SSA Toolbox 1.3 — Manual

Jan Saputra Müller, Paul von Bünau,
Frank C. Meinecke, Franz J. Király, Klaus-Robert Müller

October 25, 2011

Contents

1 Introduction and Overview

This is the manual for the SSA Toolbox, an efficient open-source implementation of the Stationary Subspace Analysis [1] algorithm. Stationary Subspace Analysis (SSA) is a general purpose algorithm for the explorative analysis of non-stationary data, i.e. data whose statistical properties change over time. SSA can help to detect, characterize and visualize temporal changes in complex high-dimensional data sets.

The SSA Toolbox is written entirely in Java and is thus platform-independent. It has been tested successfuly under Windows, Linux and MacOS. The SSA Toolbox comes with a state-of-the-art native Linear Algebra library (BLAS,LAPACK). For maximum platform independence, the user can also choose the purely Java-based library COLT. Data and results can be imported and exported as comma-separated values `.csv` files, the fail-safe format of last resort, and through Matlab's proprietary `.mat` files, a de-facto standard in the Machine Learning community.

The source code of the SSA Toolbox is fully documented (using the JavaDoc standard) and accompanied by a set of unit tests written in JUnit. The latest version is always available from github[1], a hosting service for the git versioning system. Section 7 contains further information for developers, including a high-level overview of the class structure.

There are four ways to use the SSA Toolbox

1. As a standalone application with a graphical user interface.
2. As a standalone application from the command line.
3. From within Matlab, using the wrapper script `ssa.m`.
4. As a Java library, from your own application.

2 Installation and Running

Obtaining the latest SSA Toolbox The latest version of the SSA Toolbox is available from the offical SSA homepage:

`http://www.stationary-subspace-analysis.org/toolbox`

There you can also find pointers to further references, example data and a link to the SSA mailing list.

Platforms The SSA Toolbox is written in the platform-independent Java programming language; platform-specific numerical libraries[2] are included for several target architectures. The SSA Toolbox requires the Java Runtime Environment[3] version 1.5 or later. Most operating systems have a Java Runtime

[1] See `http://https://github.com/paulbuenau/SSA-Toolbox`
[2] BLAS+LAPACK provided through jblas (see `http://www.jblas.org`).
[3] See `http://www.java.com/getjava`

Environment pre-installed, you might be able to find out the version by typing `java -version` on the command line. The SSA Toolbox has been tested on the following platforms.

- Microsoft Windows (32 and 64 bit)
- Linux (32 and 64 bit)
- Mac OS X (32 and 64 bit)

Installation and Running The SSA Toolbox comes as a single `.zip` or `.tar.gz` archive. After unpacking, you can start the SSA Toolbox by opening the file `ssa.jar` with the default method of your operating system, e.g. by double-clicking on it under Microsoft Windows, OS X and some Linux distributions.

You can also manually invoke the SSA Toolbox by typing

```
java -jar ssa.jar
```

on the command line of your operating system.

In some cases, if you want to run SSA on very large data sets, it might be necessary to start the toolbox with a higher amount of Java heap space (in those cases the SSA toolbox will inform you about this issue). There is a section in the Frequently Asked Questions which explains how to do that (see Section 6).

If you want to use the SSA Toolbox directly from Matlab, you can use the wrapper script `ssa.m`. Type `help ssa` on the Matlab command line to find out about the format of its input and output parameters. Note that if you invoke the SSA Toolbox from within Matlab, it will use its internal JVM unless you specify an external JVM, e.g. using the environment variable `MATLAB_JAVA` under Linux.

You can also use the toolbox from the command line. In this case, you have to pass options after `java -jar ssa.jar`. The following table shows the available options:

Option	Meaning/Argument
-i	Input file (in `.csv` or `.mat` format). Required.
-o	Output file or directory. If it ends with `.mat` a Matlab file will be created, otherwise `.csv` files are created in the specified directory. Required.
-d	Number of stationary sources. Required.
-r	Number of restarts. Optional. Default: 5
-n	Number of equally-sized epochs. Optional. If this option is not specified, and no custom epochization has been given, a heuristic is used to determine the number of epochs.
-e	Epochization file in `.csv` format. Optional.
-m	Use the means during optimization. Has to be 0 or 1. Optional. Default: 1
-c	Use the covariances during optimization. Has to be 0 or 1. Optional. Default: 1
-s	Random seed. Optional.
-j	Use jBlas instead of Colt. Has to be 0 or 1. Optional. Default: 0

