

Chapter 2

Mathematical Methods Applied for Hydro-meteorological Time Series Modeling

This chapter contains an overview of some methods that can be used for modeling the evolution of hydro-meteorological time series, methods employed in the next chapters. We distinguish among them: decomposition methods, Box–Jenkins methods and artificial intelligence—based methods (GEP, AdaGEP, GRNN and SVM).

For clarity sake, we remember some basic notions.

A time series is a set of successive observations of a phenomenon during a period [2]. A model is a formalized presentation of a phenomenon using an equation or a set of equations with the aim of understanding and explaining its evolution [14]. Time series modeling is the ensemble of techniques used for identifying a mathematical model representing a time series [29].

If (X_t) is a sequence of random variables and (x_t) is a sequence of the realizations of (X_t) , building a time series model for (x_t) means the detection of the joint distribution of (X_t) [16].

In the following the notion of time series will be used both in the sense of [2] and [16].

Modeling and predicting nature phenomena and processes is a difficult task due to their variability that is a consequence of their relations with other phenomena and to the influences of unpredictable factors. Hydrological series are not an exception from this rule, their evolution being usually described by stochastic processes.

Two big groups of models for time series evolution can be determined: the classical one, containing the deterministic models, and the modern ones, composed of the models issued from the artificial intelligence, as evolutionary computing (EC) and artificial neural networks (ANNs) [32].

From other viewpoint, the modeling methods could be classified as parametric, nonparametric and semiparametric. Usually, in the parametric approach there are restrictive assumptions on the data, while, in the nonparametric one, the hypotheses on the input data are relaxed. Recent studies provided good results of modeling hydro-meteorological time series using ANNs and Gene Expression Programming (GEP) [6, 7, 23, 31, 34, 35]. Support Vector Regression (SVR), the adaptation of Support Vector Machines (SVM) for solving classification and regression

problems, proved to be a good alternative to the classical methods, with applications in different domains [12, 18, 45, 62]. Also, AdaGEP, a version of GEP, was successfully used for modeling hydrological time series, as an alternative to the Box–Jenkins methods [7, 8] or together with them, in hybrid models.

Due to the temporal and spatial variability of the hydro-meteorological time series, the use of only one method is not appropriate. Therefore, different techniques are employed; they are shortly presented in this chapter.

1 Types of Models. Classical Decomposition Method

The time series, in general, and the hydrological ones, in particular, may present trend and seasonality and are influenced by random variations. The existence of such components is differently considered, depending on the type of models.

A classification of the models is provided here [4]:

- Adjustment models, written as:

$$y_t = f(t, u_t),$$

where: f is an adjustment function, t is the time and u_t is a centered random variable.

The adjustment can be deterministic or random, as all components involved in the model are deterministic, or there is at least a random component.

The most common models with random adjustment are:

- (a) the additive model, whose equation is:

$$y_t = Y_t + S_t + \varepsilon_t, \quad (1)$$

where: Y_t is the trend, S_t is the seasonal compound and ε_t is the random variable.

Taking into account the seasonality presence, (1) can be written as:

$$y_{ij} = Y_{ij} + S_j + \varepsilon_{ij}, \quad (2)$$

where: $i = \overline{1, p}$ is the period number, $j = \overline{1, m}$ is the number of the sub-period, Y_{ij} is the trend, S_j is the seasonality index and ε_{ij} is the random variable.

- (b) the multiplicative model, whose equation is:

$$y_{ij} = Y_{ij} \cdot S_j^* \cdot \varepsilon_{ij}^*, \quad (3)$$

where: Y_{ij} is the trend, S_j^* is the seasonality index and ε_{ij}^* is the random variable.

- The autoprojective models, of the type:

$$y_t = f(y_{t-1}, y_{t-2}, \dots, \varepsilon_t),$$

where ε_t is a random variable.

A special class of this type is formed by ARIMA models that will be presented in the next section.

- Explicative models, of the type:

$$y_t = f(X_t, u_t),$$

where X_t is an exogenous deterministic or random variable and u_t is an exogenous random variable.

In this case, X_t and u_t have some properties of independence or are not correlated.

Suppose that, for the moment, we analyze the additive model (2). Detecting each component is done by the classical method that supposes the steps:

1. Determining the trend, Y_{ij} , by the moving average method or an analytical method;
2. Determining the seasonal component as follows:

- (a) Determining the differences:

$$y_{ij} - Y_{ij} = S_j + \varepsilon_{ij}. \quad (4)$$

- (b) Determining the brute estimators of the seasonal component, s'_j , as averages of the differences (4) corresponding to the same season. If the sum of the brute estimators is not zero, pass to the step (c). If it is zero, pass to (d).
- (c) Determining the seasonal component, s_j , by subtracting the average of the brute estimators of the seasonal components from the brute estimator.
- (d) Supposing that $s_j = S_j$, determine the values of the random variable by:

$$\varepsilon_{ij} = y_{ij} - Y_{ij} - S_j. \quad (5)$$

For the multiplicative model (3), the decomposition is done by an analogous algorithm, whose steps are:

1. Determining the trend, Y_{ij} ;
2. Determining the seasonal component as follows:

- (a) Determining the ratio:

$$y_{ij}/Y_{ij} = S_j^* \cdot \varepsilon_{ij}^*, \quad (6)$$

- (b) Determining the brute estimators of the seasonal component, s''_j , as averages of the ratios (6) corresponding to the same season. If the product of the brute estimators is not one, pass to the step c. If it is one, pass to d.

- (c) Determining the seasonal component, s_j^* , by dividing the brute estimators by the average of the brute estimators of the seasonal components;
- (d) Supposing that $s_j^* = S_j^*$, determine the values of the random variable by:

$$\varepsilon_{ij}^* = y_{ij} / (Y_{ij} \cdot S_j^*).$$

To perform such a decomposition, ‘decompose()’ function in R may be used. It estimates the components of a time series that is described by an additive model.

The function has the following arguments:

- x—the time series that will be decomposed;
- type—type of model—“additive” or “multiplicative”; the default is the additive one;
- filter—a filter coefficient—if “NULL”, a moving average fit is performed.

The following sequence of code must be written for the decomposition of Constanta monthly series, by the multiplicative model (Fig. 1).

