R demonstration

Wolfgang Scherrer and Manfred Deistler

Juli 11, 2018

Contents

1 Introduction

This document is intended as a short guide to the modeling of multivariate time series with VARMA models or state space models. We mainly use two packages:

- the MTS package, which is kind of companion toolbox for the text book Tsay (2014), Multivariate Time Series Analysis, John Wiley & Sons.
- and the dse package by Paul Gilbert, see e.g. Gilbert, P., 2015. Brief User's Guide: Dynamic Systems Estimation. http://cran.r-project.org/web/packages/dse/vignettes/Guide.pdf.

We only discuss and use some parts of these packages. Both of them include many more models and methods, e.g. the MTS package also supports multivariate volatility models, factor models and error-correction VAR models for co-integrated time series.

Some utilities (in particular tools to convert MTS/dse objects) are collected in `tools.R` A short description/manual of these tools may be found at the end of this document.

This code is not thoroughly tested and thus should be used with some care. In particular there are almost no input checks, so be sure to use correctly specified parameters. Please feel free to use this code and to change the code according to your own needs and preferences.

Often we use a syntax like `MTS::function` or `dse::function` in order to make clear which package is used.

2 VARMA Models

We consider vector autoregressive moving average (VARMA) models of the form

$$a_0 y_t = a_1 y_{t-1} + \cdots + a_p y_{t-p} + b_0 \epsilon_t + b_1 \epsilon_{t-1} + \cdots + b_q \epsilon_{t-q}$$

where $(\epsilon_t \,|\, t \in \mathbb{Z})$ is n-dimensional white noise with variance $\Sigma = \mathbb{E}\epsilon_t \epsilon_t' = I \in \mathbb{R}^{n\times n}$ and $a_j, b_j \in \mathbb{R}^{n\times n}$ are parameter matrices. It is assumed that $a_0 = b_0$ is non singular and we most often consider the case[1] $a_0 = b_0 = I \in \mathbb{R}^{n\times n}$.

The AR/MA polynomials associated with this model are

$$\begin{aligned} a(z) &= a_0 - a_1 z - \cdots - a_p z^p \\ b(z) &= b_0 + b_1 z + \cdots + b_q z^q \end{aligned}$$

2.1 `MTS` implementation and tools

First load the `MTS` library and construct an example VARMA(2,2) model. Note that Tsay uses a different notation,

$$\phi_0 y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \phi_0 \epsilon_t - \theta_1 \epsilon_{t-1} - \cdots - \theta_q \epsilon_{t-q}$$

The AR and MA coefficients are collected in two matrices $\phi = (\phi_1, \ldots, \phi_p) \in \mathbb{R}^{n\times np}$ and $\theta = (\theta_1, \ldots, \theta_q) \in \mathbb{R}^{n\times nq}$:

```
> library(MTS)        # load package
> source('tools.R')   # load utility functions
>
> phi0 = matrix(c( 1.0,   0, 0,
+                 -1.199, 1, 0,
+                 -0.638, 0, 1), byrow = TRUE, nrow= 3)
> phi = matrix(c(0.762, 0,     0,    -0.074, 0.137,-0.313,
+               -0.142,-0.470, 0.543, 0,     0,     0,
+                0.920,-0.775, 0.064, 0,     0,     0),
+              byrow = TRUE, nrow= 3, ncol=6)
> theta = matrix(c( 0.694,-0.116,-0.150,-0.216, 0.269,-0.231,
+                  -0.540,-0.253, 0.708, 0,     0,     0,
+                   0.748,-0.760, 0.242, 0,     0,     0),
+                byrow = TRUE, nrow= 3, ncol=6)
> sigma = matrix(c(0.815, 0.154, 0.411,
+                  0.154, 0.716, 0.202,
+                  0.411, 0.202, 0.813), nrow=3)

> phi0
      [,1] [,2] [,3]
[1,]  1.000    0    0
```

[1] Of course we can easily reparametrize the model ($a_j \to a_0^{-1} a_j$ and $b_j \to a_0^{-1} b_j$) in order to satisfy the normalization constraint $a_0 = b_0 = I$.

```
[2,] -1.199    1    0
[3,] -0.638    0    1
```

```
> phi
       [,1]   [,2]  [,3]   [,4]  [,5]   [,6]
[1,]  0.762  0.000 0.000 -0.074 0.137 -0.313
[2,] -0.142 -0.470 0.543  0.000 0.000  0.000
[3,]  0.920 -0.775 0.064  0.000 0.000  0.000
```

```
> theta
       [,1]   [,2]   [,3]   [,4]  [,5]   [,6]
[1,]  0.694 -0.116 -0.150 -0.216 0.269 -0.231
[2,] -0.540 -0.253  0.708  0.000 0.000  0.000
[3,]  0.748 -0.760  0.242  0.000 0.000  0.000
```

```
> sigma
      [,1]  [,2]  [,3]
[1,] 0.815 0.154 0.411
[2,] 0.154 0.716 0.202
[3,] 0.411 0.202 0.813
```

If the stability condition

$$\det a(z) \neq 0 \quad \forall |z| \leq 1$$

holds, then the unique stationary solution of the above VARMA system has a causal MA(∞) representation

$$y_t = \sum_{j \geq 0} k_j \epsilon_{t-j}$$

The sequence $(k_j \in \mathbb{R}^{n \times n} \,|\, j \geq 0)$ is called the *impulse response function* of the VARMA system. Note that $k_0 = I$ since $a_0 = b_0$. If $\Sigma = HH'$, i.e. if $\bar{\epsilon}_t = H^{-1}\epsilon_t$ has a unit variance, then $(\bar{k}_j = k_j H \,|\, j \geq 0)$ is a so called *orthogonalized impulse response function*.

The (orthogonalized) impulse response function may be computed with the function `VARMAirf`. Note that `VARMAirf` assumes $\phi_0 = I$ ($a_0 = I$) and hence we reparametrize the model by $\phi \to \phi_0^{-1}\phi$ and $\theta \to \phi_0^{-1}\theta$.

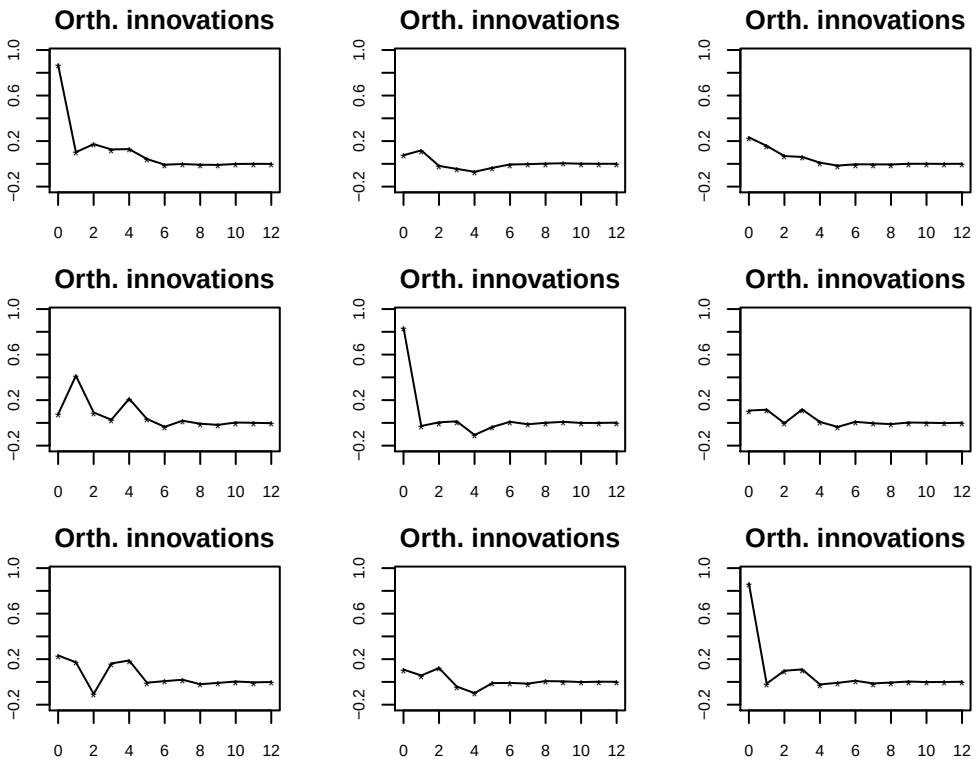
The function `VARMAirf` returns a list with components `psi` and `irf`. The component `psi` is the matrix $(k_0, k_1, \ldots, k_l) \in \mathbb{R}^{n \times n(l+1)}$. The component `irf` is the matrix[2] $(\mathrm{vec}(\bar{k}_0), \mathrm{vec}(\bar{k}_1), \ldots, \mathrm{vec}(\bar{k}_l)) \in \mathbb{R}^{n^2 \times (l+1)}$, i.e. `irf` contains the desired orthogonalized impulse response coefficients. Note that the `MTS` package here uses two different methods to represent a 3-dimensional array by a (2-dimensional) matrix!

`VARMAirf` uses the symmetric square root of Σ (computed via an eigenvalue decomposition of Σ) and hence the lag zero coefficient $\bar{k}_0 = k_0 H = H = H'$ is symmetric.

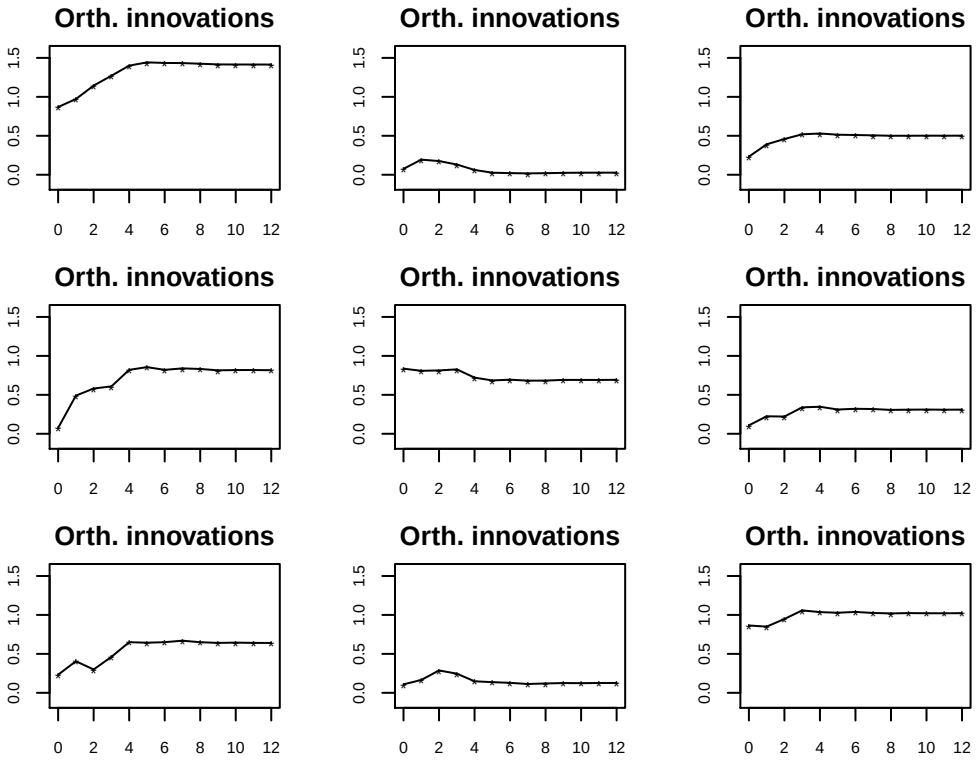
`VARMAirf` always produces two plots (one for the (orthogonalized) impulse response function and one for the cumulative (orthogonalized) impulse response function). In order to be somewhat more flexible we have implemented a simple function (`plot3d`) for plotting 3-dimensional arrays like the impulse response function or the autocovariance function. The (i,j)-th panel plots the respective (i,j)-component of $\bar{k}_l$ as a function of the lag $l = 0, 1, 2 \ldots$ and hence shows influence of the j-th "orthogonalized shock" $\bar{\epsilon}_{jt}$ on the i-th component $y_{i,t+l}$.

```
> k = MTS::VARMAirf(Phi = solve(phi0,phi),
+                   Theta = solve(phi0, theta),
+                   Sigma = sigma, lag = 12, orth = TRUE)
```

[2] vec denotes the vectorization operator, i.e. $\mathrm{vec}(X)$ is the mn-dimensional column vector obtained by stacking the columns of the (m,n)-dimensional matrix X.