For example, the following line would run SSA on the example data set:

```
java -jar ssa.jar -i example_data/example_data.mat -d 2 -o results.mat
```

3 Stationary Subspace Analysis in Brief

Stationary Subspace Analysis [1] factorizes a multivariate time-series into its stationary and non-stationary components. That is, we assume that the data generating system consists of d stationary source signals $\mathbf{s}^{\mathfrak{s}}(t) = [s_1(t), \ldots, s_d(t)]^\top$ and $D - d$ non-stationary source signals $\mathbf{s}^{\mathfrak{n}}(t) = [s_{d+1}(t), \ldots, s_D(t)]^\top$ and that the observed signals $x(t)$ are a linear superposition of these sources,

$$\mathbf{x}(t) = A\mathbf{s}(t) = \begin{bmatrix} A^{\mathfrak{s}} & A^{\mathfrak{n}} \end{bmatrix} \begin{bmatrix} \mathbf{s}^{\mathfrak{s}}(t) \\ \mathbf{s}^{\mathfrak{n}}(t) \end{bmatrix} \tag{1}$$

where A is an invertible matrix. Note that we do *not* assume that the sources $\mathbf{s}(t)$ are independent. We refer to the spaces spannend by the columns of $A^{\mathfrak{s}}$ and $A^{\mathfrak{n}}$ as the stationary ($\mathfrak{s}$-) and non-stationary ($\mathfrak{n}$-) space respectively.

The SSA algorithm factorizes the observed signals $x(t)$ according to Equation 1, i.e. it finds a linear transformation

$$\hat{A}^{-1} = \begin{bmatrix} \hat{P}^{\mathfrak{s}} \\ \hat{P}^{\mathfrak{n}} \end{bmatrix} \tag{2}$$

that separates the $\mathfrak{s}$-sources from the $\mathfrak{n}$-sources. The inverse of the estimated demixing matrix $\hat{A}^{-1}$ is the estimated mixing matrix,

$$\hat{A} = \begin{bmatrix} \hat{A}^{\mathfrak{s}} & \hat{A}^{\mathfrak{n}} \end{bmatrix}, \tag{3}$$

and the estimated stationary and non-stationary sources are thus given by

$$\hat{\mathbf{s}}^{\mathfrak{s}}(t) = \hat{P}^{\mathfrak{s}}\mathbf{x}(t) \tag{4}$$

$$\hat{\mathbf{s}}^{\mathfrak{n}}(t) = \hat{P}^{\mathfrak{n}}\mathbf{x}(t) \tag{5}$$

respectively. Note that only the $\mathfrak{s}$-projection and the $\mathfrak{n}$-space are uniquely identifiable. The projection to the $\mathfrak{n}$-sources $\hat{P}^{\mathfrak{n}}$ is found by maximizing the non-stationarity of the estimated $\mathfrak{n}$-sources.

The SSA Toolbox allows for input and output in (time $\times$ channel) and (channel $\times$ time) format. Note that the above definitions of sources, projections and basis correspond to the (channel $\times$ time) format.

4 Input, Output and Parameters

The input to the SSA Toolbox consists of

- Data: the time series $x(t)$, either as (channels $\times$ time) or (channels $\times$ time).
- Segmentation of the time series $x(t)$ into epochs, either
  - equally-sized, where the number of epochs is supplied by the user; or
  - equally-sized, where the number of epochs is set automatically by a heuristic;
  - according to a user-supplied custom epoch definition.
- Parameters to the SSA Algorithm (see Section 4.3).

The parameters are set via the graphical user interface. The time series $x(t)$ and a custom epoch definition can be loaded from comma-separated values (CSV) and Matlab (.mat) files.

4.1 Comma-Separated-Values File Format

Comma Separated Values (CSV) files are human-readable text files for storing tabular data. The columns are separated by commas and each line of the file corresponds to a row. Lines starting with a hash (`#`) are ignored. If the data has more rows than columns, then each row will be interpreted as a time point and each column as a channel. Otherwise, the format is assumed to be (channels $\times$ time). See Figure 1 for an example time series file.

