Appendix C

Software (E^4)

This Appendix describes a MATLAB toolbox for time series modeling. Its name, E^4, refers to the Spanish: *Estimación de modelos Econométricos en Espacio de los Estados*, meaning "State-Space Estimation of Econometric Models." It includes ready-to-use functions for model specification, parameter estimation, model forecasting, simulation, and signal extraction. In particular, all the examples in this book (except those of Chapter 11) have been computed with E^4.

The structure of this Appendix is as follows. Section C.1 reviews the standard models supported (including e.g., SS, VARMAX or transfer functions), describes how to define them, and shows how a model can be created by joining several building blocks through the model combination procedures that described in subsections 3.2.1 and 3.2.2. Section C.2 provides an overview of the analytical and computational procedures employed by E^4. They fall into three broad categories: standard time series methods, signal extraction procedures, and model estimation algorithms. Last,

Section C.3 provides some remarks about the design priorities of E^4, and discusses what type of user might benefit from this toolbox.

The source code for all the functions in E^4 is freely provided under the terms of the GNU General Public License. Appendix D indicates how to download E^4 and its related materials, as well as the source code and data required to replicate the examples in this book.

C.1 Models supported in E^4

C.1.1 State-Space model

The most important formulation supported by E^4 is the SS model; the standard representation for which most of its core computational algorithms are devised. The basic SS formulations supported are the MEM and SEM forms defined in Sections 2.1 and 2.2.

On the other hand, SS models are difficult to store and manipulate through software. For this reason, E^4 manages all the models supported using a convenient format called THD (THeta-Din).

C.1.2 The THD format

Defining a model in THD format requires creating two numeric matrices: `theta` and `din`, which contain respectively, the parameter values and a description of the model dynamic structure. Additionally, the model can be documented through an optional character matrix, `lab`, which contains names for the parameters in `theta`. While some analyses may require editing `theta` and `lab`, most users will never need to touch the matrix `din`.

The THD representation of an SS model can be obtained via the function `ss2thd` (SS-to-THD), which translates the matrices characterizing any MEM or SEM form into the THD format. Its syntax is:

```
[theta, din, lab] = ss2thd(Phi, Gam, E, H, D, C, Q, S, R);
```

where the input arguments are the parameter matrices in the SS model, and the outputs are the `theta`, `din`, and `lab` matrices previously described. To illustrate the use of this function, the following Example shows how to define the SS model implied by a Hodrick–Prescott filter (see Harvey and Trimbur [120]):

Example C.1.1.1 Defining a SS model in THD format

```
e4init  % Initializes E4 default options and tolerances
% *** Defines the model matrices
Phi= [1 1; NaN 1];
% The Not-A-Number value (NaN) indicates a null parameter
E= [1;NaN];
H=[1 0]; C=[1]; Q=[1/1600]; S=[0]; R=[1];
% *** Obtains the THD representation and displays
```

```
%     the resulting model
[theta, din, lab] = ss2thd(Phi, [], E, H, [], C, Q, S, R);
% The only free parameters in this model are the variances.
% Accordingly, nonzero values in the second column of theta
% indicate which parameters should remain at its current value
theta(1,2)=1; theta(2,2)=1; theta(3,2)=1;
theta(4,2)=1; theta(5,2)=1; theta(6,2)=1;
theta(7,2)=1; theta(9,2)=1;
prtmod(theta, din, lab);
```

and the resulting output is:

```
********************* Options set by user *****************
Filter. . . . . . . . . . . . . . : KALMAN
Scaled B and M matrices . . . . : NO
Initial state vector. . . . . . : AUTOMATIC SELECTION
Initial covariance of state v.  : IDEJONG
Variance or Cholesky factor? . : VARIANCE
Optimization algorithm. . . . . : BFGS
Maximum step length . . . . . . : 0.100000
Stop tolerance. . . . . . . . . : 0.000010
Max. number of iterations . . . :       75
Verbose iterations. . . . . . . : YES
************************************************************

************************** Model **************************
Native SS model
1 endogenous v., 0 exogenous v.
Seasonality: 1
SS vector dimension: 2
Parameters (* denotes constrained parameter):
PHI(1,1)     *     1.0000
PHI(1,2)     *     1.0000
PHI(2,2)     *     1.0000
E(1,1)       *     1.0000
H(1,1)       *     1.0000
H(1,2)       *     0.0000
C(1,1)       *     1.0000
Q(1,1)             0.0006
S(1,1)       *     0.0000
R(1,1)             1.0000
************************************************************
```

Note that asterisks identify which parameters are constrained to remain at their current values in any subsequent estimation.

C.1.3 Basic models

C.1.3.1 Mathematical definition

E^4 supports three basic models: VARMAX, structural econometric models, and single-output transfer functions. The VARMAX model is given by:

$$\boldsymbol{\phi_p}(\mathrm{B})\,\boldsymbol{\Phi_P}(\mathrm{B}^S)\,\boldsymbol{z_t} = \boldsymbol{G_n}(\mathrm{B})\,\boldsymbol{u_t} + \boldsymbol{\theta_q}(\mathrm{B})\,\boldsymbol{\Theta_Q}(\mathrm{B}^S)\,\boldsymbol{a_t}, \tag{C.1}$$

where S denotes the period of the seasonal cycle, B is the (already defined) backshift operator, the vectors $\boldsymbol{z_t}$, $\boldsymbol{u_t}$, and $\boldsymbol{a_t}$ are the model outputs, inputs, and errors, respectively, and the polynomial matrices in (C.1) are given by:

$$\boldsymbol{\phi_p}(\mathrm{B}) = \boldsymbol{I} + \boldsymbol{\phi_1}\mathrm{B} + \boldsymbol{\phi_2}\mathrm{B}^2 + \ldots + \boldsymbol{\phi_p}\mathrm{B}^p \tag{C.2}$$

$$\boldsymbol{\Phi_P}(\mathrm{B}^S) = \boldsymbol{I} + \boldsymbol{\Phi_1}\mathrm{B}^S + \boldsymbol{\Phi_2}\mathrm{B}^{2\cdot S} + \ldots + \boldsymbol{\Phi_P}\mathrm{B}^{P\cdot S} \tag{C.3}$$

