Explore Generalized ARMA with TSMS

Tianyang Xie, Jie Ding

09/04/2020

This vignette gives a high level overview on how to use the 'tsms' R package. The R package consists of a tool box for generalized ARMA model, as well as the MS modeling procedure for univariate time series with multiple seasonality.

Installation

The installation of 'tsms' involves using 'devtools' package. Run the following code to load 'devtools', and then install 'tsms' from the github repository. Note that for R version > 4.0.0 users, older version of dependent packages (e.g. gridExtra) would throw out warnings through the installation. We added a command line in the following code to avoid auto-converting warnings into errors.

```
install.packages("devtools")
library("devtools")
Sys.setenv("R_REMOTES_NO_ERRORS_FROM_WARNINGS" = "true") # avoid auto-converting warnings into errors
install_github('TYtianyang/tsms')
```

```
library(tsms)
#> Loading required package: gridExtra
#> Loading required package: parallel
```

Generalized ARMA

The regular ARMA model consists of AR and MA components with continued lagging orders. For example, a regular ARMA(p,q) model could be written as:

$$X_t = \sum_{i=1}^{p} \phi_i * X_{t-i} + \sum_{j=1}^{q} \psi_j * \epsilon_{t-j} + \epsilon_t$$

X_t is the data at time point t, ϕ_i is the coefficient for AR component at lag i, ψ_j is the coefficient for MA component at lag j, ϵ_t is the white noise at time point t. For the rest of the discussion, we assume the white noise following normal distribution.

As the name suggested, the generalized ARMA model doesn't require AR and MA components with continued lagging orders. Any set of lagging orders for AR and MA components would be allowed. The generalized ARMA that have AR components with lagging set S1, and MA components with lagging set S2 can be written as the following:

$$X_t = \sum_{i \in S_1} \phi_i * X_{t-i} + \sum_{j \in S_2} \psi_j * \epsilon_{t-j} + \epsilon_t$$

Our R package 'tsms' offers 3 useful functions for the generalized ARMA model: 'garma.fit' for fitting the model on a data series, 'garma.pred' for predicting on a fitted model, and 'garma.gen' for generating a data series with specified parameters. Let's elaborate their usage by some examples.

Consider a generalized ARMA model with specified parameters as follwing:

$$X_t = 0.5X_{t-1} - 0.2X_{t-2} + 0.3\epsilon_{t-1} + (0.1X_{t-15} - 0.05X_{t-20}) + (0.1\epsilon_{t-10}) + \epsilon_t, \sigma = 2$$

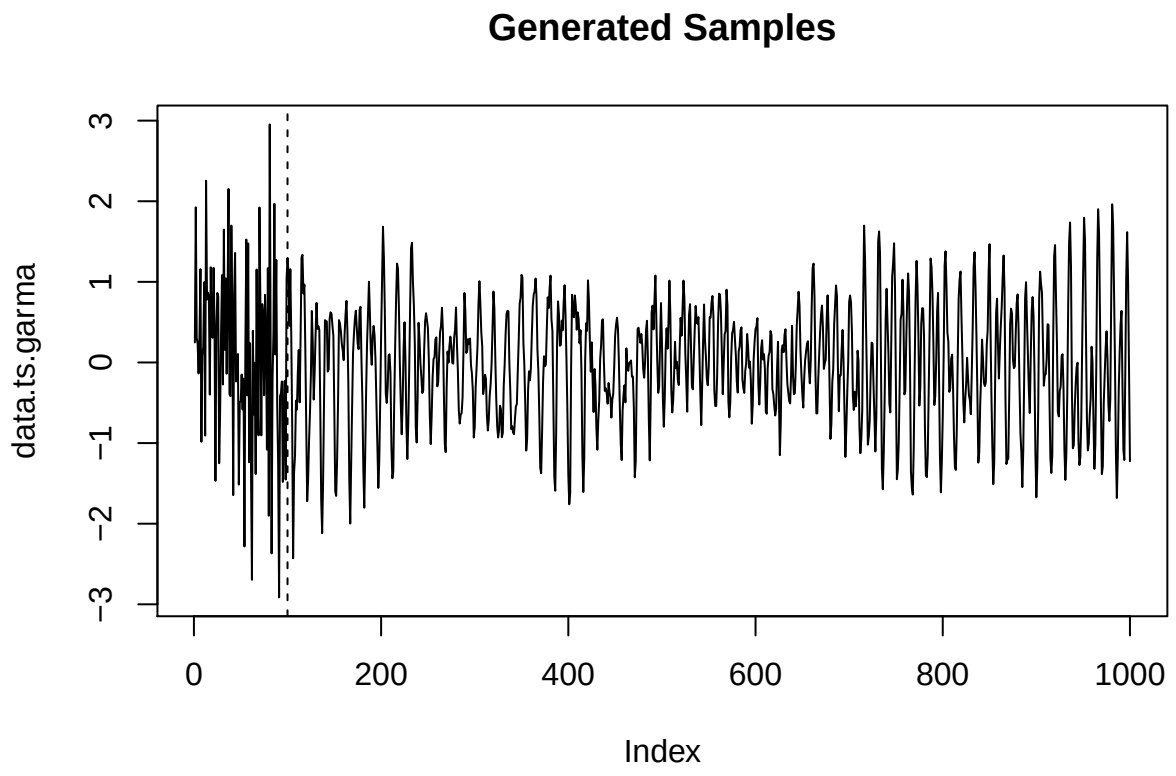
We can generate a time series with length 1000 from the generalized ARMA model above by using 'garma.gen' function:

```
data.ts.garma = garma.gen(n=1000,U=100,
              p=2,q=1,S1=c(15,20),S2=c(10),
              phi=c(0.5,-0.2),psi=c(0.3),tau1=c(0.5,-0.2),tau2=0.1,
              noise=0.2)$X
```