A custom segmentation of the time series into epochs can be specified by means of a separate CSV file, which must have the same number of rows as the time series and one column. The entries correspond to the index (starting with 1) of the epoch that a time point belongs to. Figure 2 shows an example CSV file for a segmentation of the time series into custom epochs.

```
# 2ch recording VPzj Oct 30th
-0.18671,0.11393
0.72579,1.0668
-0.58832,0.059281
2.1832,-0.095648
-0.1364,-0.83235
...
```

Figure 1: The first five time points of a timeseries in CSV format with two channels. The first line is a comment for documentation purposes.

```
# Epochs for recording VPzj Oct 30th
1
1
1
2
2
...
```

Figure 2: Custom epoch definition for the first five time points of the time series shown in Figure 1. The first three time points belong to the same epoch and the last two time points form the second epoch.

4.2 Matlab File Format

In the Matlab file format, the time series must be contained in a variable called `X`. If `X` has more rows than columns, then each row will be interpreted as a time point and each column as a channel. Otherwise, the format is assumed to be (channels $\times$ time).

```
>> X

X =

   -0.4326   -0.1867
   -1.6656    0.7258
    0.1253   -0.5883
    0.2877    2.1832
   -1.1465   -0.1364
     ...
```

Figure 3: Timeseries in Matlab with two channels and five time points.

If `X` is a cell array, then the elements are interpreted as epochs where each epoch must have the same number of channels, Figure 4 shows an example.

```
>> X

X =

    [100x2 double]    [100x2 double]    [80x2 double]
```

Figure 4: Timeseries in Matlab with custom epoch definition. The time series is split into three epochs where the first two epochs contain 100 samples each and the third epoch consists of 80 samples.

4.3 Parameters of SSA

The SSA algorithm has the following parameters which can be set via the graphical user interface.

Number of stationary sources The number of stationary sources d to be found in the time series. Depending on d, the algorithm needs a certain minimum number of distinct epochs, see next paragraph. A warning message will be issued if this condition is violated.

Number of restarts The number of times the optimization procedure should be repeated with different random initialization in order to avoid local minima. The final result is the decomposition which attained the smallest objective function value.

Number of equally-sized epochs The number of equally sized epochs N that the time series should be split into, if the user did not supply a custom segmentation. The minimum number of epochs required to avoid spurious stationary directions depends on the number of stationary sources d, see next paragraph.

Which moments of the sources should be considered Depending on the application domain, changes in mean or covariance matrix either do not occur or are not relevant. The user can therefore select whether non-stationarities in the mean, covariance matrix or both should be considered. The default is that both moments are taken into account.

Determinacy of the solution

If the number of epochs N is too small in relation to the number of non-stationary sources $D - d$, there may exist spurious stationary components that render the solution non-identifiable, i.e. components which appear stationary on the limited amount of observed data but which are in fact non-stationary. Informally speaking, spurious stationary components occur when the amount of observed variation in the distributions (i.e. the number of distinct epochs)

is insufficient to eliminate seemingly stationary directions in the non-stationary subspace. For example, if you think of two covariance matrices $\Sigma_1, \Sigma_2 \in \mathbb{R}^{2\times 2}$ in two-dimension, then one can always find a one-dimensional subspace on which the projected variances agree, unless $\Sigma_1 - \Sigma_2$ has an indefinite spectrum (i.e. the contour plot of one covariance matrix is strictly contained in the other).

It can be shown [1] that there exist no spurious stationary directions if the number of epochs N is at least

$$N > \frac{D-d}{2} + 2.$$

When only changes in one moment are considered, we need more epochs, namely

$$N > D - d + 1.$$

The SSA Toolbox issues a warning if these conditions are violated. In practice, if there is enough data, it is advisable to use more epochs than the minimum indicated by this bound.

4.4 Output

The result of the SSA algorithm is an estimated demixing matrix $\hat{A}^{-1}$ (see Equation 2). From this, the following output is generated by the SSA Toolbox (assuming the output format channels × time):

- the estimated projection to the stationary sources $\hat{P}^{\mathfrak{s}} \in \mathbb{R}^{d\times D}$
- the estimated projection to the non-stationary sources $\hat{P}^{\mathfrak{n}} \in \mathbb{R}^{(D-d)\times D}$
- the estimated basis of the stationary subspace $\hat{A}^{\mathfrak{s}} \in \mathbb{R}^{D\times d}$
- the estimated basis of the non-stationary subspace $\hat{A}^{\mathfrak{n}} \in \mathbb{R}^{D\times(D-d)}$
- the estimated stationary sources $\hat{s}^{\mathfrak{s}}(t) = \hat{P}^{\mathfrak{s}} x(t)$,
- the estimated non-stationary sources $\hat{s}^{\mathfrak{n}}(t) = \hat{P}^{\mathfrak{n}} x(t)$

These six matrices resp. time series can either be saved in individual CSV files (see Section 4.1 for a general description of the format) or to a single Matlab file (.mat).

