Chapter 10: ARMA and VARMA models

GROCER allows the estimation of ARMA and VARMA models (part 2). Before estimating such an ARMA model, you can have a look at the autocorrelation and partial autocorrelation function (part 1). As usual, the precise description of the functions and their specifications can be found in the last part (part 3).

1. Autocorrelation function and partial autocorrelation function

The autocorrelation and partial autocorrelation function are performed by grocer functions acf and pacf[1]. The simplest call is:

acf(y) (resp. pacf(y))

In that case, if N is the number of observations, the N/4 first autocorrelations are calculated and plotted, with 95% confidence band.

Some default parameters can be changed. First, you may impose that the result are not graphed. This is done simply by adding the option 'noplt' in acf (or pacf) : acf(y,'noplt').

Second, you can impose the # of calculated coefficients. This is done by adding the option: 'm=xx' where xx is the # of calculated coefficients. For instance pacf(y,'m=15') calculates the first 15 partial autocorrelation coefficients.

Lastly, you can choose the size of the confidence band by adding the option: 'size=xx'. For instance, acf(y,'size=0.1') calculates the autocorrelation function with a 90% confidence band.

The output of these functions is a tlist, that allows you to store the results. For instance: r0 = pacf('y') stores the results in a tlist called r0. This tlist contains the name of the method (r0('meth')), the values of the input variable (r0('y')), the name of this variable (r0('namey')), the autocorrelation values themselves (r0('acf')), the lower (r0('acf_l')) and upper (r0('acf_u')) bounds of the confidence band, its size (r0('size')) and a boolean indicating if y is a ts (r0('prests')).

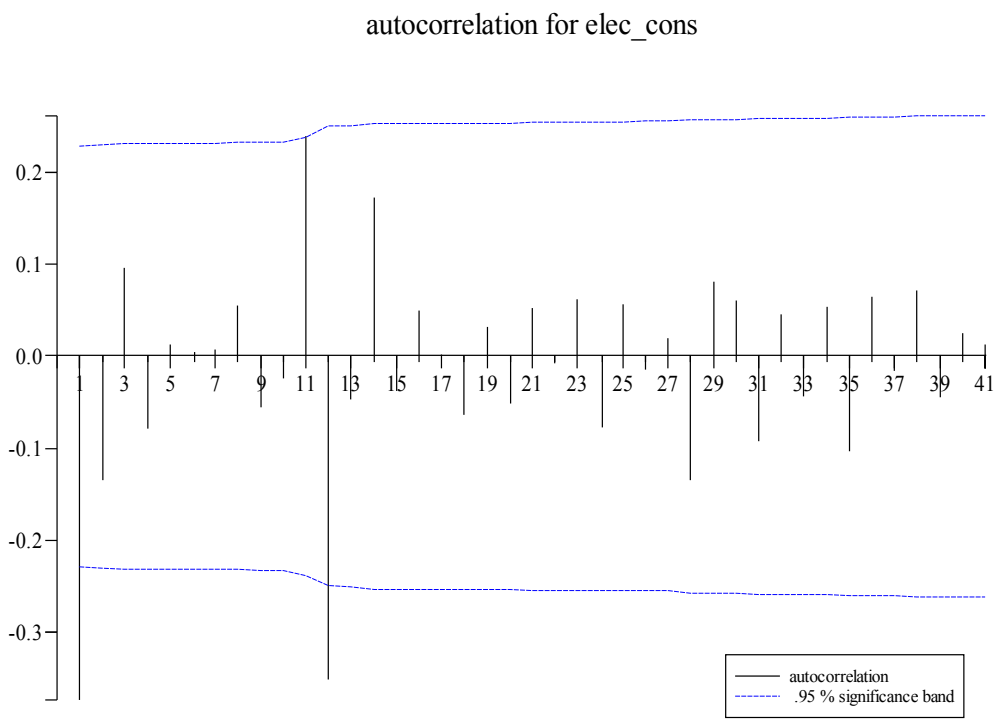
The graphs are produced by a grocer function called pltacf. Therefore, if you have saved the results of acf or pacf in a tlist result, say r0, then you can graph them again by the command pltpacf(r0).

