Práctica Guiada

Carlos Manuel Sequí Sánchez

Cargamos la biblioteca para series temporales.

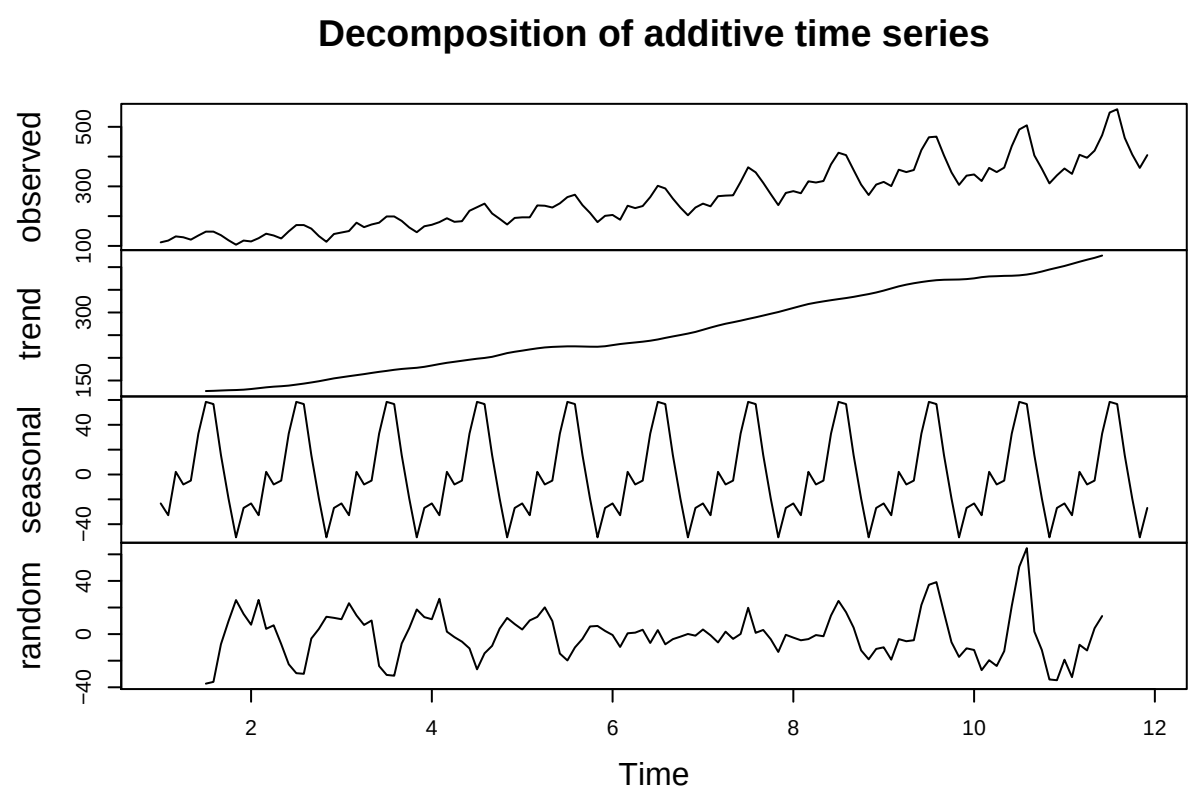
```
library(tseries)
```

Leemos los datos de la serie

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

Dividimos los datos en entrenamiento y test. Dejamos para ello el último año para la comprobación de la validez del modelo, y el resto lo usamos como train.

```
NTest = 12 # cantidad de datos a usar como test
NPred = 12 # cantidad de predicciones que queremos realizar
serie.ts = ts(serie, frequency = 12) # creamos el objeto serie temporal
                                     # suponiendo una estacionalidad de 12 meses
plot(decompose(serie.ts))
```

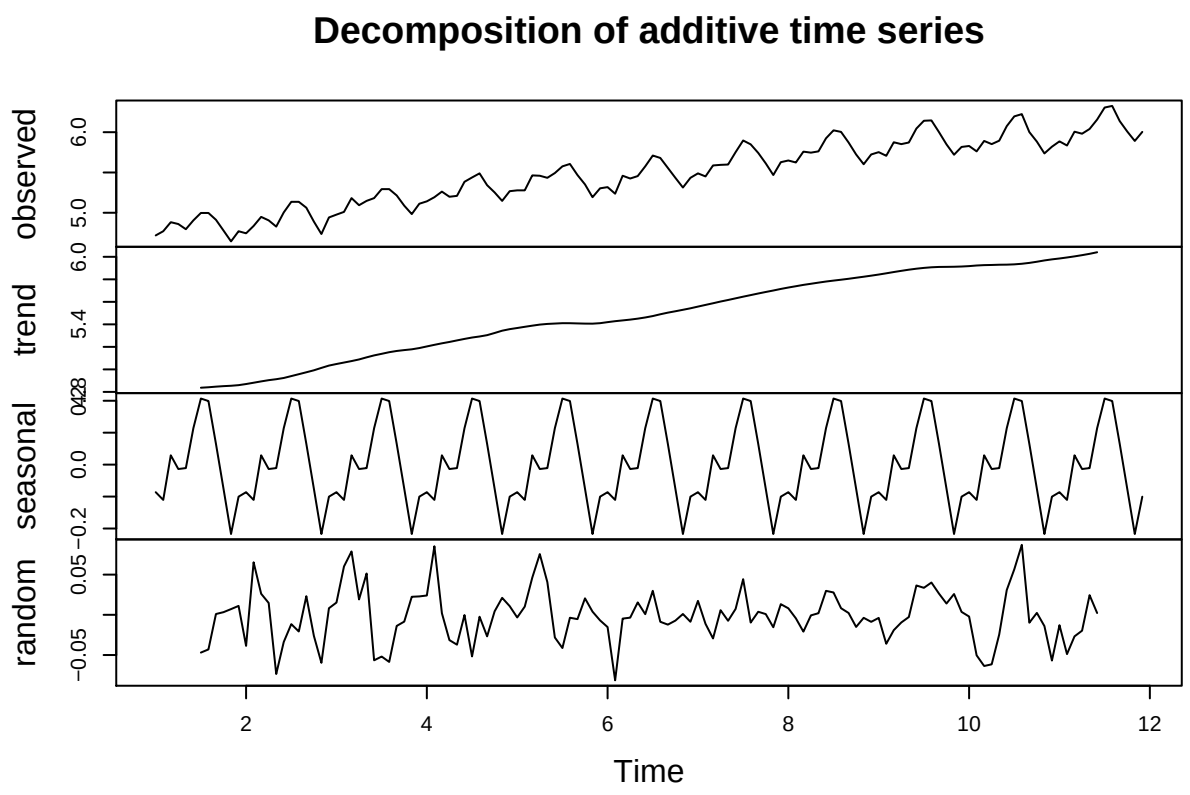


Observamos aquí:

-observed: los valores de la serie
-trend: la tendencia calculada mediante filtros
-seasonal: estacionalidad en la que cada 12 instantes de tiempo se repite la serie
-random: lo que queda de la serie una vez eliminadas tendencia y estacionalidad

Como observamos en random, la varianza es alta, lo que puede dar problemas en un futuro para la estacionariedad, ya que como sabemos, una serie con estacionariedad no varía en media ni en varianza. Para ello, a la serie inicial le calculamos el logaritmo de la serie (calculamos el logaritmo tanto a los datos como a la serie temporal):

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



Como observamos ahora, la varianza consta de menor variación a lo largo del tiempo, lo que en un futuro provocará que no tengamos problemas a la hora de calcular la estacionariedad.

Aplicando decompose sobre los datos podemos observar como los valores de cada mes en el atributo seasonal son exactamente los mismos, lo que nos hará falta para calcular la componente estacional y restarselo a la serie.