4.5 Matlab File Format

The output of the SSA Toolbox can be written to a single Matlab file which contains a structure `ssa_results` where the attributes correspond to the six outputs. See Figure 5 for an example.

```
>> ssa_results =

          Ps: [3x6 double]
          Pn: [3x6 double]
          As: [6x3 double]
          An: [6x3 double]
       s_src: [3x10000 double]
       n_src: [3x10000 double]
  parameters: [1x1 struct]
      loss_s: -1.2789
      loss_n: 73.2571
iterations_s: 13
iterations_n: 10
 description: 'SSA results (Thu Jun 10 12:58:23 CEST 2010)'
```

Figure 5: Matlab result structure of the SSA Toolbox. Five dimensional data (500 samples) are decomposed into $d = 3$ stationary sources and $D - d = 2$ non-stationary sources.

5 Examples and Toy Data

The SSA Toolbox comes with an example data set in Matlab and CSV format (`example_data.csv` and `example_data.mat`), a Matlab script to generate synthetic data (`ssa_toydata.m`), and a self-contained Matlab demonstration (`ssa_demo.m`). In the following sections, we describe each of these files.

5.1 Example Data

The example data sets in the directory `example_data` were generated using the Matlab script `ssa_toydata.m` using the command-line

```
[X, A] = ssa_toydata(10, 2, 2, 'mean_nonstat', 0.5);
```

The dataset consists of ten epochs (each 500 samples); it has two stationary and two non-stationary sources. The Matlab file `example_data.mat` also contains the true mixing matrix `A` so that you can assess the quality of the found solution using the subspace error between the true and the estimated basis of the non-stationary spaces as follows:

```
subspace_error(ssa_results.An, A(:,3:4))
```

A description of this error measure can be found in Appendix B. The comma-separated values file `example_data.csv` contains the same data as the Matlab file. Further details about the toy data generation can be found in the next section.

5.2 Toy Data Generation

The Matlab script `ssa_toydata.m` generates synthetic datasets according to the SSA mixing model. A documentation of the parameters of this function is available from the Matlab help, in this section we describe how the data is generated.

Let $X_1, \ldots, X_n$ be random variables modeling the distribution of the data in each of the n epochs. According to the SSA mixing model (Equation 1), each epoch is a linear mixture of d stationary and $D - d$ non-stationary sources,

$$X_i = A \begin{bmatrix} X^{\mathfrak{s}} \\ X_i^{\mathfrak{n}} \end{bmatrix}$$

where A is the mixing matrix and $X^{\mathfrak{s}}$ and $X_i^{\mathfrak{n}}$ are random variables representing the $\mathfrak{s}$- and the $\mathfrak{n}$-sources in the i-th epoch respectively. The entries of A are chosen uniformly at random from $[-0.5, 0.5]$ and its columns are normalized to one. The distribution of the $\mathfrak{s}$-sources is the same over all epochs,

$$X^{\mathfrak{s}} \sim \mathcal{N}(0, I),$$

and the $\mathfrak{n}$-sources are correlated with the $\mathfrak{s}$-sources,

$$X_i^{\mathfrak{n}} = C_i X^{\mathfrak{s}} + Y_i^{\mathfrak{n}},$$

where C_i is $(D - d \times d)$-matrix and $Y_i^{\mathfrak{n}} \sim \mathcal{N}(\mu_i, \Sigma_i)$ is a random variable corresponding to the conditional $X_i^{\mathfrak{n}} | X^s$. The covariance matrix $\Sigma_i^{\mathfrak{n}}$ is generated as,

$$\Sigma_i^{\mathfrak{n}} = B_i^{\mathfrak{n}} \Lambda_i (B_i^{\mathfrak{n}})^\top,$$

where $B_i^{\mathfrak{n}}$ is a random orthogonal matrix and Λ_i is a diagonal matrix of the eigenvalues $\lambda_1, \ldots, \lambda_{D-d}$. The log eigenvalues are drawn from a uniform distribution, and the sign is chosen to be positive with probability p and negative otherwise; then they normalized such that

$$\frac{1}{D-d} \sum_{i=1}^{D-d} |\log \lambda_i| = \log \lambda_{\text{mean}},$$

where $\log \lambda_{\text{mean}}$ is a measure of the distance from the standard normal distribution.