$$\boldsymbol{G_n}(\mathrm{B}) = \boldsymbol{G_1}\mathrm{B} + \boldsymbol{G_2}\mathrm{B}^2 + \ldots + \boldsymbol{G_n}\mathrm{B}^n \tag{C.4}$$

$$\boldsymbol{\theta_q}(\mathrm{B}) = \boldsymbol{I} + \boldsymbol{\theta_1}\mathrm{B} + \boldsymbol{\theta_2}\mathrm{B}^2 + \ldots + \boldsymbol{\theta_q}\mathrm{B}^q \tag{C.5}$$

$$\boldsymbol{\Theta_Q}(\mathrm{B}^S) = \boldsymbol{I} + \boldsymbol{\Theta_1}\mathrm{B}^S + \boldsymbol{\Theta_2}\mathrm{B}^{2\cdot S} + \ldots + \boldsymbol{\Theta_Q}\mathrm{B}^{Q\cdot S}. \tag{C.6}$$

The structural econometric model is similar to the VARMAX specification (C.1), but allows for contemporary relationships between the endogenous variables and their errors. In econometric terms, it is a "structural form" while the VARMAX is a "reduced form." It is defined by:

$$\bar{\boldsymbol{\phi}}_{\boldsymbol{p}}(\mathrm{B})\boldsymbol{\Phi_P}(\mathrm{B}^S)\boldsymbol{z_t} = \boldsymbol{G_n}(\mathrm{B})\boldsymbol{u_t} + \bar{\boldsymbol{\theta}}_{\boldsymbol{q}}(\mathrm{B})\boldsymbol{\Theta_Q}(\mathrm{B}^S)\boldsymbol{a_t}, \tag{C.7}$$

with:

$$\bar{\boldsymbol{\phi}}_{\boldsymbol{p}}(\mathrm{B}) = \bar{\boldsymbol{\phi}}_{\boldsymbol{0}} + \bar{\boldsymbol{\phi}}_{\boldsymbol{1}}\mathrm{B} + \bar{\boldsymbol{\phi}}_2\mathrm{B}^2 + \ldots + \bar{\boldsymbol{\phi}}_{\boldsymbol{p}}\mathrm{B}^p \tag{C.8}$$

$$\bar{\boldsymbol{\theta}}_{\boldsymbol{q}}(\mathrm{B}) = \bar{\boldsymbol{\theta}}_{\boldsymbol{0}} + \bar{\theta}_1\mathrm{B} + \bar{\boldsymbol{\theta}}_{\boldsymbol{2}}\mathrm{B}^2 + \ldots + \bar{\boldsymbol{\theta}}_{\boldsymbol{q}}\mathrm{B}^q. \tag{C.9}$$

(Note that the polynomial matrices $\boldsymbol{\Phi_P}(\mathrm{B}^S)$, $\boldsymbol{G_n}(\mathrm{B})$, and $\boldsymbol{\Theta_Q}(\mathrm{B}^S)$ in (C.7) were already defined in (C.1).)

Last, the single-output transfer function with r inputs is given by:

$$z_t = \frac{\omega_1(\mathrm{B})}{\delta_1(\mathrm{B})}u_{1,t} + \ldots + \frac{\omega_r(\mathrm{B})}{\delta_r(\mathrm{B})}u_{r,t} + \frac{\theta_q(\mathrm{B})\Theta_Q(\mathrm{B}^S)}{\phi_p(\mathrm{B})\Phi_P(\mathrm{B}^S)}a_t, \tag{C.10}$$

where, for $i = 1, 2, \ldots, r$, we have:

$$\omega_i(\mathrm{B}) = \omega_0 + \omega_1\mathrm{B} + \ldots + \omega_{m_i}\mathrm{B}^{m_i} \tag{C.11}$$

$$\delta_i(\mathrm{B}) = 1 + \delta_1\mathrm{B} + \ldots + \delta_{k_i}\mathrm{B}^{k_i}, \tag{C.12}$$

and the polynomials $\theta_q(\mathrm{B})$, $\Theta_Q(\mathrm{B}^S)$, $\phi_p(\mathrm{B})$, and $\Phi_P(\mathrm{B}^S)$ in (C.10) are scalar versions of the autoregressive and moving average matrix factors in (C.1).

C.1.3.2 Definition in THD format

The E^4 functions `arma2thd`, `str2thd` and `tf2thd` generate the THD formulation for VARMAX, structural econometric models and transfer functions, respectively. The general syntax of these functions is:

```
[theta, din, lab] = arma2thd(FR, FS, AR, AS, V, s, G, r)
[theta, din, lab] = str2thd(FR, FS, AR, AS, V, s, G, r)
[theta, din, lab] = tf2thd(FR, FS, AR, AS, V, s, W, D)
```

where the inputs to these functions are matrices of real numbers containing: (a) the initial values for the regular and seasonal autoregressive factors, `FR` and `FS`, (b) the regular and seasonal moving average factors, `AR` and `AS`, (c) the terms related to the exogenous variables in the transfer function model, `W` and `D`, and in the VARMAX and structural econometric model, `G`, (d) the noise covariance, `V`, as well as (e) scalar integers defining the seasonal period, `s`, and the number of inputs in the VARMAX and structural econometric models, `r`.