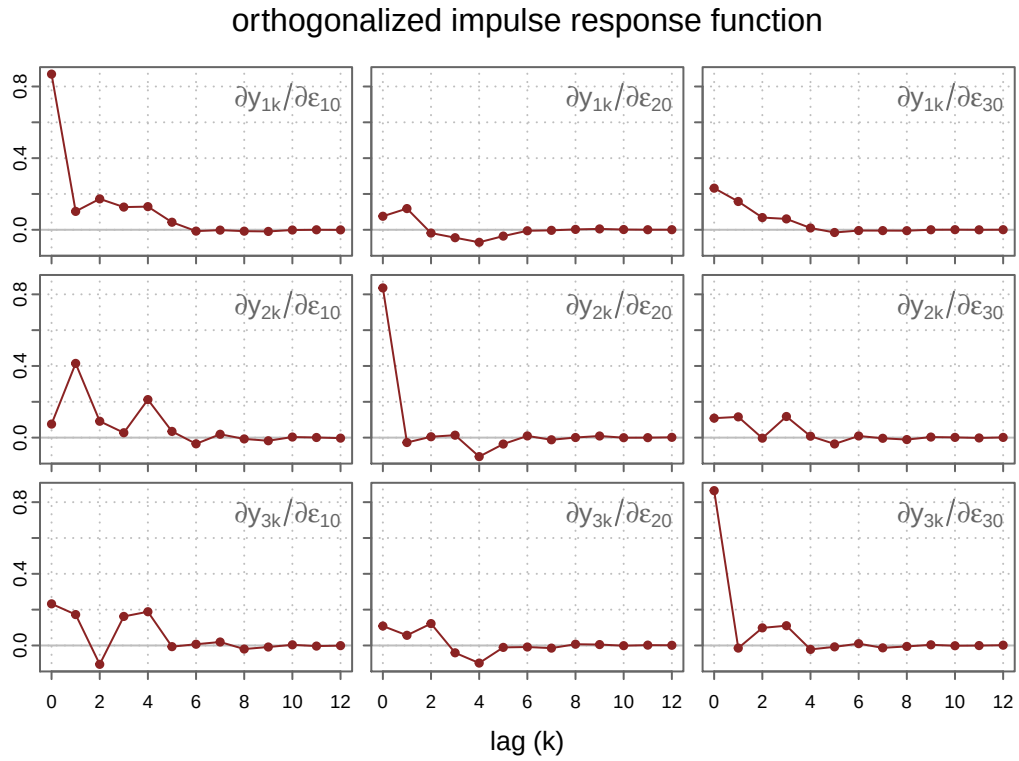



```
Press return to continue
```




```
>
> plot3d(k$irf, dim = c(3,3,13),
```

```
+     main='orthogonalized impulse response function',
+     labels.ij="partialdiff*y[i_*k]/partialdiff*epsilon[j_*0]",
+     type='o', lty='solid', col='brown4', pch=19, cex=0.5)
```



The autocovariance function

$$\gamma_j = \mathbb{E} y_{t+j} y_t' = \sum_{l \geq 0} k_{j+l} \Sigma k_j' \quad j \geq 0$$

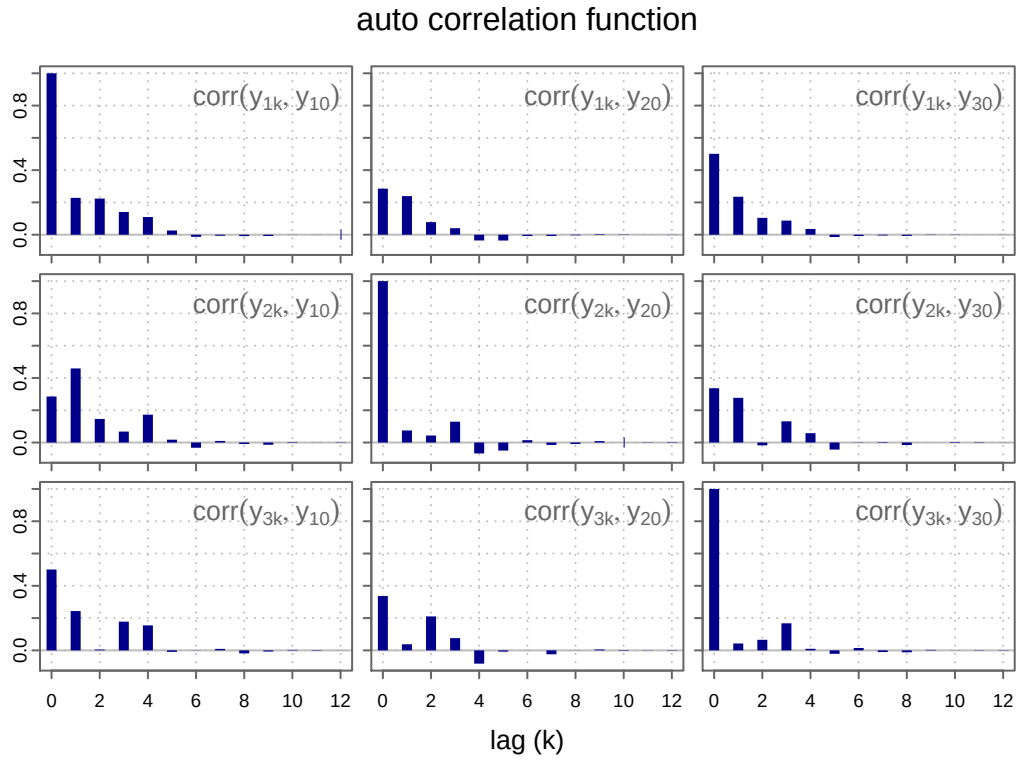
of the VARMA process (y_t) may be easily computed with the function `ARMAcov`. This function returns a list with components `autocov` and `ccm`, where `autocov` stores the autocovariances, i.e. the matrix $(\gamma_0, \gamma_1, \ldots, \gamma_l) \in \mathbb{R}^{n \times n(l+1)}$ and the $(n \times n(l+1))$ matrix `ccm` contains the autocorrelations[3] $\rho_j = \text{diag}(\gamma_0)^{-1/2} \gamma_j \text{diag}(\gamma_0)^{-1/2}$, $j = 0, 1, \ldots, l$.

To be precise `VARMAcov` computes an approximation of the autocovariance function by the finite sum $\sum_{l=0}^{m} k_{j+l} \Sigma k_j'$, where the number m of lags used corresponds to the optional parameter `trun`.

Note that `VARMAcov` always prints the computed autocovariances and autocorrelations (called cross correlation matrices) and hence here we suppress the output of the next `R` block.

```
> g = MTS::VARMAcov(Phi = solve(phi0,phi),
+                   Theta = solve(phi0,theta),
+                   Sigma = sigma, lag = 12)
>
> plot3d(g$ccm, dim = c(3,3,13),
+        main = 'auto correlation function',
+        labels.ij = "corr(list(y[i_*k],y[j_*0]))",
+        type = 'h', col = 'blue4', lwd = 5, lend = 1)
```

[3]Here $\text{diag}(\gamma_0)^{-1/2}$ denotes the diagonal matrix with diagonal elements $(\gamma_{0,ii})^{-1/2}$, i.e. the reciprocals of the standard deviations of the components y_{it}.



If in addition the miniphase assumption

$$\det b(z) \neq 0 \quad \forall |z| < 1$$

holds, then the ϵ_t's are the *innovations* of the process (y_t) and the above MA(∞) representation is the Wold representation of the process. The variance of the h-step ahead prediction errors from the infinite past then is given by

$$\Sigma_h = \mathbb{E}(y_{t+h} - \hat{y}_{t,h})(y_{t+h} - \hat{y}_{t,h})' = \sum_{l=0}^{h-1} k_l \Sigma k_l' = \sum_{l=0}^{h-1} \bar{k}_l \bar{k}_l'$$

For the variance of the i-th component of the forecast errors we obtain

$$\mathbb{E}(y_{i,t+h} - \hat{y}_{i,t,h})^2 = \sum_{l=0}^{h-1} \sum_{j=1}^{n} \bar{k}_{l,ij}^2 = \sum_{j=1}^{n} \sigma_{ij}^h \quad \text{where } \sigma_{ij}^h = \sum_{l=0}^{h-1} \bar{k}_{l,ij}^2.$$

and hence the ratio

$$c_{ij}^h = \frac{\sigma_{ij}^h}{\sum_{m=1}^{n} \sigma_{im}^h}$$

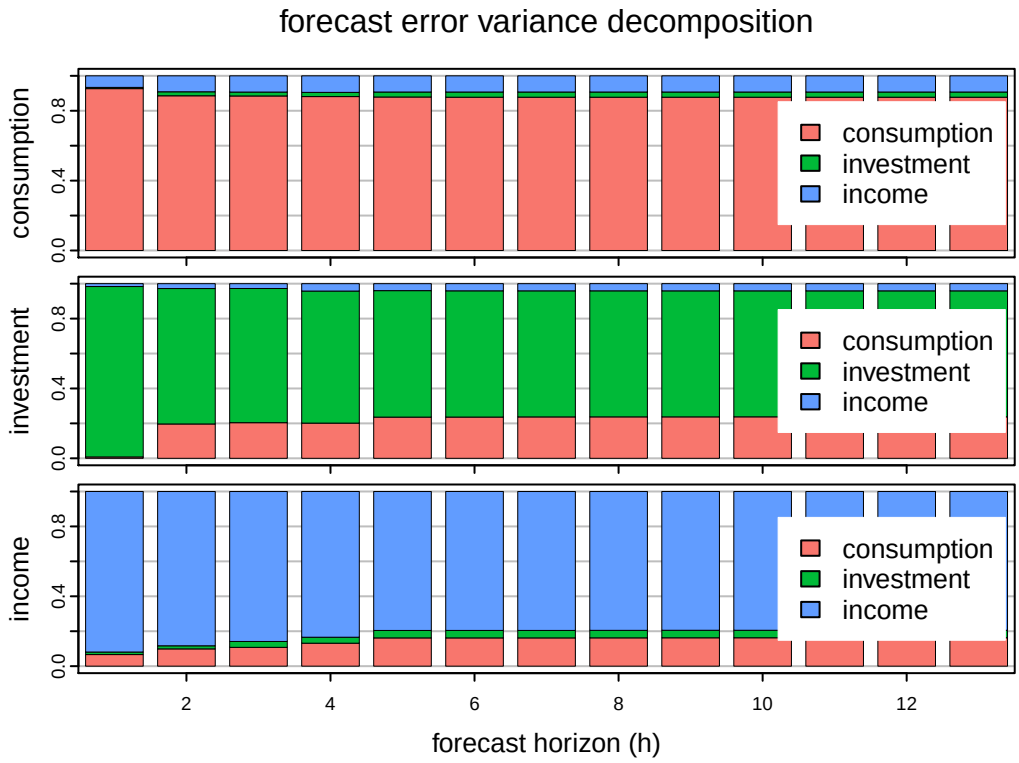
is the fraction of the forecast error variance of the i-th component due to the j-th component of the (orthogonalized) shocks $\bar{\epsilon}_t$. This is the so called *forecast error variance decomposition* which may be computed by the utility function `fevd`. This function takes as a main argument an arbitrary orthogonal impulse response function (e.g. computed by `ARMAirf`) and returns a list with two components. The first element `vd` is an $(n, n, h_{\max})$-dimensional array where the (i, j, h)-th entry is equal to c_{ij}^h and the second element `v` is an $(n, h_{\max})$-dimensional matrix where the (i, h)-th element is the variance of the i-th component of the h-step ahead forecast error, i.e. $\sum_{m=1}^{n} \sigma_{im}^h$. The maximum forecast horizon $h_{\max}$ is determined by the length of the input.

A plot[4] of this decomposition may be obtained by `plotfevd`.

[4]The choice for the `series.names` will become clear later on.