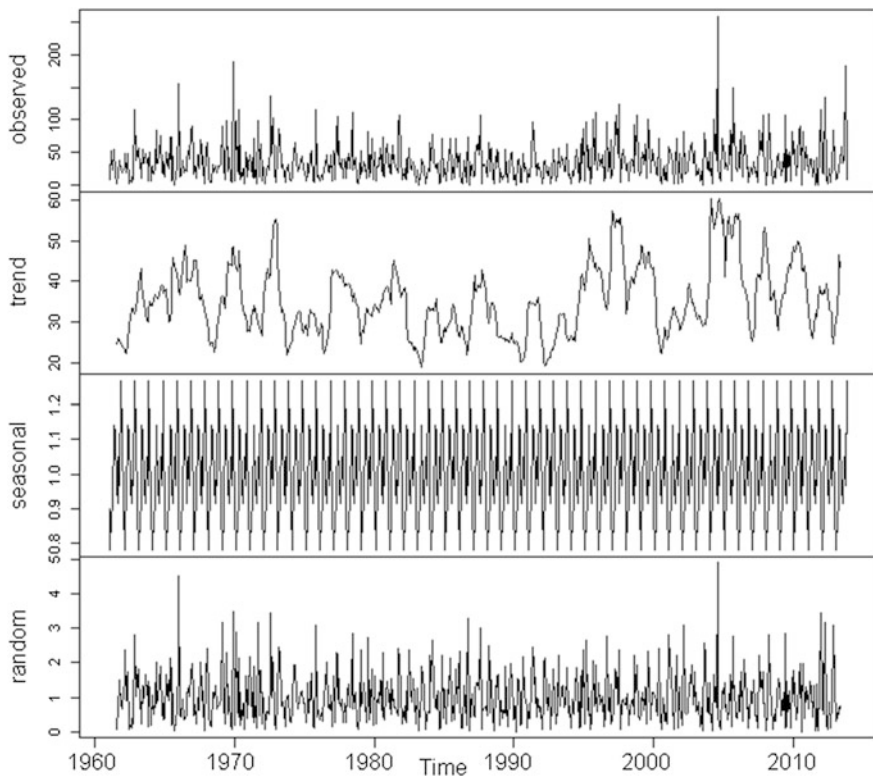


Fig. 1 Decomposition of Constanta monthly series

```

data<-read.csv("D:\\Lucrari_2.12.14\\2015_Carte\\Cta_lunar_1961_2013.csv",sep=",",
header=TRUE)
x <- data[,1]
library(stats)
x<-ts(x, start=c(1961,1), end=c(2013,12), frequency =12)
m <- decompose(x,type="multiplicative")

[1] 0.9005163 0.7813944 0.8740163 0.9073708 1.1396821 1.1196095 0.9607246
[8] 0.9139141 1.0496215 0.9650946 1.2695325 1.1185233
# gives seasonal figures

```

2 Box-Jenkins Approach and Stationarity Tests

2.1 Box-Jenkins Approach

Box and Jenkins proposed in 1970 [15] a general approach for modeling and predicting univariate time series, based on the notion of ARMA process, that can be successfully used for modeling series that don't present very high variability. In the following we summarize the basic notions [16].

The process (X_t) is said to be *weakly stationary* (stationary, for short) if it has a constant mean and its covariance depends only on the lag between two points in the series.

A process (X_t) is called *white noise* if (X_t) is uncorrelated, identically distributed, with zero mean, and a constant variance.

A discrete process (X_t) is called of AR(p) type (*autoregressive of p order*) if it can be described by the equation:

$$X_t = c + \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t, \quad \varphi_p \neq 0, \quad (7)$$

where c is a constant, $\varphi_i, i = \overline{1, p}$ are parameters and (ε_t) is a white noise.

It was proved that a sufficient condition of stationarity of an AR(p) process is that the absolute values of all the roots of the characteristic equation:

$$z^p - \sum_{i=1}^p \varphi_i z^{p-i} = 0, \quad (8)$$

are less than 1.

Particularly, if $|\varphi_1| > 1$, the AR(1) process is not stationary.

A discrete process (X_t) is called of MA(q) type (*moving average of q order*) if it has the equation:

$$X_t = \mu + \varepsilon_t - \sum_{i=1}^q \theta_i \varepsilon_{t-i}, \quad \theta_q \neq 0, \quad (9)$$

where μ is a constant, $\theta_i, i = \overline{1, q}$ are parameters and (ε_t) is a white noise.

It was proved that a sufficient condition for the stationarity of an MA(q) process is that the absolute values of the roots of the characteristic equation

$$z^q - \sum_{i=1}^p \theta_i z^{q-i} = 0 \quad (10)$$

are less than 1.

If (X_t) is a discrete process, the *partial autocorrelation function* (PACF) is defined by:

$$\tau(h) = \frac{\text{Cov}(X_t - X_t^*, X_{t-h} - X_{t-h}^*)}{[D^2(X_t - X_t^*)D^2(X_{t-h} - X_{t-h}^*)]^{1/2}}, \quad t > h \geq 0,$$

where $X_t^*(X_{t-h}^*)$ is the affine regression of $X_t(X_{t-h})$ on $X_{t-1}, \dots, X_{t-h+1}$.

The autocorrelation and the partial autocorrelation functions can be used for the selection of the model's type. The partial autocorrelation function of an AR(p) model is zero for the lag h greater than p , and its ACF is a sum of decaying exponentials or a damping sine wave. The autocorrelation function of an MA(q) model is zero for the lag h greater than q .

A process (X_t) is called of ARMA(p, q) type (*autoregressive moving average process of autoregressive order p and moving average order q*) if it can be described by the equation:

$$X_t - \sum_{i=1}^p \varphi_i X_{t-i} = \theta_0 + \varepsilon_t - \sum_{i=1}^q \theta_i \varepsilon_{t-i}, \quad \varphi_p \neq 0, \quad \theta_q \neq 0, \quad (11)$$

where θ_0 is a constant, $\varphi_i, i = \overline{1, p}, \theta_i, i = \overline{1, q}$ are parameters and (ε_t) is a white noise.

If B is the backward operator, defined by:

$$B^k(X_t) = X_{t-k}, \quad t > k, t, k \in \mathbf{N}^*,$$

and:

$$\begin{aligned} \Phi(z) &= 1 - \varphi_1 z - \varphi_2 z^2 - \dots - \varphi_p z^p, \quad \varphi_p \neq 0, \\ \Theta(z) &= 1 - \theta_1 z - \theta_2 z^2 - \dots - \theta_q z^q, \quad \theta_q \neq 0, \end{aligned}$$

the relations (7), (9), (11) can be written respectively as:

$$\begin{aligned} \Phi(B)X_t &= c + \varepsilon_t, \\ X_t &= \mu + \Theta(B)\varepsilon_t, \\ \Phi(B)X_t &= \theta_0 + \Theta(B)\varepsilon_t. \end{aligned}$$

AR(p) and MA(q) are particular cases of ARMA(p, q) processes: AR(p) = ARMA(p, 0), MA(q) = ARMA(0, q).

Preliminary estimation of the parameters in AR models can be done by Yule-Walker and Burg [17] procedures, and in ARMA models, by the innovation and Hannan and Rissanen [30] algorithms.

For selecting the best ARMA(p, q) model that fit the data, the Akaike [1] and Schwarz [57] criteria are used. These are defined by:

$$AIC(p, q) = \ln \hat{\sigma}_{p,q}^2 + 2(p + q)/n,$$

$$SCH(p, q) = \ln \hat{\sigma}_{p,q}^2 + (p + q) \ln n/n,$$

$\hat{\sigma}_{p,q}^2$ being the maximum likelihood estimation of the errors' variance and n , the volume of the sample used to fit the model.

The best ARMA(p, q) model is that with the smallest AIC (or SCH) value.

A process (X_t) is called of ARIMA(p, d, q) type (*autoregressive integrated moving average process*) if it can be described by the equation:

$$\Phi(B)(1 - B)^d X_t = \Theta(B)\varepsilon_t,$$

where $d \in \mathbb{N}^*$ is the degree of differentiation, (ε_t) is a white noise and $\Phi(z) \neq 0$, for all z with $|z| \leq 1$.

A necessary and sufficient condition for the stationarity of an ARIMA(p, d, q) process is $d = 0$, in which case this process reduces to an ARMA(p, q) process.