The following Example illustrates the use of `arma2thd` by defining a quarterly airline model in THD form, displaying its structure and obtaining the matrices in its SS representation:

Example C.1.1.2 ARIMA specification in THD and SS format

```
e4init % Initializes E4 default options and tolerances
% Defines and displays the non-stationary airline model
%    (1-B)(1-B^4) y_t = (1 - .6 B)(1 - .5 B^4) a_t
[theta,din,lab] = arma2thd([-1],[-1],[-.6],[-.5],[.1],4);
theta(1,2)=1; theta(2,2)=1; % Constrains the unit roots
prtmod(theta,din,lab);
% Now obtains the matrices of the equivalent SS representation
[Phi, Gam, E, H, D, C, Q, S, R] = thd2ss(theta, din)
```

and the resulting output is:

```
************************** Model **************************
VARMAX model (innovations model)
1 endogenous v., 0 exogenous v.
Seasonality: 4
SS vector dimension: 5
Parameters (* denotes constrained parameter):
FR1(1,1)     *    -1.0000
FS1(1,1)     *    -1.0000
AR1(1,1)          -0.6000
AS1(1,1)          -0.5000
V(1,1)             0.1000
***********************************************************
Phi =
     1     1     0     0     0
```

```
     0     0     1     0     0
     0     0     0     1     0
     1     0     0     0     1
    -1     0     0     0     0

Gam =
     []

E =
    0.4000
         0
         0
    0.5000
   -0.7000

H =
     1     0     0     0     0

D =
     []

C =
     1

Q =
    0.1000

S =
    0.1000

R =
    0.1000
```

C.1.4 Models with GARCH errors

C.1.4.1 Mathematical definition of the GARCH process

E^4 allows one to combine any SEM, VARMAX, structural model or transfer function with a vector GARCH process, for a survey on this important issue, see Bollerslev, Engle, and Nelson [25].

The general conditional heteroscedastic process is defined by the distributions $\boldsymbol{a_t} \sim \text{IID}(\mathbf{0}, \boldsymbol{Q})$ and $\boldsymbol{a_t} | \, \boldsymbol{\Omega}_{t-1} \sim \text{IID}(\mathbf{0}, \boldsymbol{\Sigma_t})$ where, according to the notation already defined in Chapter 4, $\boldsymbol{\Omega}_{t-1}$ denotes the information available up to $t-1$. In a GARCH model, the conditional covariances, $\boldsymbol{\Sigma_t}$, are such that:

$$[\boldsymbol{I} - \boldsymbol{B_m}(\text{B})] \, vech(\boldsymbol{\Sigma_t}) = \boldsymbol{w} + \boldsymbol{A_k}(\text{B}) vech\left(\boldsymbol{a_t} \boldsymbol{a_t^T}\right), \tag{C.13}$$

where *vech*() vectorizes the upper triangle and main diagonal of its argument and the polynomial matrices are given by:

$$\boldsymbol{B}_m(\mathrm{B}) = \boldsymbol{B_1}\mathrm{B} + \boldsymbol{B_2}\mathrm{B}^2 + \ldots + \boldsymbol{B_m}\mathrm{B}^m \tag{C.14}$$

$$\boldsymbol{A_k}(\mathrm{B}) = \boldsymbol{A_1}\mathrm{B} + \boldsymbol{A_2}\mathrm{B}^2 + \ldots + \boldsymbol{A_k}\mathrm{B}^k. \tag{C.15}$$

E^4 manages model (C.13) in its alternative VARMAX representation. To derive it, consider the white noise process $\boldsymbol{v_t} = vech\left(\boldsymbol{a_t}\boldsymbol{a_t^T}\right) - vech\left(\boldsymbol{\Sigma_t}\right)$, which has a null conditional mean and a very complex heteroscedastic structure. Then $vech\left(\boldsymbol{\Sigma_t}\right) = vech\left(\boldsymbol{a_t}\boldsymbol{a_t^T}\right) - \boldsymbol{v_t}$, and substituting this expression in (C.13) and rearranging some terms, yields:

$$\left[\boldsymbol{I} - \boldsymbol{A_k}(\mathrm{B}) - \boldsymbol{B_m}(\mathrm{B})\right] vech\left(\boldsymbol{a_t}\boldsymbol{a_t^T}\right) = \boldsymbol{w} + \left[\boldsymbol{I} - \boldsymbol{B_m}(\mathrm{B})\right]\boldsymbol{v_t}. \tag{C.16}$$

Equation (C.16) defines a VARMAX model for $vech\left(\boldsymbol{a_t}\boldsymbol{a_t^T}\right)$, which can be written in the alternative form:

$$vech\left(\boldsymbol{a_t}\boldsymbol{a_t^T}\right) = vech\left(\boldsymbol{Q}\right) + \boldsymbol{N_t}, \tag{C.17}$$

$$\left[\boldsymbol{I} + \bar{\boldsymbol{A}}_{max(k,m)}(\mathrm{B})\right]\boldsymbol{N_t} = \left[\boldsymbol{I} + \bar{\boldsymbol{B}}_{\boldsymbol{m}}(\mathrm{B})\right]\boldsymbol{v_t}, \tag{C.18}$$

where $\bar{\boldsymbol{A}}_{max(k,m)}(\mathrm{B}) = -\boldsymbol{A_k}(\mathrm{B}) - \boldsymbol{B_m}(\mathrm{B})$ and $\bar{\boldsymbol{B}}_{\boldsymbol{m}}(\mathrm{B}) = -\boldsymbol{B_m}(\mathrm{B})$. This is the formulation supported by E^4. Note that this representation allows for IGARCH (Integrated-GARCH) components, which would require specifying some unit roots in the AR factor of (C.18).

C.1.4.2 Defining a model with GARCH errors in THD format

To define a model with GARCH errors in THD format it is necessary to obtain: (a) the THD formulation of the model for the mean, given by `t1`, `d1` and `lab1`, (b) the THD model corresponding to the model (ARCH, GARCH, or IGARCH) for the variance, given by `t2`, `d2` and `lab2`, and (c) link both formulations using the `garc2thd` function, which has the following syntax:

```
[theta, din, lab] = garc2thd(t1, d1, t2, d2, lab1, lab2);
```