It seems that the `MTS` function `FEVdec` has a bug. Furthermore the orthogonalization scheme is "hardwired" (a Cholesky decomposition of Σ). Therefore we have implemented an own version.

```
> out = fevd(k$irf, dim = c(3,3,13))
> plotfevd(out$vd,
+     series.names = c('consumption','investment','income'))
```



The stability and the miniphase assumptions may be checked with the utility function `is.stable`. In the next subsection we will also discuss how to check these assumption with `dse` package tools.

```
> #check stability assumption
> is.stable(solve(phi0,phi))
[1] TRUE
attr(,"z")
[1] -7.072110e-01-1.402435e+00i -7.072110e-01+1.402435e+00i
[3]  1.459938e+00-1.005212e+00i  1.459938e+00+1.005212e+00i
[5]  9.596578e+15+0.000000e+00i  2.477278e+17+0.000000e+00i

> #check miniphase assumption
> is.stable(solve(phi0,theta))
[1] TRUE
attr(,"z")
[1]  8.050519e-01-1.478711e+00i  8.050519e-01+1.478711e+00i
[3] -3.306052e-01-2.326104e+00i -3.306052e-01+2.326104e+00i
[5]  3.396897e+14+0.000000e+00i -7.608720e+15+0.000000e+00i
```

The VARMA model, i.e. the degrees p, q and the parameters a_j, b_j, are not unique for a given (VARMA) process (y_t) without additional restrictions. However, note that (due to the stability and miniphase assumption) the causal MA(∞) representation is the Wold representation of the process and hence is unique. This means that the impulse response coefficients $(k_j \mid j \geq 0)$ are unique. One possibility to get a unique (identifiable) representation is to use the *echelon canonical form*. The construction of this canonical form is based on the

Hankel matrix of the impulse response coefficients

$$H = \begin{pmatrix} k_1 & k_2 & k_3 & \cdots \\ k_2 & k_3 & k_4 & \cdots \\ k_3 & k_4 & k_5 & \cdots \\ \vdots & \vdots & \vdots & \end{pmatrix}.$$

For VARMA models this matrix has a finite rank. The so called *Kronecker indices* $(\nu_1, \nu_2, \ldots, \nu_n)$ describe a basis for the row space of H. The rows with indices

$$j \in \bigcup_{\substack{1 \le i \le n \\ \nu_i > 0}} \{i, n+i, \ldots, n(\nu_i - 1) + i\}$$

form a basis for the row space of H. Clearly the rank of H is equal to the sum of the Kronecker indices ($\mathrm{rk}(H) = \sum_{i=1}^{n} \nu_i$). The echelon canonical form restricts certain elements of the AR/MA parameter matrices to zero or one. The position and the number of these restriction depends on the corresponding Kronecker indices.

The utility function `impresp2PhiTheta` computes for a given impulse response the Kronecker indices and the corresponding VARMA model in echelon canonical form. The computations are based on a finite sub matrix $H_{f,p}$ of the infinite dimensional Hankel matrix with f block rows and p block columns. The numbers f, p are determined from the length of the input sequence. The core computation is to determine a basis for the row space. This is done via a QR decomposition of the transpose $H'_{f,p}$ with the R function `qr`. The output of `impresp2PhiTheta` is a list with components `Phi`, `Theta`, `Phi0` (these matrices contain the AR/MA parameters), `kidx` (the vector of Kronecker indices), `Hrank` (the (computed) rank of the Hankel matrix H) and `Hpivot` (as returned by `qr()`). Note that the first `Hrank` elements of the vector `Hpivot` contain the indices of the basis rows of $H_{f,p}$.

The function `MTS::Kronspec` determines the zero/one restrictions imposed by the echelon canonical for given Kronecker indices (`kdx`) and prints a nice representation of these restrictions (for `output = TRUE`).

```
> out = impresp2PhiTheta(k$psi)
> out$kidx  # Kronecker indices
[1] 2 1 1
```

```
> # display the corresponding AR/MA restrictions
> junk = MTS::Kronspec(out$kidx)
Kronecker indices:  2 1 1
Dimension:  3
Notation:
 0: fixed to 0
 1: fixed to 1
 2: estimation
AR coefficient matrices:
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1    0    0    2    0    0    2    2    2
[2,]    2    1    0    2    2    2    0    0    0
[3,]    2    0    1    2    2    2    0    0    0
MA coefficient matrices:
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1    0    0    2    2    2    2    2    2
[2,]    2    1    0    2    2    2    0    0    0
[3,]    2    0    1    2    2    2    0    0    0
```

```
>
> all.equal(cbind(out$Phi0, out$Phi, out$Theta),
+           cbind(phi0,     phi,     theta))
```

```
[1] TRUE
```

The Kronecker indices are $(2, 1, 1)$ and hence the rank of the Hankel matrix is $2 + 1 + 1 = 4$ and the rows $1, 2, 3, 4$ (i.e. the first 4 rows) of H form a basis. The number of "free parameters" is 24. The last statement of the above `R` code shows that the VARMA model, we have started with, is in Echelon canonical form.

2.2 `dse` implementation and tools

The package `dse` uses an object oriented approach (with the S3 class system) and implements object classes for models, data sets and estimated models. VARMA models[5]

$$a_0 y_t + a_1 y_{t-1} + \cdots + a_p y_{t-p} = b_0 \epsilon_t + b_1 \epsilon_{t-1} + \cdots + b_q \epsilon_{t-q}$$

are represented by `ARMA` objects (which are special `TSmodel` objects). Note that the `ARMA` model class may represent more general models, in particular models with exogenous inputs (i.e. VARMAX models) and models with a trend component. In addition note that a_0 and b_0 may be different. However, here we will stick to the simple model above and assume that $a_0 = b_0 = I$.

The AR parameters $a_j \in \mathbb{R}^{n \times n}$ are stored in the $(p + 1, n, n)$-dimensional array `A`, where the i-th slot `A[i,,]` corresponds to the matrix $a_{i-1} \in \mathbb{R}^{n \times n}$. Analogously the MA parameters b_j are stored in the $(q + 1, n, n)$-dimensional array `B`.

In order to be able to easily switch between `MTS` and `dse` models and tools we have implemented two utility functions `PhiTheta2ARMA` and `ARMA2PhiTheta`.

Load the `dse` library and convert the above VARMA model to an `dse::ARMA` object. In addition we check that we can reconstruct the `Theta`, `Phi` parameters:

```
> library(dse)
Loading required package: tfplot
Loading required package: tframe

Attaching package: 'dse'
The following objects are masked from 'package:stats':

    acf, simulate

>
> arma = PhiTheta2ARMA(Phi = phi, Theta = theta, Phi0 = phi0,
+        output.names = c('consumption','investment','income'))
> arma

A(L) =
1-0.762L1+0.074L2    0-0.137L2    0+0.313L2
-1.199+0.142L1    1+0.47L1    0-0.543L1
-0.638-0.92L1    0+0.775L1    1-0.064L1

B(L) =
1-0.694L1+0.216L2    0+0.116L1-0.269L2    0+0.15L1+0.231L2
-1.199+0.54L1    1+0.253L1    0-0.708L1
-0.638-0.748L1    0+0.76L1    1-0.242L1

>
> junk = ARMA2PhiTheta(arma, normalizePhi0 = FALSE)
> all.equal(cbind(phi, theta, phi0),
```

[5] The `dse` package uses yet another convention for the sign of the AR/MA parameters!

```
+           cbind(junk$Phi, junk$Theta, junk$Phi0))
[1] TRUE
```

The stability and the miniphase assumption now may be checked with the function `polyrootDet(a))` which computes the roots of the determinant of a polynomial matrix $a(z) = a_0 + a_1 z + \cdots + a_p z^p$ with coefficients which are stored in the 3-dimensional array `a`.

```
> # check the stability assumption
> min(abs(polyrootDet(arma$A)))>1
[1] TRUE

> # check the (strict) miniphase assumption
> min(abs(polyrootDet(arma$B)))>1
[1] TRUE
```

The `dse` package contains a number of useful utilities for polynomial matrices (e.g. `characteristicPoly`, `companionMatrix`, `polydet`, ...).

3 State Space Models

State space models are an alternative way to describe processes with a rational spectral density. We consider models of the form

$$\begin{aligned} x_{t+1} &= Ax_t + B\epsilon_t \\ y_t &= Cx_t + \epsilon_t \end{aligned}$$

where (ϵ_t) is n-dimensional white noise with a variance $\Sigma = \mathbb{E}\epsilon_t\epsilon_t'$, x_t is an observed s-dimensional random vector called state and $A \in \mathbb{R}^{s\times s}$, $B \in \mathbb{R}^{s\times n}$ and $C \in \mathbb{R}^{n\times s}$ are parameter matrices. We always impose the stability assumption

$$\lambda_{\max}(A) < 1$$

and the miniphase assumption

$$\lambda_{\max}(A - BC) \leq 1.$$

Here $\lambda_{\max}(X)$ denotes the spectral radius of X, i.e. the maximum of the moduli of the eigenvalues of X. Given these assumption there exists a unique stationary solution (y_t) and this solution is of the form

$$y_t = \sum_{j\geq 0} k_j \epsilon_{t-j}$$

Furthermore the ϵ_t's are the innovations of the process (y_t) and the above MA representation is the Wold representation of the process. Therefore one says that the above state model is in *innovation form*.

State space models are implemented as `SS` objects in `dse`. However, `dse` uses a different naming convention, i.e. $A \to F$, $B \to K$ and $C \to H$. The `dse` package also handles state space models which are not in innovation form. To convert the above VARMA model to a state space model (in innovation form), we may use the function[6] `toSS`. The function `is.innovSS` checks whether the input is an "innovation form state space" object.

```
> ss = dse::toSS(arma)
> ss

 F =
     [,1] [,2] [,3]      [,4]      [,5]      [,6]
[1,]    0    0    0 -0.074000  0.137000 -0.313000
[2,]    0    0    0 -0.088726  0.164263 -0.375287
[3,]    0    0    0 -0.047212  0.087406 -0.199694
```

[6] Note that the function `toSS` does not work for ARMA(p, q) models with $q > p$!

```
[4,]    1    0    0  0.762000  0.000000  0.000000
[5,]    0    1    0  0.771638 -0.470000  0.543000
[6,]    0    0    1  1.406156 -0.775000  0.064000

 H =
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    0    0    1    0    0
[2,]    0    0    0    0    1    0
[3,]    0    0    0    0    0    1

 K =
         [,1]      [,2]      [,3]
[1,] 0.142000 -0.132000 -0.082000
[2,] 0.170258 -0.158268 -0.098318
[3,] 0.090596 -0.084216 -0.052316
[4,] 0.068000  0.116000  0.150000
[5,] 0.479532 -0.077916  0.014850
[6,] 0.215384  0.059008 -0.082300
```

```
> dse::is.innov.SS(ss)
[1] TRUE
```

We here get a model with a state space dimension $s = 6$.

The impulse response function and the autocovariance function of a process described by a state state space model in innovation form my be easily computed as follows:

$$\begin{aligned} k_0 &= I \\ k_j &= CA^{j-1}B \quad \text{for } j > 0 \end{aligned}$$

The variance of the state x_t is the solution of a so called Lyapunov equation

$$P = \mathbb{E}x_t x_t' = APA' + B\Sigma B'$$

The autocovariance function then is

$$\begin{aligned} \gamma_0 = \mathbb{E}y_t y_t' &= CPC' + \Sigma \\ \gamma_j = \mathbb{E}y_{t+j} y_t' &= CA^{j-1}(APC' + B\Sigma) \quad \text{for } j > 0 \end{aligned}$$

The utility function `SSirf` and `SScov` implement the above scheme to compute the impulse response and autocovariance function. The outputs returned by these function have the same structure as the output of the corresponding `MTS` function, e.g. `SScov` returns a list with components `autocov` and `ccm` where both of them are matrices of dimension $(n, n(l+1))$. To compute the state variance P here the function[7] `lyap` is used.

The following code computes the impulse response function and the ACF of the state space model. Of course, since this state space model and the VARMA model above describe the same process the output must be identical to the output we have computed above.

```
> k.ss = SSirf(ss, Sigma = sigma, lag.max = 12, orth = TRUE)
> all.equal(k, k.ss, check.attributes = FALSE)
[1] TRUE

>
> g.ss = SScov(ss, Sigma = sigma, lag.max = 12)
> all.equal(g, g.ss)
[1] TRUE
```

[7] Alternatively one may use the function `dse::Riccati`. However for the model we use here for testing purposes the (non iterative version) of `Riccati` stops with an error message. The function `lyap` first computes a Schur decomposition of the state transition matrix A (respectively F). To this end the `QZ` package has to be installed!

State space models (like VARMA models) are by no means unique. Even the state space dimension s is not unique. A model is called *minimal* if its state space dimension is minimal among all state space models which describe the process. A state space model is called *observable* (respectively *reachable*) if the observability matrix $O = (C', A'C', \ldots, (A')^{s-1}C')' \in \mathbb{R}^{ns \times s}$ has rank s (respectively if the reachability matrix $(B, AB, \ldots, A^{s-1}B) \in \mathbb{R}^{s \times ns}$ has rank s). It is a fundamental result in the theory of state space models that a state space model is minimal if and only the model is both observable and reachable. The minimal state dimension is equal to the rank of the Hankel matrix H of the impulse response coefficients (k_j).