As an example, take the consumption of electricity, used as an example by Newbold et al. (1994)[2]:

```
--> mtlb_load('SCI/macros/grocer/db/seriesa.dat');
--> elec_cons = transdif(seriesa,0,1,1,12);
--> resacf=acf('elec_cons','m=30')
```

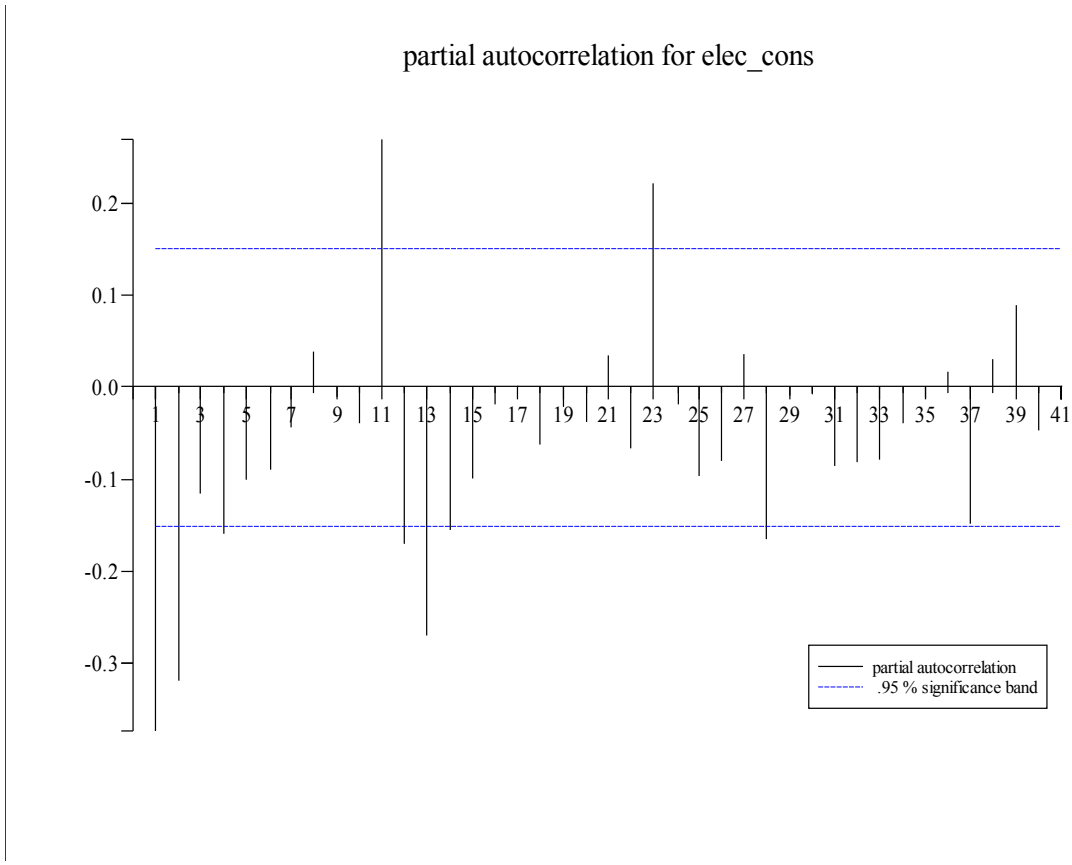
[1] acf is an extension of the function already existing in Scilab

[2] Newbold P., C. Agiakloglou and J Miller (1994) : « Adventures with ARIMA software », *International Journal of Forecasting*, 10, pp. 573-581.



Similarly, you can estimate the partial autocorrelation function:

```
--> resacf=acf('elec_cons','m=30')
```



*

The partial autocorrelation do not seem to cancel as the lag increases; but only the autocorrelations at lag 1 and 12 seem to be non zero. This seems to indicate that the process is an AR(12) of the following type :

(1-aL) (1-a L^12) elec_const = et

2. VARMA estimation.

2.1 The estimation method.

The estimation method rests on the exact maximum likelihood estimation. The likelihood is calculated with the Kalman filter, with initial state derived from De Jong and Chu-Chu-Lin (1994)[3]. The algorithm used is the one proposed by Terceiro, Casals, Jerez, Serrano and Sotoca in their E4 matlab toolbox (see "Time Series Analysis using MATLAB", June 2000), suitably translated and adapted for grocer.[4]

2.2 How to estimate an ARMA model

The function used to estimate ARMA, VARMA or VARMAX models is the same: this is the

[3] P. De Jong and S. Chu-Chun-Lin (1994): « Stationary and non-stationary state space models », *Journal for Time Series Analysis*, Vol. 15, N°2.

