Reusable components for seasonal adjustment: a new implementation of Tramo-Seats

Jean Palate
REUSABLE COMPONENTS FOR SEASONAL ADJUSTMENT:
A NEW IMPLEMENTATION OF TRAMO-SEATS.

Jean Palate
(jean.palate@nbb.be)
R&D Unit
Department of Statistics
ACKNOWLEDGEMENTS

The work presented in this paper would not have been possible without the support of A. Maravall and G.L. Caporello, and, indirectly, of the Bank of Spain. The fine solutions developed in the original Tramo-Seats program have been a great inspiration to us.

INTRODUCTION

The National Bank of Belgium produces many official statistics, like, among others, financial statistics, the balance of payments, the national accounts or the external trade statistics.

In the production and in the analysis of those statistics, we are continuously confronted with some time series problems:

- Outliers detection and estimation of missing values are a constant concern in the production process.
- The estimation of some figures relies on complex statistical methods: business surveys have to be treated by seasonal adjustment procedures, quarterly national accounts are partly based on temporal disaggregation techniques, ...
- A critical analysis of the statistics often requires the modeling of the series.
- ...

The program Tramo-Seats of Gomez-Maravall-Caporello offers efficient solutions to several of those problems. It has been used at the National Bank of Belgium for a long time. However, some of its constraints and limitations - essentially technical - appeared progressively. Keeping Tramo-Seats as a guideline, we have built a new software toolbox in the time series domain.

The paper explains in a first part our motivations and the chosen technical solution. In a second part, the content of the library is quickly presented. The third part describes our implementation of Tramo-Seats. A final part is devoted to our framework on state space forms, which comes as an additional tool.

Detailed information on Tramo-Seats itself may be obtained, among others, on the web site of the Bank of Spain (http://www.bde.es*).

The library described in this paper will be freely available on the site of the National Bank of Belgium (http://www.nbb.be/app/dqrd/index.htm) at the end of June 2006.

MOTIVATIONS

Statistical algorithms must often be integrated in completely different tasks. Outliers detection, for example, can be used in batch processing of many series; seasonal adjustment must be embedded in some automated production chains, like the business surveys; advanced graphical interfaces should also be available for detailed analysis, while, for some unskilled employees, black-box functions integrated in Excel are the preferred solution. The current implementations of Tramo-Seats (DOS programs or TSW) do not have the purpose of satisfying all those needs.

A first attempt to improve the integration of Tramo-Seats in different programs was the encapsulation of the file based input/output used by the FORTRAN module in a programming interface (using the COM technology). However, the use of file based I/O led to substantial performance penalties and to several shortcomings or complications in results retrieval. A better solution, that replaced I/O operations by direct function calls into the FORTRAN library, was then realized in collaboration with G.L. Caporello.
Despite its interest, that last solution appeared quite quickly frustrating. Indeed, it didn’t provide a true object-oriented (OO henceforth) solution: the results supplied by the FORTRAN modules were “dead” information; it was not possible to enrich those structures by the methods/properties that should have been part of their definition, as is usual in OO design; that was an annoying limitation in building advanced interfaces or extensions. But more importantly it didn’t fit with our wish to get a coherent time series framework. Tramo-Seats is designed to provide an efficient solution to some very specific problems. Nevertheless some of its concepts/algorithms can be very useful to tackle other ones. Unfortunately, the closed character of the program doesn’t allow their reuse.

These considerations explain why we decided to build a completely new open software library in the time series domain, using Tramo-Seats as a guideline.

As far as technology is concerned, OO components based on standard technologies form a very interesting solution in terms of integration, extensibility an reusability. Because their underlying technology is largely accepted by the software realm, OO components can quite easily be embedded in many different environments, from commercial software to a variety of tools for in-house developments. Java is a popular solution when portability matters, while .NET becomes the norm for Windows applications.

We provide implementations of our time series library in both technologies, using the same object-model. It should also be stressed that, compared to other more traditional development languages like FORTRAN or C, those technologies yield much more robust solutions.

**GENERAL PRESENTATION OF THE LIBRARY**

The library presented in this paper is not a program in the usual sense. It is a toolbox that can be used to solve a lot of problems in the time series domain. So it is intended more for advanced users or for programmers than for "pure" statisticians.

It is designed to provide efficient solutions that can be easily plugged in a variety of applications, ranging from rich graphical interfaces to batch processing.

The library can be seen as an extension of standard languages (C#, VB or Java) with concepts that should be meaningful to statisticians. The content of the library has been defined as follows: the main high-level concepts/algorithms handled by Tramo-Seats were identified in a first step. Then, the underlying lower-level entities were recursively defined. New high-level concepts that do not necessary belong to the Tramo-Seats sphere were gradually added (for instance temporal disaggregation methods, structural models or the X11 algorithm). Existing lower-level entities have been, as far as possible, reused or adapted. So the library provides a variety of interconnected objects, ranging from high-level entities to very basic ones. The most important concepts are listed below (appendix 1 links them with the actual structure of the library):

- **High-level concepts/algorithms**
  - Tramo-Seats
  - Temporal disaggregation
  - Estimation of structural models
  - X11
  - ...