For each epoch, the canonical correlation ρ_i is drawn uniformly at random from $[\rho_{\min}, \rho_{\max}]$, which determines the scaling of the rows of C_i,

$$C_i = BC'(B_i^{\mathfrak{n}})^\top,$$

where B is a random orthogonal matrix and C' is a $(D - d \times d)$-matrix with non-zero entries only on the diagonal,

$$(C')_{jj} = \sqrt{\frac{\lambda_{jj}}{\frac{1}{\rho_i^2} - 1}}.$$

This ensures that the canonical correlation is ρ_i for all principal vectors. Thus the covariance matrix of the sources in the i-th epoch is given by,

$$\Sigma_i = \mathrm{cov}\left(\begin{bmatrix} X^{\mathfrak{s}} \\ X_i^{\mathfrak{n}} \end{bmatrix}\right) = \begin{bmatrix} I & C_i^\top \\ C_i & C_i C_i^\top + \Sigma_i^{\mathfrak{n}} \end{bmatrix}.$$

The strength of the non-stationarity in the mean of the $\mathfrak{n}$-sources is chosen relative to the total non-stationarity $\tilde{\sigma}$ in the covariance matrix which we measure as the the sum of the absolute log eigenvalues of the whitened covariance matrices as follows. Let $\Sigma_i' = A\Sigma_i A^\top$ be the covariance matrix of the mixed sources. The whitening matrix W of the average covariance matrix is given by,

$$W = \left[\frac{1}{n}\sum_{i=1}^{n} \Sigma_i'\right]^{-\frac{1}{2}}.$$

Let $W\Sigma_i' W^\top = U\Lambda' U^\top$ be the eigenvalue decomposition of the whitened covariance matrix of the i-th epoch, where Λ' is the diagonal matrix of eigenvalues. The total non-stationarity $\tilde{\sigma}$ in the covariance matrix is measured as,

$$\tilde{\sigma} = \sum_{i=1}^{n}\sum_{j=1}^{D} \left|\log(\Lambda_i')_{jj}\right|.$$

The total non-stationarity in the mean $\tilde{\mu}$ is measured as the sum of the squared norms of the epoch mean vectors $\mu_1', \ldots, \mu_n'$ in the whitened basis,

$$\tilde{\mu} = \sum_{i=1}^{n} \|W\mu_i'\|^2.$$

Thus we first choose a random set of squared norms $m_1, \ldots, m_n$ such that $\sum_{i=1}^n m_i = \tilde{\sigma} = \tilde{\mu}$, and then a set of mean vectors $\mu_1, \ldots, \mu_n$ of the sources where the first d entries (corresponding to the $\mathfrak{s}$-sources) are zero and the last $D - d$ entries are drawn uniformly from $[-0.5, 0.5]$. These source mean vectors are then rescaled such that that $\|WA\mu_i\|^2 = m_i$ for each epoch.

5.3 Matlab Demonstration

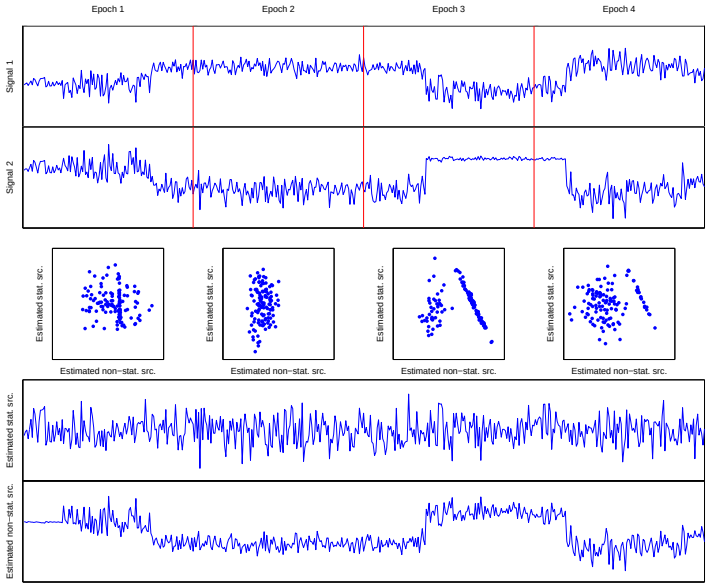

Figure 6: Output of the Matlab example `ssa_demo.m`

The plot generated by the example script is shown in Figure 6. The top panel shows the two dimensional input time series $x(t)$. Both dimensions (or variables, channels) appear non-stationary. The vertical red lines indicate how the time series is segmented into four epochs. In the middle row, we see the result of applying SSA: for every epoch, the scatter plot shows the distribution of the estimated stationary source (vertical axis) vs. the estimated non-stationary source (horizontal axis). We can see that the mean and the variance projected to the vertical axis is constant over the epochs, whereas both change significantly for the estimated non-stationary source. The bottom panel shows the time courses for the estimated stationary (top) and the estimated non-stationary (bottom) source.