[4] For the moment, I have only translated and adapted the programms needed to perform VARMA estimation. This is the reason why in some functions (such as lffast), there are useless lines of codes, corresponding to E4 capabilities not yet implemented.

function varma. This function allows also the estimation of seasonal models (SARIMA models for instance).

An ARMA model without constant as the following form:
AR(L)*AS(L^s) y = MA (L)*MAS(L^s) e

The estimation of this model on series y involves the following command:
r = varma(y, AR, ARS, MA, MAS, v, s)
AR, ARS, MA and MAS are row vectors of size p, ps, q and qs; v and s are scalars. If your ARMA has no seasonal part, then ARS and MAS should be equal to the vector []. Similarly, if you do not have MA part, then set MA and MAS to []. s is the seasonality and v the variance of the error term.

For instance, estimating an ARMA(2,1):
$(1+ar_1*L+ar_2*L^2)$ yt = $(1+ma_1*L)$ et
can be done by the following command:
r = varma(y, [0 0], [], 0,[], 0, 1)

Estimating a SARIMA(2,0,1) (1,0,1):
$(1+ar_1*L+ar_2*L^2)$ $(1+ars_1 *L^{12})$ yt = $(1+ma_1*L)$ $(1+mas_1 *L^{12})$ et
can be done by the following command:
r = varma('y', [0 0], 0, 0,0, 0, 12)

Note the quotes around y: as with other grocer estimation programs, you can estimate an ARMA model on vectors or timeseries, and, if you want to keep the name of the series, you have to put the name of the variable between quotes when calling the function.

If you want to add a constant to this model, then you should add the options: 'exo="const"','Gexo=0'. There are 2 inputs: 'exo="const"' to indicate that there is an exogenous variable called 'const'; 'Gexo=0' to indicate that this exogenous variable has only one lag. You can also add any exogenous variable, say z, with the option 'exo=z','Gexo=0' or 'exo="z"','Gexo=0' if you want to keep the names of the exogenous variable z for the results of printings. You can add several exogenous variables. The syntax is the same, but you have to enter the list of your exogenous variables in a way that cane be read by function explox: right to 'exo=', you must have either the name of a real matrix, a list of exogenous variables or a string vector of names of the exogenous variables (in that case, the names should be entered between double quotes). For instance:
'exo=list("const"), "z")'

You can also change the increment used to calculate the numerical gradient, which by default is sqrt(%eps). You just have to enter the option 'delta=x' with x your increment.

As with other estimation functions, you can avoid the printings of your results, with the option 'noprint'.

Lastly, you can enter any option to Scilab maximisation function optim (see Scilab help).

Here is the result of the estimation of an MA(1)*MAS(1) of the 12th difference on the electricity consumption tested by Newbold and al. (1994):

```
--> load('SCI/macros/grocer/db/varma_d.dat');
--> elec_cons = transdif(seriesa,0,1,1,12);
--> results=varma(elec_cons,[],[],0,0,0,12);

*************** ARMA estimation Results for model: ***************
Y = (1+MA(L))*(1+MAS(L^12)) * e
with:
- Y = endogenous
- V(e) = V

Log-likelihood: 252.57424
Information criteria: AIC = -2.988913, BIC = -2.9329011

Parameter   Estimate   Std. Dev. t-test
MA - lag 1  -0.7217599 0.0548639 -13.155467
MAS - lag 1 -0.8242452 0.0739780 -11.141761
V           0.0026089  0.0563295 0.0463147

************************* Correlation matrix *************************
MA - lag 1  1
MAS - lag 1  -.0082550 1
V            .0084479   .2372843 1

           Condition number = 1.6234575
Reciprocal condition number =  .6480967
**********************************************************************
```

These results are printed by grocer function prtvarma. If you have saved the results (as in the example), you can print them again by typing prtvarma(results).

Besides this example, taken from function varma_d1, you can find three other examples in functions varma_d2, varma_d3, and varma_d4.

