# rego

Automatic Time Series Forecasting

CHANNELATTRIBUTION

Davide Altomare and David Loris

2022-05-26

***rego*** *is a machine learning algorithm for predicting and imputing time series. It can automatically set all the parameters needed, thus in the minimal configuration it only requires the target variable and the regressors if present. It can address large problems with hundreds or thousands of regressors and problems in which the number of regressors is greater than the number of observations. Moreover it can be used not only with time series but also with any other real valued target variable. The algorithm implemented includes a bayesian stochastic search methodology for model selection and a robust estimation based on bootstrapping.* ***rego*** *is fast because all the code is C++.*

***rego*** *is a side project of ChannelAttribution: a popular machine learning library containing algorithms for marketing attribution with more than 200.000 downloads.*

channelattribution.io

## 1 Algorithm

**rego** adapts the bayesian stochastich search algorithm in ([1]) for choosing the best SARIMAX model that fits your Data.

SARIMAX model is a complete linear regression model because it includes exogenous variables, variables built considering past values of the target variable, seasonalities and variables built considering past forecasting errors. A compact way to express a general SARIMAX model is:

$$\begin{aligned}&\Phi(\mathbf{B})_p\Phi(\mathbf{B}^m)_P\Delta^d(\Delta_m)^D y_y = \beta X + \Theta(\mathbf{B})_q + \Theta(\mathbf{B}^m)_Q\varepsilon_t \\ &t = 1,\dots,T\end{aligned} \tag{1}$$

where

$$\begin{aligned} y_t &\text{ is the target variable at time } t \\ X_t &\text{ is a } K \times 1 \text{ vector of regressors at time } t \\ \varepsilon_t &\sim \mathcal{N}(0,\sigma^2) \\ t =& 1,\dots,T \\ \beta &\text{ is a } 1 \times K \text{ vector of regression parameters}\end{aligned} \tag{2}$$

and operators $\Phi(\mathbf{B})_p$, $\Delta^d$, $\Theta(\mathbf{B})_q$, $\Phi(\mathbf{B}^m)_P$, $(\Delta_m)^D$ and $\Theta(\mathbf{B}^m)_Q$ are well explained in [[3]].
In a more synthetic way it is also indicated with:

$$\text{SARIMAX}(p,d,q)(P,D,Q)_m \tag{3}$$

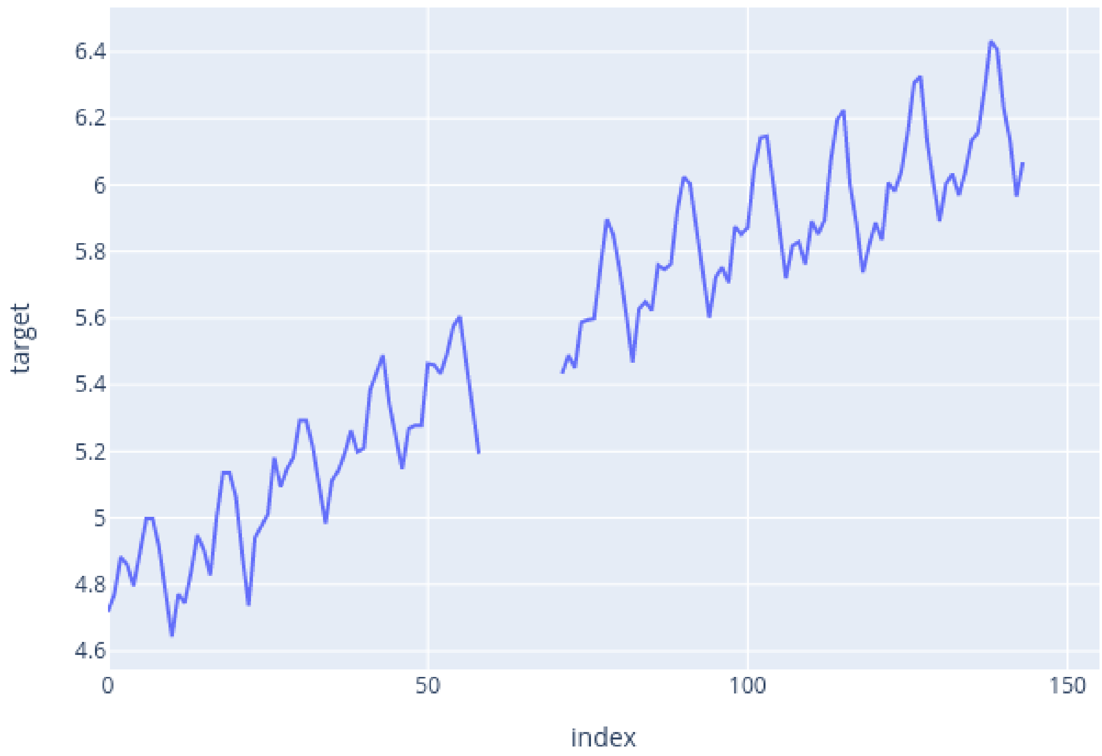


Figure 1:

A SARIMAX has $K + p + q + P + Q$ parameters. It is not difficult for a SARIMAX model to reach a high number of parameters. This is quite problematic because an high number of parameters could led to overfitting if the number of the observations are not high enough. Thus choosing the best parsimonius model that fits the data is crucial for having good predictions and realistic confidence intervals. **rego** avoids overfitting caused by high number of parameters thanks to its bayesian stochastic search algorithm oriented to sparsity, which can be thought as a bayesian alternative to LASSO.

Moreover choosing the best regression model between all the possible regression models we could fit for our data is not easy. We have seen that SARIMAX has $K+p+q+P+Q$ parameters and it means that we have:

$$\sum_{j=0}^{K+p+q+P+Q} \binom{K+p+q+P+Q}{j} \tag{4}$$

possible SARIMAX models that could be built for a generic usecase. For example if $K+p+q+P+Q$ is equal to 30, our space of models has 1,073,741,824 possible SARIMAX. It could be not feasible to evaluate all the models because it would require too much computational time. Thus an efficient search algorithm is needed for finding the best model without evaluating all the models belonging to the space of models. With **rego** a problem with $K+p+q+P+Q$ equal to 1000 can be solved in few seconds using a normal pc. If you want to have details about the algorithm implemented in **rego** please refer to [[1]] and [[2]]

## 1.1 Example 1: Seasonal data

**rego** is a general forecasting algorithm that lets you to predict your real valued target variable and/or estimate missing past values of it through the function **regpred**. You only need to feed **regpred** with a DataFrame containing your target variable located at the first column and your exogenous variables in the other ones. In the following example we will consider a seasonal target variable and we will use **regpred** to estimate some missing values in the middle of the time series and to make predictions after the last available observation. First we have to load libraries and data. Time series is shown in Figure 1

```
#Python

import pandas as pd
from rego import *
import plotly.express as px

Data=pd.read_csv("https://channelattribution.io/csv/Data_air.csv",sep=";",header=None)
Data.columns=["target"]
fig = px.line(Data,y="target")
fig.show()
```

```
#R

library(rego)

Data=read.csv("https://channelattribution.io/csv/Data_air.csv",sep=";",header=FALSE)
colnames(Data)=c("target")

plot(Data$target,type="l",ylab="target")
```

As you can see Data contains only the target variable and the series has a strong seasonality pattern. Now we can apply **regpred** to our target.

```
#Python

res=regpred(Data)

pred=res["prediction"]
fig = px.line(pred)
fig.show()
```

```
#R

res=regpred(Data)

predictions=res$prediction
plot(predictions$real,type="l",ylim=c(min(predictions,na.rm=TRUE)*0.95,max(predictions,na.rm=TRUE)*1.05))
lines(predictions$predicted,col="blue")
lines(predictions$lower_bound,col="red")
lines(predictions$upper_bound,col="red")
```

Input parameters of **regpred** are explained in Table 1.

Output of **regpred** contains three DataFrames with final, forward and backward predictions. Each DataFrame contains the columns described in Table 2.

Figure 2 gives a graphical representation of the output.

## 1.2 Example 2: High dimensional regression

Now we consider a regression problem with one target and 999 regressors each of them with 1000 observations.