6 Frequently Asked Questions

When should I use SSA? SSA can be applied whenever a multivariate dataset, that has a temporal structure is conceivable generated as a mixture of stationary and non-stationary components, where the non-stationary effects are visible in changes of the mean and covariance matrix.

Can I use SSA on a univariate time series? No.

How do I determine the number of stationary sources? To date, there exists no *automatic* selection procedure. In practice, one can apply SSA for several numbers of stationary sources and inspect the result, e.g. visually or using appropriate statistical hypothesis tests. Moreover, the value of the objective function can be used as a guideline, see Appendix A.

Why is it not possible to identify the non-stationary sources and what does this mean? The true non-stationary sources cannot be identified because adding stationary components to a non-stationary time series leaves it non-stationary. This makes the estimated non-stationary sources arbitrary. It is therefore not possible to recover the true non-stationary sources from the mixture.

Why is it not possible to identify the stationary subspace and what does this mean? The basis of the stationary subspace cannot be identified because we cannot estimate the non-stationary sources, which are dual to the basis of the stationary subspace. If we could identify the stationary subspace, then we could find the non-stationary sources by projecting orthogonal to it.

I would like to use the Toolbox from Python/Octave/or any other environment, is this possible? The SSA Toolbox can be used directly from any environment that can invoke Java code, such as e.g. Matlab. Based on the command-line interface, future releases will include wrappers for R and Python.

Can I get a p-value for my result? The objective function value is an approximate p-value for a stationarity test, see Appendix A.

The toolbox says that there is not enough Java heap space. How can I solve this issue? You have to increase the Java heap space. If you are using the standalone version of the toolbox, you can do this by running it from the command line like this:

```
java -Xmx512M -jar ssa.jar
```

This would set the Java heap space to `512M`. Of course, you can replace `512M` with your desired size.

If you are using the toolbox from within Matlab, you have to increase the Java heap space which Matlab uses. The following website explains how to do that:

`http://www.mathworks.com/support/solutions/en/data/1-18I2C`

In Matlab 2010a and later, this can be easily done from Matlab's preferences dialog.

7 Developers

This section contains some information about the technical structure of the SSA Toolbox that is helpful for developers who want to extend it or use parts in other projects. The source code distribution of the SSA Toolbox is available from the project web site (see Section 2) and from github, a public git repository server, which is the primary location of the source code:

`https://github.com/paulbuenau/SSA-Toolbox`

On github you will always find the latest, probably unreleased version of the SSA Toolbox.

Example

The code of the SSA Toolbox can be used as a library from other applications. Below you can can find a simple example which shows how to load data, parametrize the SSA algorithm, apply it and save the results to a file.

```
public class SSAJavaExample
{
    public static void main(String args[])
    {
        // create a console logger for writing log messages
        ssatoolbox.ConsoleLogger cl = new ssatoolbox.ConsoleLogger();

        // create instance of SSA main class.
        ssatoolbox.Main ssaMain = new ssatoolbox.Main(false, cl);

        // load data from CSV-file
        ssaMain.loadTimeseries(new java.io.File("data.csv"));

        // set SSA parameters
        ssaMain.parameters.setNumberOfStationarySources(3);

        // set epochization type
        ssaMain.data.setEpochType(
         ssatoolbox.Data.EPOCHS_EQUALLY_HEURISTIC);

        // Run SSA (*not* using a separate thread!)
        ssaMain.runSSA(false);

        // save stationary sources
        ssaMain.saveStationarySourcesCSV(
         new java.io.File("stationary.csv"));

        // save non-stationary sources
        ssaMain.saveNonstationarySourcesCSV(
         new java.io.File("non-stationary.csv"));
    }
}
```

Class Structure

The SSA Toolbox consists of the classes `SSA`, `SSAParameters`, `Data`, `Results`, `MathFunctions`, `GUI`, `Main`, `SSAMatrix`.

SSA This class implements the SSA algorithm. The main-part of the implementation is in the method `optimizeOnce`, which does the conjugate gradient descent. The calculation of the objective function and its gradient is done in the method `objectiveFunction`. Apart from that, the class `SSA` is responsible for checking our determinacy bounds and has to throw an exception if they are violated.