A discrete process is said to be a FARIMA(p, d, q) (*fractionally integrated ARMA*) process if:

$$(1 - B)^d \Phi(B)X_t = \Theta(B)\varepsilon_t,$$

where $\Phi(z) \neq 0, \Theta(z) \neq 0$ for all z with $|z| \leq 1$, (ε_t) is a white noise, d is the memory parameter, $0 < |d| < 0.5$ and

$$(1 - B)^d = \sum_{j=0}^{\infty} \pi_j B^j,$$

with

$$\pi_j = \prod_{1 < k \leq j} \frac{k - 1 - d}{k}, \quad j = 1, 2, \dots \text{ and } \pi_0 = 1.$$

For an extensive study of the Box-Jenkins approach, see [15].

The R stats package [40] can be used to simulate an ARIMA model. Here are some examples:

```
library(stats)
ts.sim<-arima.sim(n=40, list(ar=c(0.451, -0.485), ma=c(-0.879, 0.748)), sd=sqrt(0.234))
#generates 40 values of an ARMA(2,2) model, without constant term, whose autoregressive
# and moving average coefficients are 0.451 and -0.485, respectively 0.879 and 0.748
ts.sim #print the simulated values
```

Time Series:

Start = 1

End = 40

Frequency = 1

```
[1] 0.45851204 0.29764486 -0.48782867 1.00328887 0.23271658 -0.02816671
[7] 0.22655481 0.19091807 -0.35784716 -0.88659972 1.59331365 -0.45864452
[13] -0.36238013 1.40164581 -0.54335712 0.38079291 0.34834893 -0.69216474
[19] 0.41504097 0.22675742 -0.20430218 -0.17780706 0.14819071 0.60619994
[25] -0.65479046 -0.49896103 -0.04767047 -0.19592509 -0.98984893 0.73034419
[31] -0.58868968 0.41709830 0.52431536 0.14624364 -0.67246452 0.73284864
[37] -0.10253791 0.23828534 0.69601486 -0.32475338
```

`ts.plot(ts.sim)` #plots the generated data series (Fig. 2)

`ts.sim <- arima.sim(list(order=c(2, 1, 0)), ar= c(0.2, 0.45)), n=20)`

#generates an ARIMA(2,1,0) model

`ts.sim`

Time Series:

Start = 1

End = 21

Frequency = 1

```
[1] 0.0000000 0.5968887 2.0204269 2.2838219 2.8037469 3.2241359
[7] 3.5735482 4.0904095 4.5394847 5.0389145 5.8879888 5.2578066
[13] 3.7503221 4.1217022 2.7240303 3.0765823 1.3698995 2.0259479
[19] 0.2043944 0.9807588 -0.8276302
```

`ts.plot(ts.sim)` (Fig. 3)

`ts.sim <- arima.sim(list(order = c(0,0,0)), n = 50) # generates a white noise`

the same can be done by using `'rnorm(50)'` (Fig. 4)

Fig. 2 Simulation of an ARMA(2, 2) model

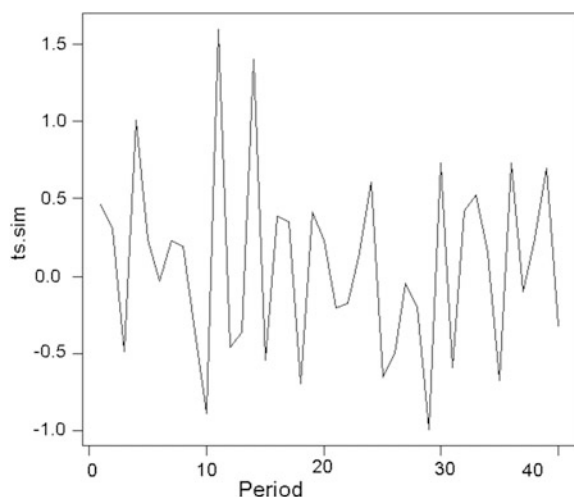


Fig. 3 Simulation of an ARIMA(2, 1, 0) model

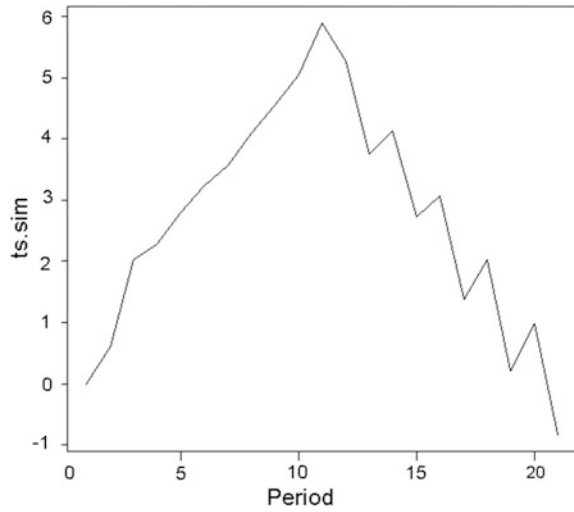
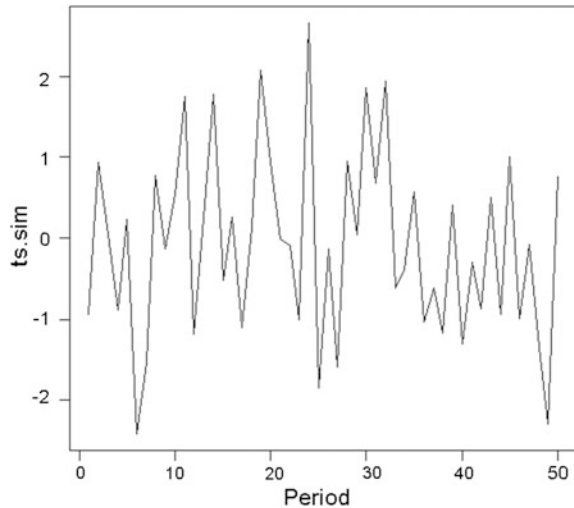


Fig. 4 Simulation of a white noise



Different ways can be followed for fitting an ARIMA model. The first one is to choose own model(s) using `Arima()` function from the R package **forecast** [36]. Among the models that fit the data, the selected one is that with the smallest AIC.

Another way is to utilize `auto.arima()` [43] which is based on the iterative algorithm introduced by Hyndman and Khandakar [44] and that combines the KPSS stationarity test (for determination of differences order d) and AIC computation (for choosing the parameters p and q for the differentiated series). If $d = 0$, a constant is also included in the model.

We exemplify here the use of `auto.arima()` on Constanta daily precipitation series recorded over a period of 10 years.

```
library(forecast)
data<-read.csv("D:\\Lucrari_2.12.14\\2015_Carte\\Cta_zilnice_1961_2009.csv", sep=";",
header=TRUE)
y<-data[1 :3850,1] #selects only the first 3850 data on the first column of the sheet
fit <- auto.arima(y) #fit the ARIMA model
plot(forecast(fit, h = 120)) #plots the forecast for the next 120 days based on the model
# (Fig. 5)
fit
```

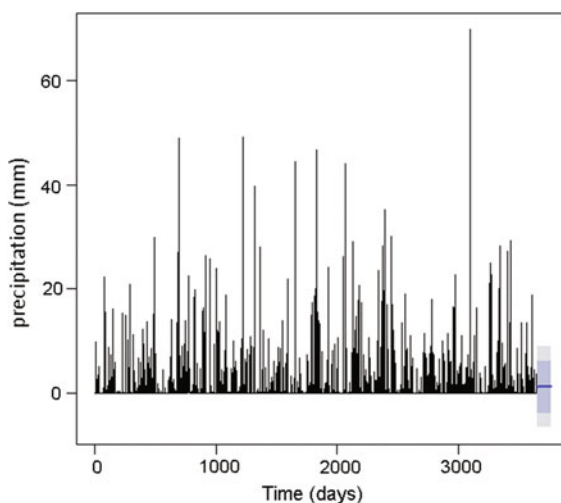
```
Series: y
ARIMA(0,0,1) with non-zero mean
```

```
Coefficients:
      ma1      intercept
      0.1825      1.1499
s.e.    0.0162      0.0766
```

```
sigma^2 estimated as 15.31: log likelihood=-10158.42
AIC=20322.85    BIC=20341.46 #BIC is the Bayesian Information Criteria
```

Remark The model is not satisfactory, so other methods must be employed.