You can also impose constraints on the coefficients. To impose an equality constraint, then you have to change the type of the corresponding input. It should be a list of 2 elements: the first one corresponds to the vector of starting values, with the constrained parameter set to its constrained value; the second one should be a string vector of the same size with values '=' for the indexes corresponding to constrained values. To impose to 0.4 for instance the value of the first coefficient to a third order AR part, then the AR part should be entered as follows:

list([0.4 0 0],['=' " "])

If you have only equality constraints and no inequality constraints, then there is a simpler possibility, which is to mix the equal signs and the values in a string vector. In the previous example, it should take the form:

['=0.4' '0' '0']

For inequality constraints, the principle is almost the same: the corresponding input should be a list with 2 elements, the first one corresponds to the vector of starting values; the second one should be a string vector of the same size with values 'value<' if the parameter should be greater than the predefined value, '<value' if the parameter should be lower than the predefined value, 'value1<value2' if the parameter should be lower than the predefined value value2 and greater than the predefined

value value1.

For instance, constraining an AR value to be between –1 and 1, then it should be entered as follows:

list(0, '-1<1').

(°) varma uses e4 starting values to feed the optimisation program. You can however if you want so impose your own starting values. To that end, you have to feed varma entries (AR, ARS, MA, MAS, v) with your starting values and enter the option 'init=own'.

For instance, to feed the previous arma model with -0.7, -0.8 and 0.003 as starting values for the MA, MAS and v respectively, enter:

--> results=varma(elec_cons,[],[],-0.7,-0.8,0.003,12)

which provides the same results!(°°)

2.3 How to estimate a VARMA model

The estimation of a VARMA model is very similar to the estimation of an ARMA model. The main difference is that the vectors are replaced with matrices. There is another difference: you can impose the variance matrix to be diagonal; to that end, you have to enter this parameter (the 6th one in function varma) as a (1 x n) or (n x 1) vector instead a (nxn) matrix

You can find an example in function mink1.

```
--> mtlb_load('SCI/scied/grocer/encours/e4sci/mink.dat');
```

// transform the variables from the database

```
--> z1 = transdif(mink(:,3),1,1);
--> z2 = mink(:,2)-mean(mink(:,2));
--> muskrat=z1
--> mink=z2(2:62)
```

// define the input parameters; note the 0 equality constraints imposed on the AR parameters

```
--> phi1 = ['0','=0';'=0','0'];
--> phi2 = ['0','=0';'=0','0'];
--> phi3 = ['0','=0';'=0','=0'];
--> phi4 = ['0','=0';'=0','=0'];
--> theta = [0,0;0,0];
--> sigma = [0,0;0,0];
```