SSAParameters The class `SSAParameters` stores the parameters, such as the assumed number of stationary sources, the number of restarts for the optimization and whether to use means/covariance matrices. It is responsible to

check whether the entered parameters are valid and has to throw an exception otherwise.

Data The class `Data` stores our time series and does the epochization (equally sized epochs or custom epochization). It has to check whether the epochization is valid (and possible) and has to throw an exception otherwise.

Results This class stores the results of the SSA algorithm.

MathFunctions This class contains auxilliary functions.

GUI This class is responsible for the GUI of the standalone-version of the toolbox. It implements the interface `Logger`.

Main This is the main class of the standalone-version of the toolbox.

SSAMatrix `SSAMatrix` is a wrapper class for different matrix libraries. All matrices in the toolbox are instances of this class. It passes the matrix operations to the currently used matrix library.

Logger This is an interface for logging. It contains only one method, which has to be implemented by all implementing classes: `appendToLog`.

ConsoleLogger This class implements the interface `Logger`. It writes all messages which are passed to the method `appendToLog` directly to the console (needed for the matlab-wrapper).

External Dependencies

All necessary libraries are included in the distribution archive. These are the following.

- `JMatIO` for reading and writing Matlab files;
- `COLT` and `jblas` for matrix computations; and
- `java-getopt` for evaluation command-line arguments.

Unit Tests

The unit tests are realized using `JUnit` and can be found in the subdirectory `tests` of the repository. There are tests for the classes `Data` and `SSA`.

Tests for the Data class

`testSetNumberOfEqualSizeEpochs()` This test checks whether an exception is thrown if we have less data points than epochs (in case of equally sized epochs).

`testSetCustomEpochDefinition()` This test checks whether an exception is thrown if we have less data points than epochs (in case of a custom epoch definition).

Tests for the SSA class

`testOptimizeMeanCov()` This test checks whether an exception is thrown if the determinacy bound is violated (when using both means and covariance matrices).

`testOptimizeMean()` This test checks whether an exception is thrown if the determinacy bound is violated (when using means only).

`testOptimizeCovariance()` This test checks whether an exception is thrown if the determinacy bound is violated (when using covariance matrices only).

`testOptimize()` This function contains a simple test which checks if the SSA optimization works correctly by running the optimization on a very simple data set. This data set is 2-dimensional, where the first dimension is stationary and the second one is non-stationary (this means the mixing matrix to be estimated is the identity matrix). For checking the solution of the optimization, the angle between the estimated stationary projection direction and the correct one $\begin{pmatrix}1\\0\end{pmatrix}$ is considered. The same is done with the estimated basis vector of the non-stationary subspace, where the correct direction is $\begin{pmatrix}0\\1\end{pmatrix}$.

8 Contact and Support

If you have any question, bug report or want to discuss the application of SSA to a specific problem, please join the SSA mailing list:

`http://groups.google.com/group/ssa-list`

You can find further information on the official SSA homepage:

`http://www.stationary-subspace-analysis.org`

Appendices

A Interpreting the objective function value

In practice, in order to assess a demixing found by the SSA algorithm, we need an interpretable measure of stationarity resp. non-stationarity of the estimated stationary sources $\hat{s}^{\mathfrak{s}}(t)$ and the estimated non-stationary sources $\hat{s}^{\mathfrak{n}}(t)$. To that end, we can interpret the SSA objective function from an hypothesis testing point of view. The SSA objective function is equivalent to the likelihood-ratio test statistic for comparing two models for the data: all epochs follow a multivariate standard normal distribution (null hypothesis H_0) vs. the more complex model where every epoch has a different mean and covariance matrix (alternative hypothesis H_A). This also suggests a procedure for determining the number of the stationary sources: we increase the number of stationary sources as long as the stationarity test on the estimated $\mathfrak{s}$-sources is not rejected.