Fig. 5 MA(1) model for the daily precipitation registered a Constanta (1961–1970)



2.2 Stationarity Tests

Different statistical tests have been proposed for checking the hypothesis of time series stationarity. Some of them can be applied without knowledge on the model designed for the data series; others are used knowing the autoregressive models fitted on the data series, for testing the existence of a unit root. All can be grouped in two categories:

- Tests whose null hypothesis is that the series is non-stationary, as: the DF (Dickey and Fuller) [22, 25], ADF (Augmented Dickey–Fuller), PP (Phillips–Perron) [54] and SP tests (Schmidt and Phillips) [56] tests;
- Tests whose null hypothesis is that the series is stationary, as the KPSS (the Kwiatkowski, Phillips, Schmidt and Shin) test [47].

Generally, a time series (X_t) may have a stochastic trend (unit root) (TS_t), a deterministic trend and/or a cyclical component (C_t), such that it could be written as:

$$X_t = TD_t + TS_t + C_t. \quad (12)$$

Therefore, for testing the hypothesis $TS_t \neq 0$, the first kind of tests is used, and for checking $TS_t = 0$, the second one is employed.

In the Dickey–Fuller test, the following models are investigated: (1) autoregressive of first order: (2) autoregressive of first order, with an average different from zero, (3) autoregressive with trend. They are described respectively by the equations:

$$\Delta X_t = \varphi X_{t-1} + \varepsilon_t, \quad (13)$$

$$\Delta X_t = \varphi X_{t-1} + \beta + \varepsilon_t, \quad (14)$$

$$\Delta X_t = \varphi X_{t-1} + \beta + \gamma t + \varepsilon_t, \quad (15)$$

where ε_t is an independent random variable and Δ is the backshift operator.

Therefore, the null hypothesis $H_0: \varphi = 0$ (or, equivalent, the existence of a unit root) is tested against the stationarity one, $H_1: \varphi < 0$.

The null hypothesis is rejected if the p -value calculated for the test is less than the established significance level (0.05, if no other specification is done) [25, 53].

ADF is a generalization of DF test, which relies on the model:

$$\Delta X_t = \varphi X_{t-1} + \sum_{j=1}^{p-1} \alpha_j \Delta X_{t-j} + \beta' D_t + \varepsilon_t,$$

where: ε_t is a white noise and D_t is a deterministic trend. The null hypothesis is, again, the existence of a unit root ($\varphi = 0$).

The test statistic is the usual t test. A value of this statistics greater than the critical value (from the table of the Student t statistics) corresponding to the significance level indicates that the null hypothesis cannot be rejected.

In the case of PP test, the series' heteroskedasticity is also considered, so that the series model is:

$$\Delta X_t = \phi X_{t-1} + \beta' D_t + \varepsilon_t,$$

where ε_t is stationary and may be heteroskedastic.

The test statistics is a modified version of the ADF test statistics, which corrects the autocorrelation or heteroskedasticity of ε_t [39].

Belonging to the second group of tests, KPSS tests the null hypothesis that the study series is stationary in mean or in trend (linear deterministic).

The series' model is:

$$X_t = \beta' D_t + r_t + u_t, \quad r_t = r_{t-1} + \varepsilon_t,$$

where D_t is the deterministic component (constant or linear trend), r_t is a random walk, ε_t is stationary, with the variance σ_ε^2 , u_t is stationary, but may be heteroskedastic.

The null hypothesis is equivalent with: $H_0 : \sigma_\varepsilon^2 = 0$ and the test statistics used to test it is [39, 53]:

$$KPSS = \left(T^{-2} \sum_{t=1}^T \hat{S}_t^2 \right) / \hat{\lambda}^2,$$

where $\hat{S}_t = \sum_{j=1}^t \hat{u}_j$, \hat{u}_j is the residual in the regression model of X_t on D_t and $\hat{\lambda}^2$ is a consistent estimate of the variance parameter:

$$\lambda^2 = \lim_{T \rightarrow \infty} \sum_{t=1}^T E[T^{-1} S_T^2].$$

The null hypothesis is rejected at a significance level if the test statistic is greater than the critical value.

For a deeper insight on these tests, the reader may refer to [53].

In [21] some of these tests have been applied to analyze the stationarity of precipitation series from Benin. Here, we present the code in R for performing some tests and the results of their application on Constanta daily series (1961–2009).

```
library(tseries)
data<-read.csv("D:\\Lucrari_2.12.14\\2015_Carte\\Cta_zilnice_1961_2009.csv", sep="," ,
header=TRUE)
y<-data[,1]
adf.test(y)
```

Augmented Dickey-Fuller Test

data: y

Dickey-Fuller = -8.7952, Lag order = 8, p-value = 0.01

alternative hypothesis: stationary

Warning message:

In adf.test(y): p-value smaller than printed p-value

```
PP.test(y)
```

Phillips-Perron Unit Root Test

data: y

Dickey-Fuller = -24.8673, Truncation lag parameter = 6, p-value = 0.01

```
kpss.test(y, "Level")
```

KPSS Test for Level Stationarity

data: y

KPSS Level = 0.873, Truncation lag parameter = 5, p-value = 0.05952

```
kpss.test(y, "Trend")
```

KPSS Test for Trend Stationarity

data: y

KPSS Trend = 0.1382, Truncation lag parameter = 5, p-value = 0.06437.

The results of ADF and PP tests prove that the hypothesis that the series has a unit root can be rejected. KPSS test could not reject the hypothesis that the series is stationary in level and trend.

The same tests are implemented in the packages **fUnitRoots** [37] and **urca** [38]. In the following we present some examples of the use of these tests.

```
library(fUnitRoots)
adfTest(y) # computes test statistics and p values along the implementation of Trapletti
# [37] for unit roots
```

Title:

Augmented Dickey-Fuller Test

Test Results:

PARAMETER:
 Lag Order: 1
 STATISTIC:
 Dickey-Fuller: -7.4657
 P VALUE:
 0.01

Warning message:
 In adfTest(y) : p-value smaller than printed p-value

*unitrootTest(y) # computes test statistics and p values along the implementation of
 # McKinnon [50] for unit roots*

Title:
 Augmented Dickey-Fuller Test

Test Results:

PARAMETER:
 Lag Order: 1
 STATISTIC:
 DF: -7.4657
 P VALUE:
 t: 1.737e-12
 n: 0.05854

```
library(urca)
data<-read.csv("D:\\Lucrari_2.12.14\\2015_Carte\\Cta_zilnice_1961_2009.csv", sep= ",",
header=TRUE)
y<-data[,1]
z <- ur.pp(y, type="Z-tau", model="trend", lags="short") # performs the PP test
summary(z)
```