- **Medium-level concepts**
  - Time series
  - ARIMA models
  - UCARIMA models
  - Wiener-Kolmogorov filters
  - State space forms, Kalman filters/smoothers
  - GLS with ARIMA noise
The goal of this introductory paper is certainly not to offer a complete overview of the library - it contains several hundreds of classes - , but to give some feeling of its capabilities. In that prospect, examples of code (written in C#\(^1\)) are systematically provided; they are enclosed in frames and they can be skipped in a first reading. The exact syntax of those examples is not the important point; it will never be explained. However, we hope that they will be sufficiently clear to arouse the interest of the reader.

NEW IMPLEMENTATION OF TRAMO-SEATS

INTRODUCTION

Our implementation of Tramo-Seats is not, except for some very specific section of the code, a translation of the original FORTRAN program. Instead of a closed algorithm, it has to be considered as a sophisticated application build on our time series library.

In spite of the completely different structure of the modules, we have tried to get results as similar as possible to those produced by the FORTRAN program.

Our solution implements most of the possibilities provided by the original program. Some features have been added. The most important ones are:

- calendar effects for "end of period" observations,
- new kinds of outliers: seasonal outlier, displacement outlier (shift between two adjacent periods), ...
- estimation of shorter annual series (8 years).
- estimation of the unobserved components of Seats by means of the exact Kalman smoother.

The whole library can be used to extend the analysis of the results. An example of new features in that domain is the derivation of the one-step-ahead residuals in Tramo and their inspection by means of a periodogram.

\(^1\) Standard language for the .NET framework. C# is very similar to Java and to C++.
OBJECT-ORIENTED IMPLEMENTATION.

Compared to the more traditional procedural approach, the OO paradigm implies an important shift in the reasoning. While algorithms are the central notion in the procedural approach, OO programming focuses on the underlying theoretical concepts. Algorithms can be seen as a way to link different conceptual objects. For example, if Seats is (partly) build around the canonical decomposition of an ARIMA model, an OO implementation of the same problem will be organized around the concepts of ARIMA and UCARIMA models and their properties. The canonical decomposition will then be just a tool that derives an UCARIMA model from an ARIMA model. Complex algorithms themselves are often encapsulated in entities that have to monitor the processing. These entities are sometimes composed of parts that can be easily changed and that can store intermediary information. The final results are provided by the monitoring object.

Our implementation of Tramo-Seats relies on that model. In the next sections, we shall put forward the main concepts handled by the library and the main algorithms with their controlling entities.

BASIC CONCEPTS

Tramo-Seats is the acronym of "Time Series Regression with Arima Noise, Missing Observations and Outliers" and "Signal Extraction in Arima Time Series". Two central notions appear immediately from that description, namely time series and ARIMA model.

Time series

The time series class of the library (\texttt{Nbb.TimeSeries.SimpleTS.TS}) describes a simple time series defined on a regular periodicity (from monthly to yearly observations), and with possible missing values\(^3\). Time series can be created from a starting period and an array of values or from a set of couples (date/value). Simple operations on time series are part of the class.

```csharp
// TIME SERIES
// CREATION
// creation from a starting period (frequency/year/0-based period) and an array of values
double[] data = ...;
TS ts1 = new TS(TSFrequency.Monthly, 1980, 0, data);

// creation from [(date/obs)], through the TSDataCollector object
TSDataCollector collector = new TSDataCollector();
for (int i=0; i<n; ++i)
{
    DateTime date = ...;
    double val = ...;
    collector.AddObservation(date, val);
}

// immediate conversion to other frequencies are possible, following different aggregation modes
TS ts2 = collector.Make(TSFrequency.Monthly, TSAggregationType.Sum);

// OPERATIONS
TS sum = ts1 + ts2;

double[] weights = new double[] { .125, .25, .25, .25, .125 };
TS ma = sum.MovingAverage(weights, true, true);

// DATA RETRIEVAL
```

\(^2\) The classes mentioned in this paper belong to the .NET libraries.

\(^3\) Missing values are identified by the value \texttt{Double.NaN} (= Not a Number).
Any link to databases is excluded from the model; this is an imperative condition for the portability of our solution. However, building time series objects from any data provider (databases, files or applications) is usually straightforward.

**ARIMA model**

Tramo deals with SARIMA models, as defined in Box and Jenkins. Some restrictions are set on the length of the auto-regressive and moving average polynomials. That kind of model is handled in the `Nbb.SArima.SArimaModel` class, using the same restrictions as the original program. In its decomposition process, Seats needs a more general form of ARIMA model, identified by unrestricted auto-regressive and moving average polynomials. The `Nbb.Arima.ArimaModel` class provides an implementation of those models. Both classes can be used interchangeably in a lot of routines.

Usual properties or methods of ARIMA models, like their ACGF (of their stationary transformations) or their (pseudo-)spectrum are available.