Note that 'garma.gen' return a list of objects. By calling 'X' from it, the generated data can be obtained. X will be a vector with length 1000. 'garma.gen' function doesn't require the stationarity for the generated time series. However, it does filter out some of the generated results so that the generated series won't diverge drastically in the end.

Also note that the parameter U controls the number of burn-in samples. For those burn-in samples, the data are generated from white noise process with standard deviation specified in noise parameter. Only the samples after U are generated from the generalized ARMA model. Therefore the distributions of burn-in samples and samples after integer U would be different.

```
plot(data.ts.garma,main='Generated Samples',type='l')
abline(v=100,lty=2)
```



For generating a time series from a regular ARMA model, we can also utilize 'garma.gen' function. For example, for the following regular ARMA model:

$$X_t = 0.5X_{t-1} - 0.2X_{t-2} + 0.3\epsilon_{t-1} + \epsilon_t, \sigma = 2 \quad (1)$$

Use:

```
data.ts.arma = garma.gen(n=1000,U=100,
               p=2,q=1,S1=NULL,S2=NULL,
               phi=c(0.5,-0.2),psi=c(0.3),tau1=NULL,tau2=NULL,
               noise=0.2)$X
```

For fitting a generalized ARMA model, we can use 'garma.fit' function.

```
model.ts.garma = garma.fit(data.ts.garma[1:850],U=100,p=2,q=1,S1=c(15,20),S2=c(10),crit='AIC')
print(model.ts.garma)
#> $U
#> [1] 100
#>
#> $p
#> [1] 2
#>
#> $phi
#> [1]  0.4976974 -0.1885049
#>
#> $q
#> [1] 1
#>
#> $psi
#> [1] 0.326785
#>
#> $r1
#> [1] 2
#>
#> $S1
#> [1] 15 20
#>
#> $tau1
#> [1]  0.5064084 -0.1840115
#>
#> $r2
#> [1] 1
#>
#> $S2
#> [1] 10
#>
#> $tau2
#> [1] 0.1062488
#>
#> $gamma
#> [1] 0
#>
#> $sigma
#> [1] 0.2068554
#>
#> $dim
#> [1] 7
```

```
#>
#> $ic
#> [1] -221.1953
```