// estimate the varma model; note that there are no seasonals, that the AR part has degree 4 and is
// supposed to be diagonal; the MA part is of order 1, without constraints; the variance matrix is not
// supposed to be diagonal

```
--> result = varma(['muskrat' 'mink'],[phi1,phi2,phi3,phi4],[],theta,[],sigma,1);


*************** VARMA estimation Results for model: ***************
(1+AR(L))*Y = (1+MA(L))*e
with:
      [ muskrat  ]
```

```
- Y = [ mink     ]
- V(e) = V

Log-likelihood: 9.0059411
Information criteria: AIC =  .1309527, BIC =  .5808111

Parameter                        Estimate   Std. Dev. t-test
AR - eq # 1 - lag # 1 - var # 1  -.6887092  .1363550 -5.0508541
AR - eq # 2 - lag # 1 - var # 2 -1.2679082  .1098543 -11.541722
AR - eq # 1 - lag # 2 - var # 1  .5941062   .1402932 4.2347478
AR - eq # 2 - lag # 2 - var # 2  .5592537   .0921302 6.0702509
AR - eq # 1 - lag # 3 - var # 1  -.0681946  .1171420  -.5821529
AR - eq # 1 - lag # 4 - var # 1  .2815736   .0855952 3.2895966
MA - eq # 1 - lag # 1 - var # 1  -.2965995  .1606131 -1.8466711
MA - eq # 2 - lag # 1 - var # 1  .6027021   .0803646 7.4995979
MA - eq # 1 - lag # 1 - var # 2  -.8641751  .1387191 -6.2296763
MA - eq # 2 - lag # 1 - var # 2  -.8351469  .1529343 -5.4608218
V(1,1)                           .2492625   .0225682 11.044879
V(2,1)                           .0424334   .0147947 2.8681379
V(2,2)                           .2013021   .0182819 11.011036

************************* Correlation matrix **************************
AR - eq # 1 - lag # 1 - var # 1 1
AR - eq # 2 - lag # 1 - var # 2  -.0152609 1
AR - eq # 1 - lag # 2 - var # 1  -.6789892  .3357148  1
AR - eq # 2 - lag # 2 - var # 2  -.2311956  -.8421604  -.1502274 1
AR - eq # 1 - lag # 3 - var # 1  .3913329   -.1942320  -.8865285  .1444133  1
AR - eq # 1 - lag # 4 - var # 1  .3544018    .1474168    .3531851    -.1745917
-.6498412 1
MA - eq # 1 - lag # 1 - var # 1  .695909     -.3598524   -.3469839   .0429098
.0681844   .3914261  1
MA - eq # 2 - lag # 1 - var # 1  -.1737098   .2757770    .3879273    -.3466258
-.2637317  .1241977   .0854211  1
MA - eq # 1 - lag # 1 - var # 2  -.5098756   .4000510    .4033164    -.3752331
-.1850609  -.2234434  -.5844278  .2478797
1
MA - eq # 2 - lag # 1 - var # 2  -.4418512   .6946614    .3507546    -.4527361
-.1188363  -.2083229  -.7039318  -.0030171
 .6418551 1
V(1,1)                               .0058166    .0235944    -.0064164  -.0268380
.0069480   -.0103007  -.0024881  -.0249897
 .0174591  .0219656 1
V(2,1)                               -.0092355  .0225302    .0034581    -.0163655
.0019412   -.0104281  -.0153218  .0074602
 .0263244  .0223605  .2718836 1
V(2,2)                               .0116761    .0082854    -.0125729  -.0143127
.0068019   -.0075062  -.0001659  -.0218261
 .0211852  .0224291  .0364676  .2475205 1

           Condition number = 333.30747
Reciprocal condition number =  .0024437
***********************************************************************
```

3. The ARMA and VARMA functions and their specifications.

acf_____autocorrelation function

CALLING SEQUENCE
[resacf]=acf(namey,varargin)

PARAMETERS

INPUT:
* namey = a time series, a real (nx1) vector or a string equal to the name of a time series or a (nx1) real vector between quotes
* argi = (optional) string arguments that can be:
  - 'noplt' if the user does not want to plot the autocorrelation coefficients
  - 'm=xx' where xx is the number of calculated coefficients (default: floor(size(y,'*')/4))
  - 'size=xx' where xx is the size of the confidence band (default: 0.05)

OUTPUT:
resacf = a tlist with
  . resacf('meth') = 'acf'
  . resacf('y') = values of the input variable
  . resacf('acf') = autocorrelation coeffcients
  . resacf('acf_l') = low bound of the confidence interval
  . resacf('acf_u') = upper bound of the confidence interval
  . resacf('size') = size of the confidence band
  . resacf('namey') = name of variable
  . resacf('prests') = %t if variable is a ts

DESCRIPTION
Calculates sample autocorrelation coefficients

Example:

mtlb_load('SCI/macros/grocer/db/seriesa.dat');elec_cons = transdif(seriesa,0,1,1,12);
resacf=acf(elec_cons,'m=30')

Calculates the 30 first autocorrelations for the index of electricty consumption provided in matlab database seriesa.dat

arma_dv_____partial derivatives of VARMA matrices

CALLING SEQUENCE
[F,dF,A,dA,V,dV,G,dG]=arma_dv(theta,theta2mat,di)

PARAMETERS

INPUT:
* theta = (npx1) vector of starting values for the parameters
* theta2mat = a string vector of instructions that transforms theta into the matrices FR, FS, AR, AS, V and G
* di = a scalar

OUTPUT:
* F = the whole AR part of the model
* A = the whole MA part of the model
* V = the variance of the residuals
* G = the coefficients of the exogenous variables
* dF, dA, dV, dG = the derivatives of these matrices with respect to the i-th parameter in theta

DESCRIPTION
Computes the partial derivatives of the matrices in reduced form of a VARMAX model with respect to i-th parameter of the theta vector.