The library implements also more tricky operations, like the sum of two orthogonal ARIMA models or their transformation to non-invertible models. This part of the library is heavily used in our implementation of Seats.

```java
// CREATION
// ARIMA model
Polynomial ar = new Polynomial(2);
ar[0]=1; ar[1]=-.3; ar[2]=.1;
Polynomial ma = new Polynomial(1);
ma[0]=1; ma[1]=-2;
// Polynomials in the Backshift operator are represented by the BFilter class
BFilter bar = new BFilter(ar), bma = new BFilter(ma);
ArimaModel arima = new ArimaModel(bar, bma, 1);

// SARIMA model
SArimaSpecification spec = new SArimaSpecification(12);
spec.Airline();
SArimaModel sarima = new SArimaModel(spec);
sarima.SetTheta(1, -.8);
sarima.SetBTheta(1, -.4);

// OPERATIONS and PROPERTIES
ArimaModel sum = arima + sarima;
Spectrum spectrum = sum.Spectrum;

// non invertible model...
Spectrum.Minimizer min = new Spectrum.Minimizer();
min.Minimize(spectrum);
ArimaModel noninvertible = sum - min.Minimum;

// ACGF of the stationary transformation.
URBFilter ur = new URBFilter(); // ur will contain the unit roots
ArimaModel stationary = noninvertible.DoStationary(ur);
ACGF acgf = stationary.ACGF;
```

---

4 We consider SARIMA model \((P \ D \ Q)(BP \ BD \ BQ)\) with \(P<=3, D<=2, Q<=3, BP<=1, BD<=1, BQ<=1\).

5 From an OO point of view, both classes implement the same abstract interface, namely the `Nbb.Arima.IArimaModel`; many operations are defined in terms of that abstract interface.
UCARIMA models and Wiener-Kolmogorov-filters

Seats is centered around the notion of UCARIMA models and their estimation by means of the Wiener-Kolmogorov filters (WK henceforth). An UCARIMA model, which is simply a list of orthogonal ARIMA components, is derived in Seats from an SARIMA model through the canonical decomposition.

The new implementation has been guided by the following considerations: the framework developed in the program of Gomez and Maravall is not specifically linked to the canonical decomposition of an ARIMA model; it can indeed be applied to any kind of UCARIMA-type models; other popular time series models, including the structural models, belong to that class. So, the library has been designed to handle very general UCARIMA models. Some of the key-features related to that point are listed below.

An UCARIMA model -Nbb.UCARima.UModell - can be defined in one of two ways. On the one hand, the models of its components can be specified directly ⁶. On the other hand, it can be derived (if possible) from an ARIMA model following a decomposition scheme similar to the one used in Seats; the identification of the components and the choice of a noninvertible component can be defined in a flexible way.

The complete analysis framework developed in Seats around the WK filters has been derived for any valid UCARIMA model. Detailed information on the theoretical models of the preliminary and final estimators and of the final and revision errors is available in the Nbb.UCARima.WKEstimators and Nbb.UCARima.WKPreliminaryEstimatorProperties classes. The Nbb.UCARima.WKEstimates class provides estimates of the components of any valid UCARIMA model using the fast Burman-Wilson algorithm.

The following code shows some of those facets.

```java
// WK implementation of the Hodrick-Prescott filter
Polynomial d = new Polynomial(1);
d[0] = 1;
d[1] = -1;
BFilter D = new BFilter(d); //filter defined with the Backshift operator
BFilter D2 = D * D;
double alpha = 1600;
ArimaModel signal = new ArimaModel(D2, null, 1);
ArimaModel noise = new ArimaModel(null, null, alpha);
UCModel hp = new UCModel();
hp.AddComponent(signal);
hp.AddComponent(noise);
WKEstimates wk = new WKEstimates();
wk.UCModel = hp;
wk.Data = data;
double[] es = wk.Estimates(0, true);
double[] en = wk.Estimates(1, true);

// BASIC STRUCTURAL MODEL
BsmSpecification spec = new BsmSpecification();
spec.SeasonalModel = SeasModel.HarrisonStevens;
BasicStructuralModel bsm = new BasicStructuralModel(spec, 12);
bsm.LevelVar = 1; bsm.SeasVar = .5; bsm.NoiseVar = .1;
```

⁶ The library contains an advanced implementation of structural models, using, among others, different kinds of seasonal components. UCARIMA models are automatically derived for those models.
ALGORITHMS

Tramo

More specific concepts linked to Tramo are regression models with Arima noise and specialized regression variables, like calendar effects and outliers. Several classes facilitate the use of these concepts. Their description is out of the scope of this introductory paper. However, the concept of outlier is shortly presented in appendix 2.

The whole processing of Tramo is described in the following (simplified) scheme.

```java
UCModel bsmUcm = bsm.UCModel();
wk.UCModel = bsmUcm;

double[] trend = wk.Estimates(0, true);
double[] seas = wk.Estimates(1, true);
double[] irr = wk.Estimates(2, true);

WKEstimators estimators = new WKEstimators(bsmUcm);
// Model of the concurrent revision error for the first component (trend)
LinearModel tmodel = estimators.RevisionError(0, 0);
```

The class `Nbb.TramoSeats.Tramo` monitors that processing. It contains sub-entities that correspond to the main building blocks of the schema.

The complete execution of Tramo can be decomposed in the following steps:

1. Preliminary tests (log/level, calendar)
2. Model identification (Differentiation / Stationary ARMA)
3. Outliers detection
4. Final estimation and validity tests
5. Analysis (residuals)
3. Initialization of that entity with the specifications.
4. Actual execution for a given time series (step 3 and 4 can be put together)
5. Results retrieval from the monitoring object and from its sub-entities.