The `dse` tools `observability` and `reachability` compute the singular values of the observability respectively of the reachability matrices.

```
> svO = dse::observability(ss)
> svR = dse::reachability(ss)
Singular values of reachability matrix for noise:  0.7443343 0.4589284 0.3712107 0.2409465 2.989818e-17
```

```
> signif(rbind(svO,svR),4)
      [,1]   [,2]   [,3]   [,4]      [,5]      [,6]
svO 2.9990 2.6280 1.7220 0.9879 9.206e-01 4.450e-01
svR 0.7443 0.4589 0.3712 0.2409 2.990e-17 1.083e-18
```

Inspecting these singular values shows that the model is not reachable and hence is *not minimal*. This observation is also verified by the fact that the state variance P is not regular. The eigenvalues of P are

```
> P = lyap(ss$F, ss$K %*% sigma %*% t(ss$K)) # compute the state variance P
> eigen(P, only.values = TRUE)$values
[1]  4.211826e-01  1.233055e-01  7.788831e-02  3.473577e-02
[5] -7.009378e-19 -1.415094e-17
```

One possibility to achieve a minimal model is to use a "balancing and truncation" scheme. The `dse` package offers the function `balanceMittnik(model,n)` to this end. The (optional) parameter `n` is the desired state dimension. Here we use `n=4` based on the fact that only 4 of the reachability singular values are significantly greater than zero. The following code computes a (minimal) state space model with a state space dimension 4 and checks that this model really is an equivalent description of the process (y_t):

```
> ssb = dse::balanceMittnik(ss, n=4)
> ssb

 F =
           [,1]       [,2]       [,3]       [,4]
[1,] -0.1327857  0.1813460  0.3120305  0.3311117
[2,] -0.7805031  0.1090193  0.2904374 -0.1461123
[3,]  0.0715883 -0.6736422  0.6819328  0.2285732
[4,] -0.2674132 -0.4954136 -0.3930425 -0.2987692

 H =
            [,1]        [,2]       [,3]         [,4]
[1,]  0.02484203  0.07963300 -0.4383435  0.184895023
[2,] -0.56526277  0.02998341 -0.2627269  0.183926049
[3,] -0.19050836 -0.45709445 -0.3611779 -0.001188339

 K =
            [,1]       [,2]       [,3]
[1,] -0.76035926  0.2878963  0.1622077
[2,]  0.01509112 -0.1126601  0.2372431
[3,] -0.21419099 -0.1738510 -0.1596715
[4,] -0.04079621  0.2276468  0.3100379
```

```
> k.ss = SSirf(ss, Sigma = sigma, lag.max = 12, orth = TRUE)
```

```
> all.equal(k, k.ss, check.attributes = FALSE)
[1] TRUE
```

To check the stability assumption we may use the function `stability`. For the miniphase assumption there is no corresponding tool. However, it is not difficult to check this assumption "manually" by computing the eigenvalues of $(A - BC)$ (respectively using the `dse` notation of $(F - KH)$):

```
> dse::stability(ssb)
The system is stable.
[1] TRUE
attr(,"roots")
          Eigenvalues of F       moduli
[1,] -0.2859941+0.5749215i 0.6421272+0i
[2,] -0.2859941-0.5749215i 0.6421272+0i
[3,]  0.4656927+0.3115639i 0.5603051+0i
[4,]  0.4656927-0.3115639i 0.5603051+0i
```

```
> lambda = eigen(ssb$F - ssb$K %*% ssb$H)$values
> cat(ifelse((max(abs(lambda))<1),
+            'The system is strictly miniphase\n',
+            'The system is not strictly miniphase\n'))
The system is strictly miniphase
```

Even minimal systems are not identifiable. There is still the freedom to apply a state space transformation $x_t \longrightarrow Tx_t$, where $T \in \mathbb{R}^{s \times s}$ is non singular. The state parameter matrices then are transformed as

$$\begin{array}{rcl} A & \rightarrow & TAT^{-1} \\ B & \rightarrow & TB \\ C & \rightarrow & CT^{-1} \end{array}$$

See also the function `dse::gmap`.

Analogously to the VARMA case one may define an echelon canonical form for state space models. The parameter matrices have certain elements which are restricted to be one or zero and the position of these restricted elements again are determined by the Kronecker indices of the Hankel matrix of the impulse response coefficients. The utility function `impresp2SS` computes the state space model in echelon form for a given impulse response function. (This provides an alternative to construct a (unique, minimal) state space model which is equivalent to a given VARMA model.)

```
> sse = impresp2SS(k$psi, type = 'echelon')$ss
> sse

 F =
       [,1]   [,2]   [,3]  [,4]
[1,]  0.000  0.000  0.000 1.000
[2,] -0.142 -0.470  0.543 1.199
[3,]  0.920 -0.775  0.064 0.638
[4,] -0.074  0.137 -0.313 0.762

 H =
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0

 K =
         [,1]      [,2]     [,3]
[1,] 0.068000  0.116000  0.15000
```

```
[2,] 0.479532 -0.077916  0.01485
[3,] 0.215384  0.059008 -0.08230
[4,] 0.193816 -0.043608  0.03230
```

```
> k.ss = SSirf(sse, Sigma = sigma, lag.max = 12, orth = TRUE)
> all.equal(k, k.ss, check.attributes = FALSE)
[1] TRUE
```

For this model the number of "free parameters" is 24.

4 Estimation

4.1 Data

In order to illustrate the estimation of VARMA and state space models here we use a data set from the "FRED (Federal Reserve Economic Data)" database (https://fred.stlouisfed.org). We use the following three quarterly time series series:

- `DPIC96` Real Disposable Personal Income
- `GPDIC1` Real Gross Private Domestic Investment
- `PCECC96` Real Personal Consumption Expenditures

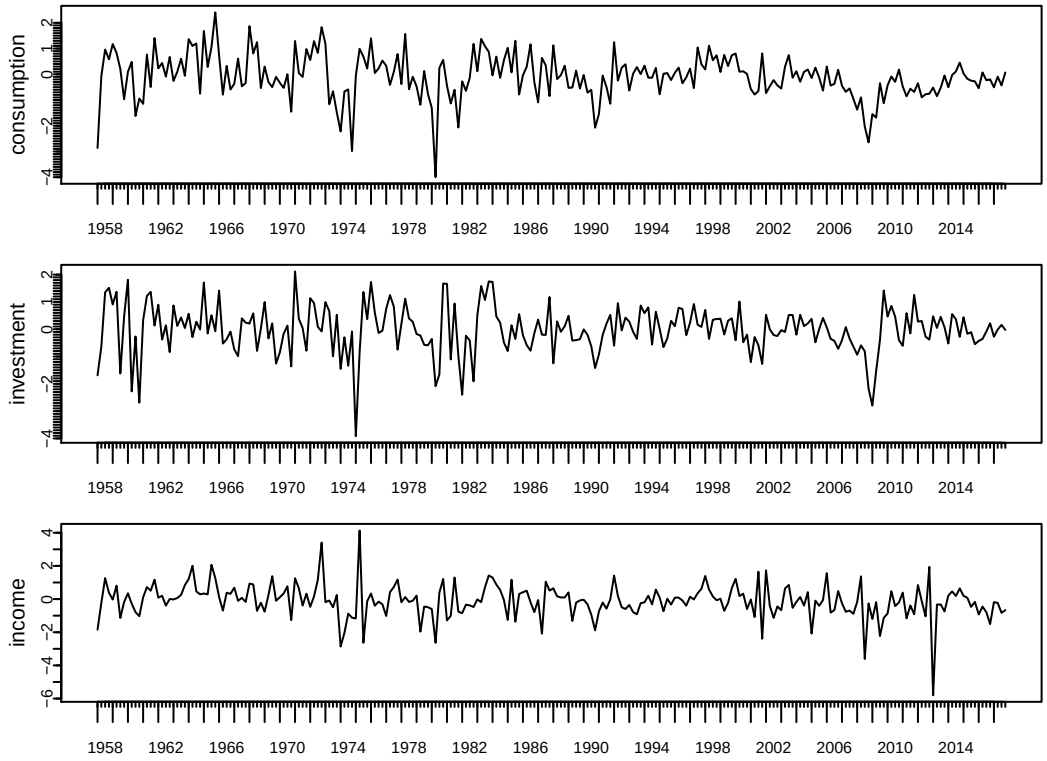
The variables are measured in Billions of Chained 2009 Dollars.

We consider the quarterly growth rates (i.e. the differences of the log values) and demean and scale the growth rates such that the sample variance is equal to one. The whole date set is split into two parts: The data from 1985 to 1991 is used for the estimation of the models and the data from 1992 to the end of 2017 is used for the comparison of the models (in terms of their predictive power).

We use a matrix `y` to store the data (for estimation with the `MTS` tools) and a `TSdata` object which is used for the estimation by `dse` tools.

```
> # read the data into an R data.frame object
> data = read.delim(file = 'prCoIn_Quarterly.txt',header=TRUE)
> data$DATE = as.Date(data$DATE)
>
> # compute the quarterly growth rates
> data$consumption = c(NA,diff(log(data$PCECC96)))
> data$investment = c(NA,diff(log(data$GPDIC1)))
> data$income = c(NA,diff(log(data$DPIC96)))
>
> # skip the data before 1958
> d = as.POSIXlt(data$DATE)
> data = data[d >= as.POSIXlt('1958-01-01'), ]
> d = d[d >= as.POSIXlt('1958-01-01')]
>
> # collect the time series to be analyzed in the matrix "y"
> y = cbind(data$consumption,data$investment,data$income)
> colnames(y) = c('consumption','investment','income')
> rownames(y) = paste(format(data$DATE,'%Y'),
+                     quarters(data$DATE),sep=' ')
>
> n = ncol(y)      # number of variables
> T.obs = nrow(y)  # total sample size
> T.est = sum(d < as.POSIXlt('1992-01-01')) # estimation sample
>
```

```
> # demean and scale the data
> y = scale(y, center = colMeans(y[1:T.est,]),
+               scale =  sqrt((T.est-1)/T.est)*apply(y[1:T.est,],
+                                       MARGIN = 2, FUN = sd))
>
> data.start = c(1900 + d[1]$year, d[1]$mon %/% 3 +1)
> data.end = c(1900 + d[T.obs]$year, d[T.obs]$mon %/% 3 +1)
> est.end  = c(1900 + d[T.est]$year, d[T.est]$mon %/% 3 +1)
>
> # construct "dse" TSdata objects
> sample = dse::TSdata(output = y)
> sample = tframed(sample, list(start=data.start, frequency=4))
> estimation.sample = tfwindow(sample, end = est.end)
>
> # plot the data
> par(oma = c(0,0,0,0), mar = c(2,2,1,0)+0.1, tcl = -0.2,
+     mgp = c(1.25, 0.15, 0), cex.main = 1, cex.axis = 0.75)
> tfplot(sample)
```




```
>
> cat('(quarterly) data: ', paste(colnames(y), collapse=','), '\n',
+     'total sample:       ', rownames(y)[1],'-',rownames(y)[T.obs], '  ',
+     T.obs, ' observations\n',
+     'estimation sample:  ', rownames(y)[1],'-',rownames(y)[T.est], '  ',
+     T.est, ' observations\n',
+     'validation sample:  ', rownames(y)[T.est+1],'-',rownames(y)[T.obs], '  ',
+     T.obs-T.est, ' observations\n',sep='')
(quarterly) data: consumption,investment,income
total sample:       1958 Q1-2017 Q4  240 observations
```

```
estimation sample:  1958 Q1-1991 Q4  136 observations
validation sample:  1992 Q1-2017 Q4  104 observations
```