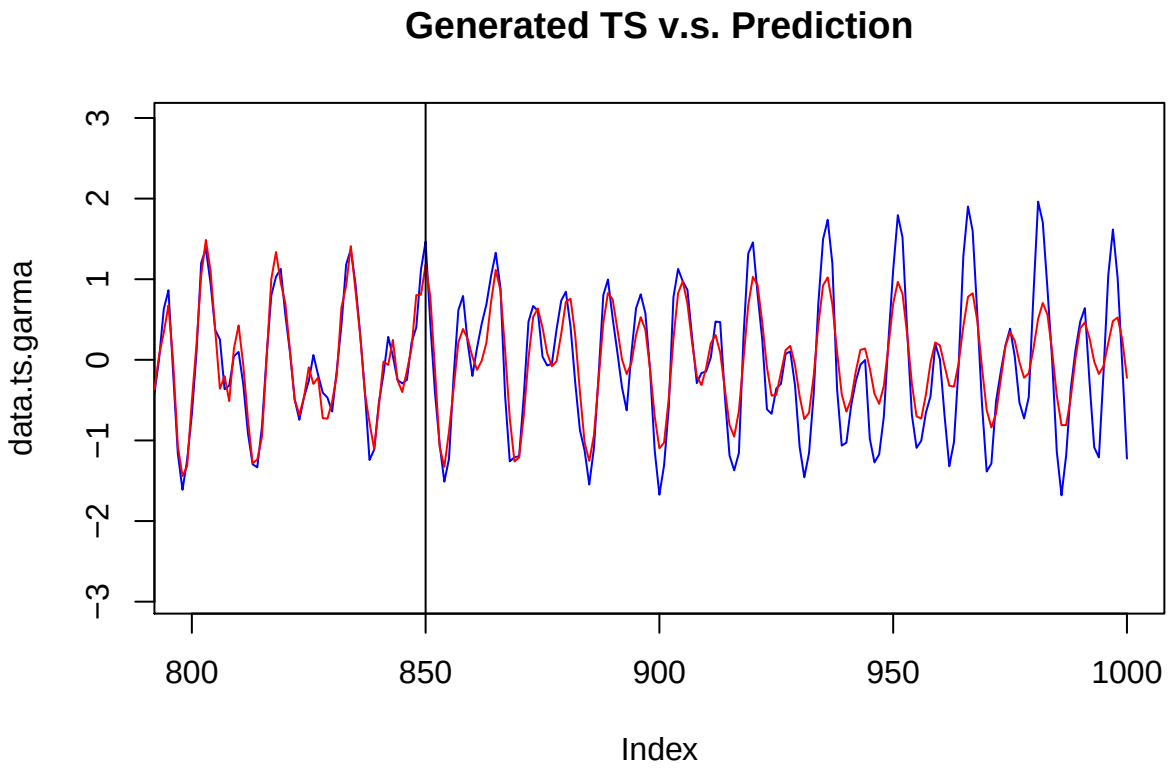
Note that in the example above, we only utilized the former 850 points for training.

The 'garma.fit' function returns a list of objects including the specified parameters and estimated coefficients. An information criterion value will also be recorded at the end of the list. One can also change the IC type to others such as 'BIC' or 'BC' by simply change the crit parameter.

```
model.ts.garma = garma.fit(data.ts.garma[1:850],U=100,p=2,q=1,S1=c(15,20),S2=c(10),crit='BIC')
model.ts.garma = garma.fit(data.ts.garma[1:850],U=100,p=2,q=1,S1=c(15,20),S2=c(10),crit='BC')
```

After fitting a model, we can pass on our fitted results to the prediction function 'garma.pred' to perform dynamic forecasting. For example, use the following code to perform 150 step forward forecasting.

```
pred.ts.garma = garma.pred(model.ts.garma,X=data.ts.garma[1:850],pred_t=150)
plot(data.ts.garma,type='l',xlim=c(800,1000),main='Generated TS v.s. Prediction',col='blue')
lines(pred.ts.garma,col='red')
abline(v=850)
```



MS modeling procedure

Another feature of the 'tsms' R package is the MS modeling procedure for univariate time series with multiple seasonality. Unlike the other methods for multiple seasonality setting, MS doesn't require a pre-determined seasonality periods. It only needs to know how many seasonality cycles should be inferred from the data. The procedure will first suggest potential seasonality periods by performing a Discrete Fourier Transformation based on the number of seasonality periods. Then, it uses a parallel fitting module to fit generalized ARMA

models for different parameters. Finally, it select the best model by minimizing IC value and perform forecasting.

The following code gives an example of the basic usage of MS modeling procedure. It performs a MS procedure with AR and MA components limited to at most 3 lags, allowing 2 seasonality cycles in the data. The procedure utilized the former 850 points of the series, burned in 100 points in the beginning, and forecast for 150 steps ahead at last.