Test regression with intercept and trend

Call:
 lm(formula = y ~ y.l1 + trend) # builds the regression equation of y

Residuals:

Min	1Q	Median	3Q	Max
-38.785	-19.877	-6.802	12.143	221.550

Coefficients:
Estimates the regression coefficients, the values of t statistics and the p-values

```

      Estimate   Std. Error   t value   Pr(>|t|)
(Intercept) 34.964872  1.833083  19.074   <2e-16 ***
y.l1         0.010005  0.039808   0.251   0.8016
trend        0.011200  0.006425   1.743   0.0818 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# y.l1 is not significant, the trend coefficient is significant at 0.01 and the trend is
# also significant.

Residual standard error: 29.6 on 632 degrees of freedom
Multiple R-squared:  0.005011, Adjusted R-squared:  -0.001862
# there is no correlation between the exogenous and endogenous variables
F-statistic: 1.591 on 2 and 632 DF,  p-value: 0.2044
# the model is not significant in its whole
Value of test-statistic, type: Z-tau  is: -24.8674

      aux. Z statistics
Z-tau-mu      19.3696
Z-tau-beta    1.7425

Critical values for Z statistics:
      1pct      5pct     10pct
critical values -3.977072 -3.419005 -3.131727
# The value of Z-tau is less than the critical values, so that the null hypothesis can
# be rejected.

x<-ur.sp(y, type="tau", pol.deg=1, signif=0.01) # performs the SP test
summary(x)

Call:
lm(formula = sp.data)

Residuals:
    Min     1Q   Median     3Q    Max
-38.785 -19.877  -6.802   12.143  221.550

Coefficients:
      Estimate   Std. Error   t value   Pr(>|t|)
(Intercept) 31.397725  2.672152  11.750   <2e-16 ***
y.lagged     0.010005  0.039808   0.251   0.8016
trend.exp1   0.011200  0.006425   1.743   0.0818 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
summary(x)
```

Residual standard error: 29.6 on 632 degrees of freedom
 Multiple R-squared: 0.005011, Adjusted R-squared: 0.001862
 F-statistic: 1.591 on 2 and 632 DF, p-value: 0.2044

Value of test-statistic is: -22.2254

Critical value for a significance level of 0.01 is: -3.58

The hypothesis that the series has a unit root is rejected.

3 Genetic Algorithms

Evolutionary Algorithms (EAs) are part of artificial intelligence techniques, inspired by the theory of natural evolution and selection of species. So, a population generates new populations of individuals and among them, the best individuals survive. In these algorithms, the individuals are candidate solutions of the problem at hand. The environment in which the individuals live is the search space. The evolution is produced by the application of different genetic operators, and the selection process is done by using the fitness function, based on the principle that the best individual in a population survives.

In biology, the genes of an organism encode the rules that describe how the organism is built up. At their turn, the genes are connected in chromosomes.

In Evolutionary Algorithms, the genotype defines the individuals' content and structure, and the phenotype refers to the behavior of an individual genotype.

Before running an evolutionary algorithm for solving a problem, the potential solutions must be encoded as bit strings, referred as chromosomes, in this context. The flowchart of an EA is:

1. Generate an initial population of candidate solutions (randomly).
2. Assign a fitness score to each chromosome accordingly to his quality for solving the problem.
3. Apply genetic operators to the population.
4. Check the stop criteria. If it is fulfilled, go to 5. If not, restart from 2.
5. Stop.

The individuals' selection can be done by different methods, as the fitness-proportionate selection or the tournament selection [10].

Genetic Algorithm (GA) and Genetic Programming (GP) belong to the family of EAs. In GA the genes are encoded by fixed length binary strings, and the chromosomes of the population have the same size.

For evolving the initial population, Holland [33] defined the one-point cross-over, as a main genetic operator, and the one-point mutation and the inversion, as secondary operators. GA mutation acts by flipping a bit in an individual. By crossover, two offspring are built from two individuals by swapping some segments of genetic code between them.

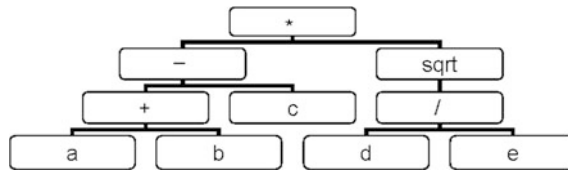


Fig. 6 GP individual that encodes the expression $[(a + b) - c] * \sqrt{d/e}$. The functions are the root node and inner nodes. The terminals are the leaf nodes

The main drawback of GA is the constant size of the chromosomes that does not permit the change of the model' structure during evolution. To eliminate this inconvenient, GP represents the individual using a tree structure, with a variable length. Figure 6 presents such an example. The trees can evolve, altering their shapes, sizes, and contents.

The function set can include algebraic operators, usual mathematical functions, Boolean operations, conditional operators (If—Then—Else), recursive functions, iterative functions (Do—Until). Terminals are variables or constants.

The symbols' set must be selected so as to meet the necessary and sufficient conditions to model the problem.

GP individuals encode computer programs, or mathematical functions expressed as complex compositions of functions and variables or constants [46]. If GAs are mainly used in optimization problems, GP is designed for models' identification, for given datasets. Contrary to what happens in nature, in GP there is no delimitation between the phenotype and the genotype. Therefore, the genetic operators act directly on the phenotype [7] and limit the operators' search power [24].

For an insight on this topic, one may refer to [46, 48].

3.1 Gene Expression Programming

Gene Expression Programming (GEP) has been introduced in 2001, by Candâda Ferreira [23]. It belongs to the family of GAs and is an automatic programming method based on the principle of natural selection. GEP combines features of GA (linear chromosomes of fixed length) and GP (hierarchical structures of different forms and dimensions). Its basic idea is the representation of the solutions of the problem at hand as individuals that evolve over generations by means of genetic operators.

Our aim is to present here a short overview of GEP based on [6, 10, 23, 24], not to explore all its capabilities.

Unlike GP, that does not allow multiple genes per individual for building solutions, GEP individuals are formed by many genes of equal length, coding nonlinear expressions. The individuals are strings of symbols (mathematical functions, constants, or variables). At its turn, a GEP gene is formed by a head (that can contain constants, variables, and symbols) and a tail (that can contain only constants and variables). Every gene encodes a sub-tree. The sub-trees attached to

every gene of an individual are linked together to the root node using linking functions that belong to a set specified by the user.

The gene number is a parameter that has to be set in the algorithm.

The proportionate roulette-wheel scheme [26] and simple elitism (cloning the best individual) are used for individuals' selection in GEP. The individuals of each generation are evaluated function of a performance measure. It is based on the error of the expression coded by the individual with respect to the input data. As a result, a fitness value is attached to each individual. The higher the fitness value is, the better an individual is. Therefore, the selection of the individuals for replication is done function of their fitness and the luck of the roulette. Then, the genomes of the best individuals (the selected ones) are copied as many times as the outcome of the roulette, during the replication process.

The operations involved in the genetic evolution are:

- Mutation, in which a part of an individual changes, preserving the chromosome's structural organization.
- Transposition (IS, RIS and gene transposition) that is used for copying a randomly selected part of a gene and moving it into another position of a chromosome. In the transposition process, a gene is deleted from its initial position and is moved at the beginning of the chromosome, such that the chromosome's length is maintained.
- Crossover, which combines futures of two parents in offspring. The GA crossover and its specific operators are also used in GEP, but a specific GEP operator for genes' crossover is also defined.