4.2 Evaluate and Forecast

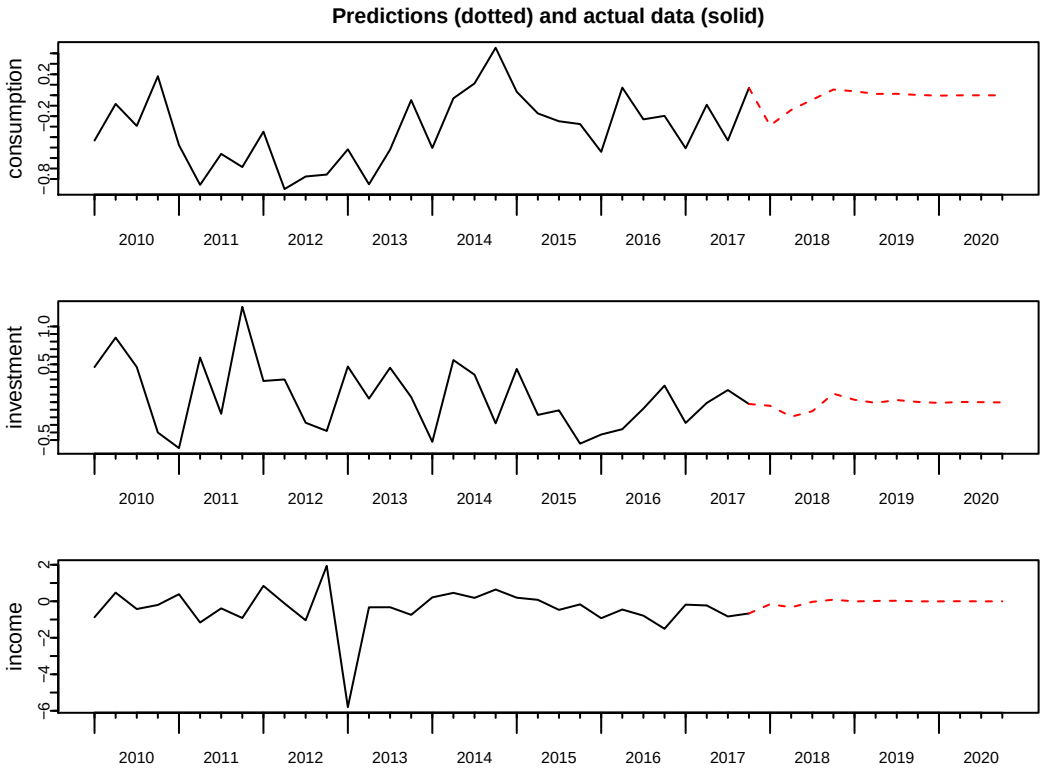
In order to evaluate the "fit" of a given model on a data set we may use the `dse::l()` function which in particular computes the (log) likelihood of the model. The output of this command is a `TSestModel` which is an object which stores the model, the data and the estimation results.

```
> arma = l(arma, estimation.sample)
> summary(arma)
neg. log likelihood = 506.1354    sample length = 136
     consumption investment    income
RMSE   0.9304949   1.232342 0.7825009
ARMA:
inputs :
outputs:  consumption investment income
     input  dimension =  0     output dimension =  3
     order A =  2     order B =  2     order C =
      26  actual parameters      6  non-zero constants
     trend not estimated.
```

Forecasts may be computed by the `dse` functions `forecast`, `horizonForecasts` and `featherForecasts`. The function `dse::forecast` computes the out-of-sample forecasts for a given model and sample. In the example below we compute the forecasts for 2018-2020, i.e. for forecast horizons $h = 1, 2, \ldots, 12$. The corresponding `MTS` function (for estimated VARMA models) is `MTS::VARMApred`.

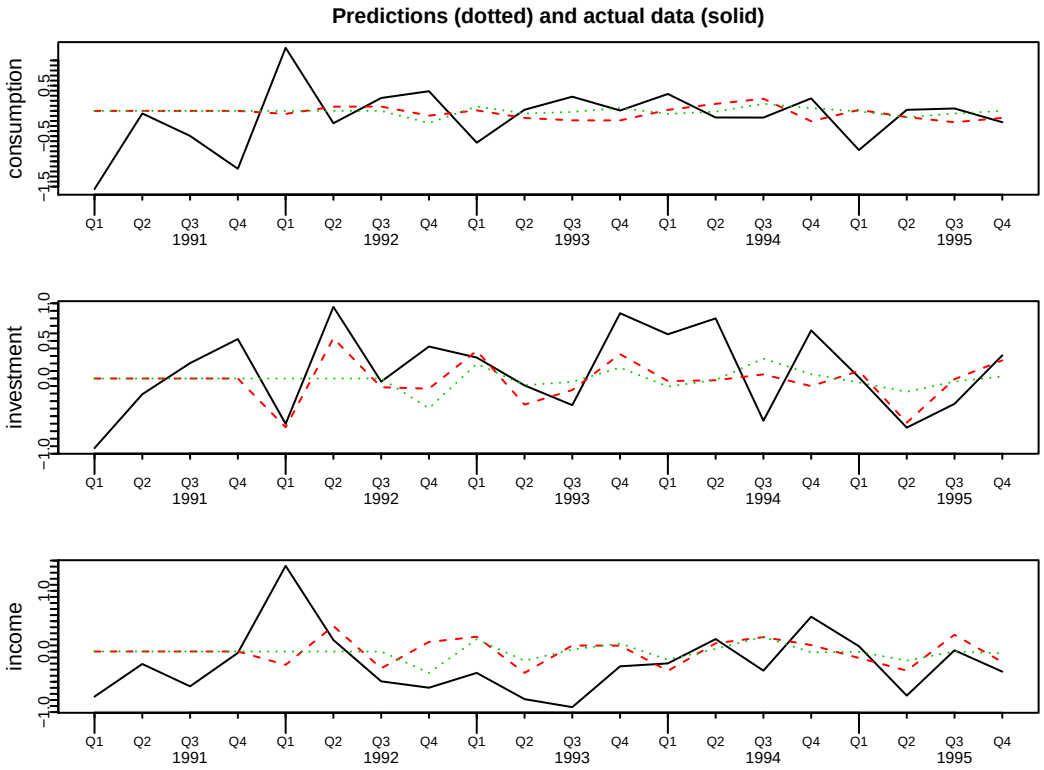
The function `dse::horizonForecasts` computes the h-step ahead forecasts for all time points within a sample. The forecasts are "aligned" with the original data, such that it is easy to compute the forecast errors. In the code below we use the state space model `sse` (which is equivalent to the VARMA model `arma`) in order to show that the syntax is independent of the object class. The optional parameter `discard.before` means that predictions based on data up to this time point should be discarded (i.e. are set to zero).

```
> # compute the "out-of-sample" forecast for 3 years
> z = suppressWarnings(forecast(arma,data=sample,horizon=4*3))
>
> par(oma = c(0,0,0,0), mar = c(2,2,2,0)+0.1, tcl = -0.2,
+     mgp = c(1.25, 0.15, 0), cex.main = 1, cex.axis = 0.75)
> tfplot(z,start=c(2010,1)) # plot the end of the sample
```




```
>
> # extract the "out-of-sample" forecast for 2018
> window(z$forecast[[1]], end=c(2018,4))
          Series 1    Series 2    Series 3
2018 Q1 -0.28900612 -0.04738126 -0.15876220
2018 Q2 -0.14087985 -0.19181475 -0.32920728
2018 Q3 -0.04276265 -0.11987410 -0.02930487
2018 Q4  0.05460322  0.11196985  0.08652213

>
> z = suppressWarnings(horizonForecasts(sse, sample,
+             horizons = c(1,4), discard.before = T.est))
> # plot a subsample
> tfplot(z, start = c(1991,1), end = c(1995,4))
```




```
>
> # extract the forecasts/errors for "income"
> junk = cbind(sample$output[,3], t(z$horizonForecasts[,,3]),
+        sample$output[,c(3,3)] - t(z$horizonForecasts[,,3]))
> colnames(junk) = c(colnames(z$data$output)[3],
+        t(outer(c('pred h=','err h='),c(1,4),paste,sep='')))
> rownames(junk) = rownames(y)
> round(junk[(T.est-3):(T.est+8),],4)
         income pred h=1 pred h=4 err h=1 err h=4
1991 Q1 -0.7447   0.0000   0.0000 -0.7447 -0.7447
1991 Q2 -0.2067   0.0000   0.0000 -0.2067 -0.2067
1991 Q3 -0.5754   0.0000   0.0000 -0.5754 -0.5754
1991 Q4 -0.0239   0.0000   0.0000 -0.0239 -0.0239
1992 Q1  1.4128  -0.2206   0.0000  1.6334  1.4128
1992 Q2  0.1890   0.4206   0.0000 -0.2316  0.1890
1992 Q3 -0.4929  -0.2781   0.0000 -0.2148 -0.4929
1992 Q4 -0.5981   0.1570  -0.3521 -0.7550 -0.2460
1993 Q1 -0.3536   0.2433   0.2014 -0.5970 -0.5550
1993 Q2 -0.7867  -0.3533  -0.1527 -0.4334 -0.6340
1993 Q3 -0.9186   0.0992   0.0316 -1.0178 -0.9502
1993 Q4 -0.2453   0.0967   0.1324 -0.3421 -0.3778
```

4.3 Estimate Models

4.3.1 Estimate VAR Models

Of course it is easy to estimate autoregressive models with both packages. Here we use `dse::estVARar` which is based on the Yule-Walker equations. The order is estimated by the AIC information criterion. (The

estimated order here is $p = 2$.) This VAR model will serve as a kind of benchmark model.

```
> model.VAR =  dse::estVARXar(estimation.sample, aic = TRUE)
> summary(model.VAR)
neg. log likelihood = 510.5801    sample length = 136
     consumption investment   income
RMSE   0.9402565  0.8669979 0.930077
ARMA: model estimated by estVARXar
inputs :
outputs:  consumption investment income
     input  dimension =  0     output dimension =  3
     order A =  2     order B =  0     order C =
      18  actual parameters      6  non-zero constants
     trend not estimated.
```

4.3.2 Estimate State Space Models

The `dse` package offers a number of functions to estimate state space models. The package author suggest to use `bft` (brute force technique), so we we follow this advice. The main idea of this technique (which is a particular *subspace algorithm*) is as follows. First estimate a "long" AR model, i.e. with a high order p. Next convert this model to a state space model and then use a model reduction technique to construct a state space model of the desired order s. This procedure is repeated for a number of AR orders p and state space dimensions s and finally the model which is the best in terms of an information criterion is returned.

The following `R` code estimates two state space models, the first one uses "BIC" as selection criterion (and returns a state model of order $s = 1$) and the second one uses "AIC" (which results in $s = 3$).

```
> model.BFTbic = dse::bft(estimation.sample,
+                         criterion = 'tbic', verbose = FALSE)
> summary(model.BFTbic)
neg. log likelihood = 531.3362    sample length = 136
     consumption investment    income
RMSE   0.9710178  0.8961374 0.9726227
innovations form state space: nested model a la Mittnik
inputs :
outputs:  consumption investment income
   input  dimension =  0   state  dimension =  1   output dimension =  3
   theoretical parameter space dimension =  6
   7  actual parameters    0  non-zero constants
   Initial values not specified.

>
> model.BFTaic = dse::bft(estimation.sample,
+                         criterion = 'taic', verbose = FALSE)
> summary(model.BFTaic)
neg. log likelihood = 513.1865    sample length = 136
     consumption investment    income
RMSE   0.9312625  0.8933566 0.9239323
innovations form state space: nested model a la Mittnik
inputs :
outputs:  consumption investment income
   input  dimension =  0   state  dimension =  3   output dimension =  3
   theoretical parameter space dimension =  18
   27  actual parameters    0  non-zero constants
   Initial values not specified.
```

Next we try to enhance these models with maximum likelihood. The corresponding function is `dse::estMaxLik` which simply takes an "initial" model as main argument. Note that both `model.BFTbic` and `model.BFTaic` are `TSestModel` objects and thus contain the (estimation) data.

```
> model.BFTbicML = estMaxLik(model.BFTbic)
> summary(model.BFTbicML)
neg. log likelihood = 528.7699    sample length = 136
     consumption investment    income
RMSE   0.9584305  0.8920208 0.9712616
innovations form state space: Estimated with max.like/optim  ( converged ) from initial model:  nested m
inputs :
outputs:  consumption investment income
   input  dimension =  0   state  dimension =  1   output dimension =  3
   theoretical parameter space dimension =  6
   7  actual parameters    0  non-zero constants
   Initial values not specified.

>
> model.BFTaicML = estMaxLik(model.BFTaic)
> summary(model.BFTaicML)
neg. log likelihood = 509.5183    sample length = 136
     consumption investment    income
RMSE   0.9301279  0.8753634 0.9262648
innovations form state space: Estimated with max.like/optim  ( converged ) from initial model:  nested m
inputs :
outputs:  consumption investment income
   input  dimension =  0   state  dimension =  3   output dimension =  3
   theoretical parameter space dimension =  18
   27  actual parameters    0  non-zero constants
   Initial values not specified.
```

4.3.3 Estimation of VARMA Models

The function `dse::estMaxLik` can also deal with VARMA models. However, one first has to find an appropriate initial estimate and we could not a find a suitable `dse` function for this purpose. (Of course one could first estimate a state space model and then convert this to a VARMA model.) `estMaxLik` uses a simple (and flexible) scheme to deal with constraints. It simply treats coefficients (of the initial model) which are zero or one as fixed. One may also impose more complicated constraints with the function `dse::fixConstants`. However, it is not clear how one could impose a constraint like $a_0 = b_0$ and thus it is not possible to estimate VARMA model in echelon canonical form.