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

```
## $x
##         Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
## 1  4.718499 4.770685 4.882802 4.859812 4.795791 4.905275 4.997212 4.997212
## 2  4.744932 4.836282 4.948760 4.905275 4.828314 5.003946 5.135798 5.135798
## 3  4.976734 5.010635 5.181784 5.093750 5.147494 5.181784 5.293305 5.293305
## 4  5.141664 5.192957 5.262690 5.198497 5.209486 5.384495 5.438079 5.488938
## 5  5.278115 5.278115 5.463832 5.459586 5.433722 5.493061 5.575949 5.605802
## 6  5.318120 5.236442 5.459586 5.424950 5.455321 5.575949 5.710427 5.680173
## 7  5.488938 5.451038 5.587249 5.594711 5.598422 5.752573 5.897154 5.849325
## 8  5.648974 5.624018 5.758902 5.746203 5.762051 5.924256 6.023448 6.003887
```

```
## 9  5.752573 5.707110 5.874931 5.852202 5.872118 6.045005 6.142037 6.146329
## 10 5.828946 5.762051 5.891644 5.852202 5.894403 6.075346 6.196444 6.224558
## 11 5.886104 5.834811 6.006353 5.981414 6.040255 6.156979 6.306275 6.326149
##         Sep      Oct      Nov      Dec
## 1  4.912655 4.779123 4.644391 4.770685
## 2  5.062595 4.890349 4.736198 4.941642
## 3  5.214936 5.087596 4.983607 5.111988
## 4  5.342334 5.252273 5.147494 5.267858
## 5  5.468060 5.351858 5.192957 5.303305
## 6  5.556828 5.433722 5.313206 5.433722
## 7  5.743003 5.613128 5.468060 5.627621
## 8  5.872118 5.723585 5.602119 5.723585
## 9  6.001415 5.849325 5.720312 5.817111
## 10 6.001415 5.883322 5.736572 5.820083
## 11 6.137727 6.008813 5.891644 6.003887
##
## $seasonal
##             Jan         Feb         Mar          Apr         May          Jun
## 1  -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
## 2  -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
## 3  -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
## 4  -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
## 5  -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
## 6  -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
## 7  -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
## 8  -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
## 9  -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
## 10 -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
## 11 -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726  0.11403832
##             Jul         Aug         Sep         Oct         Nov         Dec
## 1   0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
## 2   0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
## 3   0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
## 4   0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
## 5   0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
## 6   0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
## 7   0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
## 8   0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
## 9   0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
## 10  0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
## 11  0.20689984  0.19914832  0.06503613 -0.07542627 -0.21722154 -0.10035385
##
## $trend
##         Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
## 1        NA       NA       NA       NA       NA       NA 4.837280 4.841114
## 2  4.869840 4.881389 4.893411 4.904293 4.912752 4.923701 4.940483 4.957406
## 3  5.047776 5.060902 5.073812 5.088378 5.106906 5.124312 5.138282 5.152751
## 4  5.203909 5.218093 5.231553 5.243722 5.257413 5.270736 5.282916 5.292150
## 5  5.367695 5.378309 5.388417 5.397805 5.403849 5.407220 5.410364 5.410294
## 6  5.419628 5.428330 5.435128 5.442237 5.450659 5.461103 5.473655 5.489713
## 7  5.557864 5.572693 5.587498 5.602730 5.616658 5.631189 5.645937 5.659812
## 8  5.727153 5.738856 5.750676 5.760658 5.770846 5.780430 5.788745 5.796524
## 9  5.842665 5.853541 5.864863 5.875490 5.885654 5.894475 5.901555 5.907026
## 10 5.917360 5.922887 5.926146 5.927563 5.929657 5.930458 5.932964 5.938377
```

```
## 11 5.985269 5.994078 6.003991 6.014899 6.026589 6.040709       NA       NA
##         Sep      Oct      Nov      Dec
## 1  4.846596 4.851238 4.854488 4.859954
## 2  4.974380 4.991942 5.013095 5.033804
## 3  5.163718 5.171454 5.178401 5.189431
## 4  5.304079 5.323338 5.343560 5.357427
## 5  5.408381 5.406761 5.406218 5.410571
## 6  5.503974 5.516367 5.529403 5.542725
## 7  5.674172 5.687636 5.700766 5.714738
## 8  5.804821 5.814072 5.823075 5.832692
## 9  5.910012 5.910708 5.911637 5.913829
## 10 5.946188 5.956352 5.967813 5.977291
## 11       NA       NA       NA       NA
##
## $random
##               Jan           Feb           Mar           Apr           May
## 1              NA            NA            NA            NA            NA
## 2  -0.0386339970  0.0653225337  0.0261932467  0.0147482216 -0.0736314277
## 3   0.0152313141  0.0601629324  0.0788155515  0.0191380326  0.0513961090
## 4   0.0240281013  0.0852933536  0.0019817093 -0.0314592700 -0.0371192467
## 5  -0.0033066372  0.0102350395  0.0462589183  0.0755466804  0.0406806494
## 6  -0.0152342638 -0.0814587149 -0.0046979403 -0.0035211005  0.0154697303
## 7   0.0173472978 -0.0112246683 -0.0294051083  0.0057470561 -0.0074287676
## 8   0.0080946404 -0.0044088265 -0.0209297699 -0.0006885087  0.0020126383
## 9  -0.0038180964 -0.0360012050 -0.0190885497 -0.0095213382 -0.0027287804
## 10 -0.0021409158 -0.0504060472 -0.0636581078 -0.0615944124 -0.0244470388
## 11 -0.0128909429 -0.0488378968 -0.0267934492 -0.0197188603  0.0244726346
##               Jun           Jul           Aug           Sep           Oct
## 1              NA -0.0469674232 -0.0430504999  0.0010228263  0.0033113203
## 2  -0.0337929898 -0.0115840295 -0.0207556255  0.0231791098 -0.0261668558
## 3  -0.0565667896 -0.0518768430 -0.0585941210 -0.0138188466 -0.0084314293
## 4  -0.0002791708 -0.0517365147 -0.0023602730 -0.0267807069  0.0043612826
## 5  -0.0281966990 -0.0413144046 -0.0036404389 -0.0053568900  0.0205235777
## 6   0.0008077175  0.0298726519 -0.0086891533 -0.0121823358 -0.0072186807
## 7   0.0073449512  0.0443173338 -0.0096358733  0.0037950438  0.0009179914
## 8   0.0297872004  0.0278023678  0.0082145728  0.0022607399 -0.0150607236
## 9   0.0364915439  0.0335829803  0.0401549408  0.0263671358  0.0140430437
## 10  0.0308492379  0.0565803816  0.0870329719 -0.0098095757  0.0023969741
## 11  0.0022314988            NA            NA            NA            NA
##               Nov           Dec
## 1   0.0071245946  0.0110841849
## 2  -0.0596746089  0.0081925943
## 3   0.0224266863  0.0229108650
## 4   0.0211557510  0.0107849852
## 5   0.0039607447 -0.0069125020
## 6   0.0010246364 -0.0086488767
## 7  -0.0154847585  0.0132371681
## 8  -0.0037344751 -0.0087532126
## 9   0.0258967636  0.0036357185
## 10 -0.0140188337 -0.0568544240
## 11            NA            NA
##
## $figure
##  [1] -0.08627390 -0.11042950  0.02915584 -0.01376615 -0.01080726
```

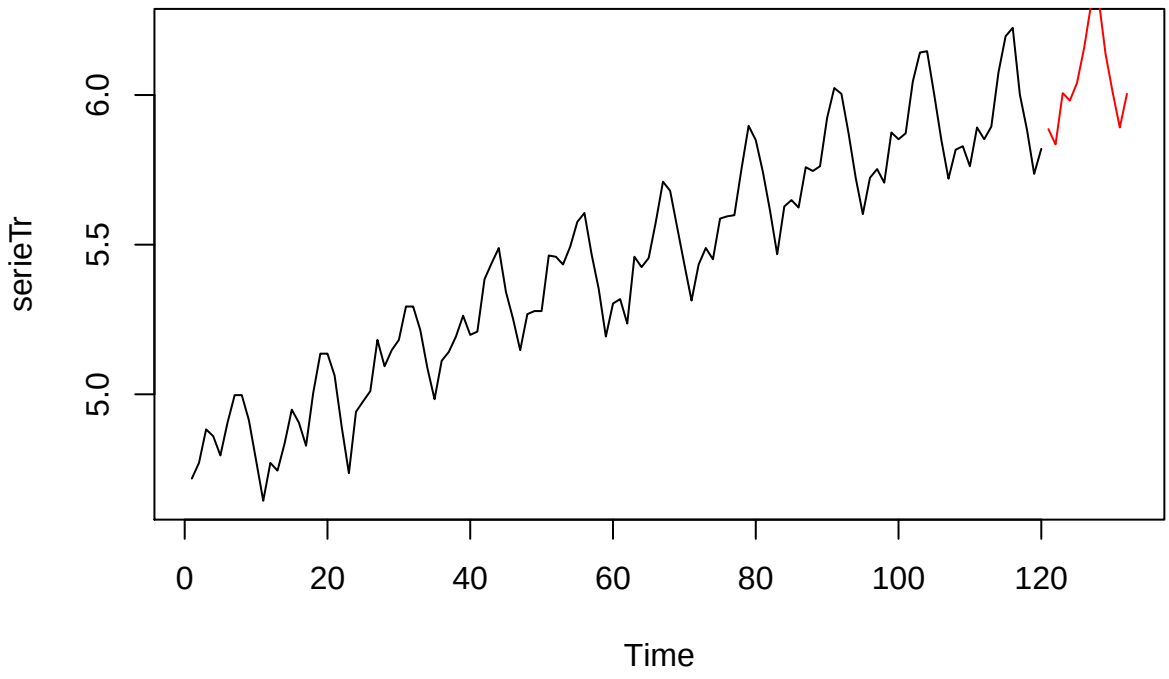
```
##  [6]  0.11403832  0.20689984  0.19914832  0.06503613 -0.07542627
## [11] -0.21722154 -0.10035385
##
## $type
## [1] "additive"
##
## attr(,"class")
## [1] "decomposed.ts"
```