The execution of the Tramo processing itself needs only a few lines of code. Common results retrieval from the Tramo object are also straightforward. However the model allows a deep inspection of the processing through, notably, its different sub-blocks. It should be pointed out that many results are computed on request. So the processing is not burdened by unnecessary computations.

Finally, we do not consider the analysis module of Tramo to be part of the algorithm. Any modification or extension in that domain has no impact on the processing itself. Behind the splitting up of Tramo in sub-entities, appears the idea of interchangeability: it should be possible to replace any block by another one that provides the same services without disturbing the rest of the processing. For instance, a completely new module for the selection of the ARMA specification could be put in the place of the current one, while keeping unchanged the rest of the process. Such an advanced feature is not yet available.

However, the use of each of the current blocks is possible outside the whole processing.

The following example illustrates those different points.

```csharp
// Creation of a time series
double[] data = ...;
TS ts = new TS(TSFrequency.Monthly, 1980, 0, data);

// Step 1. Creation of the specification
TramoSpecification tramospec = new TramoSpecification();
tramospec.ModelIdentification.IsEnabled = true;
tramospec.OutliersDetection.All();

// Step 2. Controlling object
Tramo tramo = new Tramo();

// Step 3-4. Execution (with a pre-initialization by the given spec)
if (tramo.Process(ts, tramospec))
{
    // step 5. Simple results retrieval
    ILIst<OutlierEstimation> outliers = tramo.OutliersEstimation();
    foreach (OutlierEstimation outlier in outliers)
    {
        //...
    }

    // Advanced results retrieval. We examine the preferred ARMA specifications in the last run of the automatic identification procedure.
    for (int i = 0; i < 5; ++i)
    {
        SArmaSpecification curspec =
            tramo.ArmAIdentifier.PrefereedModels[i].HR.Specification;
        double BIC = tramo.ArmAIdentifier.PrefereedModels[i].BIC;
    }

    // Analysis.
    // dynamic making of the one step ahead residuals, and creation of a periodogram
    TS res = tramo.OneStepAheadResiduals();
    Periodogram periodogram = new Periodogram(res.Values.DataCopy());
}

>Date/Time

// Individual use of a subroutine of Tramo. Identification of the unit roots
UnitRootsSearcher ursearcher = new UnitRootsSearcher();
```
Seats

A large part of Seats is devoted to the analysis of an UCARIMA decomposition. That part, which has been considered above, doesn't belong to the processing itself. So, the scheme of Seats is quite simple.

It should be noted that our implementation of Seats doesn't provide any estimation procedure of the parameters of the ARIMA model. It relies on the routines developed in Tramo for that purpose. An estimation based on the Kalman smoother is also provided. It will be discussed later on.

Tramo-Seats

The complete Tramo-Seats algorithm is just the concatenation of the two modules, with some final adjustments.
The ideas developed in the implementation of Tramo - namely the use of an entity to monitor the processing - can be transposed to the case of (Tramo-)Seats. This is illustrated in the code below:

```java
// Creation of a time series
double[] data = ...;
TS ts = new TS(TSFrequency.Monthly, 1980, 0, data);

// SEATS. Comparison between WK and Kalman estimates and their variances.
// Step 1. Creation of the specification
SeatsSpecification seatsspec = new SeatsSpecification();
TS wk_trendstde, kalman_trendstde, wk_stochsa, kalman_stochsa;

// Step 2. Controlling object
Seats seats = new Seats();

// Step 3-4. Execution (with a pre-initialization by the given spec)
if (seats.Process(ts, seatsspec))
{
    // step 5. Simple results retrieval
    wk_stochsa = seats.StochasticComponent(CmpType.SeasonallyAdjusted);
    wk_trendstde = seats.StdevComponent(CmpType.TrendCycle);
}

seatsspec.Method = ComponentsEstimationMethod.Kalman;
if (seats.Process(ts, seatsspec))
{
    kalman_stochsa = seats.StochasticComponent(CmpType.SeasonallyAdjusted);
    kalman_trendstde = seats.StdevComponent(CmpType.TrendCycle);
}

// TRAMO-SEATS

// Step 1. Creation of the specification
TramoSpecification tramospec = new TramoSpecification();
tramospec.ModelIdentification.IsEnabled = true;
tramospec.OutliersDetection.All();

// Step 2. Controlling object
TramoSeats tramoseats = new TramoSeats();
```
// Step 3-4. Execution (with a pre-initialization by the given spec)
if (tramoseats.Process(ts, tramospec, seatsspec))
{
    // step 5. Simple results retrieval
    TS sa = tramoseats.FinalComponent(CmpType.TrendCycle);

    // Estimated model, from the Tramo sub-entity.
    SArimaModel sarima = (SArimaModel) tramoseats.Tramo.Estimation.Model;

    // Variances of the revision error for the first component (trend) from
    // the concurrent estimator to the next 100 estimators), from the Seats sub-
    // entity.
    WKEstimators wks = tramoseats.Seats.WKEstimators;
    double[] vrev = wks.RevisionErrorVariance(0, true, 0, 100);
}

SUMMARY

Our OO implementation of Tramo-Seats provides, in the first analysis, an easy way to plug the
powerful algorithm of Gomez-Maravall-Caporello in other programs. The model allows default
processing as well as advanced uses.
The concepts and the algorithms developed around Tramo-Seats can be the starting point for new
developments or extensions that are not possible using the current program.