Example:
[F,dF,A,dA,V,dV,G,dG] = arma_dv(theta,theta2mat,di);

Examples taken from function fr_dv(); this is an intermedaite in the calculation of the derivative of the information matrix with respect to the parameters (see function imod())

arma2param_____explodes a Varmax model for estimation uses

CALLING SEQUENCE
[theta,theta2mat,V2theta,thetalab,AR,ARS,MA,MAS,G,V,p,P,q,Q,g,s,k,n,np,%type,vdiag,ineq] = arma2param(m,AR,ARS,MA,MAS,V,s,G,r,namexos)

PARAMETERS

INPUT:
* m = # of endogenous variables
* AR = a string, representing the name of the AR part of the model
* ARS = a string, representing the name of the seasonal AR part of the model
* MA = a string, representing the name of the MA part of the model
* MAS = a string, representing the name of seasonal MA part of the model

The names of FR, FS, AR and AS represent objects that can be either of constant type or list type; in the last case, the first item of the list should represent as in the first case the starting value of the corresponding matrix, and second one a string matrix, of same size, of constraints
(" for no constraint, '=' for equality constraint, 'value<*', '*<value' or 'value1<*<value2' for inequality constraints)
* V = a (mxm) (var-covar) matrix or a (mx1) vector (the diagonal of a var-covar matrix, supposed to have 0 outside the diagonal)

* s = order of seasonality
* G = the coefficients martrix for the endogenous variables (if any)
* r = # of exogenous variables (if any)

OUTPUT:
* theta = (npx1) vector of parameters that will be estimated
* theta2mat = vector of strings, representing the instructions that transform back theta into the input parameters
* V2theta = vector of strings, representing the instructions that transform V into the corresponding parameters in theta
* theatlab = (npx1) vector of strings, representing theta names used when printing the results
* FR = the AR part of the model
* FS = the seasonal AR part of the model
* AR = the MA part of the model
* AS = the seasonal MA part of the model
* G = the coefficients matrix for the endogenous variables
* V = the (mxm) (var-covar) matrix
* p = degree of the AR part of the model
* P = degree of the seasonal AR part of the model
* q = degree of the MA part of the model
* Q = degree of the seasonal MA part of the model
* s = order of seasonality
* k = maximum degree of the total AR, MA and G parts
* n = k*m
* np = # of estimated parameters
* %type = type of the e4 model
* vdiag = 0 if V is diagonal, 1 if not

DESCRIPTION

Explodes a Varmax model for estimation uses
The input arguments corresponding to the model:

(I + AR1·B + ... +ARp·B^p)(I + ARS1·B^s + ... + ARSps·B^ps·s) y(t) =
(G0 + G1·B + ... + Gt·B^l) u(t) +
(I + MA1·B + ... + MAq·B^q)(I + MAS1·B^s + ... + MASqs·B^qs·s) e(t)

are:

AR = [AR1 | ... | ARp] ARS = [ARS1 | ... | ARSps]
G = [G0 | G1 | ... | Gg]
MA = [MA1 | ... | MAq] MAS = [MAS1 | ... | MASqs]

Examples:

1) [theta,grocer_e4_theta2mat,grocer_e4_V2theta,grocer_e4_lab,grocer_AR,grocer_ARS, grocer_MA,grocer_MAS,G,grocer_V, grocer_e4_p,grocer_e4_P,grocer_e4_q,grocer_e4_Q, grocer_e4_g, grocer_e4_s,grocer_e4_k,grocer_e4_n,grocer_e4_np,grocer_e4_type,grocer_e4_vdiag, grocer_e4_ineq] = arma2param(grocer_e4_m,grocer_AR,grocer_ARS,grocer_MA,grocer_MAS, grocer_v,grocer_s,grocer_e4_Gexo,grocer_e4_r,grocer_e4_namexo);

2) [theta,grocer_e4_theta2mat,grocer_e4_V2theta,grocer_e4_lab,grocer_AR,grocer_ARS, grocer_MA,grocer_MAS,G,grocer_V,grocer_e4_p,grocer_e4_P,grocer_e4_q,grocer_e4_Q,

grocer_e4_g,grocer_e4_s,grocer_e4_k,grocer_e4_n,grocer_e4_np,grocer_e4_type,grocer_e4_vdiag, grocer_e4_ineq]=arma2param(grocer_e4_m,grocer_AR,grocer_ARS,grocer_MA,grocer_MAS, grocer_v,grocer_s);

Both examples taken from function varma(); the first example deals with the case when there are exogenous variables (for instance a constant), the second one when there are no exogenous variables (in particular no constant).