Procedemos a la división de los datos en train y test

```
serieTr = serie.log[1:(length(serie.log)-NTest)] # tomamos todos los valores de la serie
                                                 # como train excepto los 12 últimos
tiempoTr = 1:length(serieTr) # instantes de tiempo de esta serie train

# mismo proceso para la serie de test...
serieTs = serie.log[(length(serie.log)-NTest+1):length(serie)]
tiempoTs = (tiempoTr[length(tiempoTr)]+1):(tiempoTr[length(tiempoTr)]+NTest)
```

Representamos la serie de entrenamiento y la linea de la serie de test en rojo con los parametros necesarios para que salga de forma correcta dentro de los límites de la gráfica y en el lugar adecuado.

```
plot.ts(serieTr, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempoTs, serieTs, col="red")
```



A continuación prodecemos a modelar la tendencia. Parece ser que la tendencia es lineal y creciente, por lo que tendrá la forma de:
serie = parametroA*tiempo + parametroB.

Con la funcion lm calculamos esos dos parametros para modelar dicha tendencia.

```
parametros.H1 = lm(serieTr ~tiempoTr)
parametros.H1
```

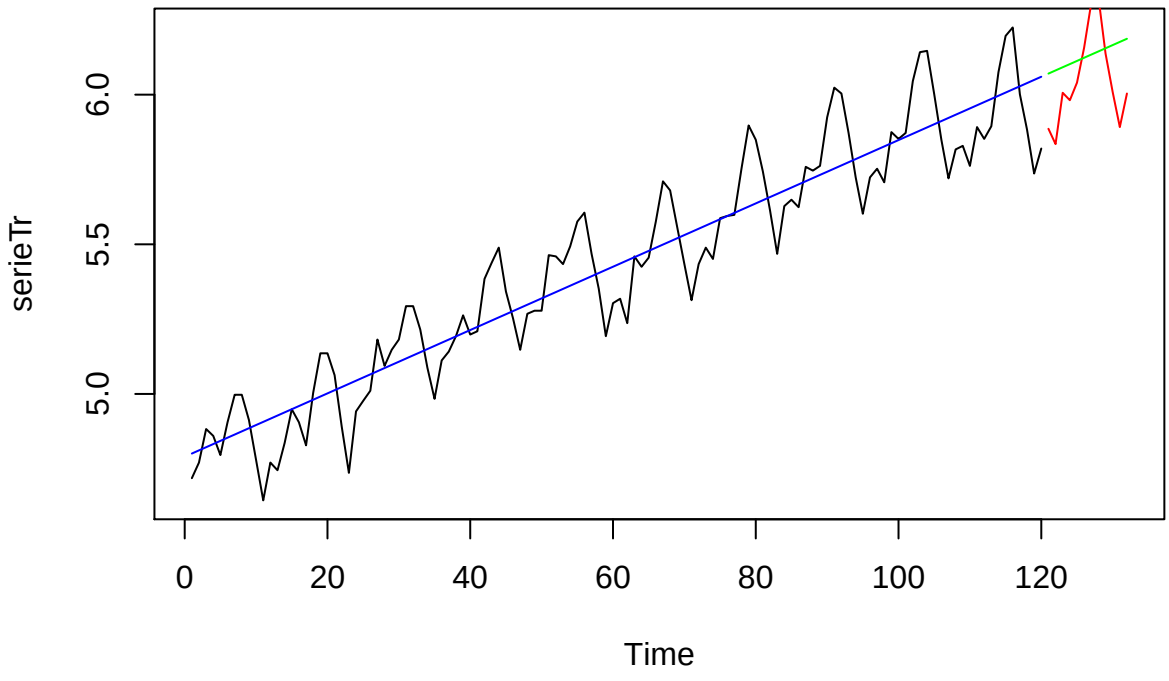
```
##
## Call:
## lm(formula = serieTr ~ tiempoTr)
##
## Coefficients:
## (Intercept)     tiempoTr
##     4.79025      0.01058
```

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

```
# tendencia en entrenamiento
tendEstimadaTr = parametros.H1$coefficients[1] + tiempoTr*parametros.H1$coefficients[2]
# tendencia en test
tendEstimadaTs = parametros.H1$coefficients[1] + tiempoTs*parametros.H1$coefficients[2]
```

Comprobamos de forma visual si la tendencia se ajusta al modelo que tenemos de la serie temporal.

```
plot.ts(serieTr, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempoTs, serieTs, col="red")
lines(tiempoTr, tendEstimadaTr, col = "blue")
lines(tiempoTs, tendEstimadaTs, col = "green")
```



Validamos a continuación de forma estadística que el modelo lineal creado sea correcto. Para ellos hemos de

comprobar que los errores a lo largo del rango de la serie tanto en entrenamiento como en test son normales, es decir, que se distribuyen mediante distribución normal a lo largo de todo el tiempo.

Aplicamos el test de normalidad de Jarque Bera para comprobar la normalidad en los errores tanto en los residuos del entrenamiento como en los del test, calculados como la tendencia estimada en test menos la serie temporal.

```
JB.tr = jarque.bera.test(parametros.H1$residuals)
JB.ts = jarque.bera.test(tendEstimadaTs-serieTs)
JB.tr
```

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

```
JB.ts
```

```
##
##  Jarque Bera Test
##
## data:  tendEstimadaTs - serieTs
## X-squared = 0.87957, df = 2, p-value = 0.6442
```

Asumiendo confianza del 95% podemos asumir que no hay diferencia significativa de los datos de error con respecto a los de una distribución normal ni en train ni en test (ya que ambos p-value son mayores a 0.05).

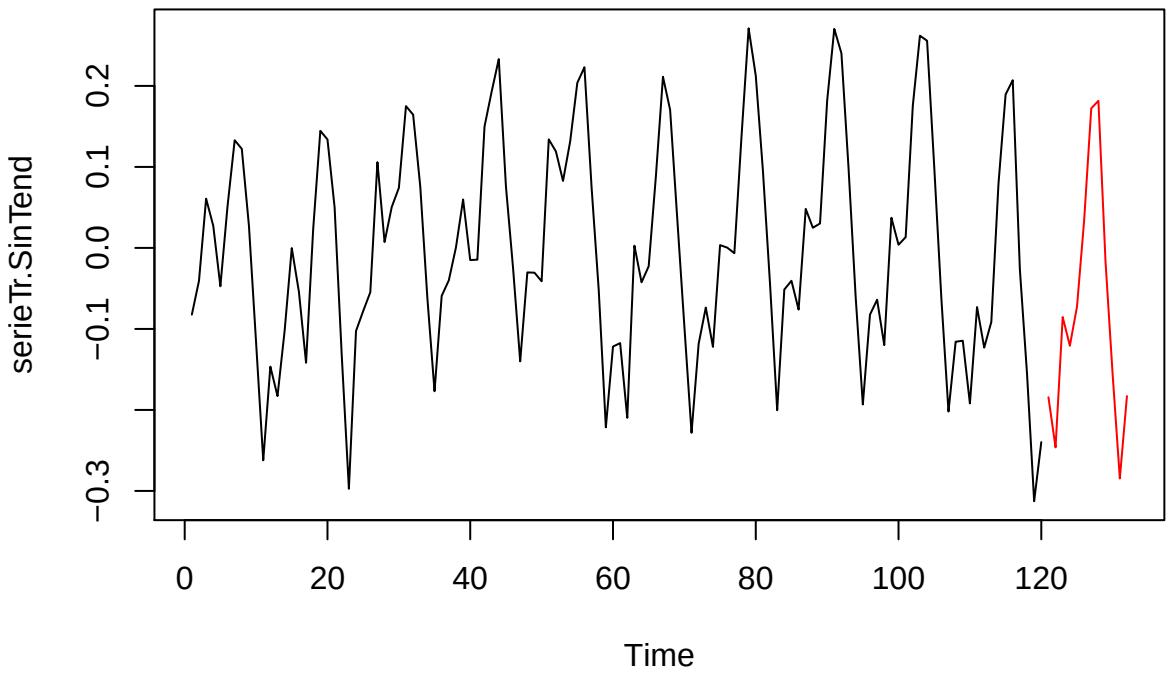
Comparamos las medias de error para comprobar si el error producido en la parte de train es equivalente al producido en la parte de test. Aplicamos el Test de Student para comparar dos distribuciones de datos diferentes (mediante sus medias).

```
TT = t.test(c(parametros.H1$residuals, tendEstimadaTs-serieTs))
TT
```

```
##
##  One Sample t-test
##
## data:  c(parametros.H1$residuals, tendEstimadaTs - serieTs)
## 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 p-value nos indica que esta distribución de errores es de media cero (casi: 0.007), y que no hay una desviación significativa con respecto a esta media, por tanto ambos errores de test y entrenamiento tienen la misma media y el modelo lineal es válido, por lo que la tendencia calculada es correcta. Procedemos por tanto a eliminar la tendencia de las series iniciales de entrenamiento y test.