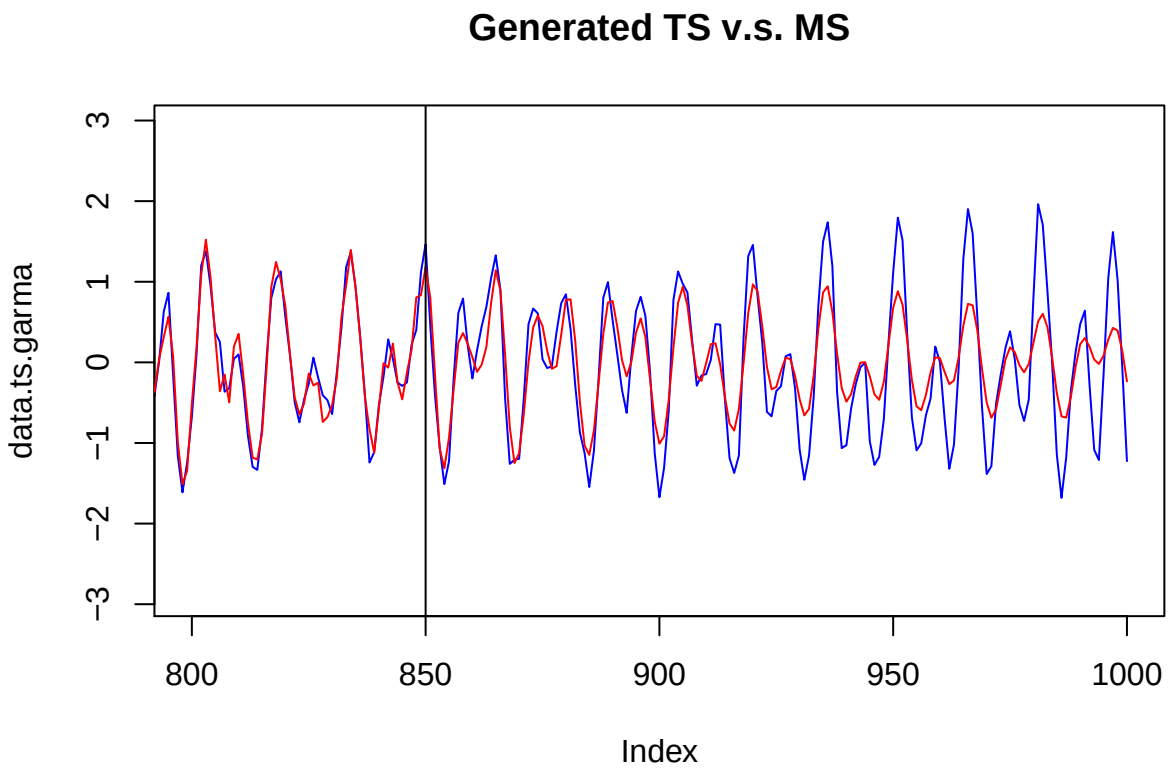
```
result = MS(X=data.ts.garma[1:850],U=100,p_range=c(0:3),q_range=c(0:3),r=2,pred_t=150)
```

The MS function outputs a list of objects: the best model object, the IC panel, and the prediction result.

```
print(names(result))
#> [1] "obj"        "ic_panel"   "prediction"
print(result$ic_panel)
#>    p q                            S1                            S2             W
#> 1  0 0 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 2  0 1 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 3  0 2 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 4  0 3 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 5  1 0 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 6  1 1 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 7  1 2 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 8  1 3 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 9  2 0 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 10 2 1 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 11 2 2 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 12 2 3 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 13 3 0 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 14 3 1 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 15 3 2 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#> 16 3 3 6 7 8 9 10 12 13 14 15 16 17 6 7 8 9 10 12 13 14 15 16 17 sea1 sea2 exo
#>          ic dim
#> 1  553.0418  23
#> 2  187.4047  24
#> 3  147.2208  25
#> 4  150.4980  26
#> 5  267.1572  24
#> 6  159.7091  25
#> 7  149.9470  26
#> 8  153.2477  27
#> 9  139.0143  25
#> 10 139.4761  26
#> 11 132.9600  27
#> 12 135.6686  28
#> 13 142.3898  26
#> 14 139.4367  27
#> 15 131.4033  28
#> 16 141.9166  29
```

The prediction result compares to the original series by the following codes.

```
plot(data.ts.garma,type='l',xlim=c(800,1000),main='Generated TS v.s. MS',col='blue')
lines(result$prediction,col='red')
abline(v=850)
```



In general, the parameters 'p_range','q_range' and 'r' would be the most important parameters for the MS. But MS modeling procedure also has many other parameters so that the users can play with them to fully explore the flexibility of the method.

By setting the parameter 'blur.out' to a different value, we can allow the procedure to select the best model on a wider bandwidth of seasonality components. For example, we allow 'blur.out' to be 'c(3,3)' in the following, so that the procedure will also consider including the AR and MA components with lag-3 and lag+3 for each seasonality order. The default value for 'blur.out' is 'c(2,2)'

```
result = MS(X=data.ts.garma[1:850],U=100,p_range=c(0:3),q_range=c(0:3),r=2,pred_t=150,blur.out=c(3,3))
```

By setting the parameter 'sar,sma' to different logical values, we control the procedure whether to include generalized AR or MA components. For example, we avoid the generalized MA components to be included in the following code.

```
result = MS(X=data.ts.garma[1:850],U=100,p_range=c(0:3),q_range=c(0:3),r=2,sma=F)
```

We can also change the selecting criterion by setting 'crit' to a different IC. There are three options for this parameter: 'AIC', 'BIC', and 'BC'. The default is 'BC'.

```
result = MS(X=data.ts.garma[1:850],U=100,p_range=c(0:3),q_range=c(0:3),r=2,crit='BC')
```

In the end, one can shut down the parallel computing function by setting 'multicore=F'. The procedure will then perform fitting on each generalized ARMA model one by one. In most of the situation, it will raise the computational cost thus not recommended.

```
result = MS(X=data.ts.garma[1:850],U=100,p_range=c(0:3),q_range=c(0:3),r=2,multicore=F)
```