The data has been generated using the following formula

$$\begin{aligned} y_t &= x_{175,t} + x_{542,t} + x_{815,t} + \varepsilon_t \quad \varepsilon_t \sim \mathcal{N}(0,1) \quad \forall t = 1,\dots,1000 \\ x_{k,t} &\sim \mathcal{N}(0,1) \quad \forall k = 1,\dots,999 \quad \forall t = 1,\dots,1000 \end{aligned} \tag{5}$$

*Data* is a DataFrame containing the target variable at first position and the 999 regressors at position from second to last column. First we load and visualize target:

```
#Python

import pandas as pd
from rego import *
import plotly.express as px

Data=pd.read_csv("https://channelattribution.io/csv/Data_sim_1000.csv",sep=",",header=None)
fig = px.line(Data.iloc[:,0]  )
fig.show()
```

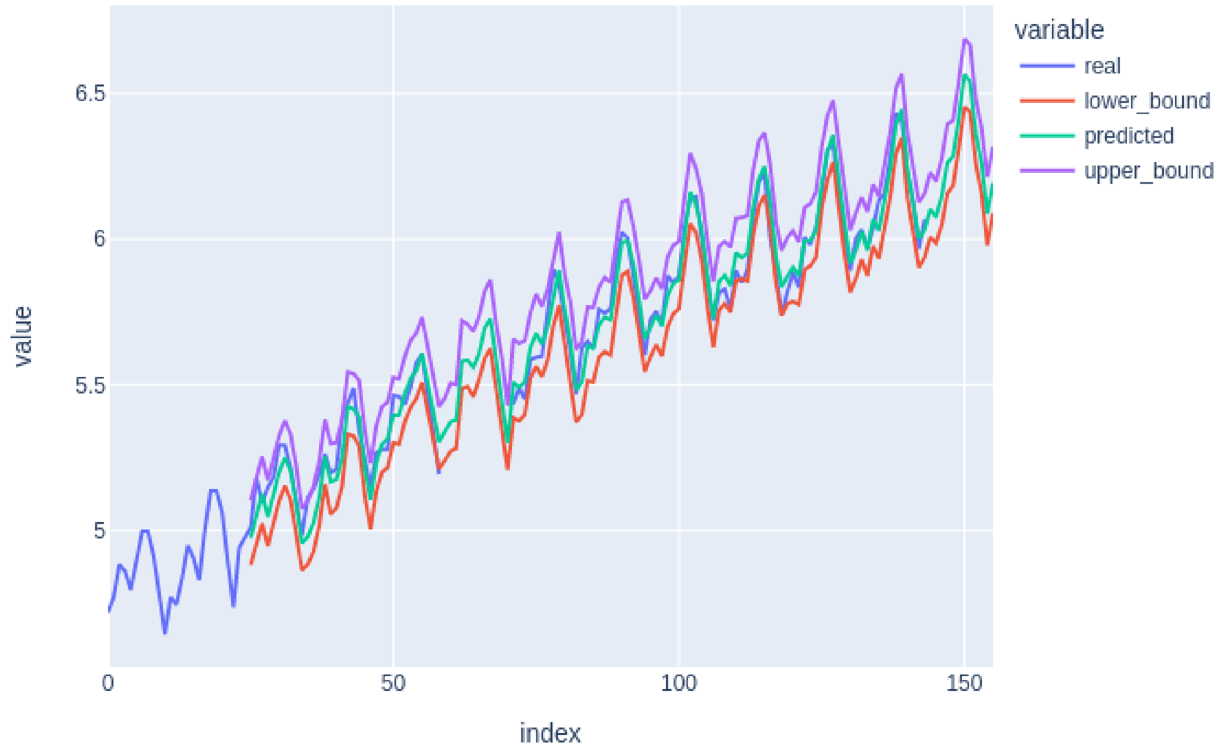
variable
real
lower_bound
predicted
upper_bound
value
6.5
6
5.5
5
0
50
100
150
index

Figure 2:

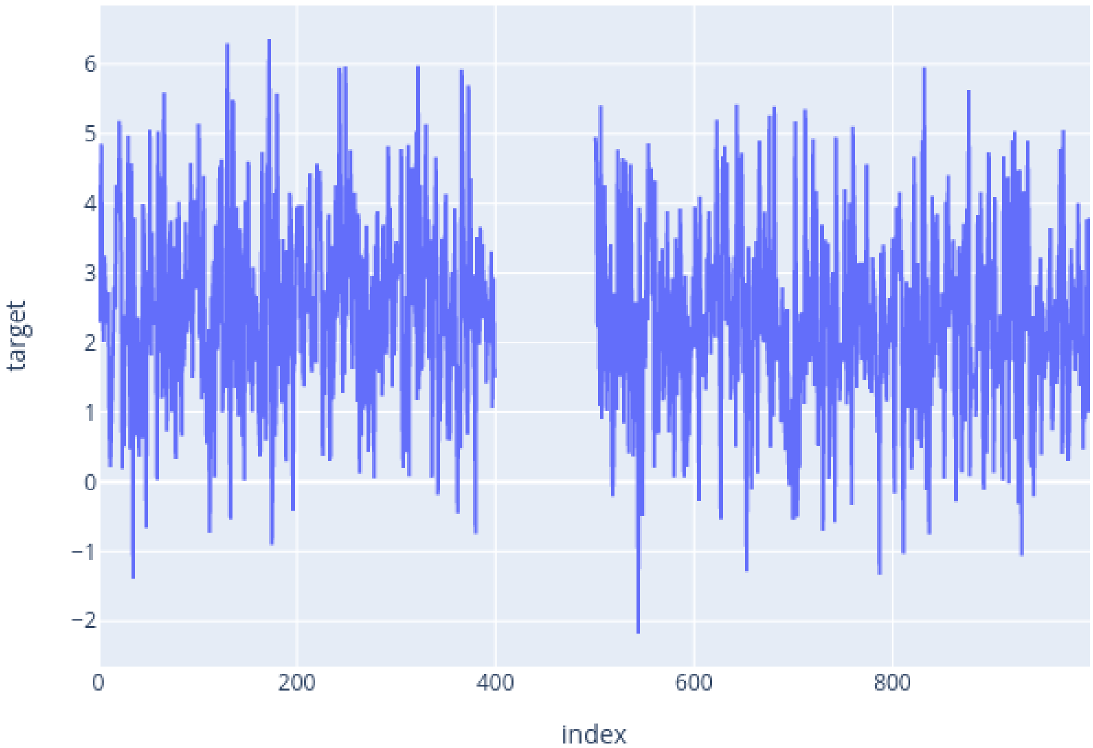


Figure 3:

Table 1: Input parameters

| | |
|---|---|
| **Data** | data.frame containing target variable at first column and regressors if present from second to last column |
| **from_lag** | minimum time lag to be considered in the autoregressive moving average part of the algorithm |
| **max_lag** | maximum time lag to be considered in the autoregressive moving average part of the algorithm. If "auto" then the algorithm will set a suitable value. Set to 0 or NULL if you want to remove the autoregressive moving average part as in case of non time series data |
| **alpha** | significance level for the confidence interval produced around predictions. If 0.05 then the algorithm will calculate a 95% confidence interval around predictions |
| **nsim** | number of bootstrap replications used for producing confidence interval around predictions |
| **flg_print** | if 1 some information during the evaluation will be printed |
| **direction** | if "->" then only a forward prediction will be executed, if "<-" then only a backward prediction will be executed, if "<->" then both a forward than a backward prediction will be executed if possible. For imputing missing values is convenient to leave default "<->" |
| **flg_const** | if 1 then a constant is included into the model |
| **flg_diff** | if 1 and no regressor is present then if the target variable exhibits a trend, it is one-step differentiated up to two times |
| **model** | estimated models from a previous train to be used in new data prediction without retraining |

Table 2: Output: predictions

| | |
|---|---|
| **real** | observed values for the target variable |
| **lower bound** | lower bound of the estimated confidence interval for predictions |
| **predicted** | point-wise prediction for target variable |
| **upper bound** | upper bound of the estimated confidence interval for predictions |

```
#R

library(rego)

Data=read.csv("https://channelattribution.io/csv/Data_sim_1000.csv",sep=",",header=FALSE)
colnames(Data)=c("target",paste0("X",rep(1:999)))

plot(Data$target,type="l",ylab="target")
```