For these reason we use the `MTS` package for estimation of VARMA models. A VARMA(p,q) model (without further structure) may be estimated with `MTS::VARMA` (or `MTS::VARMAcpp` where the computation of the likelihood is implemented in C++). The initial estimate is computed by the HRK algorithm. The output of this function is a list which in particular contains the estimated AR (`$Phi`) and MA (`$Theta`) parameters. In order to be able to easily compare this ARMA model with the above estimated state space models, we convert the result of `MTS::VARMA` to a `dse::TSestModel` object.

```
> out = MTS::VARMACpp(y[1:T.est,], p=1, q=1,
+                     include.mean = FALSE, details = FALSE)
Number of parameters:  18
initial estimates:  0.3866 0.2347 -0.1595 0.4197 -0.0479 0.2851 0.7382 -0.2265 -0.1591 -0.3344 -0.2207 0
Par. lower-bounds:  -0.0731 -0.105 -0.5698 -0.0226 -0.3748 -0.1097 0.2556 -0.5831 -0.5899 -0.8385 -0.619
Par. upper-bounds:  0.8462 0.5743 0.2508 0.862 0.279 0.6799 1.2207 0.1301 0.2717 0.1696 0.1777 0.9228 0
Final   Estimates:  0.2272542 0.305018 0.2507603 0.2214837 0.0851363 -0.1097409 0.255627 0.1300933 -0.39
```

```
Warning in sqrt(diag(solve(Hessian))): NaNs wurden erzeugt

Coefficient(s):
             Estimate  Std. Error  t value Pr(>|t|)
consumption   0.22725     0.21633    1.050   0.2935
investment    0.30502     0.16703    1.826   0.0678 .
income        0.25076     0.17564    1.428   0.1534
consumption   0.22148          NA       NA       NA
investment    0.08514          NA       NA       NA
income       -0.10974     0.11571   -0.948   0.3429
consumption   0.25563     0.26977    0.948   0.3433
investment    0.13009          NA       NA       NA
income       -0.39245     0.22162   -1.771   0.0766 .
             -0.15836     0.25570   -0.619   0.5357
              0.17765     0.18415    0.965   0.3347
              0.02029     0.16094    0.126   0.8997
             -0.23158     0.10339   -2.240   0.0251 *
             -0.18158          NA       NA       NA
             -0.02620     0.11179   -0.234   0.8147
             -0.03042     0.20729   -0.147   0.8833
             -0.16413          NA       NA       NA
              0.36665     0.19469    1.883   0.0597 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
---
Estimates in matrix form:
AR coefficient matrix
AR( 1 )-matrix
      [,1]   [,2]   [,3]
[1,] 0.227 0.3050  0.251
[2,] 0.221 0.0851 -0.110
[3,] 0.256 0.1301 -0.392
MA coefficient matrix
MA( 1 )-matrix
       [,1]   [,2]    [,3]
[1,] 0.1584 -0.178 -0.0203
[2,] 0.2316  0.182  0.0262
[3,] 0.0304  0.164 -0.3666

Residuals cov-matrix:
           [,1]       [,2]      [,3]
[1,] 1.02815846 0.06541776 0.4179541
[2,] 0.06541776 1.11927252 0.2405532
[3,] 0.41795407 0.24055319 0.9005358
----
aic=  0.02975727
bic=  0.4152557

>
>
> model.ARMA11 = PhiTheta2ARMA(out$Phi, out$Theta,
+                   output.names = seriesNames(sample)$output)
> model.ARMA11 = dse::l(model.ARMA11, estimation.sample)
>
```

```
> summary(model.ARMA11)
neg. log likelihood = 566.4133    sample length = 136
     consumption investment    income
RMSE    1.039683   1.063948 0.9585755
ARMA:
inputs :
outputs:  consumption investment income
     input  dimension =  0     output dimension =  3
     order A =  1     order B =  1     order C =
      18  actual parameters      6  non-zero constants
     trend not estimated.
```

Next we estimate VARMA models in echelon canonical form (with `MTS::Kronfit`) for Kronecker indices $\nu = (1, 0, 0)$, $\nu = (1, 1, 0)$, $\nu = (1, 1, 1)$, and $\nu = (2, 1, 1)$. The output of these computations is quite lengthy and thus we don't include it here.

```
> out = suppressWarnings(MTS::Kronfit(da = y[1:T.est,],
+               kidx = c(1,0,0), include.mean = FALSE))
> model.ARMA100 = PhiTheta2ARMA(out$Phi, out$Theta, out$Ph0,
+       fix = T, output.names = seriesNames(sample)$output)
> model.ARMA100 = dse::l(model.ARMA100, estimation.sample)
> summary(model.ARMA100)
>
> out = suppressWarnings(MTS::Kronfit(da = y[1:T.est,],
+               kidx = c(1,1,0), include.mean = FALSE))
> model.ARMA110 = PhiTheta2ARMA(out$Phi, out$Theta, out$Ph0,
+       fix = T, output.names = seriesNames(sample)$output)
> model.ARMA110 = dse::l(model.ARMA110, estimation.sample)
> summary(model.ARMA110)
>
> out = suppressWarnings(MTS::Kronfit(da = y[1:T.est,],
+               kidx = c(1,1,1), include.mean = FALSE))
> model.ARMA111 = PhiTheta2ARMA(out$Phi, out$Theta, out$Ph0,
+       fix = T, output.names = seriesNames(sample)$output)
> model.ARMA111 = dse::l(model.ARMA111, estimation.sample)
> summary(model.ARMA111)
>
> out = suppressWarnings(MTS::Kronfit(da = y[1:T.est,],
+               kidx = c(2,1,1), include.mean = FALSE))
> model.ARMA211 = PhiTheta2ARMA(out$Phi, out$Theta, out$Ph0,
+       fix = T, output.names = seriesNames(sample)$output)
> model.ARMA211 = dse::l(model.ARMA211, estimation.sample)
> summary(model.ARMA211)
```

Note that the VARMA model which has been used throughout the first two sections of this demonstration is a "rounded" version of the last model estimated, i.e. the model for $\nu = (2, 1, 1)$.

```
> all.equal(list(Ph0=phi0,Phi=phi,Theta=theta),
+           lapply(out[c('Ph0','Phi','Theta')],round,3))
[1] "Component \"Phi\": Mean relative difference: 0.004395604"
```

4.4 Compare Models

The `dse` package provides some nice tools to compare and evaluate a set of estimated models. We start with evaluation the "in-sample" performance of the models with `dse::informationTests`.

```
> info = dse::informationTests(model.VAR, model.BFTbic,
+             model.BFTbicML, model.BFTaic, model.BFTaicML,
+             model.ARMA11, model.ARMA100, model.ARMA110,
+             model.ARMA111, model.ARMA211)
                    based on no.of parameters     based on theoretical parameter dim.
       PORT  -ln(L)  AIC   BIC   GVC  RICE   FPE   AIC   BIC   GVC   RICE   FPE
 [1]  129.6  510.6 1057.2 1129.4 1058.0 1058.8 1057.2     NA     NA     NA     NA     NA

 [1]  144.0  531.3 1076.7 1104.8 1076.8 1076.9 1076.7 1074.7 1098.7 1074.8 1074.9 1074.7

 [1]  148.7  528.8 1071.5 1099.6 1071.7 1071.8 1071.5 1069.5 1093.6 1069.6 1069.7 1069.5

 [1]  123.7  513.2 1080.4 1188.7 1082.2 1084.3 1080.5 1062.4 1134.6 1063.2 1064.1 1062.4

 [1]  124.7  509.5 1073.0 1181.3 1074.9 1077.0 1073.1 1055.0 1127.2 1055.9 1056.7 1055.1

 [1]  249.3  566.4 1168.8 1241.0 1169.6 1170.5 1168.9     NA     NA     NA     NA     NA

 [1]  156.1  529.5 1070.9 1095.0 1071.0 1071.1 1070.9     NA     NA     NA     NA     NA

 [1]  138.6  519.4 1062.9 1111.0 1063.2 1063.6 1062.9     NA     NA     NA     NA     NA

 [1]  124.6  509.5 1055.0 1127.2 1055.9 1056.7 1055.1     NA     NA     NA     NA     NA

 [1]  121.4  506.1 1060.3 1156.5 1061.7 1063.3 1060.3     NA     NA     NA     NA     NA

opt     10       10       9       7       9       9       9       5       3       5       5       5
  PORT  - Portmanteau test                        -ln(L)- neg. log likelihood
  AIC   - neg. Akaike Information Criterion      BIC   - neg. Bayes  Information Criterion
  GVC   - Generalized Cross Validation           RICE  - Rice Criterion
  FPE   - Final Prediction Error
 WARNING - These calculations do not account for trend parameters in ARMA models.
```

The information criteria of course heavily depend on the number of "free" parameters of the respective models. As noted above the default strategy of `dse` is to consider the coefficients of the parameter matrices which are not equal to one or zero as "free" and to consider the zero/one coefficients as "fixed". The `dse::summary(model)` command prints this number of "free" parameters as `actual number of parameters`.

- For a general state space model this gives $2ns + s^2$ parameters. However, this does not account for the fact that the parameter matrices are only unique up to state space transformations. Therefore `dse::informationTests` also uses the so called "theoretical number of parameters": $2ns$. In the summary table below we report the information criteria based on this "theoretical number of parameters".

- For VARMA models in echelon form the "actual number of parameters" is not correct since it does not account for the constraint $a_0 = b_0$.
  The function `dse::fixConstants` allows to set any coefficient as "fixed" or as "free". In order to force `dse::informationTests` to use the correct number of free parameters $2n(\nu_1 + \cdots + \nu_n)$ for a model in echelon form, we use `dse::fixConstants` and "fix" all zero/one coefficients **and** all entries of b_0. This is accomplished in the code above by calling `PhiTheta2ARMA` with the optional argument `fix=TRUE`.

The following code extracts the relevant information criteria and produces a `LaTeX` table.

```
> library(kableExtra)
> estimates = c('VAR',
+               'BFTbic', 'BFTbicML', 'BFTaic', 'BFTaicML',
+               'ARMA11', 'ARMA100', 'ARMA110', 'ARMA111', 'ARMA211')
```

Table 1: In-sample (information) criteria of the estimated models: **#par** - number of parameters, **port** - Portmanteau test, **like** - neg. log likelihood, **aic** - Akaike Information Criterion, **bic** - Bayes Information Criterion, **gvc** - Generalized Cross Validation, **rice** - Rice Criterion, **fpe** - Final Prediction Error.

