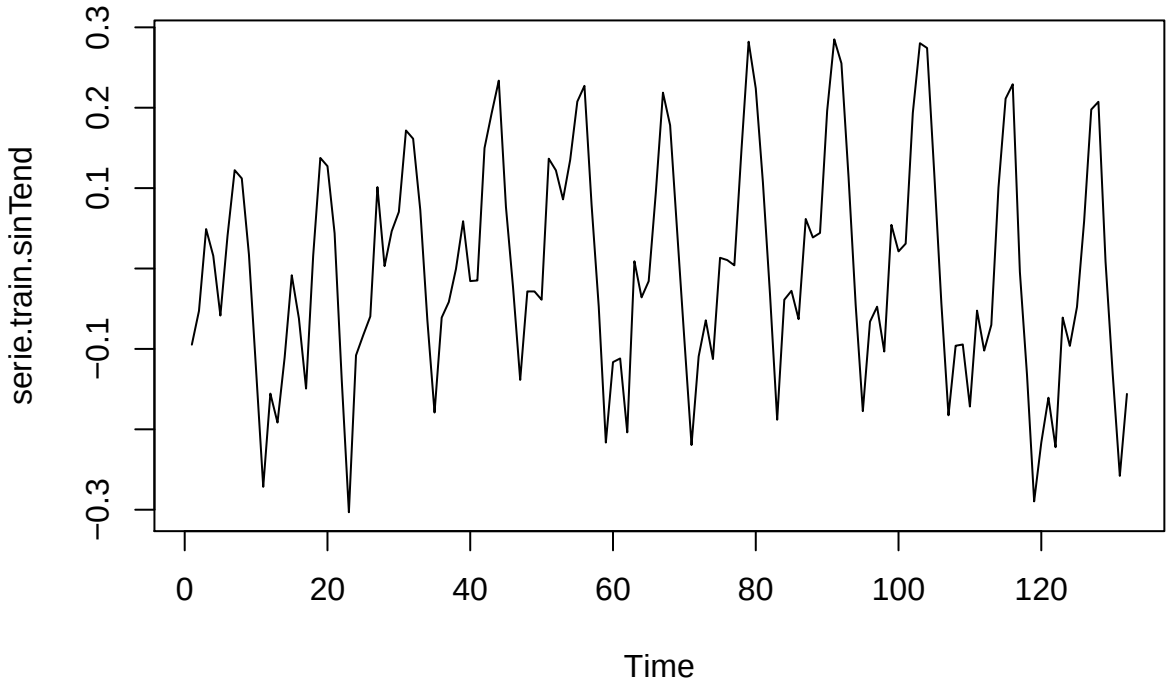
Estimamos la nueva tendencia, ahora con todos los datos.

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

tendencia.train.H1 = parametros.H1$coefficients[1]+tiempo.train*parametros.H1$coefficients[2]
tendencia.test.H1 = parametros.H1$coefficients[1]+tiempo.test*parametros.H1$coefficients[2]
```

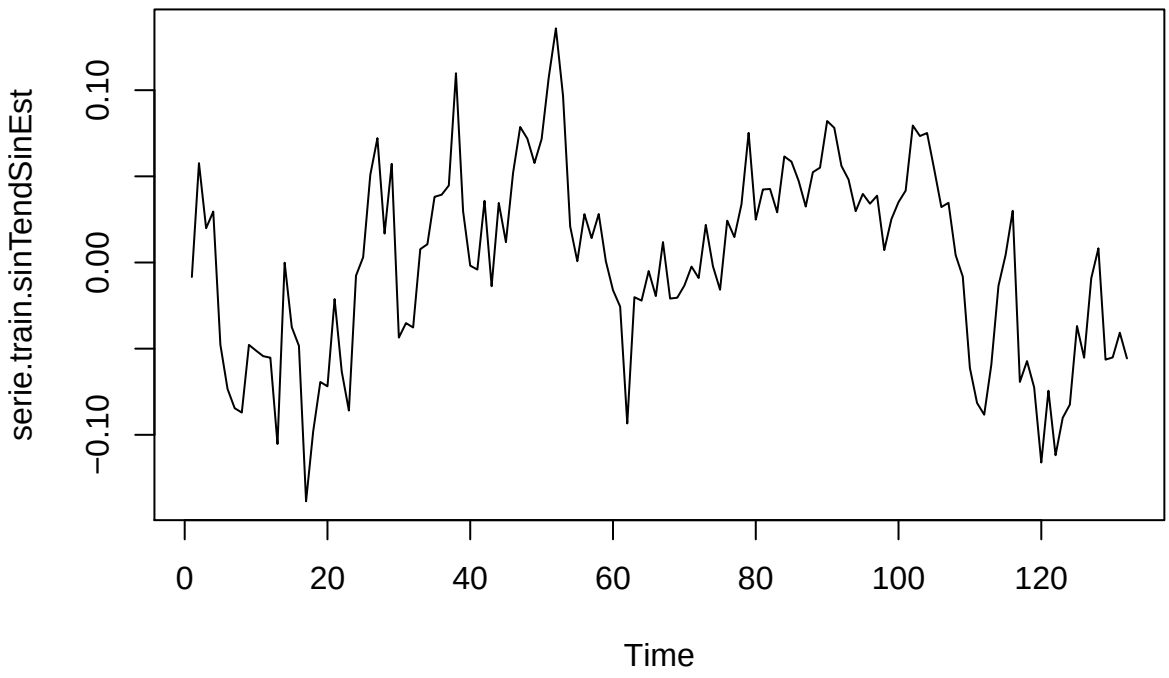
Una vez obtenida la nueva tendencia, eliminamos dicha tendencia para pasar a tener la siguiente serie.