```
serieTr.SinTend = serieTr - tendEstimadaTr
serieTs.SinTend = serieTs - tendEstimadaTs

# comprobamos como queda:
plot.ts(serieTr.SinTend, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempoTs, serieTs.SinTend, col="red")
```

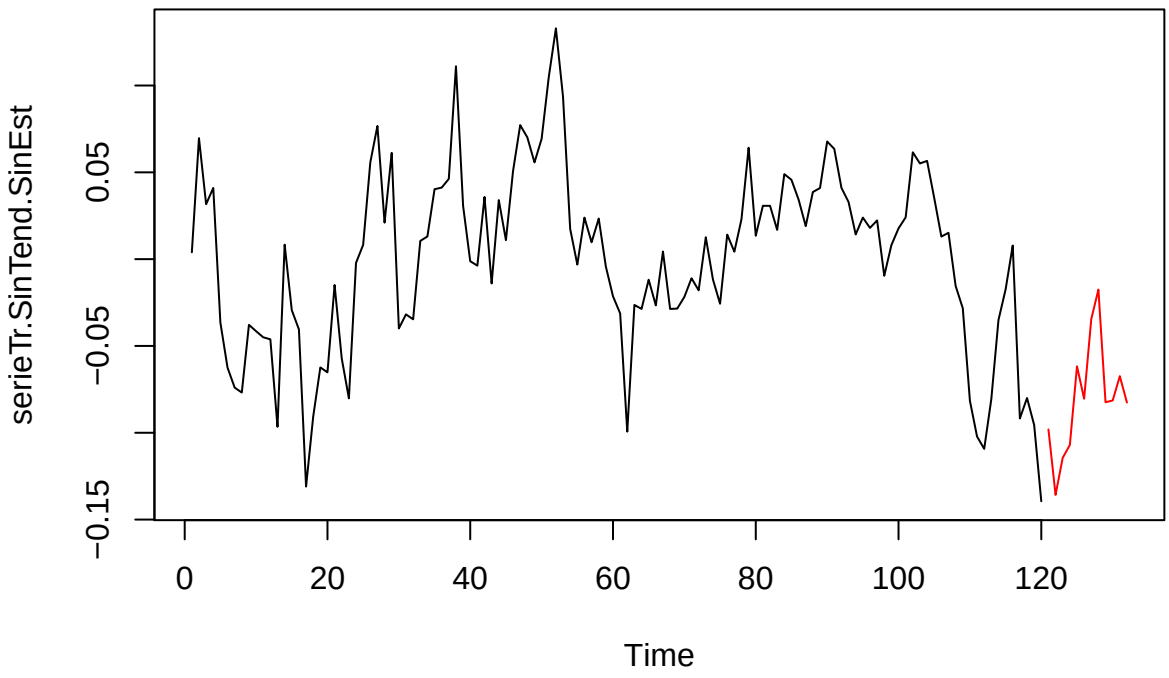


Una vez eliminda la tendencia quitamos la estacionalidad. Usamos para ello la funcion decompose

```
k = 12 # aquí guardamos los datos estacionales
estacionalidad = decompose(serie.ts.log)$seasonal[1:k] # tal como dijimos antes usamos
                                                       # el atributo seasonal
                                                       # para calcular los k=12 valores
                                                       # de estacionalidad
                                                       # que se van repitiendo en la estacionalidad
```

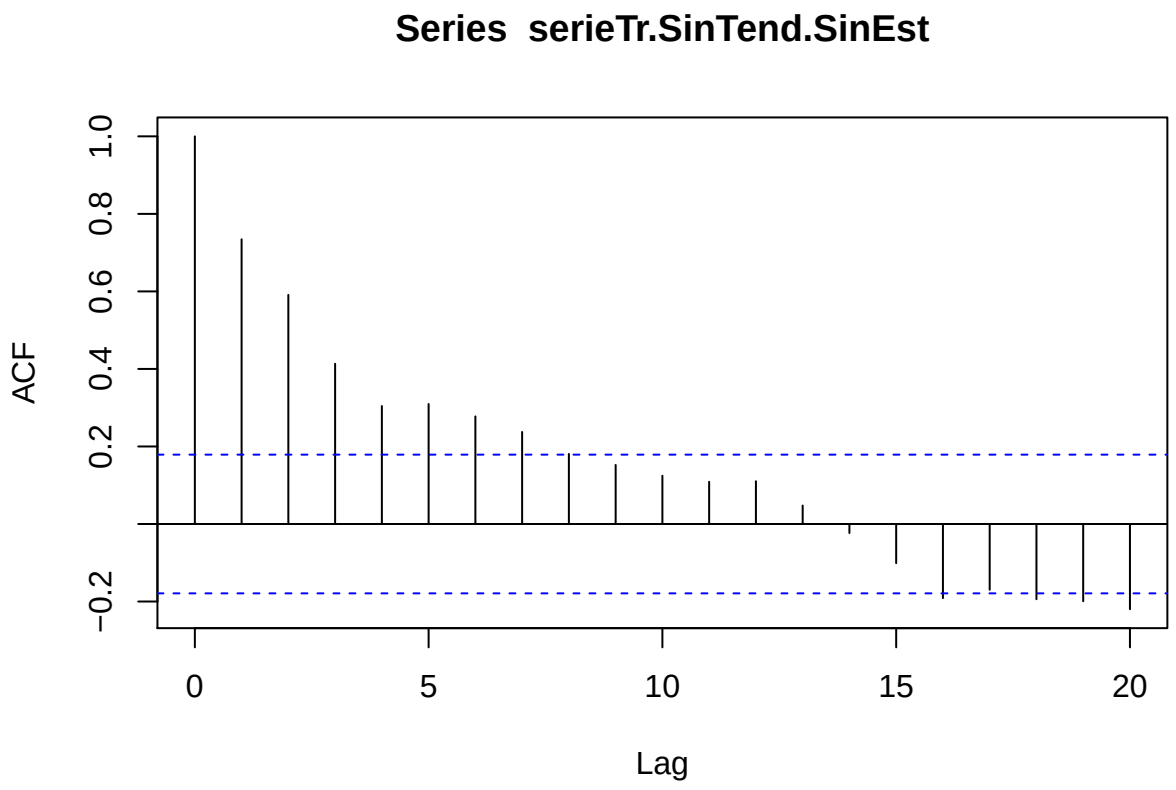
Ahora mismo tenemos una serie sin tendencia con 120 valores (train) y una estacionalidad con 12 valores. A esta serie sin tendencia hemos de restarle la estacionalidad de 12 en 12 valores (serie[1:12] - estacionalidad, serie[13:25] - estacionalidad . . .).

```
# repetimos la estacionalidad para ello, tantas veces como valores haya en la serie
aux = rep(estacionalidad, length(serieTr.SinTend)/length(estacionalidad))
# quitamos la estacionalidad a train y test
serieTr.SinTend.SinEst = serieTr.SinTend - aux
serieTs.SinTend.SinEst = serieTs.SinTend - estacionalidad # no usamos aux porque estacionalidad
                                                          # tiene la misma cantidad de valores que
                                                          # serieTs.SinTend
# comprobamos como queda:
plot.ts(serieTr.SinTend.SinEst, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempoTs, serieTs.SinTend.SinEst, col="red")
```



Comprobamos si la serie es estacionaria o no con el test ACF

```
acf(serieTr.SinTend.SinEst)
```



No llega a ser un resultado visualmente aclaratorio para determinar si posee o no estacionariedad debido a que posee una bajada muy suave entre instantes de tiempo, por tanto procedemos a aplicar un test de Dickey-Fuller aumentado para asegurarnos.

```
ADFTr = adf.test(serieTr.SinTend.SinEst)
ADFTr
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  serieTr.SinTend.SinEst
## Dickey-Fuller = -1.8407, Lag order = 4, p-value = 0.6427
## alternative hypothesis: stationary
```

Con un nivel de confianza del 95% no podemos asegurar que la serie sea estacionaria. Lo normal es no sea estacionario en media, es decir, que la media a lo largo del tiempo varie. Debemos hacer la serie estacionaria, por lo que vamos a diferenciarla.