	#par	port	like	aic	bic	gvc	rice	fpe
VAR	18	129.6	510.6	1057.2	1129.4	1058	1058.8	1057.2
BFTbic	6	144	531.3	1074.7	1098.7	1074.8	1074.9	1074.7
BFTbicML	6	148.7	528.8	1069.5	1093.6	1069.6	1069.7	1069.5
BFTaic	18	123.7	513.2	1062.4	1134.6	1063.2	1064.1	1062.4
BFTaicML	18	124.7	509.5	1055	1127.2	1055.9	1056.7	1055.1
ARMA11	18	249.3	566.4	1168.8	1241	1169.6	1170.5	1168.9
ARMA100	6	156.1	529.5	1070.9	1095	1071	1071.1	1070.9
ARMA110	12	138.6	519.4	1062.9	1111	1063.2	1063.6	1062.9
ARMA111	18	124.6	509.5	1055	1127.2	1055.9	1056.7	1055.1
ARMA211	24	121.4	506.1	1060.3	1156.5	1061.7	1063.3	1060.3

```
> n.estimates = length(estimates)
> n.par = integer(n.estimates)
> names(n.par) = estimates
> n.par['VAR'] = (n^2) * (dim(model.VAR$model$A)[1]-1)
> n.par['BFTbic']   = 2*n* nrow(model.BFTbic$model$F)
> n.par['BFTbicML'] = 2*n* nrow(model.BFTbicML$model$F)
> n.par['BFTaic']   = 2*n* nrow(model.BFTaic$model$F)
> n.par['BFTaicML'] = 2*n* nrow(model.BFTaicML$model$F)
> n.par['ARMA11'] = (n^2)*2
> n.par['ARMA100'] = 2*n*1
> n.par['ARMA110'] = 2*n*2
> n.par['ARMA111'] = 2*n*3
> n.par['ARMA211'] = 2*n*4
>
> # for the state space models use the "theoretical number of paramaters"
> info1 = info[,1:7]
> info1[2:5,3:7] = info[2:5,8:12]
>
> junk = data.frame(n.par,info1)
> rownames(junk) = estimates
> colnames(junk)[1] = '\\#par'
> for (i in 2:ncol(junk)) {
+   x = junk[[i]]
+   is.min = (x == min(x))
+   x = round(x,1)
+   junk[[i]] = cell_spec(x, background = ifelse(is.min,"#90EE90","white"))
+ }
> kable(junk, caption = "In-sample (information) criteria of the estimated models:
+ \\textbf{\\#par} - number of parameters, \\textbf{port} - Portmanteau test,
+ \\textbf{like} - neg. log likelihood, \\textbf{aic} - Akaike Information Criterion,
+ \\textbf{bic} - Bayes  Information Criterion, \\textbf{gvc} - Generalized Cross Validation,
+ \\textbf{rice}  - Rice Criterion,  \\textbf{fpe} - Final Prediction Error.", digits = 1,
+       escape = F, booktabs = T, linesep = "") %>%
+   row_spec(0, bold = TRUE)
```

The "out-of-sample" performance of these model is evaluated by considering the 1-step ahead prediction

errors.

```
> z = dse::forecastCov(model.VAR, model.BFTbic,
+              model.BFTbicML, model.BFTaic, model.BFTaicML,
+              model.ARMA11, model.ARMA100, model.ARMA110,
+              model.ARMA111, model.ARMA211, data = sample,
+              horizons = 1:4, discard.before = T.est)
>
> # extract MSE for each series and the total MSE
> mse = array(0, dim = c(ncol(y)+1,4,n.estimates),
+             dimnames = list(c(colnames(y),'total'),
+                             paste('h=',1:4,sep=''), estimates))
> for (k in (1:n.estimates)) {
+   for (h in (1:4)) {
+     mse[,h,k] = c(diag(z$forecastCov[[k]][h,,]),
+                   sum(diag(z$forecastCov[[k]][h,,])))
+ }}
> mse = aperm(mse,c(2,3,1))
> round(mse,4)
, , consumption

       VAR BFTbic BFTbicML BFTaic BFTaicML ARMA11 ARMA100 ARMA110 ARMA111 ARMA211
h=1 0.2828 0.3570   0.3082 0.2466   0.2596 0.2774  0.2812  0.2950  0.2595  0.2730
h=2 0.2764 0.3501   0.3020 0.2430   0.2546 0.2695  0.2746  0.2903  0.2546  0.2619
h=3 0.3213 0.3778   0.3463 0.2706   0.2970 0.3397  0.3183  0.3165  0.2970  0.3045
h=4 0.3710 0.3989   0.3830 0.3067   0.3374 0.3811  0.3601  0.3851  0.3372  0.3549

, , investment

       VAR BFTbic BFTbicML BFTaic BFTaicML ARMA11 ARMA100 ARMA110 ARMA111 ARMA211
h=1 0.2909 0.3476   0.3039 0.3300   0.2990 0.4578  0.3388  0.2770  0.2997  0.3549
h=2 0.2886 0.3428   0.3007 0.3259   0.2973 0.4591  0.3117  0.2751  0.2980  0.3213
h=3 0.3858 0.3889   0.3803 0.3967   0.4056 0.3878  0.3840  0.3815  0.4055  0.4015
h=4 0.4019 0.3939   0.3927 0.4028   0.4166 0.3951  0.3950  0.3920  0.4163  0.4228

, , income

       VAR BFTbic BFTbicML BFTaic BFTaicML ARMA11 ARMA100 ARMA110 ARMA111 ARMA211
h=1 1.0064 0.9881   0.9886 1.0167   0.9871 0.9856  1.0061  0.9935  0.9893  0.9729
h=2 1.0158 0.9974   0.9980 1.0242   0.9954 0.9949  1.0097  1.0013  0.9977  0.9555
h=3 1.0408 1.0269   1.0161 0.9703   1.0020 1.0040  1.0027  1.0359  1.0026  1.0194
h=4 1.0053 1.0544   1.0467 0.9800   1.0014 1.0546  1.0368  1.0568  1.0018  1.0222

, , total

       VAR BFTbic BFTbicML BFTaic BFTaicML ARMA11 ARMA100 ARMA110 ARMA111 ARMA211
h=1 1.5800 1.6926   1.6007 1.5933   1.5456 1.7208  1.6261  1.5654  1.5486  1.6007
h=2 1.5808 1.6902   1.6007 1.5931   1.5473 1.7235  1.5960  1.5668  1.5503  1.5386
h=3 1.7479 1.7936   1.7426 1.6376   1.7045 1.7315  1.7050  1.7339  1.7051  1.7254
h=4 1.7783 1.8472   1.8224 1.6895   1.7553 1.8309  1.7918  1.8339  1.7552  1.7999
```

The following code extracts the MSE of the one-step ahead predictions produces a LaTeX table.

```
> junk = data.frame(mse[1,,])
> junk$rVAR = (1 -junk$total / junk$total[1]) * 100
```

Table 2: This table shows the (out-of-sample) mean squared errors of the estimated models for the 1-step ahead prediction. The column `total` is the sum of the MSE values for the three series (consumption, investment and income). The last column reports the percentage improvement as compared to the VAR model.

	consumption	investment	income	total	rVAR
VAR	0.283	0.291	1.006	1.580	0.0%
BFTbic	0.357	0.348	0.988	1.693	-7.1%
BFTbicML	0.308	0.304	0.989	1.601	-1.3%
BFTaic	0.247	0.330	1.017	1.593	-0.8%
BFTaicML	0.260	0.299	0.987	1.546	2.2%
ARMA11	0.277	0.458	0.986	1.721	-8.9%
ARMA100	0.281	0.339	1.006	1.626	-2.9%
ARMA110	0.295	0.277	0.993	1.565	0.9%
ARMA111	0.260	0.300	0.989	1.549	2.0%
ARMA211	0.273	0.355	0.973	1.601	-1.3%

```
> rownames(junk) = estimates
> for (i in 1:ncol(junk)) {
+   x = junk[[i]]
+   if (i <ncol(junk)) {
+     is.min = (x == min(x))
+   } else {
+     is.min = (x == max(x))
+   }
+   if (i < ncol(junk)) {
+     x = sprintf('%1.3f',junk[[i]])
+   } else {
+     x = sprintf('%1.1f%%',junk[[i]])
+   }
+   junk[[i]] = cell_spec(x, background = ifelse(is.min,"#90EE90","white"))
+ }
> kable(junk, caption = "This table shows the (out-of-sample) mean squared
+ errors of the estimated models for the $1$-step ahead prediction.
+ The column \\texttt{total} is the sum of the MSE values for the
+ three series (consumption, investment and income). The last column
+ reports the percentage improvement as compared to the VAR model.",
+       align = 'r', escape = F, booktabs = T, linesep = "") %>%
+   row_spec(0, bold = TRUE)
```

4.4.1 Discussion and notes

- In-sample-performance:
  - The `ARMA211` model is the most complex model (24 free parameters) and thus it is no surprise that this model is optimal in terms of the likelihood.
  - The models `BFTaicML`, `ARMA11`, `ARMA111` are obtained by optimizing the likelihood over essentially the same set of models (VARMA$(1,1)$ or state space models with a state space dimension $s = 3$). Accordingly `BFTaicML`, `ARMA111` have the same likelihood value. However, `ARMA11` is much worse. This is an indication that the initial estimate is crucial for the ML estimation.
  - The models `BFTaicML`, `ARMA111` are the best models with respect to the information criteria. Only the BIC criterion picks the more parsimonious model `BFTbicML`.

- Out-of-sample performance, prediction quality:
  - The MSE values of the predictors are quite similar for most of the models (and time series). Therefore the ranking has to be interpreted with some care. For a careful analysis one should test whether the differences are "statistically significant", e.g. by a Diebold Mariano test.
  - The models `BFTaicML`, `ARMA111` are also the best models in terms of out-of-sample prediction.
  - By construction the ML estimates `BFTbicML`, `BFTaicML` yield better likelihood values than the corresponding initial estimates `BFTbic` and `BFTaic`. For the data considered here the ML estimates also give better predictions, i.e. there is a (small) performance gain.
  - The ARMA11 model performs badly.
- Of course the above notes cannot easily be generalized. The results heavily depend on the data considered.
- None of the above estimation schemes guarantees stable and miniphase models. So to be sure one should check the estimated models as described above.
- The maximum likelihood methods use a simplified version of the (negative log) likelihood.
- The computations have been carried out with the following versions: `R`: R version 3.4.4 (2018-03-15), `MTS`: 1.0, `dse`: 2015.12.1, `QZ`: 0.1.6, `kableExtra`: 0.9.0.

5 R-Tools

In this section we give a short description of the tools used in this R-demonstration.

5.1 `ARMA2PhiTheta`:

Extracts the AR/MA coefficients of a `dse::ARMA` object and returns these coefficient matrices in the style of the `MTS` package.

Usage

```
ARMA2PhiTheta = function(arma, normalizePhi0 = TRUE)
```

Arguments

`arma` `dse::ARMA` object.

`normalizePhi0` if TRUE then the lag zero coefficent matrix is normalized to the identity matrix and the other coefficients are correspondingly transformed.

Value

`Phi` $(n \times np)$ matrix containing the AR parameters $\phi = [\phi_1, \ldots, \phi_p]$.

`Theta` $(n \times np)$ matrix containing the MA parameters $\theta = [\theta_1, \ldots, \theta_q]$.

`Phi0` $(n \times n)$ matrix containing the lag zero coefficient matrix $\phi_0 = \theta_0$.

Examples

```
A = array(c(1, .5, .3, 0.4, .2, .1, 0, .2, .05, 1, .5, .3) ,c(3,2,2))
B = array(c(1, .2, 0.4, .1, 0, 0, 1, .3), c(2,2,2))
arma = dse::ARMA(A = A, B = B, C=NULL,
```

```
                output.names = paste('y', 1:2, sep = ''))
ARMA2PhiTheta(arma)
m = ARMA2PhiTheta(arma, normalizePhi0 = FALSE)
arma.test = PhiTheta2ARMA(m$Phi, m$Theta, m$Phi0)
all.equal(arma, arma.test)
```

5.2 `basis2kidx`: Kronecker indices

Determine the Kronecker indices, given the indices of the basis rows of the Hankel matrix of the impulse response function.

Usage

```
basis2kidx = function(basis, n)
```

Arguments

`basis`	(integer) vector with the indices of the basis rows of the Hankel matrix of the inmpulse response.
`n`	(integer) dimension of the process.

Value

n-dimensional (integer) vector with the Kronecker indices.

Examples

```
basis2kidx(c(1,2,3), 2)
basis2kidx(c(1,2,4), 2)
## Not run:
## basis2kidx(c(1,2,5),2) # this is not a 'nice' basis!
## End(Not run)
```

5.3 `fevd`: Forecast Error Variance Decomposition

Computes the Forecast Errors Variance Decomposition from a given orthogonalized impulse response function. It seems that the `MTS:FEVdec` implementation has a bug. Furthermore the orthogonalization scheme via a Cholesky decomposition of the covariance of the innovations is "hard coded". Therefore, here we offer an alternative, more flexible approach.

Usage

```
fevd = function(irf, dim = NULL)
```

Arguments

`irf` orthogonalized impulse response function, as computed eg. by `MTS:VARMAirf` or `SSirf`. `irf` may be a 3-dimensional array of dimension $(n \times n \times l)$ or a matrix which is then coerced to such an array, see the paramater `dim` below.

`dim` integer vector of length 3. If this (optional) parameter is given then `irf` is coerced to an array with `irf = array(irf, dim = dim)`. Note that `dim[1]==dim[2]` must hold.

Value

`vd` $(n \times n \times l)$ dimensional array which contains the forecast error variance decomposition: `vd[i,j,h]` is the percentage of the variance of the h-step ahead forecast error of the i-th component due to the j-th orthogonalized shock.

`v` $(n \times l)$ matrix which contains the forecast error variances: `v[i,h]` is the variance of the h-step ahead forecast error for the i-th component.