In the process of function finding, the goal is to determine an expression that links the endogenous variable to the exogenous ones. Particularly, in the time series modeling, given the sample $\{x_1, x_2, \dots, x_n\}$, the aim is to determine a model that approximates as well as possible the given values.

Here are the steps in solving this type of problem:

- Define the fitness function for an individual program;
- Choose the terminals' set, T , and the functions' set, F , for creating the chromosomes. In the simplest case, $F = \{+, -, *, /\}$;
- Choose the chromosome' structure, i.e. the length of the head and the number of genes;
- Choose the linking function;
- Choose the set of genetic operators and their rates.

An important issue is the determination of the window size, w , i.e. the number of the previous values used to estimate the actual one.

Depending on w , the preprocessed time series provides n -dimensional elements, $d_{t-w} = (x_{t-w}, x_{t-w+1}, \dots, x_{t-1})$, $w+1 \leq t \leq n$, function of which x_t is estimated by GEP, as:

$$x_t = f(x_{t-w}, x_{t-w+1}, \dots, x_{t-2}, x_{t-1}) + \varepsilon_t, \quad w+1 \leq t \leq n.$$

In our experiments, the fitness is considered to be the mean squared error (MSE) of the model coded by the chromosomes with respect to the registered data, which is defined by:

$$MSE = \frac{1}{n-1} \sum_{t=1}^n (x_t - \hat{x}_t)^2,$$

where x_t is the registered value, \hat{x}_t is the estimated one, and n is the sample's volume.

3.2 Adaptive Gene Expression Programming

In GEP the genes' number in a chromosome is constant for all individuals in a population. The modification of this number affects the length and the form of the solution.

Determining the optimum genes' number is a difficult, time-consuming process, involving the user's experience, requesting a big number of experiments. For avoiding this restriction, Adaptive Gene Expression Programming (AdaGEP) algorithm [13] is designed to identify automatically the genes' number in GEP, using a deactivation mechanism [7]. In this approach, a genemap is attached to each chromosome, in order to select the genes that participate in the decoding process. It is a bit string whose dimension is equal to the number of genes of a GEP individual. If the bit value is 1, the gene participates to the decoding, as in GEP case; if it is zero, it does not participate in this process.

The genemaps evolve similarly to the population in a GA; the mutation and crossover operations on them are exactly as in GA case. GEP iterations are accompanied by genemaps' iterations. Also, a genemap survives in the selection process only if the corresponding chromosome survives [7].

So, the steps of AdaGEP algorithm are:

1. Create the initial population of individuals (randomly)
2. Evolve individuals with GEP operators (crossover, mutation, transpositions)
3. Apply Gene Map Evolution operator on the population of genemaps
4. Evaluate each AdaGEP individual over the set of fitness cases
5. Select the next generation of individuals (by roulette wheel selection)
 - the fitness value of a genemap is that of the individual to whom it is attached.
 - survival of a GEP chromosome implies the survival of its genemap also.
6. Go to 2 if the stop criterion is not fulfilled.

4 Support Vector Regression (SVR)

Support Vector Regression (SVR) [55] is a nonlinear regression method derived from the Support Vector Machines technique (SVM) that was developed for solving problems of supervised learning and classification, based on the principle of errors' minimization. The idea behind this algorithm is the construction of a model function, f , which forecasts the output of a system that depends on a set of variables, using a set of input data for which the output is known.

The main characteristic of SVM is that the prediction function is developed on a support set. The algorithms based on Support Vector have been extended to classification problems, using different loss functions [3].

SVR is a category of SVM. ε -SVR, introduced by Vapnik [63] uses so-called ε -insensitive loss function, which minimizes the generalized error bound instead of minimizing the learning error, using the structural risk minimization principle. This algorithm searches a function f , as smooth as possible, that has at most a ε deviation from the specified output of all learning data. These constraints lead to a convex optimization problem.

Below we present the formulation of the algorithm, based on [3, 63].

Firstly, let us consider the case of a linear function defined by:

$$f(x) = \langle w, x \rangle + b, \quad b \in \mathbf{R}, \quad x \in X,$$

where X is the input space and $\langle \cdot, \cdot \rangle$ is the inner product in X .

Suppose that the learning data are denoted by $(x_i, y_i) \subset X \times \mathbf{R}, i = \overline{1, m}$.

For taking into account the existence of an infeasible convex optimization problem, the slack variables ξ, ξ^* are introduced, so that the problem becomes:

$$\text{minimize} \left\{ \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m (\xi_i + \xi_i^*) \right\},$$

under the constraints:

$$\begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases}$$

Using the dual problem, the function f can be written as:

$$f(x) = \sum_{i=1}^m (\alpha_i - \alpha_i^*) \langle x_i, x \rangle + b, \quad (12)$$

with

$$f(x) = \sum_{i=1}^m (\alpha_i - \alpha_i^*) = 0, \quad \alpha_i, \alpha_i^* \in [0, C].$$

Relation (12) is the support—vector expansion of f . C and ε are parameters that must be determined.

In ν -SVR, which is a version of SVR, ε is considered to be a variable in the optimization process and a most convenient parameter $\nu \in (0, 1)$ is introduced.

For solving the non-linear problem, a projection of input data on a Hilbert space, F , of higher dimension is done and then (12) is written as:

$$f(x) = \sum_{i=1}^m (\alpha_i - \alpha_i^*) K(x_i, x) + b,$$

utilizing a kernel function, K , as linear, polynomial, radial basis function (RBF), sigmoid [35, 58] or other functions that fulfill Mercer's conditions [60].

Since there are no general criteria to choose a particular kernel, in most cases the choice is done in an empirical way.

The prediction requires the choice of kernel parameters, which depend on data. This is, generally, a difficult task. In many cases parameters are tuned by hand, based on experience, and are adjusted taking into account the experimental results.

Kernlab package from R can be used to run SVR. Other software may also be employed for this purpose. For some of them, the reader may refer to [41].

5 General Regression Neural Network (GRNN)

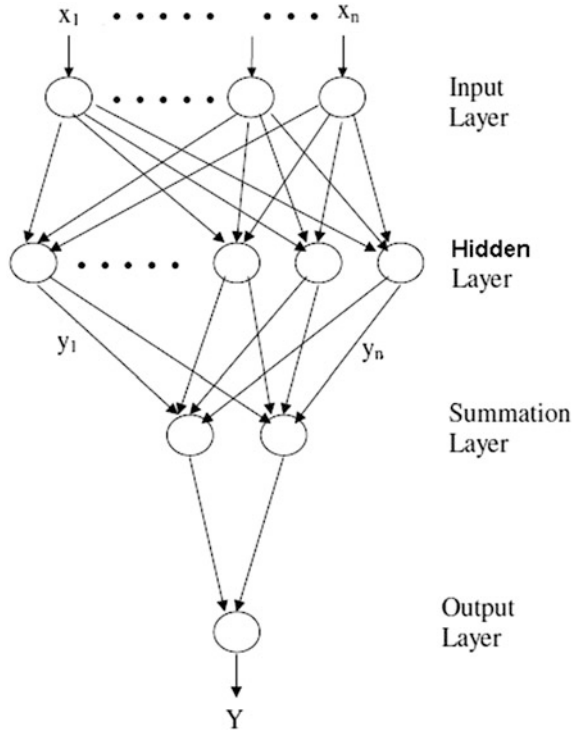
Artificial Neural Networks (ANNs) have been introduced by McCulloch and Pitts [49] for modeling logical functions. They are formed by artificial neurons that sum the input data producing the output. An activation function is applied to the weighted sums to produce the result, which becomes the input of the next layer.