STATE SPACE FORMS

INTRODUCTION

In the perspective of building a coherent framework for time series analysis, the need for an
enlarged state space forms (SSF henceforth) library emerged quickly. An efficient handling of
problems like the estimation of structural models, the temporal disaggregation of time series or the
handling of multivariate models rely often on the availability of flexible SSF.

More specifically, as far as Tramo-Seats is concerned, it seems interesting to compare the WK
estimates with the exact Kalman smoother solution for UCARIMA models; more particularly, the
variances of the different estimators should be investigated more thoroughly\(^7\).

Tramo already contains an advanced handling of Kalman filters (KF henceforth) for ARIMA models.
To achieve an efficient processing, it relies on some important characteristics:

- In function of the needs, different solutions are used: fast estimation is handled by means
  of Chandrasekhar recursions, while forecasts are provides by means of ordinary KF; for
  numerically unstable problems, square root filters are also available.
- Direct matrix computations are avoided; the specific structure of the model is used to
  provide efficient transformations of the state vector and of the prediction errors covariance
  matrix.

Our SSF framework exploits those ideas and enlarges them to deal with a variety of other time
series models, including UCARIMA models, structural models and multivariate SUTSE models.

\(^7\) As expected, the discrepancies between WK and KF estimates (resulting from the semi-infinite/finite
approach) are nearly always negligible. However, the difference between their variances can become very
large, even for long series, when the MA part of the model contains quasi-unit roots.
GENERAL PRESENTATION

The general linear gaussian state-space model can be written in many different ways; our framework uses the presentation of Durbin and Koopman \[11\] (see appendix 3). Despite its simple form, it can handle many common time series problems.

General algorithms for filtering and smoothing, based on matrix computation, can be quite easily developed. However, they often lead to poor performances. Faster solutions, like Tramo, replace matrix computations by direct transformations of the states/covariance matrices. This is of course only possible when the exact nature of the SSF is known. The OO-design of the library is based on these considerations. Common features, like filtering or smoothing, are defined as much as possible in terms of such transformations. The whole logic of filtering and smoothing is handled by common controlling entities that delegate the actual computation to specific implementations for each kind of model.

Fast univariate SSF implementations are provided for AR(I)MA, UCARIMA and structural models. Multivariate SSF implementations are limited to SUTSE models. An automatic transformation of multivariate problems to univariate ones is also available. Regression effects can be treated by GLS, as in Tramo, or by incorporating the regression effects in the state vector. Finally, the diffuse initialization is handled by the device of Durbin and Koopman \[11\] unless another method is specified.

Filtering can be handled through different algorithms, that can be classified according to the following schema:

- **Univariate**
  - Ordinary Kalman filter (TRAMO forecast)
  - Array algorithm (square root)
  - Chandrasekhar recursions (TRAMO estimation)
  - Fast Array algorithm (square root)

- **Multivariate**
  - Univariate treatment
  - Multivariate Array algorithm (square root)
  - Fast Array algorithm (square root)

The right filter has to be chosen in function of the model (time-invariant or not), of the data (with or without missing values) and of the usage (smoothing, ...).

The framework also provides ordinary smoothing and disturbance smoothing algorithms.

That part of the library is used to produce the ARIMA forecasts and the one-step-ahead residuals of Tramo, and the Kalman smoother estimates of the unobserved components of Seats. More detailed information on the SSF models used in these cases can be found in the appendix 4.
Some simple uses of the SSF framework are presented below.

```java
UCModel ucm = ...;
double[] data = ...;

// Wiener-Kolmogorov estimates
WKEstimates wk = new WKEstimates();
wk.UCModel = ucm;
wk.Data = data;

double[] wktrend = wk.Estimates(0, true);
double[] wkseas = wk.Estimates(1, true);
double[] wkirr = wk.Estimates(2, true);

// Ssf solutions
SsfUCArima ssf = new SsfUCArima(ucm);
SsfData ssfdata = new SsfData(data, null);

// Disturbance smoother + ordinary filter (default)
DisturbanceSmoothes dsmoother = new DisturbanceSmoothes();
dsmoother.Ssf = ssf;

if (dsmoother.Process(ssfdata))
{
    SmoothingResults srlts = dsmoother.CalcSmoothedStates();
    double[] kftrend = srlts.Component(ssf.CmpPos(0));
    double[] kfseas = srlts.Component(ssf.CmpPos(1));
    double[] kfirr = srlts.Component(ssf.CmpPos(2));
}

// Disturbance smoother + fast Chandrasekhar filter (diffuse initialization)
FastFilter<ISsf> fastfilter = new FastFilter<ISsf>();
fastfilter.Ssf = ssf;

DiffuseFilteringResults drslts = new DiffuseFilteringResults(true);
drslts.FilteredData.IsSavingA = true;

fastfilter.Process(ssfdata, drslts);
if (dsmoother.Process(ssfdata, drslts))
{
    SmoothingResults srlts = dsmoother.CalcSmoothedStates();
    ...
}

// Ordinary smoother witht variance estimation
Smoothersmoother = new Smoothers();
SmoothingResults rslts = new SmoothingResults();
smoother.CalcVar = true; // =false=without variance
smoother.Ssf = ssf;
if (smoother.Process(ssfdata, rslts))
{
    double[] kftrend = rslts.Component(ssf.CmpPos(0));
    ...
}

// Idem for a Basic Structural Model
BsmSpecification spec = new BsmSpecification();
spec.SeasonalModel = SeasModel.Trigonometric;
BasicStructuralModel bsm = new BasicStructuralModel(spec, 12);
```
Finally, the library provides a lot of different ways to compute smoothed estimates of different time series models. The table below compared the performances of the different solutions proposed in the previous example.