```
serieTr.SinTend.SinEst.Diff = diff(serieTr.SinTend.SinEst) # diferenciación en una unidad
                                                           # de la serie anterior
serieTs.SinTend.SinEst.Diff = diff(serieTs.SinTend.SinEst)

# le pasamos el test de nuevo para comprobar si es estacionaria
ADFTr2 = adf.test(serieTr.SinTend.SinEst.Diff)
```

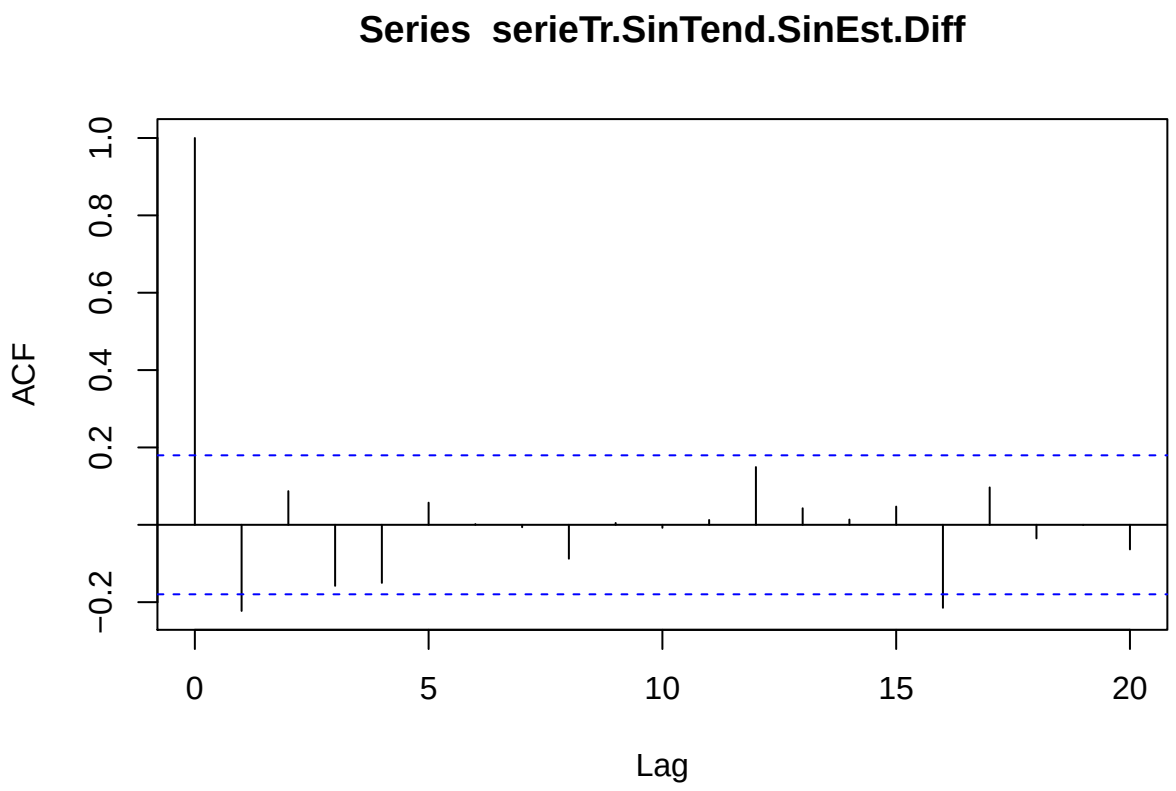
```
## Warning in adf.test(serieTr.SinTend.SinEst.Diff): p-value smaller than
## printed p-value
```

```
ADFTr2
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  serieTr.SinTend.SinEst.Diff
## Dickey-Fuller = -6.2153, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

Al ser p-value < 0.05 podemos asumir que ya sí es estacionaria. Lo comprobamos también de forma visual:

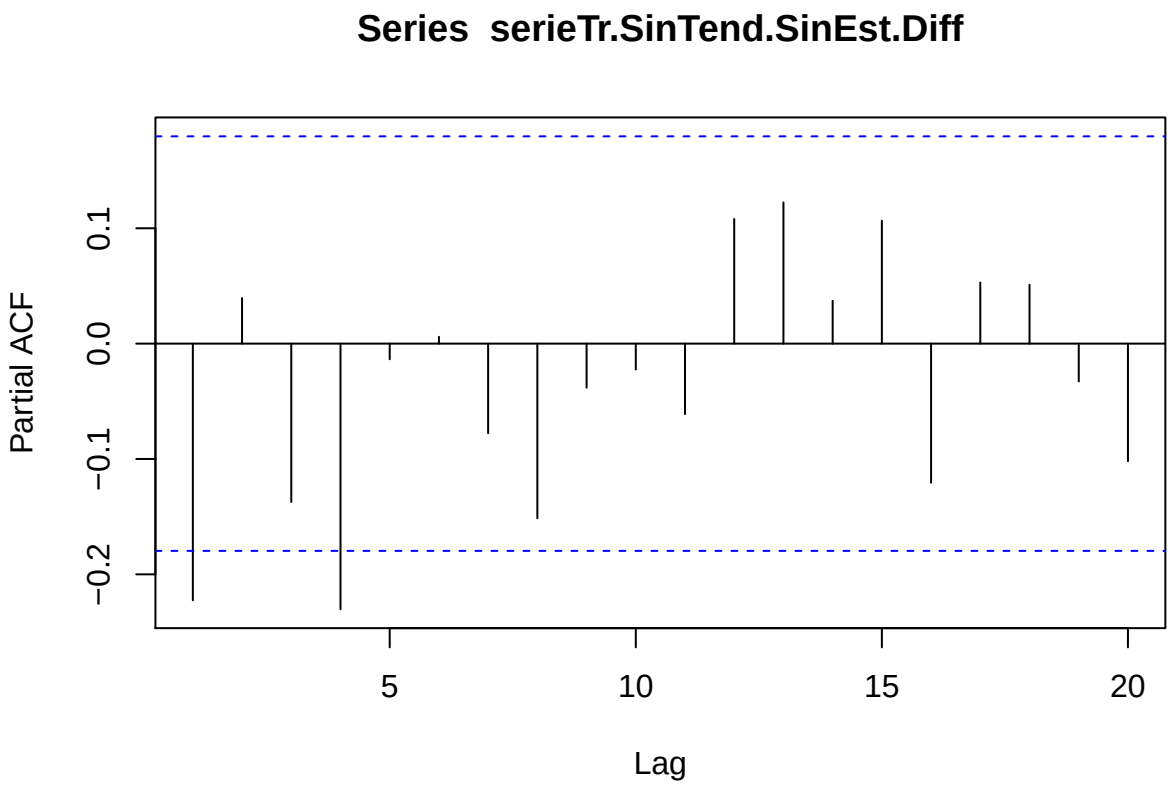
```
acf(serieTr.SinTend.SinEst.Diff)
```



Observamos la existencia de estacionaridad con la gran bajada entre los instantes 0 y 1.

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

```
pacf(serieTr.SinTend.SinEst.Diff)
```



Podemos pensar que estas gráficas de acf y pacf son típicos de un modelo autoregresivo de grado 4 (el último instante donde la línea pasa el umbral en pacf). Por ello deberíamos poder modelar la componente irregular del sistema restante tras quitar la tendencia y la estacionalidad mediante un modelo autoregresivo de orden 4. Podemos usar para ello un modelo ARIMA con una diferenciación. Entrenamos dicho modelo a continuación.

```
# calculamos el modelo
# la diferenciación la introducimos dentro del modelo, por tanto usamos serieTr.SinTend.SinEst
modelo = arima(serieTr.SinTend.SinEst, order = c(4,1,0)) # 0 indica que no es de medias moviles
# vemos los residuos del modelo, que al ser un modelo autoregresivo lo calculamos como residuos+serie
valoresReconstruidos = serieTr.SinTend.SinEst + modelo$residuals # valores reconstruidos de la serie
                                                                 # con el modelo que tenemos


# calculamos las predicciones
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
```

```
# calculamos el error cuadrático acumulado del ajuste en ajuste y test
errorTr = sum((modelo$residuals)^2)
errorTs = sum((valoresPredichos - serieTs.SinTend.SinEst) ^2)
```

```
errorTr
```