General Regression Neural Networks (GRNN) were designed by Specht [59] as feed-forward networks, with four layers: Input layer, Pattern layer, Summation layer and Output layer (Fig. 7). Unlike Probabilistic Neural Networks that perform classifications on discrete variables, GRNN performs regressions and the target variables are continuous [60].

In GRNN, the numbers of neurons in the Input layer and that of the predictor variables are equal.

GRNN uses nonparametric estimators of the density of probability function. The measure of the estimation's quality realized by each training sample is the Euclidean or city-block distance between the training sample and the prediction point [59]. The distances between the assessed point and the other points are calculated and a kernel function is applied to them for computing the weight of each point. The best estimated value of the new point is determined by summing the

Fig. 7 GRNN diagram



values of the other points, weighted by the kernel function. These operations are done in the Hidden layer, which is also used for storing the predictors' values and the target values and whose number of neurons is equal to the number of data in the training set.

The Summation layer is formed by two neurons: the numerator and the denominator summation neurons. They are used for the storage of the sum of the weights from the Hidden layer and the weights multiplied by the actual target values, respectively.

The Output layer is composed of one neuron that contains the result of the division of the values stocked by the numerator and the denominator of the previous layer [5].

6 Wavelets

Wavelets are waves with some specific properties, utilized for representing different functions [52], based on the wavelets transform (WT), a recent technique developed by Grossman and Morlet [28]. Due to its advantage, as the absence of the restrictions related to the process' stationarity or its long range dependence property

[27], and the preservation of the local phenomena, multiscale and periodical features, WT is a valuable tool for time series modeling.

Let us consider a classical problem of non-parametric regression:

$$y_i = f(x_i) + \varepsilon_i, \quad i = 1, \dots, n,$$

where (ε_i) are independent, identically distributed variables, with zero mean and the variance σ^2 .

If $f \in L^2(\mathbf{R})$ and ψ is the so-called mother wavelet, there are functions, called wavelets, defined by:

$$\psi_{jk}(x) = 2^{j/2} \psi(2^j x - k),$$

such as (ψ_{jk}) form an orthonormal basis for $f \in L^2(\mathbf{R})$ and, therefore, f can be written as:

$$f(x) = \sum_{k \in \mathbf{Z}} c_k \phi_{0k}(x) + \sum_{j < J, k \in \mathbf{Z}} d_{jk} \psi_{jk}(x),$$

where

$$c_k = \int_{\mathbf{R}} f(x) \phi_{0k}(x) dx, \quad d_{jk} = \int_{\mathbf{R}} f(x) \psi_{jk}(x) dx,$$

$$\phi_{j_0 k}(x) = 2^{j_0/2} \phi(2^{j_0} x - k),$$

ϕ is the father wavelets and J is a natural number giving the maximum resolution [20].

Practically, d_{jk} are firstly estimated and then the shrinkage is applied to the estimated values, \tilde{d}_{jk} , providing the new ones, \hat{d}_{jk} . The filtering process depends on a threshold, λ , as well as on the used filter-hard or soft.

In our studies [9, 11] we employed the hard threshold, which is defined by:

$$\delta_{\lambda}^H(d_{jk}) = d_{jk} I(|d_{jk}| > \lambda),$$

where I is the indicator function.

The main advantage of the wavelets procedure is the conservation of the coordinates d_{jk} that contain the significant information on the function f and the noise d_e removal, following the decomposition rule:

$$d_{jk} = \hat{d}_{jk} + d_e.$$

Stein's Unbiased Risk Estimate (SURE) and the universal threshold are the most used in applications [9, 11].

Finally, the function f is built as:

$$f(x) = \sum_{k=1}^{2^J-1} \hat{c}_k \phi_{0k}(x) + \sum_{j=0}^{J-1} \sum_{k=0}^{2^j-1} \hat{d}_{jk} \psi_{jk}(x),$$

where \hat{c}_k is an estimation of c_k [19].

The alternative of the wavelets continuous transform (CWT) for discrete data is the discrete wavelets transform (DWT).

For example, DWT can be defined using the Daubechies wavelets, for a given the data sets x_1, x_2, \dots, x_n , $n = 2^J$, following the steps [61]:

1. Consider the sets of coefficients:

$$\{h_0 = 1/\sqrt{2}, h_1 = 1/\sqrt{2}\}, \quad \{g_0 = 1/\sqrt{2}, g_1 = -1/\sqrt{2}\}.$$

2. Build the filters H, G and their duals, H^*, G^* such that:

$$\sum_k h_k = \sqrt{2}, \quad \sum_k g_k = 0, \quad H^*H + G^*G = Id,$$

where Id is the unit matrix.

3. Build the continuous function f , defined by:

$$f = \sum_k f_k \phi_H(t - k),$$

where

$$\phi_H(t) = \begin{cases} 1, & t \in [0, 1) \\ 0, & t \in \mathbf{R} - [0, 1) \end{cases}.$$

4. Use the CWT procedure for f built at the previous step.

For simulation using wavelets, **wavelets** and **wavethresh** packages from R can be used. For an extensive study on wavelets, readers can refers to [19, 20, 51, 52, 61].

References

1. Akaike, H.: Fitting autoregressive models for prediction. Ann. Inst. Stat. Math. **21**, 243–247 (1969)
2. Alder, H.L., Roessler, E.B.: Introduction to Probability and Statistics, 5th edn. W.H. Freeman and Company, San Francisco (1972)