pacf_____partial autocorrelation function

CALLING SEQUENCE
[respacf]=pacf(namey,varargin)

PARAMETERS

INPUT:
* namey = a time series, a real (nx1) vector or a string equal to the name of a time series or a (nx1) real vector between quotes
* argi = (optional) string arguments that can be:
  - 'noplt' if the user does not want to plot the partial autocorrelation coefficients
  - 'm=xx' where xx is the number of calculated coefficients (default: floor(size(y,'*')/4)
  - 'size=xx' where xx is the size of the confidence band (default: 0.05)

OUTPUT:
respacf = a tlist with
  . respacf('meth') = 'pacf'
  . respacf('y') = values of the input variable
  . respacf('pacf') = partial autocorrelation coeffcients
  . respacf('pacf_l') = low bound of the confidence interval
  . respacf('pacf_u') = upper bound of the confidence interval
  . respacf('size') = size of the confidence band
  . respacf('namey') = name of variable
  . respacf('prests') = %t if variable is a ts

DESCRIPTION
Calculates sample partial autocorrelation coefficients.

Example:
mtlb_load('SCI/macros/grocer/db/seriesa.dat');elec_cons = transdif(seriesa,0,1,1,12);
respacf=pacf(elec_cons,'m=30')

Calculates the 30 first partial autocorrelations for the index of electricty consumption provided in matlab database seriesa.dat.

theta2arm2_____recovers arma parameters from the vector of estimated parameters

CALLING SEQUENCE
[AR,ARS,MA,MAS,V,G]=theta2arm2(theta,theta2mat,fromgrad)

PARAMETERS

INPUT:
* theta = (npx1) vector of parameters
* theta2mat = a string vetor of instructions that transforms theta into the matrices FR, FS, AR, AS, V and G
* fromgrad = if not given, forces the matrix V to be positive definite
---
OUTPUT:
FR,FS,AR,AS,V,G = matrices of the process:
(I + FR1·B +...+FRp·B^p)(I + FS1·B^s +...+ FSps·B^ps·s) y(t)
= (G0 + G1·B +...+ Gt·B^l) u(t) +
(I + AR1·B +...+ARq·B^q)(I + AS1·B^s +...+ ASqs·B^qs·s) a(t)
with Var(a(t)) = V

DESCRIPTION

Recovers the matrices of an ARMA process from the values of the estimated parameters.

Examples:
mtlb_load('SCI/scied/grocer/encours/e4sci/seriesa.dat'); elec_cons = transdif(seriesa,0,1,1,12);
results=varma(elec_cons,[],[],0,0,0,12); grocer_AR=[];grocer_ARS=[];grocer_MA=[];
grocer_MAS=[]; grocer_V=[];grocer_G[];
[AR,ARS,MA,MAS,V,G]=theta2arm2(results('coeff'),results('theta2mat'),1)

Although theta2arm2 is mainly useful for the estimation of a VARMA model, this example shows how it can be used to recover the estimated matrices from the tlist results produced by the function VARMA. Note however that it imposes to define the matrices grocer_AR,..., grocer_G; so this capability should be used with care...

theta2arma_____VARMA model to VARMAX reduced form

CALLING SEQUENCE
[F,A,V,G]=theta2arma(theta,theta2mat)

PARAMETERS

INPUT:
* theta = (npx1) vector of parameters
* thetamat = a string vetor of instructions that transforms theta into the matrices AR, ARS, MA, MAS, V and G

OUTPUT:
* F = the MA part of the model
* A = the MA part of the model
* G = the coefficients matrix for the endogenous variables
* V = the (mxm) (var-covar) matrix

DESCRIPTION
Converts a VARMA model to reduced form VARMAX notation.
The input arguments corresponding to the model:
(I + AR1·B + ... +ARp·B^p)(I + ARS1·B^s + ... + ARSps·B^ps·s) y(t) =
(G0 + G1·B + ... + Gt·B^l) u(t) +
(I + MA1·B + ... + MAq·B^q)(I + MAS1·B^s + ... + MASqs·B^qs·s) a(t)
are transformed to
F(B)y(t) = G(B)u(t) + A(B)e(t)

Examples:
[F,A,V,G]=theta2arma(theta,theta2mat)

Taken from function theta2fr. Since this function is an intermediate calculation in an VARMA estimation, it would be difficult to use it outside this framework.