Example C.1.1.3 Defining a regression model with GARCH errors

The following code generates and displays the THD representation of a regression model with GARCH errors:

```
e4init % Initializes E4 default options and tolerances

% Defines and displays a regression model with GARCH(1,1) errors:
%   y_t = .7 x_t1 + 1 x_t2 + a_t
%   a_t^2 = .1 + N_t
%   (1 - .8 B) N_t =  (1 - .7 B) v_t
```

```
[t1, d1, lab1] = arma2thd([], [], [], [], [.1], 1,[.7 1],2);
[t2, d2, lab2] = arma2thd([-.8], [], [-.7], [], [.1], 1);
[theta, din, lab] = garc2thd(t1, d1, t2, d2, lab1, lab2);
prtmod(theta, din, lab);
```

and the resulting output is:

```
************************** Model ****************************
GARCH model (innovations model)
1 endogenous v., 2 exogenous v.
Seasonality: 1
SS vector dimension: 0
Endogenous variables model:
  White noise model (innovations model)
  1 endogenous v., 2 exogenous v.
  Seasonality: 1
  SS vector dimension: 0
  Parameters (* denotes constrained parameter):
  G0(1,1)            0.7000
  G0(1,2)            1.0000
  V(1,1)             0.1000
  --------------
GARCH model of noise:
  VARMAX model (innovations model)
  1 endogenous v., 0 exogenous v.
  Seasonality: 1
  SS vector dimension: 1
  Parameters (* denotes constrained parameter):
  FR1(1,1)          -0.8000
  AR1(1,1)          -0.7000
  --------------
*************************************************************
```

C.1.5 Nested models

A "nested" model is built by combining several simple models. E^4 allows for two types of model combination: nesting in inputs and nesting in errors.

Nesting in inputs, or "endogeneization" was defined mathematically in Subsection 3.2.1. It consists in augmenting a SS model with the models for its exogenous variables which, after doing so, become endogenous variables. This operation is useful, for example, when one wants to combine a transfer function and a VARMAX model for its inputs to compute forecasts for all these variables in a single operation. Another situation where nesting is useful is when there are missing values in the inputs. "Endogeneizing" is then required because SS methods only allow for missing values in the endogenous variables; see Section 11.5.

On the other hand, nesting in errors was defined in Subsection 3.3.1. It consists in augmenting a model in SEM form with another model for its disturbances. Intuitively, it can be seen as defining a model by the product of different polynomials. This is handy, e.g., when one wants to separate the factors containing different types of roots, such as unit, complex, or real roots. It is also useful when modeling a series with several seasonal factors, corresponding to different periods.

The following examples illustrate how E^4 implements the definition of nested models.

Example C.1.1.4 Nesting in inputs

Given the models:

$$z_t = \frac{0.3+0.6\,\mathrm{B}}{1-0.5\,\mathrm{B}}\,u_t + \frac{1-0.8\mathrm{B}}{1-0.6\,\mathrm{B}+0.4\,\mathrm{B}^2}\,a_{1,t}, \qquad \sigma_{a1}^2 = 1, \tag{C.19}$$

$$(1-0.7\,\mathrm{B})\,u_t = a_{2,t}, \qquad \sigma_{a2}^2 = 0.3, \tag{C.20}$$

the code required to define and combine both models in inputs is:

```
e4init

% Obtains the THD representation for both models:
[t1,d1,l1]=tf2thd([-.6 .4],[],[-.8],[],[1.0],1,[ .3 .6],[-.5]);
[t2,d2,l2]=arma2thd(-.7,[],[],[],.3,1);

% Combines the models for the endogenous variable and the input
[theta, din, lab] = stackthd(t1, d1, t2, d2, l1, l2);
[theta, din, lab] = nest2thd(theta, din, 1, lab);
% The 3rd input argument to nest2thd determines the combination:
%   0 : nesting in errors,  1 : nesting in inputs
%   and displays the resulting model structure
prtmod(theta,din,lab);
```

and the output from the `prtmod` command is:

```
************************** Model ***************************
Nested model in inputs (innovations model)
2 endogenous v., 0 exogenous v.
Seasonality: 1
SS vector dimension: 4
Submodels:
{
  Transfer function model (innovations model)
  1 endogenous v., 1 exogenous v.
  Seasonality: 1
  SS vector dimension: 3
  Parameters (* denotes constrained parameter):
```

```
  FR(1,1)           -0.6000
  FR(1,2)            0.4000
  AR(1,1)           -0.8000
  W1(1,1)            0.3000
  W1(2,1)            0.6000
  D1(1,1)           -0.5000
  V(1,1)             1.0000
  --------------
  VARMAX model (innovations model)
  1 endogenous v., 0 exogenous v.
  Seasonality: 1
  SS vector dimension: 1
  Parameters (* denotes constrained parameter):
  FR1(1,1)          -0.7000
  V(1,1)             0.3000
  --------------
}
************************************************************
```

Note that the final model has two endogenous variables, z_t and $u_{1,t}$, and no inputs.

Example C.1.1.5 Nesting in errors

Consider now the model:

$$\left(1-0.5\mathrm{B}+0.3\mathrm{B}^2\right)(1-\mathrm{B})\left(1-\mathrm{B}^{12}\right)z_t = a_t, \qquad \sigma_a^2 = 0.2, \tag{C.21}$$

which has nonstationary

regular and seasonal AR factors. The following code defines and displays model (C.21) through nesting in errors.

```
e4init
% First obtains the THD representation of the factors
[t1, d1, l1] = arma2thd([-1], [-1], [], [], [0], 12);
[t2, d2, l2] = arma2thd([-.5 .3], [], [], [],[.2], 12);

%  and then the stacked and nested models
[ts, ds, ls] = stackthd(t1, d1, t2, d2, l1, l2);
[theta, din, lab] = nest2thd(ts, ds, 0, ls);
% The 3rd input argument to nest2thd determines the combination:
%   0 : nesting in errors,  1 : nesting in inputs

% Constrains the regular and seasonal unit roots
theta(1,2)=1; theta(2,2)=1;
prtmod(theta, din, lab);
% The variance of the model t1-d1 is ignored
%  so we can specify any arbitrary value
```

and the output from prtmod is:

```
************************** Model **************************
Nested model in errors (innovations model)
1 endogenous v., 0 exogenous v.
Seasonality: 12
SS vector dimension: 15
Submodels:
{
  VARMAX model (innovations model)
  1 endogenous v., 0 exogenous v.
  Seasonality: 12
  SS vector dimension: 13
  Parameters (* denotes constrained parameter):
  FR1(1,1)     *    -1.0000
  FS1(1,1)     *    -1.0000
  --------------
  VARMAX model (innovations model)
  1 endogenous v., 0 exogenous v.
  Seasonality: 12
  SS vector dimension: 2
  Parameters (* denotes constrained parameter):
  FR1(1,1)          -0.5000
  FR2(1,1)           0.3000
  V(1,1)             0.2000
  --------------
}
************************************************************
```

C.2 Overview of computational procedures

E^4 includes three basic groups of functions: (a) standard procedures for time series analysis, (b) signal extraction methods and (c) model estimation algorithms.