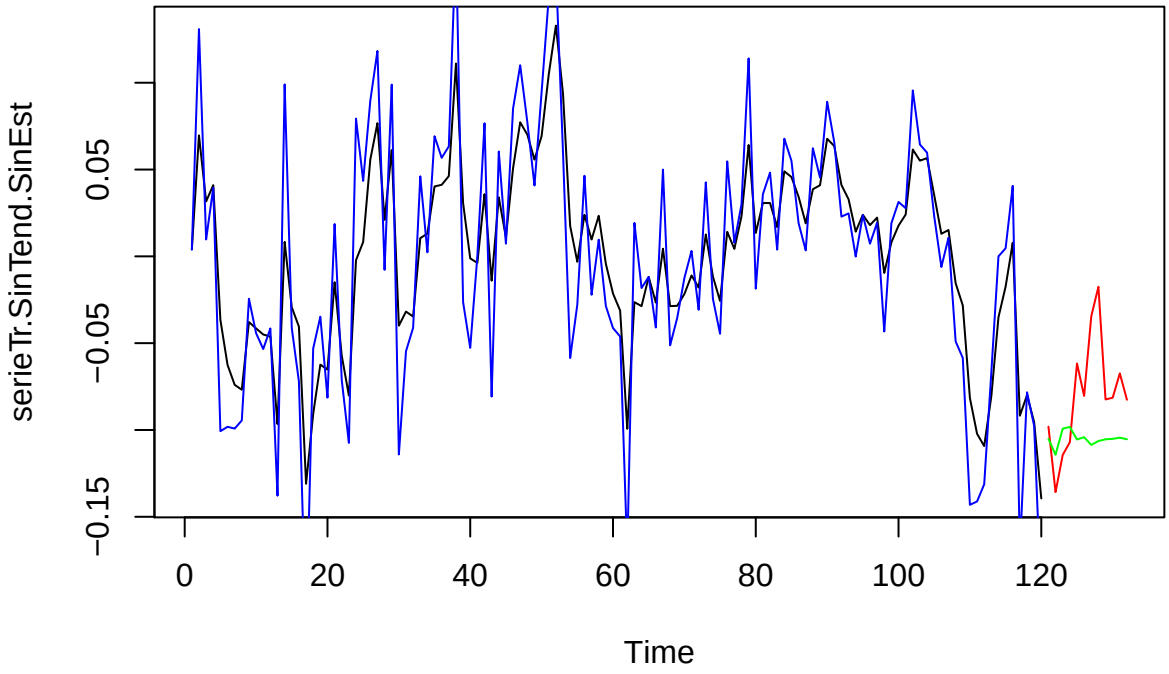
```
## [1] 0.1344656
```

```
errorTs
```

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

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

```
plot.ts(serieTr.SinTend.SinEst, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(valoresReconstruidos, col="blue")
lines(tiempoTs,serieTs.SinTend.SinEst, col="red")
lines(tiempoTs,valoresPredichos, col="green")
```



La reconstrucción para el train es razonablemente fiel, sin embargo para el test no, posiblemente debido a que, al observar acf y pacf, podemos ver que los instantes distintos al actual están muy cercanos a los límites, por lo que quizás no sean lo suficientemente importantes para calcular el valor en el instante actual. Podemos decir que nuestra serie es prácticamente ruido blanco, por ello nos aparece la predicción verde observada en la gráfica. Finalmente comprobamos la validez del modelo (sus errores) mediante tests estadisticos. Para ello aplicamos varios:

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

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

```
##
##  Box-Pierce test
##
```

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

Con una confianza del 95% no podemos pensar que los residuos no sean valores aleatorios, por tanto pasa el test determinando que sí, siguen una distribución aleatoria. Si no fuesen aleatorios significaría que nuestro modelo no se ajustaría a todas las características de la señal, por lo que sería un modelo incorrecto. En este caso es válido con respecto al valor de los residuos.

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

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

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

Saphiro-Wilk: mismo cometido que Jarque Bera

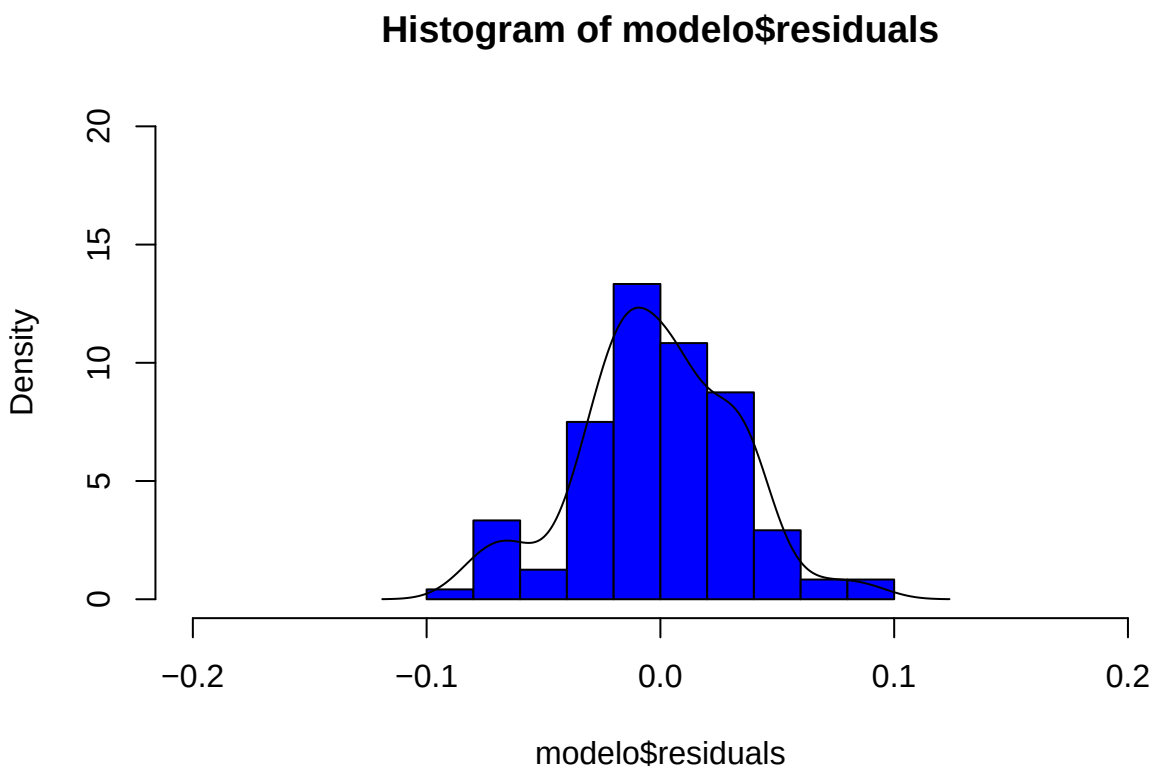
```
swt = shapiro.test(modelo$residuals)
swt
```

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

Asumimos en ambos la normalidad de los residuos con el resultado de los p-value obtenidos, ya que la hipótesis nula propone que los datos no siguen una distribución normal.

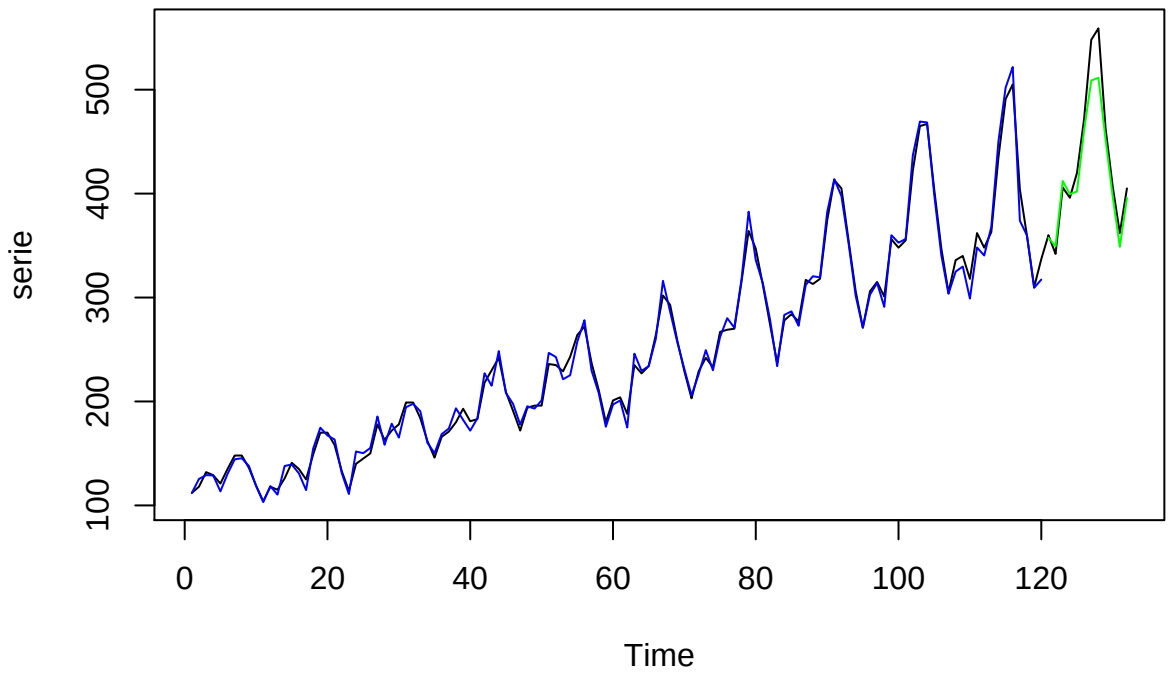
A continuación mostramos un histograma y función de densidad para obtener resultados gráficos de confirmación.