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of estimations by second (monthly UCARIMA model on 20 years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wiener Kolmogorov filter (Burman-Wilson)</td>
<td>400</td>
</tr>
<tr>
<td>Diffuse initialization + Fast Chandrasekhar filter + Disturbance smoother</td>
<td>400</td>
</tr>
<tr>
<td>Diffuse initialization + Ordinary Kalman filter + Disturbance smoother</td>
<td>160</td>
</tr>
<tr>
<td>Diffuse initialization + Ordinary Kalman filter + Ordinary smoother (without variance estimation)</td>
<td>120</td>
</tr>
<tr>
<td>Diffuse initialization + Ordinary Kalman filter + Ordinary smoother (with variance estimation)</td>
<td>25</td>
</tr>
</tbody>
</table>

SUMMARY

The SSF framework developed in the library extends the solutions implemented in Tramo. They open the door to new and powerful solutions, while keeping a unified solution.

CONCLUSIONS

OO components based on standard technologies are a powerful and flexible software solution. They allow fast developments, facilitate the reusability and allow extensions. However, that approach is still quite uncommon in the statistical world. The library that we have built using Tramo-Seats as a guideline shows that complex statistical developments, like seasonal adjustments, can benefit much from such a solution.

That library - which will be freely available on a web site of the National Bank of Belgium - should not be considered as a finished solution, but more as a first attempt to build an integrated framework for time series analysis. We hope that it will arouse interest and collaborations in that way.

---

8 PC with 3.0Ghz-CPU.
Bibliography


[28] Ladiray D. and Quenneville B. (1999), "Comprendre la méthode X11".


## Contents of the modules (.NET)

<table>
<thead>
<tr>
<th>Module</th>
<th>Namespace</th>
<th>Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>NbbUtilities.dll</td>
<td>Nbb.Utilities</td>
<td>General exception</td>
</tr>
<tr>
<td></td>
<td>Nbb.Data</td>
<td>Definition/handling of data sets</td>
</tr>
<tr>
<td>NbbStats.dll</td>
<td>Nbb.Stats</td>
<td>Descriptive statistics</td>
</tr>
<tr>
<td></td>
<td>Nbb.Random</td>
<td>Random numbers</td>
</tr>
<tr>
<td></td>
<td>Nbb.DStats</td>
<td>Statistical distributions</td>
</tr>
<tr>
<td>NbbMath.dll</td>
<td>Nbb.Maths</td>
<td>Complex numbers</td>
</tr>
<tr>
<td></td>
<td>Nbb.Maths.Polynomial</td>
<td>Polynomials</td>
</tr>
<tr>
<td></td>
<td>Nbb.Maths.LinearFilters</td>
<td>Linear filters</td>
</tr>
<tr>
<td></td>
<td>Nbb.Maths.Functions</td>
<td>Numerical derivatives</td>
</tr>
<tr>
<td></td>
<td>Nbb.Maths.Matrices</td>
<td>Matrices</td>
</tr>
<tr>
<td>NbbTs.dll</td>
<td>Nbb.TimeSeries.SimpleTS</td>
<td>Simple time series</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Calendar effects</td>
</tr>
<tr>
<td>NbbEco.dll</td>
<td>Nbb.Regression</td>
<td>Regression model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OLS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Outliers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Likelihood</td>
</tr>
<tr>
<td></td>
<td>Nbb.TsRegression</td>
<td>Regression model on time series</td>
</tr>
<tr>
<td></td>
<td>Nbb.Arima</td>
<td>General ARIMA models</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GLS with ARIMA noise</td>
</tr>
<tr>
<td></td>
<td>Nbb.SArima</td>
<td>Fast KF (ARIMA models)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hannan-Rissanen</td>
</tr>
<tr>
<td></td>
<td>Nbb.UCArima</td>
<td>SARIMA (box-Jenkins) models</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UCarima models</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WK filters</td>
</tr>
<tr>
<td>NbbSsf.dll</td>
<td>Nbb.Ssf</td>
<td>Univariate Ssf models</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Filters, smoothers</td>
</tr>
<tr>
<td></td>
<td>Nbb.MSsf</td>
<td>Multivariate Ssf models</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Univariate treatment</td>
</tr>
<tr>
<td>NbbTramoSeats</td>
<td>Nbb.TramoSeats</td>
<td>TramoSeats</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified Hodrick-Prescott</td>
</tr>
<tr>
<td>NbbBsm.dll</td>
<td>Nbb.Bsm</td>
<td>Basic structural models</td>
</tr>
<tr>
<td></td>
<td>Nbb.Sutse</td>
<td>SUTSE models</td>
</tr>
<tr>
<td>NbbX11.dll</td>
<td>Nbb.X11</td>
<td>X11</td>
</tr>
</tbody>
</table>
Appendix 2.