C.2.1 Standard procedures for time series analysis

The main standard procedures included are detailed in Table C.1. These functions cover basic needs such as plotting a time series, computing its autocorrelations or its descriptive statistics. They are included to make the Toolbox self-contained, but its functionality is not innovative.

The following example illustrates the use of these functions with simulated data. We will not show the outputs resulting from this and other simulation-based examples, because the exact results obtained will differ in each run.

Example C.2.1 ARIMA identification

Table C.1 *Standard statistics procedures to specify and simulate a time series model.*

Function	Description	References:
`augdft`	Computes the augmented Dickey-Fuller test	Dickey and Fuller [78]
`descser`	Descriptive statistics for a vector of time series	
`histsers`	Computes and displays histograms for a vector of time series	
`lagser`	Generates lags and leads from a data matrix	
`midents`	Multiple sample autocorrelation and partial autoregression functions	Wei [221]
`plotsers`	Displays standardized plots for a vector of time series	
`rmedser`	Displays the mean/std. deviation plot of a vector of time series	Box and Cox [28]
`transdif`	Computes differences and Box-Cox transforms	Box and Cox [28]
`uidents`	Sample autocorrelation and partial autocorrelation functions	Box, Jenkins, and Reinsel [30]
`simmod`	Simulates a homoscedastic model	
`simgarch`	Simulates a model with GARCH errors	Bollerslev, Engle, and Nelson [25]

Consider the ARMA(2,1) model:

$$\left(1-0.5\,\mathrm{B}+0.3\,\mathrm{B}^2\right) z_t = (1-0.7\mathrm{B})\, a_t, \qquad \sigma_a^2 = 0.2. \tag{C.22}$$

The following code defines it, simulates a sample, and performs a standard univariate identification process:

```
e4init
% Obtains the THD definition of the model
[theta, din, lab] = arma2thd([-.5 .3], [], [-.7], [], [.2], 1);
prtmod(theta,din,lab)

% Simulates the sample and omits the first 50 observations
z=simmod(theta,din,250); z=z(51:250,1);

% Applies several standard time series analysis tools
% Standard-deviation mean plot
rmedser(z);
% Time series plot
plotsers(z);
```

```
% Augmented Dickey-Fuller test. The second argument (2)
%  specifies the number of lags for the dynamic regression
augdft(z,2);
% Histogram
histsers(z);
% Descriptive statistics
descser(z);
% Sample autocorrelation and partial autocorrelation functions
% The second argument specifies the number of lags
uidents(z,10);
```

Example C.2.2 VARMA identification

The following code illustrates the use of the multiple time series identification functions included in E^4 by simulating a VAR(1) process and then identifying it:

```
e4init
% Obtains the THD definition of the model
Phi=[-.5 .3;NaN -.6];
Sigma=[1 .5;.5 1];
[theta,din,lab]=arma2thd([Phi],[],[],[],[Sigma],1);
prtmod(theta,din,lab)

% Simulates the sample and omits the first 50 observations
y=simmod(theta,din,250); y=y(51:250,:);

% Descriptive statistics for multiple time series
descser(y);

% Computes 10 lags of the multiple autocorrelation, partial
%  autocorrelation and cross-correlation functions
midents(y,10);
```

C.2.2 Signal extraction methods

As we saw in Chapter 4, the SS literature considers two main signal extraction problems: filtering and smoothing. Both have as their focus the estimation of the sequence of states, but based on different information sets. E^4 computes these sequences for different purposes. Filtered states are typically used to compute the Gaussian likelihood in prediction-error decomposition form (see Chapters 5 and 6), so that filtering algorithms are embedded in the likelihood evaluation functions of E^4; see Section C.2.3.

On the other hand, smoothed estimates have many uses, such as interpolating missing in-sample values (Kohn and Ansley [137]), calculating the residuals of a model allowing for missing values (Kohn and Ansley [138]), cleaning noise-contaminated samples (Kohn and Ansley [136]), decomposing a time series into

meaningful unobserved components (Harvey, [118]; Casals, Jerez, and Sotoca [41]), and detecting outliers (De Jong and Penzer [70]). Table C.2 summarizes the main E^4 functions implementing signal-extraction algorithms, with corresponding literature references where relevant.

Table C.2 *Signal extraction functions. These are the main procedures offered for time series disaggregation, structural decomposition, smoothing and forecasting.*

Function	Description	References
`aggrmod`	Computes smoothed estimates for a high-frequency time series from: (a) a sample with low and high-frequency data, and (b) the high-frequency model for the endogenous variable(s)	Casals, Sotoca and Jerez [46]
`e4trend`	Decomposes a vector of time series into the trend, seasonal, cycle and irregular components implied by an econometric model in THD format	Casals, Jerez, and Sotoca [41]
`fismod`, `fismiss`	Computes smoothed estimates of the state and observable variables of a model in THD form. The function `fismiss` allows for in-sample missing values	Casals, Jerez, and Sotoca [40]
`foremod`	Computes forecasts for a homoscedastic model in THD format	
`foregarc`	Computes forecasts for a model in THD format, allowing for GARCH errors	
`residual`	Computes the residuals for a model in THD format	

Basic Fixed-Interval Smoothing is addressed by the E^4 function `fismod`, which implements the algorithm of Casals, Jerez, and Sotoca [40]. Its syntax is:

```
[xhat, Px, e] = fismod(theta, din, z);
```

The input arguments of these functions are a THD format specification (`theta`, `din`) and the data matrix (`z`) where the missing values, if any, are coded as `NaN` (the "not-a-number" MATLAB special symbol). The output arguments of `fismod` are: `xhat`, the sequence of expected values of the state vector conditional on the sample; `Px`, a matrix with the corresponding covariances; and `e`, a matrix of fixed-interval smoothed errors. There is another smoother function, `fismiss`, which allows for missing values in z.