Figure 3 shows the target. Now we can apply *rego* to your Data.

```
#Python

res=regpred(Data)

pred=res["prediction"]
fig = px.line(pred)
fig.show()
```

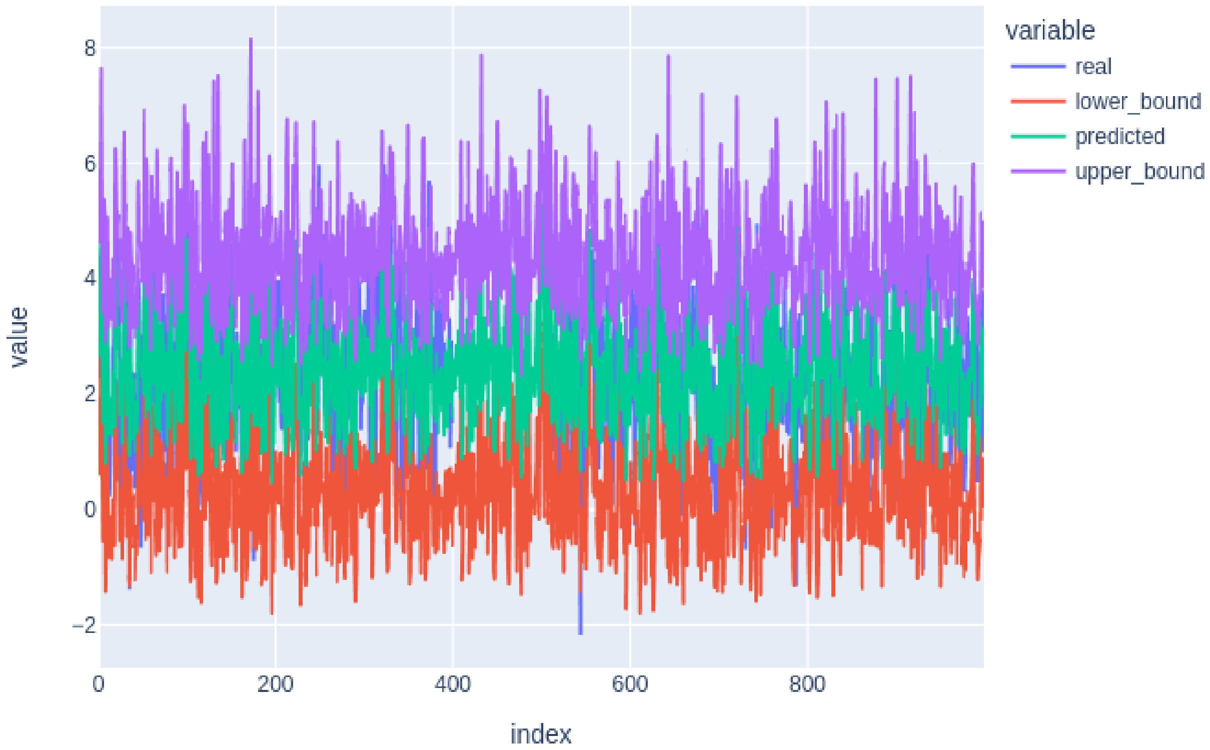


Figure 4:

```
#R

res=regpred(Data)

predictions=res$prediction
plot(predictions$real,type="l",ylim=c(min(predictions,na.rm=TRUE)*0.95,max(predictions,na.rm=TRUE)*1.05))
lines(predictions$predicted,col="blue")
lines(predictions$lower_bound,col="red")
lines(predictions$upper_bound,col="red")
```

Figure 4 gives a graphical representation of the output.

## References

[1] Altomare,Consonni,La Rocca (2011), Objective Bayesian Search of Gaussian DAG Models with Non-local Priors.

[2] Berger, J. and Molina, G. (2005).Posterior model probabilities via pathbased pairwise priors. Statistica Neerlandica 59, 3–15.

[3] A complete introduction to time series analysis with r sarima models, medium.com