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



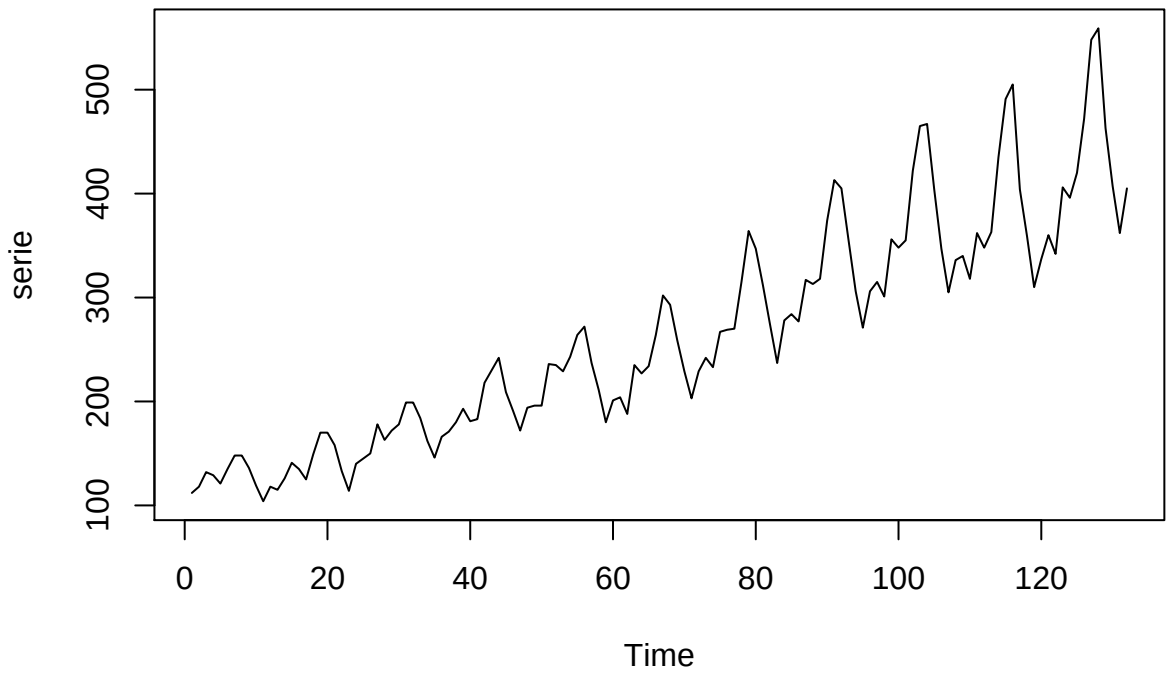
Con estas comprobaciones el modelo queda validado, ya que los errores que nos dan son aleatorios que provienen de una distribución normal. A continuación hemos de deshacer los cambios para comprobar la predicción con el conjunto de datos de test. Procedemos a ello:

```
# Incluimos la estacionalidad
valoresReconstruidos.Est = valoresReconstruidos + aux
valoresPredichos.Est = valoresPredichos + estacionalidad

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

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

plot.ts(serie)
lines(tiempoTr, valoresReconstruidos.Est.Tend.exp, col = "blue")
lines(tiempoTs, valoresPredichos.Est.Tend.exp, col = "green")
```



A continuación procedo a calcular la predicción para el año 1960 a partir de este mismo conjunto de train. Partimos de la siguiente serie

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

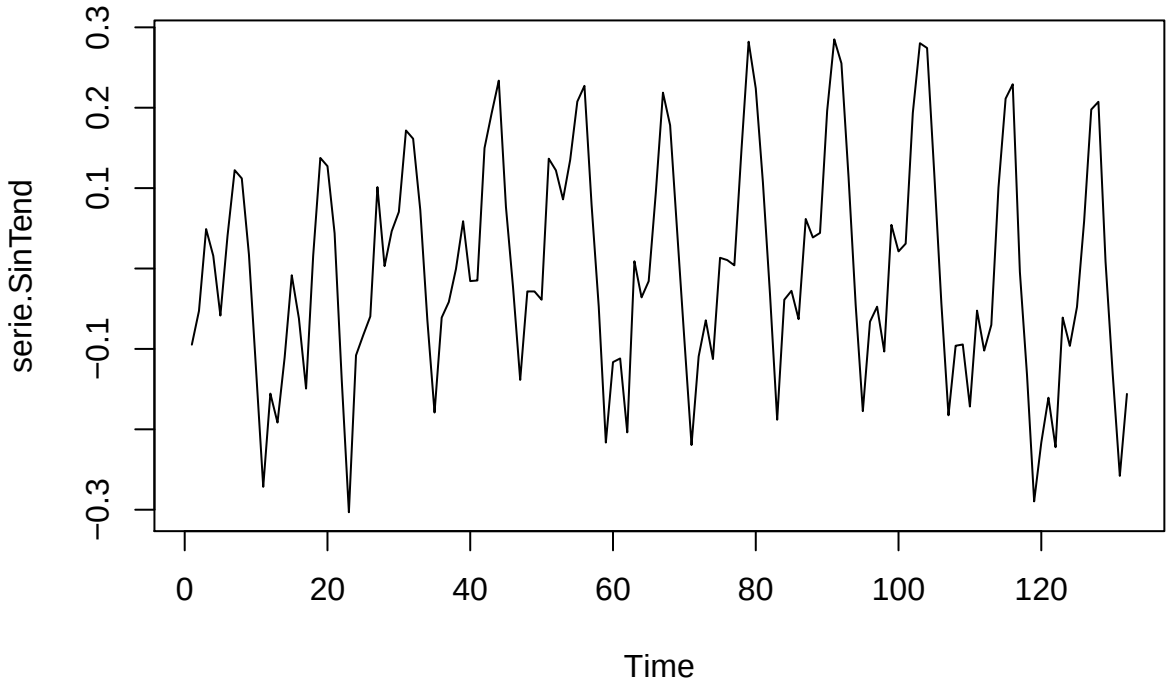


Dividimos en train y test:

```
serieTr = serie.log # tomamos todos los valores de la serie
tiempo = 1:length(serieTr) # instantes de tiempo de esta serie train
tiempoTs =(length(tiempo)+1):(length(tiempo)+NPred)
```

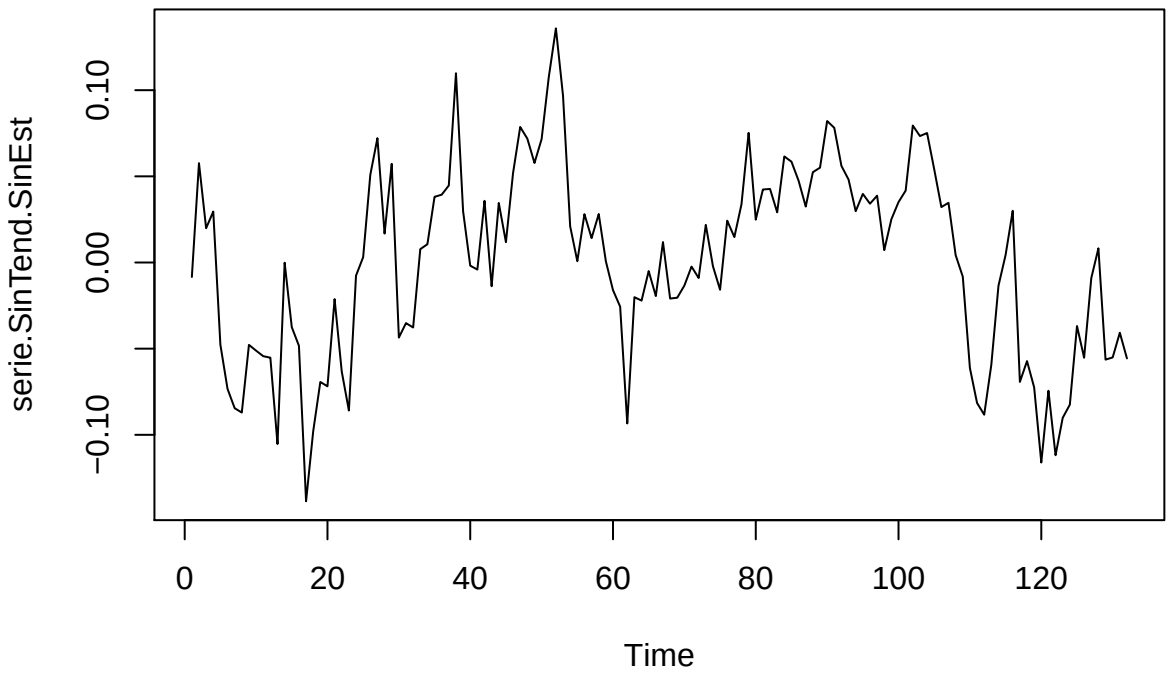
Modelamos tendencia, la estimamos y se la quitamos a la serie:

```
parametros.H1 = lm(serieTr ~tiempo)
# tendencia en entrenamiento
tendEstimada = parametros.H1$coefficients[1] + tiempo*parametros.H1$coefficients[2]
# tendencia en test
tendEstimadaTs = parametros.H1$coefficients[1] + tiempoTs*parametros.H1$coefficients[2]
serie.SinTend = serieTr - tendEstimada
plot.ts(serie.SinTend, xlim=c(1, tiempo[length(tiempo)]))
```