Examples

```
phi = matrix(c( 0.5, -0.7,  0.3, 0.75,
               -0.2, -0.5,  0.4,-0.90),
byrow = TRUE, nrow = 2)
theta = matrix(c(-0.1, -0.8,
                  0.7, -0.1),
                byrow = TRUE, nrow = 2)
k = MTS::VARMAirf(Phi = phi, Theta = theta,
                  lag = 24, orth = TRUE)
out = fevd(k$irf, dim = c(2,2,25))
plotfevd(out$vd)
```

5.4 `impresp2PhiTheta`: Construct a VARMA model from an impulse response

The model returned is in echelon canonical form. Note that this means in particular that $\phi_0 = \theta_0$ is (in general) a lower triangular matrix.

Usage

```
impresp2PhiTheta = function(k, tol = 1e-8)
```

Arguments

`k` $(n \times n(l_{\max}+1))$ matrix which contains the impulse response coefficients, e.g. computed by `MTS:VARMAirf`.

`tol` tolerance used by `qr` to estimate the rank and the basis of the Hankel matrix H of the impulse response function.

Value

`Phi` $(n \times np)$ matrix containing the AR parameters $\phi = [\phi_1, \ldots, \phi_p]$.

`Theta` $(n \times np)$ matrix containing the MA parameters $\theta = [\theta_1, \ldots, \theta_q]$.

`Phi0` $(n \times n)$ matrix, lag zero coefficient $\phi_0 = \theta_0$.

`kidx` n-dimensional (integer) vector with the Kronecker indices.

`Hrank` estimated rank of the Hankel matrix, as computed by `qr`.

`Hpivot` vector of pivot elements, returned by `qr`. Note that the first `Hrank` elements of this vector are the indices of the basis rows of the Hankel matrix.

5.5 `impresp2SS`: Construct a state space model from an impulse response

This function implements the Ho-Kalman realization algorithm which determines a state space realization for a given impulse response function.

For the case `type=="echelon"` a state space system in echelon canonical form is returned and for `type=="balanced"` a kind of balanced realization. The core step of this algorithm is to determine a basis for the row space of the Hankel matrix H of the impulse response coefficients.

In the first case this is done via a QR decomposition of the transposed Hankel matrix with the R function `qr` using the tolerance parameter `tol`. If the optional parameter `s` is missing or `NULL` then the rank of the Hankel matrix and hence the state space dimension is determined from this QR decomposition. If `s` is given then the algorithm constructs a state space model with this specified state space dimension. However, this option should be used with care. There is no guarantee that the so constructed model is a reasonable approximation for the true model.

For the "balanced" case an SVD decomposition of the Hankel matrix is used. For missing `s` the rank of the Hankel matrix is determined from the singular values of the matrix using `tol` as a threshold, i.e. the rank is estimated as the number of singular values which are larger than or equal to `tol` times the largest singular value. If `s` is specified then the first `s` right singular vectors are used as a basis for row space of the Hankel matrix and a state space model with state space dimension `s` is returned.

Usage

```
impresp2SS = function(k, type = c('echelon','balanced'), tol = 1e-8, s)
```

Arguments

`k` $(n \times n(l_{\max}+1))$ matrix which contains the impulse response coefficients, e.g. computed by `MTS:VARMAirf`.

`type` (string) determines the type of the realization, see the description above.

`tol` tolerance parameter, see the description above.

`s` desired state dimension, see the description above.

Value

`ss` a `dse::SS` object which represents the constructed innovation form state space model.

`Hsv` for `type=='balanced'`: a vector with the singular values of the Hankel matrix of the impulse response.

`kidx` for `type=='echelon'`: n-dimensional (integer) vector with the Kronecker indices.

`Hrank` for `type=='echelon'`: estimated rank of the Hankel matrix, as computed by `qr`.

`Hpivot` for `type=='echelon'`: vector of pivot elements, returned by `qr`. Note that the first `Hrank` elements of this vector are the indices of the basis rows of the Hankel matrix.

5.6 `is.stable`: Check the stability of a matrix polynomial

This function checks the roots of the determinant of a polynomial matrix

$$I - a_1 z - \cdots - a_p z^p$$

and returns `TRUE` if all roots are outside the unit circle. The roots are determined from the eigenvalues of the companion matrix, i.e. the roots are the reciprocals of the non-zero eigenvalues of the companion matrix.

Usage

```
is.stable = function(A)
```

Arguments

`A` $(n \times np)$ matrix with the polynomial coefficients $A = (a_1, \ldots, a_p)$.

Value

a scalar logical with an attribute `z` which contains the roots.

See Also

`dse:polyrootDet`.

5.7 `lyap`: Lyapunov equation

Solve the Lyapunov equation:

$$P = APA' + Q$$

It is assumed that Q is symmetric and that A is stable, i.e. that the eigenvalues of A have moduli less than one.

The procedure uses the Schur decomposition method as in *Kitagawa, An Algorithm for Solving the Matrix Equation $X = FXF' + S$, International Journal of Control, Volume 25, Number 5, pages 745–753 (1977).* and the column-by-column solution method as suggested in *Hammarling, Numerical Solution of the Stable, Non-Negative Definite Lyapunov Equation, IMA Journal of Numerical Analysis, Volume 2, pages 303–323 (1982).*

The Schur decomposition of A is computed with `QZ::qz.zgees`.

Usage

```
lyap = function(A,Q)
```

Arguments

`A` $(n \times n)$ (stable) matrix.

`Q` $(n \times n)$ (symmetric) matrix.

Value

`P` solution of the Lyapunov equation.

Examples

```
A = matrix(rnorm(4),nrow=2,ncol=2)
Q = matrix(rnorm(4),nrow=2,ncol=2)
Q = Q %*% t(Q)
P = lyap(A,Q)
```

5.8 PhiTheta2ARMA:

This function constructs an `dse::ARMA` object for given AR/MA parameter matrices (in the style of the MTS package).

The dse Package uses a simple strategy to impose restrictions on the parameters of a models. It simply treats all coefficients equal to one or zero as fixed and the others as "free". However for a model in *echelon form* in addition $a_0 = b_0$ must hold. Ignoring this constraint has to effects. First `dse::informationTests` does not use the correct number of free parameters and hence it does not return the correct values for information criteria like AIC. Second, if the model is feed into `dse::estMaxLik` then the estimated model will in general not be in echelon canonical form.

In order to circumvent the first problem this function offers the switch `fix`. For `fix=TRUE` the function calls `dse::fixConstants` and sets all zero/one entries as fixed and in addition all entries of b_0. By this trick `dse::informationTests` then uses the right number of free parameters corresponding to a model in echelon form. However this trick does not solve the second problem.

Usage

```
PhiTheta2ARMA = function(Phi, Theta, Phi0 = NULL, fix = FALSE,
                         output.names = paste('y',1:nrow(Phi),sep=''))
```

Arguments

`Phi` $(n \times np)$ matrix containing the AR parameters $\phi = [\phi_1, \ldots, \phi_p]$.

`Theta` $(n \times np)$ matrix containing the MA parameters $\theta = [\theta_1, \ldots, \theta_q]$.

`Phi0` $(n \times n)$ matrix containing the lag zero coefficient matrix $\phi_0 = \theta_0$. In the default case `Phi0=NULL` the n-dimensional identity matrix is used.

`fix` see the discussion above.

`output.names` vector of strings with the respective names for the n components of the ARMA process.

Value

`dse::ARMA` object.

Examples

```
phi = matrix(c(-0.5, -0.2, -0.3, -0.05, -0.2, -0.5, -0.1, -0.30),
byrow = TRUE, nrow = 2)
theta = matrix(c(-0.2,  0.0, -0.1, -0.3),
               byrow = TRUE, nrow = 2)
phi0 = matrix(c(1.0, 0, 0.4, 1),
              byrow = TRUE, nrow = 2)
arma = PhiTheta2ARMA(solve(phi0,phi), solve(phi0,theta))
arma = PhiTheta2ARMA(phi, theta, phi0)
m = ARMA2PhiTheta(arma, normalizePhi0 = FALSE)
all.equal(cbind(phi0,phi,theta),cbind(m$Phi0,m$Phi,m$Theta))
```

5.9 `plot3d`: Plot 3-dimensional arrays

This is as simple function for plotting three dimensional structures, like the impulse response function or the autocovariance function of an ARMA process.

Usage

```
plot3d = function(x, dim = NULL, labels.ij = NULL,
                  main = '', xlab = 'lag (k)', ...)
```

Arguments

`x` $(m \times n \times l)$ dimensional array, or a vector/matrix which can be coerced to an array, see below.

`dim` integer vector of length 3. If this (optional) parameter is given then x is coerced to an array with `x = array(x, dim = dim)`.

`labels.ij` string which is used to label the panels. E.g.
`labels.ij = 'partialdiff*y[i_*k]/partialdiff*epsilon[j_*0]'`
may be used for a plot of an impulse response function. The string should contain `'i_'`, `'j_'` as place holders for the indices `i,j` of the respective (i,j)-th subpanels.

`main, xlab` main title for the plot and label for the x axis.

`...` other graphical parameters. These parameters are passed on to the `lines()` method.

Value

invisibly returns the input `x` after beeing coerced to an array.

Examples

```
plot3d(rnorm(2*3*11), dim = c(2,3,11), main ='test one',
       labels.ij = expression(corr(x[list(i_,t+k)],x[list(j_,t)])) )
plot3d(array(rnorm(4*4*21), dim = c(4,4,21)), main ='test two', xlab = 'delay (s)',
       labels.ij = expression(epsilon[j_*0] %->% y[i_*s]) )
plot3d(matrix(rnorm(4*100),nrow = 4), dim = c(4,1,100), main ='test three',
       xlab = 'time t-1', labels.ij = expression(series~i_) )
```

5.10 plotfevd: Plot a Forecast Error Variance Decomposition

Generate a plot of a Forecast Error Variance Decomposition.

Usage

```
plotfevd = function(vd, main = 'forecast error variance decomposition',
                    xlab = 'forecast horizon (h)', series.names, col)
```

Arguments

`vd` $(n \times n \times h)$ array, which contains the forecast error variance decomposition, as computed e.g. by `fevd`.

`main` main title for the plot.

`xlab` label for the x axis.

`series.names` vector of strings the names for the component series.

`col` n-dimensional vector of colors.

Value

invisible copy of the input `vd`.

Examples

```
vd = array(runif(4*4*20), dim = c(4,20,4))
v = apply(vd, MARGIN = c(1,2), FUN = sum)
vd = vd / array(v, dim = c(4,20,4))
vd = aperm(vd, c(1,3,2))
plotfevd(vd)
# of course this sub-sample is not a valid FEVD
plotfevd(vd[1:2,1:2,1:10], series.names = c('x','y'), col = c('red','blue'))
```

5.11 SScov: Autocovariance and autocorrelation function of a state space model

This function computes the autocovariance and autocorrelation function of a stationary process represented by a state space model in innovation form. The syntax and the return value is analogous to `MTS:VARMAcov`.

Note that the stability of the state space model is not checked.

Usage

```
SScov = function(ss, Sigma = NULL, lag.max = 12L)
```

Arguments

`ss` `dse:SS` object, which represents an innovation form state space model.

`Sigma` covariance of the innovations ($(n \times n)$ symmetric, positive definite matrix). If `NULL` then the n-dimensional identity matrix is used.

`lag.max` maximum lag (integer)

Value

`autocov` $(n \times n(l_{\max} + 1))$ matrix with the autocovariances.

`ccm` $(n \times n(l_{\max} + 1))$ matrix with the autocorrelations.

5.12 SSirf: Impulse response function of a state space model

This function computes the impulse response function and the orthogonalized impulse response function of a state space model in innovation form. The syntax and the return value is analogous to `MTS::VARMAirf`.

The orthognalized innovations are defined by the *symmetric* square root of the covariance matrix Σ. Note that the stability of the state space model is not checked.

Usage

```
SSirf = function(ss, Sigma = NULL, lag.max = 12L, orth = TRUE)
```

Arguments

`ss` `dse:SS` object, which represents an innovation form state space model.

`Sigma` covariance of the innovations ($(n \times n)$ symmetric, positive definite matrix). If `NULL` then the n-dimensional identity matrix is used.

`lag.max` maximum lag (integer)

`orth` if `TRUE` then the orthogonalized impulse response function is computed.

Value

`psi` $(n \times n(l_{\max} + 1))$ matrix with the impulse response coefficients.

`irf` $(nn \times (l_{\max} + 1))$ matrix with the orthogonalized impulse response coefficients. If `!orth` then `irf` is just a 'reshaped' version of `psi`. Note that `psi` and `irf` have different dimensions.