An extensible definition of outliers.

The outliers detection of Tramo is based on the assumption that an outlier can be defined by a rational filter in the backshift operator.

Our library implements the following outliers types:

<table>
<thead>
<tr>
<th>Outlier type (*)</th>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additive outlier</td>
<td>$1$</td>
</tr>
<tr>
<td>Level shift outlier</td>
<td>$1/(1 - c \cdot B)$</td>
</tr>
<tr>
<td>Transitory change outlier</td>
<td>$\left(1 - \frac{s}{s-1} B + \frac{1}{s-1} B^s\right)/(1 - B - B^s + B^{s+1})$</td>
</tr>
<tr>
<td>Seasonal outlier*</td>
<td>$B - 1$</td>
</tr>
<tr>
<td>Displacement outlier*</td>
<td>$B - 1$</td>
</tr>
</tbody>
</table>

(*) new type.

The outliers detection procedure of Tramo relies on that definition to provide a fast processing. The IOutlier interface describes additional functions that are necessary to completely integrate a type of outlier in the complete Tramo processing. Besides the filter, the most important one is the ability to fill a buffer with the outlier effects, provided that the outlier occurs in a specified position.$^9$

Any new class that implements the IOutlier interface can be added to the outliers detection procedure and will be correctly handled by Tramo.

---

$^9$ That function can be derived from the filter. However, it has to be defined for performance considerations.
Appendix 3.

State space formulation of Durbin and Koopman.

Measurement equation:

\[ y_t = Z_t \alpha_t + \varepsilon_t \]
\[ \varepsilon_t \sim N(0, \sigma^2 I) \]

State equation:

\[ \alpha_{t+1} = T_t \alpha_t + \eta_t \]
\[ \eta_t \sim N(0, \sigma^2 V_t) \]

with \( 0 \leq t < n \)

\( y_t \) is the \( m \times 1 \) observations vector at period \( t \), \( \alpha_t \) is the \( r \times 1 \) state vector. \( \varepsilon_t \) and \( \eta_t \) are assumed to be serially independent and independent of each other at all time points.

The residuals of the state equation will often be modelled as

\[ \eta_t = RW_t \xi_t \]
\[ \xi_t \sim N(0, \sigma^2 Q_t) \]

where \( \xi_t \) is a vector of \( e \times 1 \) residuals (\( 0 < e \leq r \)), \( Q_t \) is a \( e \times e \) covariance matrix, \( W_t \) is a \( m \times e \) matrix (weights of the disturbances) and \( R_t \) is a \( r \times m \) matrix composed of columns of \( I_r \), that identifies the items of the state vector modified by the residuals.

(Diffuse) Initialisation

The initial conditions of the filter are defined as follows:

\[ \alpha_0 = a_0 + \Lambda \delta_0 + \eta_0 \]
\[ E(\delta_0) = 0 \]
\[ \text{var}(\delta_0) = \kappa I \]
\[ E(\eta_0) = 0 \]
\[ \text{var}(\eta_0) = \sigma^2 P_0 \]
\[ \text{cov}(\delta_0, \eta_0) = 0 \]

where \( K \) is arbitrary large\(^10\).

\( P_0 \) is the variance of the stationary elements of the initial state vector and \( \kappa \Lambda \Lambda' = \kappa P_\infty \) models the diffuse part. Both matrices are \( r \times r \).\(^{11} \) The rank of \( P_\infty \) is the number of independent constraints.

\(^{10}\) The \( \sigma^2 \) factor is absorbed in \( K \)

\(^{11}\) We do not require that diffuse/non diffuse elements reside in separate items of the initial state vector. Diffuse/non diffuse effects must simply be split in independent parts.
Appendix 4.

State space representation of ARIMA and UCARIMA models

**ARIMA model**

The ARIMA process is defined by

\[
\Delta(B)\Gamma(B)y(t) = \Theta(B)\epsilon(t),
\]

where

- \(\Delta(B) = 1 + \delta_1 B + \ldots + \delta_d B^d\)
- \(\Gamma(B) = 1 + \phi_1 B + \ldots + \phi_p B^p\)
- \(\Theta(B) = 1 + \theta_1 B + \ldots + \theta_q B^q\)

are the differencing, auto-regressive and moving average polynomials. We also write:

\[
\Phi(B) = \Delta(B)\cdot \Gamma(B) = 1 + \phi_1 B + \ldots + \phi_p B^{p+d}
\]

Let \(\psi_i\) be the psi-weights of the Arima model, \(\gamma_{st,j}\) and \(\gamma_{st,j}\), the psi-weights and the autocovariances of the differenced Arma model. We also define:

\[
r = \max(p + d, q + 1)
\]

\[
s = r - 1
\]

Using those notations, the state-space model can be written as follows\(^\text{12}\):

**State vector:**

\[
a(t) = \begin{pmatrix}
y(t) \\
y(t+1|t) \\
\vdots \\
y(t+s|t)
\end{pmatrix}
\]

where \(y(t+i|t)\) is the orthogonal projection of \(y(t+i)\) on the subspace generated by \(\{y(s) : s \leq t\}\). Thus, it is the forecast function with respect to the semi-infinite sample.