```
serie.train.sinTend = serie.train - tendencia.train.H1
plot.ts(serie.train.sinTend, xlim=c(1, tiempo.train[length(tiempo.train)]))
```



Ahora que tenemos la serie sin tendencia, toca quitarle la estacionalidad para dejarla en formato estacionaria. La estacionalidad es la misma que antes por lo que no requiere que la editemos.

```
serie.train.sinTendSinEst = serie.train.sinTend - estacionalidad
plot.ts(serie.train.sinTendSinEst, xlim=c(1, tiempo.train[length(tiempo.train)]))
```



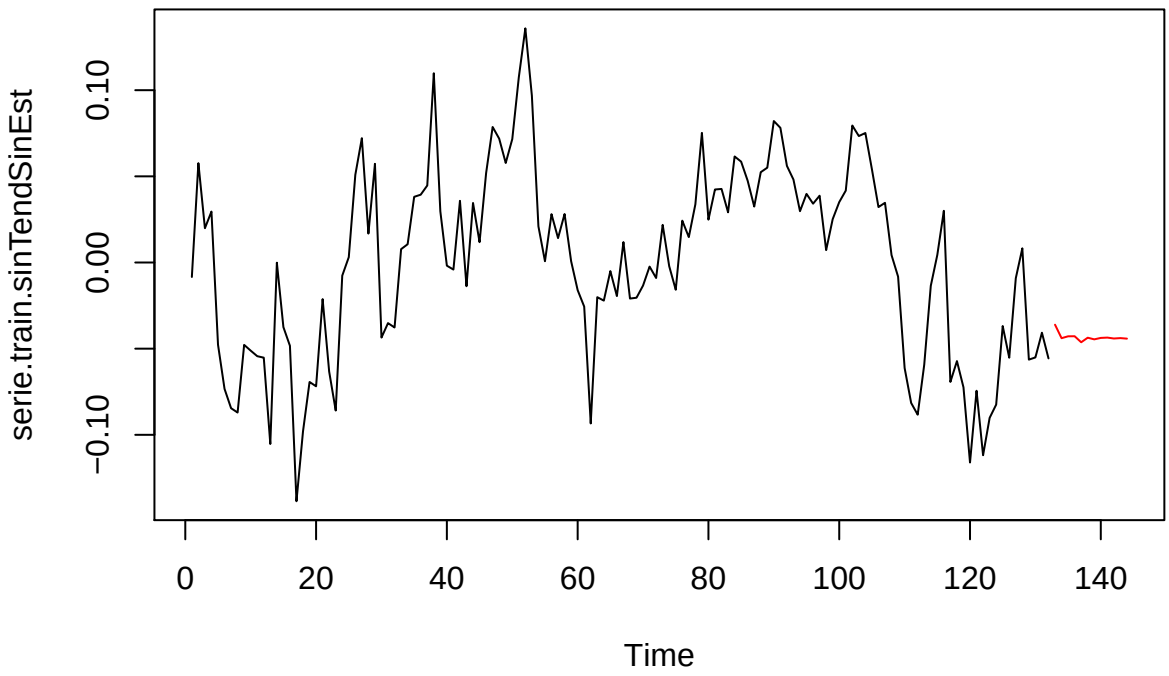
Una vez conseguida esta serie estacionaria, volvemos a calcular un modelo ARIMA con los parámetros que nos funcionaron en el caso anterior. Este modelo se usará para predecir los nuevos datos de 1960.

```
modelo = arima(serie.train.sinTendSinEst, order = c(4,1,0))
```

Con el modelo construido predecimos los nuevos datos, un total de 12 (para ocupar todo el año), y las vamos a mostrar junto al resto de la serie, la parte conocida.

```
predicciones = predict(modelo, n.ahead = nPred)
valoresPredichos = predicciones$pred

plot.ts(serie.train.sinTendSinEst, xlim=c(1, tiempo.test[length(tiempo.test)]))
lines(tiempo.test, valoresPredichos, col="red")
```


serie.train.sinTendSinEst
0.10
0.00
−0.10
0
20
40
60
80
100
120
140
Time