3. Basak, D., Pal, S., Patranabis, D.C.: Support vector regression. *Neural Inf. Process. Lett. Rev.* **11**(10), 203–224 (2007)
4. Bărbulescu, A.: *Time Series with Applications*. Junimea, Iași (2002) (in Romanian)
5. Bărbulescu, A., Barbeș, L.: Mathematical models for inorganic pollutants in Constanța area, Romania. *Rev. Chim.* **64**(7), 747–753 (2013)
6. Bărbulescu, A., Băutu, E.: Alternative models in precipitation analysis. *Analele Științifice ale Universității Ovidius, Matematică* **17**(3), 45–68 (2009)
7. Bărbulescu, A., Băutu, E.: Time series modeling using an adaptive gene expression programming. *Int. J. Math. Models Meth. Appl. Sci.* **3**(2), 85–93 (2009)
8. Bărbulescu, A., Băutu, E.: Mathematical models of climate evolution in Dobruja. *Theor. Appl. Climatol.* **100**(1–2), 29–44 (2010)
9. Bărbulescu, A., Deguenon, J.: Models for trend of precipitation in Dobruja. *Environ. Eng. Manage. J.* **13**(4), 873–880 (2014)
10. Bărbulescu, A., Maftai, C., Băutu, E.: *Modeling the Hydro-Meteorological Time Series. Applications to Dobruja Region*. Lambert Academic Publishing, Germany (2010)
11. Bărbulescu, A., Petac, A.: Statistical assessment of precipitation evolution. *Case Study. Autom. Comput. Appl. Math.* **22**(1), 7–15 (2013)
12. Băutu, E., Bărbulescu, A.: Forecasting meteorological time series using soft computing methods: an empirical study. *Appl. Math. Inf. Sci.* **7**(4), 1297–1306 (2013)
13. Băutu, E., Băutu, A., Luchian, H.: AdaGEP—an adaptive gene expression programming algorithm. In: *Proceedings of the Ninth international Symposium on Symbolic and Numeric Algorithms For Scientific Computing*, pp. 403–406. SYNASC, IEEE Computer Society, Washington, DC, 26–29 Sept 2007
14. Bourbonais, J.: *Cours et Exercices d'économétrie*. Dunod, Paris (1993)
15. Box, G.P.E., Jenkins, G.M., Reinsel, G.C.: *Time Series Analysis: Forecasting and Control*, 4th edn. Wiley, Hoboken (2008)
16. Brockwell, P., Davies, R.: *Introduction to Time Series Analysis and Forecasting*. Springer, New York (2002)
17. Burg, J.P.: *A new analysis technique for time series data*. NATO Advanced Study Institute of Signal Processing, Enschede, Netherlands (1968)
18. Camps-Valls, G., Chalk, A.M., Serrano-López, A.J., Martín-Guerrero, J.D., Sonnenhammer, E. L.L.: Profiled support vector machines for antisense oligonucleotide efficacy prediction. *BMC Bioinform.* **5**:135 (2004) <http://www.biomedcentral.com/1471-2105/5/135>
19. Chui, C.K.: *An Introduction to Wavelets*. Academic Press Inc., San Diego (1992)
20. Daubechies, I.: *Ten lectures on Wavelets*. SIAM, New Delhi (1992)
21. Deguenon, J., Bărbulescu, A.: Fractal characterization of rainfall in Benin. *Int. J. Ecol. Econ. Stat.* **30**(3), 46–55 (2013)
22. Dickey, D.A., Fuller, W.A.: Distribution of the estimators for autoregressive time series with a unit root. *J. Am. Stat. Assoc.* **74**(366), 427–431 (1979)
23. Ferreira, C.: Gene expression programming: a new adaptive algorithm for solving problems. *Complex Syst.* **13**(2), 87–129 (2001)
24. Ferreira, C.: *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. Springer, Berlin (2006)
25. Fuller, W.A.: *Introduction to Statistical Time Series*. Wiley, New York (1996)
26. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Boston (1989)
27. Granger, C.W.J., Joyeux, R.: An introduction to long memory time series models and fractional differencing. *J. Time Ser. Anal.* **1**, 15–39 (1980)
28. Grossman, A., Morlet, J.: Decomposition of Hardy functions into square integrable wavelets of constant shape. *SIAM J. Math. Anal.* **1**, 723–736 (1984)
29. Haidu, I.: *Time Series Analysis. Applications in Hydrology*. HGA, București (1997) (in Romanian)
30. Hannan, E.J., Risannen, J.: Recursive estimation of mixed autoregressive moving—average order. *Biometrika*, **69**(1), 81–94 (1981)

31. Hashimi, M.Z., Shamseldin, A.Y., Melville, B.W.: Statistical downscaling of watershed precipitation using Gene Expression Programming (GEP). *Environ. Model. Softw.* **26**(12), 1639–1646 (2011)
32. Haykin, S.: *Neural networks and Learning Machines*, 3rd edn. Pearson Education Inc., New York (2009)
33. Holland, J.H.: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge (1992)
34. Hong, Y.-S.T., White, P.A., Scott, D.M.: Automatic rainfall recharge model induction by evolutionary computational intelligence. *Water Resour. Res.* **41**, W08422, 13 PP (2005)
35. Hsu, C.-W., Chang, C.-C., Lin, C.-J.: A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei (2010) <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>
36. <https://cran.r-project.org/web/packages/forecast/forecast.pdf>
37. <https://cran.r-project.org/web/packages/fUnitRoots/fUnitRoots.pdf>
38. <https://cran.r-project.org/web/packages/urca/urca.pdf>
39. <http://faculty.washington.edu/ezivot/econ584/notes/unitroot.pdf>
40. <https://stat.ethz.ch/R-manual/R-devel/library/stats/html/arima.sim.html>
41. http://www.support-vector-machines.org/SVM_soft.html
42. Hung, N.Q., Babel, M.S., Weesakul, S., Tripathi, N.K.: An artificial neural network model for rainfall forecasting in Bangkok, Thailand. *Hydrol. Earth Syst. Sci.* **13**, 1413–1425 (2009)
43. Hyndman, R., Athanasopoulos, G.: *Forecasting: Principles and Practice*. OTexts: Melbourne (2014) <https://www.otexts.org/fpp/8/7>
44. Hyndman, R.J., Khandakar, Y.: Automatic time series forecasting: the forecast package for R. *J. Stat. Softw.* **26**(3) (2008)
45. Kaneko, H., Funatsu, K.: Application of online support vector regression for soft sensors. *AIChE J.* **60**, 600–612 (2014)
46. Koza, J.R.: *Genetic programming: on the programming of computers by means of natural selection*. MIT Press Cambridge, Massachusetts (1992)
47. Kwiatkowski, D., Phillips, P.C.B., Schmidt, P., Shin, Y.: Testing the null hypothesis of stationarity against the alternative of a unit root. *J. Econometrics* **54**, 159–178 (1992)
48. Langdon, W.B., Poli, R.: *Foundations of Genetic Programming*. Springer, Berlin (2002)
49. McCullogh, W., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **5**, 115–133 (1943)
50. MacKinnon, J.G.: Numerical distribution functions for unit root and cointegration tests. *J. Appl. Econometrics* **11**, 601–618 (1996)
51. Nason, G.P.: Choice of the threshold parameter in wavelet function estimation. In: Antoniadis, A., Oppenheimer, G. (eds.) *Wavelets and Statistics, Lecture Notes in Statistics*, vol. 103, pp. 261–280. Springer, Berlin (1995)
52. Nason, G.P.: *Wavelets Methods in Statistics with R*. Springer, Berlin (2008)
53. Pfaff, B.: *Analysis of Integrated and Cointegrated Time Series with R*, 2nd edn. Springer, Berlin (2008)
54. Phillips, P.C.B., Perron, P.: Testing for a unit root in time series regression. *Biometrika* **75**(2), 335–346 (1998)
55. Sapankevych, N., Sankar, R.: Time series prediction using support vector machines: a survey. *IEEE Comput. Intell. Mag.* **4**(2), 24–38 (2009)
56. Schmidt, P., Phillip, P.C.B.: LM tests for a unit root in the presence of deterministic trends. *Oxford Bull. Econ. Stat.* **54**(3), 257–287 (1992)
57. Schwarz, G.E.: Estimating the dimension of a model. *Ann. Stat.* **6**(2), 461–464 (1978)
58. Smola A.J., Schölkopf B.: A tutorial on support vector regression. *Statistics and Computing*, **14**, 199–222 (2004)
59. Specht, D.F.: General regression neural network. *IEEE Trans. Neural Netw.* **2**(6), 568–576 (1991)
60. Specht, D.F.: Enhancements to probabilistic neural networks. *Int. Jt. Conf. Neural Netw.* **1**, 761–768 (1992)

61. Stark, H.-G.: Wavelets and Signal Processing. Springer, Berlin (2005)
62. Tay, F.E.H., Cao, L.: Application of support vector machines in financial time series forecasting. *Omega: Int. J. Manage. Sci.* **29**(4), 309–317 (2001)
63. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (2000)

Studies on Time Series Applications in Environmental
Sciences

Bărbulescu, A.

2016, XIII, 187 p. 86 illus., 24 illus. in color., Hardcover

ISBN: 978-3-319-30434-2