(°)
Transf_roots_____reverses roots lower than 1 in a L polynom

CALLING SEQUENCE
[AR,corrv_ar]=transf_roots(AR)

PARAMETERS

INPUT:
* AR = a (k x 1) vector of coefficients of a polynom 1+AR(L)

OUTPUT:
* AR = the (k x 1) vector of transforms coefficients such as all roots of (1+AR(L)) are greater than 1
* corrv_ar = the coefficient that lust be used to correct the variance of residuals

DESCRIPTION
Reverses roots lower than 1 in a L polynom and calculates the correction to apply to the variance of residuals.

Examples:
[AR,corrv_ar]=transf_roots([-2.4 0.8])

Reverses the roots of 1-2.4*L+0.8*L^2 that are lower than 1 (here it is 0.5). Provides:
corrv_ar = 0.25
AR = [- 0.9 0.2]
(°°)

varma_____VARMA estimation

CALLING SEQUENCE
result=varma(endo,AR,ARS,MA,MAS,v,s, arg1,…,argn)

PARAMETERS

INPUT:
* endo = either
 - a matrix of strings, each one being the name of a variable
 - a (Txn) real matrix
 - a ts
 - a list of variables; each element could be a timeseries, a real vector, a real matrix or a string (the name of a variable with one of the types cited above, between quotes)
* AR = the matrix [] or a (nx(n*p)) matrix
 with:
 - n = # of endogenous variables in endo
 - p = # of lags in the AR part of the process
* ARS = the matrix [] or a (nx(n*ps)) matrix
 with ps = # of lags in the seasonal AR part of the process
* MA = the matrix [] or a (nx(n*q)) matrix
 with: q = # of lags in the AM part of the process
* MAS = the matrix [] or a (nx(n*qs)) matrix
 with qs = # of lags in the seasonal MA part of the process
* v = a (nx1) vector if the user wants to impose independence between residuals or a (nxn) matrix in the other case
* s = a scalar representing the order of the seasonality
* arg1,…,argn:
 - 'noprint' if the user does not want to print the results
- 'delta=xx' where xx is the value used as increment in the calculation of the numerical gradient (default=sqrt(%eps))
 - 'exo=x' with x:
 ° a list of variables each element could be a timeseries, a real vector, a real matrix or a string (the name of a variable with one of the types cited above, between quotes)
 ° a matrix of strings, each one being the name of a variable or the string 'cte' or 'const' if the user wants a constant to be included automatically
 - 'Gexo=x' with x a matrix of values corresponding to the exogenous variables; the dimension of x should be (n x (r*g)) with r=# of exogenous variables, g=# of lags of the endogenous variables
 - any option to optim

OUTPUT:
result = a results tlist with:
- result('meth') ='varma'
- result('y') = (nobsxnendo) matrix of values for the endogenous variables
- result('namey') = (nvarx1) vector of names for the endogenous variables
- result('nobs') = # of observations
- result('coeff') = (npx1) vector of estimated parameters
- result('lab') = (npx1) string vector of names for the estimated parameters

- result('llike') = log-likelihood of the model
- result('tstat') = Student's t of the coefficients
- result('std') = (npx1) Student's t of the coefficients
- result('corr') = (npxnp) correlation matrix
- result('AIC') = Akaike information criterion
- result('BIC') = Schwartz information criterion
- result('theta2mat') = (npx1) string vector making the transformation of the vector of estimated parameters into the matrices of the problem
- result('seas') = order of the seasonality
- result('nexo') = # of exogenous variables in the model
- result('resid') = (nobsx1) vector of filtered residuals

DESCRIPTION

Estimates a VARMA model using E4 functions. The ARMA model has the following form:
AR(L)*ARS(L^s) y = MA(L)*MAS(L^s) e [+G(L)X]
where L is the lag operator, X is an optional vector of exogenous variables

Examples:
mtlb_load('SCI/scied/grocer/encours/e4sci/seriesa.dat'); elec_cons = transdif(seriesa,0,1,1,12);
results=varma(elec_cons,[],[],0,0,0,12)

Provides the estimation of a VARMA model without AR part and without exogenous variables (in particular without constant). Series are monthly and starting values for the 3 estimated parameters (the first MA parameter, the first seasonal MA and the variance of residuals) are all 0.