Smoothing is often used to decompose a time series into the sum of trend, cycle, seasonal, and irregular components. The function `e4trend` implements the exact decomposition described in Casals, Jerez, and Sotoca [41]. Its simplified syntax is:

```
[trend,season,cycle,irreg] = e4trend(theta,din,z)
```

where the input arguments `theta`, `din`, and `z` are identical to those of `fismod`, and the output arguments are fixed-interval smoothed estimates of the corresponding structural components.

The function `foremod` predicts future values of the endogenous variables for a given model. Its syntax is:

```
[zf, Bf] = foremod(theta, din, z, k, u)
```

where the input arguments `theta`, `din`, and `z` are identical to those of `fismod`, `k` is the number of forecasts to be computed, and `u` is a matrix with the out-of-sample values of the exogenous variables, if any. The forecasts and the corresponding sequence of covariances are returned in the output arguments `zf` and `Bf`. The analogous functions, `foremiss` and `foregarc`, compute forecasts for samples with missing values and for models with conditional heteroscedastic errors, respectively. Last, the residuals for a model are calculated through the function `residual` whose syntax is:

```
[z1, vT, wT, vz1, vvT, vwT] = residual(theta, din, z)
```

where the input arguments `theta`, `din` and `z` are identical to those of `fismod`. On the other hand, the output arguments are `z1`, `vT`, and `wT` which are, respectively, the model residuals, the fixed-interval smoothed estimates of the observation errors, and the smoothed estimates of the state errors. The arguments `vz1`, `vvT`, and `vwT` store the corresponding sequences of covariances.

Example C.2.3 HP filtering

The following code simulates the HP filter model defined in Example C.1.1.1 and extracts the HP trend using different procedures:

```
e4init
% Working with standard deviations improves the model scaling
sete4opt('var','fac');

Phi= [1 1; NaN 1];
E= [1;NaN];
H=[1 0]; C=[1]; Q=[sqrt(1/1600)]; S=[0]; R=[1];
% Obtains the THD representation and displays the model
[theta, din, lab] = ss2thd(Phi, [], E, H, [], C, Q, S, R);
prtmod(theta, din, lab);
% Simulates the data and omits the first 50 observations
y=simmod(theta,din,250); y=y(51:250,:);

% Extracts and plots the trend and error components
[trend,season,cycle,irreg] = e4trend(theta,din,y);
plotsers([trend,irreg]);

% e4trend does not provide variances for the trend component
```

```
% The following commands do this using fismod
[xhat, px, e] = fismod(theta, din, y); varhptrend=px(1:2:400,1);
figure; hold on
plot(varhptrend.^.5)
hold off
```

C.2.3 Likelihood and model estimation

Table C.3 summarizes the main functions in connection with model estimation. These functions specialize in the computation of likelihood values, information matrices, and parameter estimates.

E^4 includes several functions to compute the Gaussian log-likelihood. The most advanced one is `lfsd`, whose general syntax is:

```
[f, innov, ssvect] = lfsd(theta, din, z)
```

where the input arguments are identical to those defined for previous functions, and the output arguments are: (a) `f`, the value of the log-likelihood, (b) `innov`, a matrix of one-step-ahead forecast errors, and (c) `ssvect`, a matrix of estimates of the state variables organized so that its t-th row contains the filtered estimate of the state vector at time t, conditional on sample values up to $t-1$. The technical details of this function can be found in Casals, Sotoca and Jerez [47]. Other functions to compute the log-likelihood of a homoscedastic model are `lfmod` (Terceiro [206]), `lffast` (Casals, Sotoca and Jerez [44]), and `lfcd` (Casals, Sotoca and Jerez [47]). They differ only in minor details, and all have the same inputs and outputs. Last, `lfmiss` and `lfgarch` are two specialized variants of `lfsd`, allowing for missing values in the endogenous variables and GARCH errors, respectively.

As in the case of the likelihood, there are three functions dealing with computation of the information matrix: `imod`, `imiss`, and `igarch`. They all have the same syntax. For example, any call to `imod` has the following structure:

```
[std, corrm, varm, Im] = imod(theta, din, z)
```

This function computes the information matrix of `lfmod` and `lffast` (`Im`) as well as the parameter analytical standard errors (`std`), the correlation matrix of the estimates (`corrm`), and the corresponding covariance matrix (`varm`). The functions `imiss` and `igarch` do the same for `lfmiss` and `lfgarch`, respectively. If the model is misspecified or is non-Gaussian, the function `imodg` computes a robust information matrix, alternative to `imod`; see Ljung and Caines [153] and White [226].

Being able to compute the Gaussian log-likelihood is not enough to estimate the parameters of a model: one needs an iterative procedure to compute the optimal value of the parameters. The E^4 function `e4min` implements two main optimization algorithms, BFGS and Newton–Raphson (Dennis and Schnabel [74]). Its general syntax is:

```
[pnew,iter,fnew,gnew,hessin]= ...
        e4min(func,theta,dfunc,P1,P2,P3,P4,P5)
```

Table C.3 *Model estimation functions. These are the main procedures which can be applied to compute the Gaussian likelihood, its first-order derivatives, and information matrix. These functions can be fed to the standard E^4 optimizer (`e4min`) to compute ML estimates. Initial estimates can be computed using a specialized function (`e4preest`).*

Function	Description	References
Likelihood computation and derivatives		
`lfmod`, `lffast`, `lfsd`, `lfcd`, `lfper`, `lfgarch`, `lfmiss`	Compute the log-likelihood function for a model in THD form. `lfmod`, `lffast`, `lfsd` and `lfcd` support models with homoscedastic errors. `lfper` supports periodic models. Last, `lfgarch` admits for GARCH errors and `lfmiss` allows for in-sample missing values	Terceiro [206], Casals, Sotoca and Jerez [44], Casals, Sotoca and Jerez [47]
`gmod`, `gmiss`, `ggarch`	Computes the analytical gradient for `lfmod` and `lffast` (`gmod`), `lfmiss` (`gmiss`) and `lfgarch` (`ggarch`)	Casals, Sotoca and Jerez [45], Casals, and Sotoca [43]
`imod`, `imiss`, `igarch`, `imodg`	Compute the Gaussian analytical information matrix for `lfmod`, `lffast`, `lfsd` and `lfcd` (`imod`), `lfmiss` (`imiss`) and `lfgarch` (`igarch`). The function `imodg` computes the robustified information matrix for `lfmod` and `lffast`	Terceiro [206], Ljung and Caines [153], White [226]
Model estimation		
`e4min`	Computes the unconstrained numerical minimum of a nonlinear function using the BFGS or the Newton-Raphson algorithms	Dennis and Schnabel [74]
`e4preest`	Computes a fast estimate of the parameters for a model in THD format	Garcia–Hiernaux, Jerez, and Casals [104]
`prtest`	Displays estimation results	
Toolbox defaults and options		
`e4init`	Initializes the global variables and defaults	
`sete4opt`	Modifies the toolbox options	