Le quitamos ahora la estacionalidad a la serie:

```
k = 12 # aquí guardamos los datos estacionales
estacionalidad = decompose(serie.ts.log)$seasonal[1:k]
aux = rep(estacionalidad, length(serie.SinTend)/length(estacionalidad))
serie.SinTend.SinEst = serie.SinTend - aux
plot.ts(serie.SinTend.SinEst, xlim=c(1, tiempo[length(tiempo)]))
```

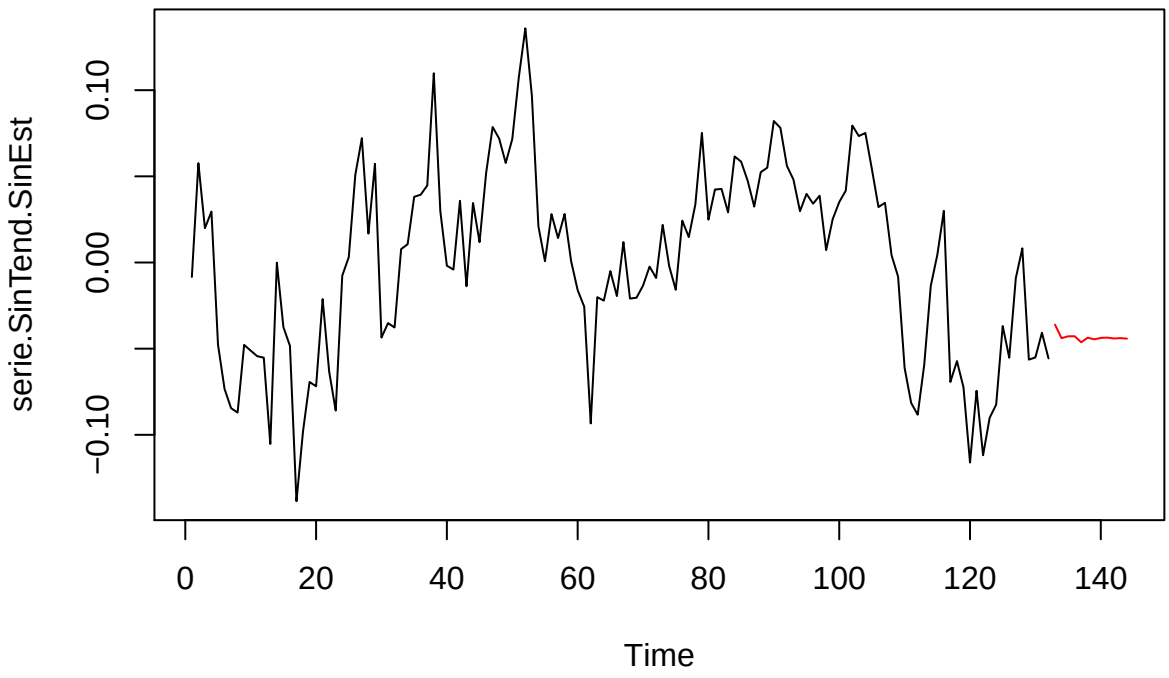


Calculamos el modelo:

```
# la diferenciación la introducimos dentro del modelo, por tanto usamos serieTr.SinTend.SinEst
modelo = arima(serie.SinTend.SinEst, order = c(4,1,0)) # 0 indica que no es de medias moviles

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

```
## Time Series:
## Start = 133
## End = 144
## Frequency = 1
##  [1] -0.03608365 -0.04392406 -0.04284180 -0.04279409 -0.04628586
##  [6] -0.04363918 -0.04460773 -0.04376963 -0.04357871 -0.04411222
## [11] -0.04387764 -0.04417832
```

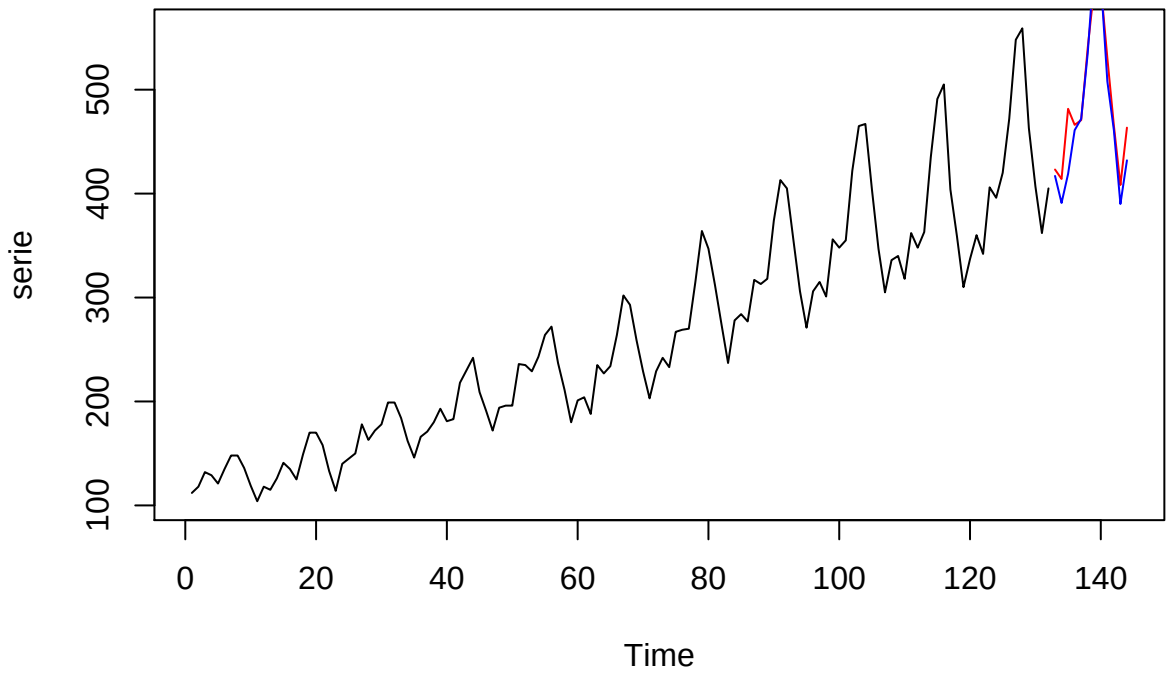
```
# observamos el resultado
plot.ts(serie.SinTend.SinEst, xlim=c(1,tiempoTs[length(tiempoTs)]))
lines(tiempoTs,valoresPredichos, col="red")
```



Aquí tenemos el ajuste del modelo con la componente regular, al igual que antes, podemos intuir que es bastante malo.

Volvemos hacia atrás para comparar las predicciones con el dataset de test:

```
# Incluimos la estacionalidad
valoresPredichos.Est = valoresPredichos + estacionalidad

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

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

# leemos los datos de test para comprobar los resultados
predReales = scan("pasajeros_1960.predict")

plot.ts(serie, xlim=c(1, tiempoTs[length(tiempoTs)]))
lines(tiempoTs,valoresPredichos.Est.Tend.exp, col="red")
lines(tiempoTs, predReales, col="blue")
```



Como vemos el modelo ajusta sus predicciones de buena manera a los datos de test del año 1960 aplicando exactamente las mismas transformaciones realizadas sobre los datos de train de 1949-1959.