**System matrices:**

Using the notations of appendix 3, the matrices of the model are

\[
Z(t) = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}
\]

\[
h(t) = 0
\]

\(^\text{12}\) See for example Gomez-Maravall [9]
\[
T(t) = \begin{pmatrix}
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1 \\
-\phi_r & \cdots & \cdots & -\phi_1
\end{pmatrix}
\]

\[
W(t) = \begin{pmatrix}
\psi_0 \\
\psi_1 \\
\vdots \\
\psi_s
\end{pmatrix}
\]

\[
R(t) = I_r
\]

\[
Q(t) = \sigma^2
\]

and the initial conditions can be written:

\[
a_0 = (0 \quad \cdots \quad 0)
\]

\[
P_{x,0} = \Sigma V \Sigma'
\]

\[
P_{v,0} = \Lambda \Lambda'
\]

\(V\) is the variance/covariance of the stationary model; it can be derived by the relationships:

\[
V[i,0] = V[0,i] = \gamma_{st,i}
\]

\[
V[i,j] = V[i-1, j-1] - \psi_{st,i} \psi_{st,j}
\]

\(\Sigma\) is a \(r \times r\) lower triangular matrix with ones on the main diagonal; other cells are defined by the recursive relationship:

\[
\Sigma[i,j] = -\delta_1 \Sigma[i-1, j] - \cdots - \delta_d \Sigma[i-d, j]
\]

with the convention \(\Sigma[i,j] = 0\) if \(i < 0\)

\(\Lambda\) is a \(r \times d\) matrix; its first \(d\) rows form an identity matrix; other cells are defined as above:

\[
\Lambda[i,j] = -\delta_1 \Lambda[i-1, j] - \cdots - \delta_d \Lambda[i-d, j]
\]
**UCARIMA model**

A state space representation of an UCARIMA model can be directly derived from the state space representation of each of its components, as defined above. More specifically, if the model contains \( n \) components, its SSF representation will be defined by:

\[
\alpha_t = \begin{bmatrix}
\alpha_{1,t} \\
\vdots \\
\alpha_{n,t}
\end{bmatrix}
\]

\[
T = \begin{bmatrix}
T_1 & 0 & 0 \\
0 & \ddots & 0 \\
0 & 0 & T_n
\end{bmatrix}
\]

\[
Z = [Z_1 \ \ldots \ \ Z_n]
\]

\[
W = \begin{bmatrix}
W_1 & 0 & 0 \\
0 & \ddots & 0 \\
0 & 0 & W_n
\end{bmatrix}
\]

\[
Q = \begin{bmatrix}
\sigma^2_1 & 0 & 0 \\
0 & \ddots & 0 \\
0 & 0 & \sigma^2_n
\end{bmatrix}
\]

\[
P_{s,0} = \begin{bmatrix}
P_{s,0,1} & 0 & 0 \\
0 & \ddots & 0 \\
0 & 0 & P_{s,0,p}
\end{bmatrix}
\]

with the different sub-matrices as above.

**UCARIMA model with mean effect.**

When the model contains a mean effect, Seats estimates by WK the following model:

\[
\Delta(B) \Gamma(B) y(t) = \mu + \Theta(B) \varepsilon(t).
\]

The mean effect is affected to the trend.

An equivalent SSF solution can be obtained as follows:
if we note the unit roots associated with the trend by $\Delta_{\text{trend}}(B)$ and $\Delta_{\text{others}}(B) = \Delta(B)/\Delta_{\text{trend}}(B)$, the mean effect on the trend component, written $m_t$, is defined by the relationship.

$$\Delta_{\text{trend}}(B) = \tilde{\mu} \quad (1)$$

with $\tilde{\mu} = \mu / \Delta_{\text{others}}(1) \Gamma(1)$

and with the starting values $m_0 = \begin{bmatrix} \mu_0 \\ \vdots \\ \mu_{s-1} \end{bmatrix}$, where $s$ is the degree of $\Delta_{\text{trend}}(B)$.

It is easy to see that the total mean effect as the form $m_t = \overline{\mu}_t + X_t m_0$, where $\overline{\mu}_t$ is the mean effect generated by (1) with zero initial values, and where $X_t m_0$ is the effect of the initial values in $t$. $m_0$ can be obtained by solving the OLS regression $y_t - \overline{\mu}_t = X_t m_0$.

Then, we consider the model $\Delta(B) \Gamma(B) (y_t - m_t) = \Theta(B) \epsilon_t$, that we can decompose in trend, seasonal and irregular components as usual. The final trend is obtained by adding $m_t$ to the trend of the model without the mean.