The operation of `e4min` is as follows. Starting from the initial estimate of the parameters in `theta`, it iterates on the objective function whose name is stored in the argument `func`. The iteration can be based on the analytical gradient or on a numerical approximation, depending on whether `dfunc` contains the name of the analytical gradient function or is an empty string, `''`. Finally, the stopping criterion takes into account the relative changes in the values of the parameters and/or the size of the gradient vector. The parameters `P1, ..., P5` are optional. If they are specified, its values are fed as additional input arguments to the objective function `func`.

Once the iterative process is stopped, the function returns: `pnew`, the value of the parameters; `iter`, the number of iterations performed; `fnew`, the value of the objective function at `pnew`; `gnew`, the analytical or numerical gradient, depending on the contents of `dfunc`; and finally `hessin`, a numerical approximation to the Hessian of the objective function.

The auxiliary function `sete4opt` manages different options controlling the performance of the optimizer, including type of algorithm, tolerance for the stopping criteria, or the maximum number of iterations allowed.

Another important function for model estimation is `e4preest`. It computes fast and consistent estimates of the parameters in `theta` which, in most cases, are adequate starting values for likelihood optimization. The syntax of this function is:

```
thetanew = e4preest(theta, din, z)
```

where the input arguments are identical to those of `lffast`, and the estimates are returned in `thetanew`. The algorithm implemented in `e4preest` is described by Garcia–Hiernaux, Jerez, and Casals [104].

The following code simulates, estimates and applies different signal-extraction procedures to the nonstationary

airline model structure defined in Example C.1.1.2:

Example C.2.4 Simulation, estimation and signal extraction

```
e4init
% Defines and displays the non-stationary airline model
%    (1-B)(1-B^12) y_t = (1 - .6 B)(1 - .5 B^12) a_t
[theta,din,lab] = arma2thd([-1],[-1],[-.6],[-.5],.1,12);
theta(1,2)=1; theta(2,2)=1;  % Constrains the unit roots

% Simulates the sample and omits the first 50 observations
z=simmod(theta,din,250); z=z(51:250,1);

% Computes preliminary estimates
theta=e4preest(theta,din,z);
% ... and then ML estimates
[thopt, it, lval, g, h] = e4min('lffast', theta,'', din, z);
% Computes the analytical standard errors
[std, corrm, varm, Im] = imod(thopt, din, z);
% Summary of estimation results
prtest(thopt, din, lab, z, it, lval, g, h, std, corrm)

% Residual diagnostics
ehat=residual(thopt, din, z);
descser(ehat);
plotsers(ehat);
uidents(ehat,39);
```

```
% Structural decomposition
[trend,season,cycle,irreg] = e4trend(thopt,din,z);
plotsers([trend,season,irreg]);

% Computes and displays 10 forecasts and their standard errors
[zfor, Bfor] = foremod(thopt, din, z, 10);
[zfor Bfor.^.5]

% Marks obs. 10 and 40 as missing and estimates them
z1=z; z1(10)=NaN; z1(40)=NaN;
[zhat, pz] = fismiss(thopt, din, z1);
[z(1:50) z1(1:50) zhat(1:50)]
```

C.3 Who can benefit from E^4?

E^4 combines the intellectual input from many people who faced special needs that commercial software could not satisfy. If these needs coincide with yours, perhaps you should consider using E^4 for your work.

First of all, we needed **transparency**. Statistical software often hides relevant computational outputs, such as the likelihood, gradient, or the information matrix conditioning number. This happens because these results convey one of two messages: either that everything worked as expected, or that there is a problem with the model, the data, or the software. Providing this information may not be the best way to do "business" but, for some users at least, they are important pieces of the puzzle.

Second, we needed **reliability**. Mainstream software tends to prioritize computational speed at the expense of numerical accuracy. This can be done in different ways such as, e.g., choosing fast but sloppy computational algorithms or easy-to-satisfy default convergence criteria. Speed is crucial for a real-time computational system. However, for academic work and many professional applications, speed is attractive but not so valuable when compared to precision or computational reliability.

Third, we needed **consistency**. Common time series packages implement different procedures to perform the same task for different models, and this may produce substantial inconsistencies. These inconsistencies are often due to the intrinsic difficulty of making a clean distinction between models and procedures. For example, a standard time series package may have a specific procedure to forecast an ARIMA model and another one for VARMA, without any guarantee that their outputs will be mutually consistent. E^4 avoids this shortcoming by translating internally all the models supported to the SS form, so that, for example, in the ARIMA and VARMA case, a unique forecasting procedure is employed.

Our fourth and last need was strictly personal: we wanted to exploit the software generated by research. Academic work often produces neat pieces of code which do a specific task well. This software could be useful to other people in an adequate framework, but if this framework does not exist, it will probably vanish forever into the "limbo" reserved for neat pieces of code after the paper is published. On the other

hand, a coordinated effort to put together, organize, and document these scientific by-products, may be academically profitable for both authors and end-users.

Our personal answer to these questions was E^4, a MATLAB toolbox intended to harbor the collection of results from a research line in SS methods, whose origins extend back into the early 1980s.

E^4 is our main research tool and, accordingly, it prioritizes academic needs. As a consequence, its functions provide a transparent, well-documented, and user-configurable suite of outputs that provides many clues about the problem such as lack of convergence or ill-conditioning. Also, the code has been carefully optimized for numerical accuracy and robustness, making an intensive use of stability-inducing methods, such as factorization and scaling. Finally, computational tolerances are transparent, user-configurable, and have strict default values.

E^4 does all the important calculations using the equivalent SS form of the models employed. As noted before, this feature is important in enforcing consistency, since all the core algorithms are developed for a SS model. Therefore, when an E^4 function receives an input model, the first thing it does is obtain the equivalent SS representation, and only then does it apply the corresponding procedure. This approach cleanly separates models from procedures, assures a high degree of coherence in the results, simplifies code development, and, last but not least, makes the toolbox very easy to extend, because implementing new models with equivalent SS representations only requires coding new user interface functions. Conversely, a new procedure only needs to support the SS model, because, thanks to the bridge functions provided, it can be immediately applied to all the models supported.

E^4 has been an effective repository to organize and distribute the code resulting from our research. Every new research project we undertake adds new functionality to E^4, which is documented not only in the user manual, but also, with exceptional depth, in the corresponding papers. In 2000 we launched a website (`www.ucm.es/e-4/`), containing the source code for the Toolbox as well as complete documentation. Appendix D provides more information about these issues.

Appendix D

Downloading E^4 and the examples in this book

D.1 The E^4 website

You can find the E^4 website at the URL: `https://www.ucm.es/e-4/`. This website distributes many E^4-related materials, including the complete toolbox code, a user manual, as well as all the code and data required to replicate the examples in this book.

D.2 Downloading and installing E^4

To download E^4 you need to:

1. Access `https://www.ucm.es/e-4` using your WEB browser.
2. Click the "Downloads" item in the left-hand-side menu (alternatively, you can type the address `https://www.ucm.es/e-4/downloads` in the navigation bar of your browser).
3. Download the file `E4.zip` in a folder of your choice and extract the files contained in it. By doing so, you will obtain 150+ text files, most of them with the MATLAB `.m` extension, which constitute the core of E^4 code.
4. Optionally, download other optional materials such as: (a) a user manual for E^4, (b) source code and data for the examples in the user manual, (c) source code and data for the examples in this book, as well as (d) the documentation for several groups of specialized functions which are distributed with E^4 but not covered by the user manual, and (e) beta-versions of some functions which are not yet included in the core of E^4.

The only installation required consists of including the folder(s) with E^4 code in the MATLAB search path[1]. To this end, you will need to select "Set Path" from the File menu in the initial MATLAB window and the add the E^4 folder using the menu options. Alternatively, you can access the same menu by typing `pathtool` at the MATLAB Command Window prompt. To edit this search path you may need to execute MATLAB with full administrative rights. Refer to your Operating System documentation, or to specialized Internet resources, to see how to do this.

To use the E^4 functions, you will also need to type the command:

```
e4init;
```

at least once at the beginning of each E^4 session. This command initializes the toolbox options and parameters, and its output is similar to:

```
 EEEEEEEEE   444  444
EEEEEEEEEEE  444  444
EEE          44444444
EEE           4444444
EEEEEEE           444
EEEEEEE           444
EEE
EEE
EEEEEEEEEE
 EEEEEEEE

Toolbox for State Space Estimation of Econometric Models
                  Version  APR-2013

Web: www.ucm.es/e-4/

********************** Options set by user ********************
Filter. . . . . . . . . . . . . : KALMAN
Scaled B and M matrices . . . . : NO
Initial state vector. . . . . . : AUTOMATIC SELECTION
Initial covariance of state v.  : IDEJONG
Variance or Cholesky factor? .  : VARIANCE
Optimization algorithm. . . . . : BFGS
Maximum step length . . . . . . : 0.100000
Stop tolerance. . . . . . . . . : 0.000010
Max. number of iterations . . . :       75
Verbose iterations. . . . . . . : YES
***************************************************************
```

Bear in mind that E^4 will not work properly if this `e4init` is not run.

[1]This search path is a list of folders where MATLAB will look for the code of any function which is not part of its core.

After these installation and initialization steps have been completed, MATLAB is able to use the E^4 library.

D.3 Downloading the code for the examples in this book

In the Downloads section of [`https://www.ucm.es/e-4/downloads`] you will find an item titled "Source code and data for the examples in the book: State-Space Methods for Time Series Analysis: Theory, Applications and Software." By clicking on this item you will download a file titled `SSBookExamples.zip`. You should extract its content to any folder of your choice. By doing this, you will create the folders: `ExamplesChapter3`, `ExamplesChapter4`,... and so on until `ExamplesChapter11`. These folders contain the source code and data required to replicate the examples in the corresponding Chapter.

In each of these folders you will find either code files or new folders. In both cases, the name of the file/folder refers to a specific example, e.g. the files contained in `ExamplesChapter3` are:

```
Example0311.m
Example0312.m
Example0313.m
Example0321.m
Example0322.m
Example0331.m
```

where the first nine characters, `Example03`, refer to the Chapter, the tenth character refers to the Section containing the example, and the last character is the sequential number of the example. Therefore, the name `Example0311.m` refers to the first example in Chapter 3, Section 1, `Example0312.m` corresponds to the second example in the same Chapter and Section, and so on.

In other cases the main folder will contain several sub-folders. For example, `ExamplesChapter8` includes the sub-folders:

```
Example0851 (Simulated data)
Example0852 (Wheat price)
Example0853 (Lydia Pinkham data)
```

where the first part of the folder name follows the naming convention described above, and the free text within parentheses refers to the specific dataset employed. Each of these folder contains MATLAB code (in `.m` files) and data (in text files with the extension `.dat`). In some cases you will also find and Excel worksheets with additional information about the dataset and the results.

The `ExamplesChapter11` folder is a little different since the examples were coded in `R`, and it contains just three files:

```
Example1161.R
Example1162.R
whale.txt
```

where `whale.txt` is the dataset required by the second example in Section 6 (`Example1162.R`). The first two files are chunks of `R` code meant to be executed

sequentially, with accompanying comments to indicate what exactly is being computed. Each of these may require the user to load specific R libraries[2]. In particular, both examples make liberal use of the "astsa" library that accompanies the book by Shumway and Stoffer [196]

[2]See the website https://www.r-project.org/ for detailed information on R.