More formally, let $X_1, \ldots, X_N$ be d-variate random variables modeling the distribution of the data in the N epochs. Formally, the hypothesis can be written as follows.

$$H_0 : X_1, \ldots, X_N \sim \mathcal{N}(0, I)$$
$$H_A : X_1 \sim \mathcal{N}(\mu_1, \Sigma_1), \ldots, X_N \sim \mathcal{N}(\mu_N, \Sigma_N)$$

In other words, the statistical test tells us whether we should reject the simple model H_0 in favor of the more complex model H_A. This decision is based on the value of the test statistics, whose distribution is known under the null hypothesis H_0. Since H_0 is a special case of H_A and since the parameter estimates are obtained by Maximum Likelihood, we can use the likelihood ratio test statistic Λ, which is the ratio of the likelihood of the data under H_0 and H_A, where the parameters are their maximum likelihood estimates. The likelihood ratio test statistic $\Lambda(\mathcal{X})$ is,

$$\Lambda(\mathcal{X}) = -2 \log \frac{L_{H_0}(\mathcal{X})}{L_A(\mathcal{X})}, \tag{6}$$

where $L_{H_0}(\mathcal{X})$ is the likelihood of the data set under the simpler null model and $L_A(\mathcal{X})$ is the likelihood of the data set under the more complex alternative model. In order to find the estimated stationary sources $\hat{s}^{\mathfrak{s}}(t)$, the SSA algorithm minimizes the test statistic $\Lambda(\mathcal{X})$, i.e. it maximizes the p-value to not reject the null hypothesis of stationarity. Conversely, to estimate the most non-stationary sources $\hat{s}^{\mathfrak{n}}(t)$, we maximize $\Lambda(\mathcal{X})$ and thereby minimize the p-value.

Under the null hypothesis H_0, the test statistic $\Lambda(\mathcal{X})$ is approximately χ^2 distributed with $k = \frac{1}{2}Nd(d+3)$ degrees of freedom. It can be shown that $\sqrt{2\Lambda(\mathcal{X})}$ converges (faster) to a Gaussian distribution with mean $\mu_0 = \sqrt{2k-1}$ and unit standard deviation $\sigma_0 = 1$. We therefore report the normalized objective function value,

$$\Lambda'(\mathcal{X}) = \sqrt{2\Lambda(\mathcal{X})} - \sqrt{2k-1},$$

both for the estimated stationary and the estimated non-stationary sources. We can thus derive approximate p-values for the stationarity test from the value of $\Lambda'(\mathcal{X})$ and the standard normal distribution.

A.1 Derivation of the test statistic

In this section we note the mathematical details for deriving the value of the test statistic (Equation 6). Let $\mathcal{X} \subset \mathbb{R}^d$ be the data set which is divided into N epochs $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_n = \mathcal{X}$ with corresponding sample sizes $N_i = |\mathcal{T}_i|$ for $1 \leq i \leq N$ and let $\hat{\mu}_1, \ldots, \hat{\mu}_N \in \mathbb{R}^d$ and $\hat{\Sigma}_1, \ldots, \hat{\Sigma}_N \in \mathbb{R}^{d \times d}$ be estimates of the epoch mean and epoch covariance matrices respectively. Let $p_{\mathcal{N}}(x; \mu, \Sigma)$ be the probability density function of the multivariate Gaussian distribution. We have centered and whitened the data set such that,

$$\sum_{i=1}^{N} \frac{N_i}{\sum_{j=1}^{N} N_j} \hat{\mu}_i = 0 \quad \text{and} \quad \sum_{i=1}^{N} \frac{N_i - 1}{\sum_{j=1}^{N} (N_j - 1)} \hat{\Sigma}_i = I.$$

The log-likelihood under H_0 is given by,

$$\begin{aligned}
\log L_{H_0}(\mathcal{X}) &= \sum_{i=1}^{N} \sum_{x \in \mathcal{T}_i} \log p_{\mathcal{N}}(x; 0, I) \\
&= -\frac{1}{2} \sum_{i=1}^{N} N_i d \log(2\pi) + \sum_{x \in \mathcal{T}_i} x^\top x \\
&= -\frac{1}{2} \sum_{i=1}^{N} N_i d \log(2\pi) + \operatorname{tr}\left[\left(\sum_{x \in \mathcal{T}_i} x^\top x - \hat{\mu}_i \hat{\mu}_i^\top \right) + N_i \hat{\mu}_i \hat{\mu}_i^\top \right] \\
&= -\frac{1}{2} \sum_{i=1}^{N} N_i d \log(2\pi) + N_i \hat{\mu}_i^\top \hat{\mu}_i + \operatorname{tr}\left[(N_i - 1) \hat{\Sigma}_i \right] \\
&= -\frac{1}{2} \sum_{i=1}^{N} \left(N_i d \log(2\pi) + N_i \hat{\mu}_i^\top \hat{\mu}_i \right) + \left(\sum_{i=1}^{n} (N_i - 1) \right) \operatorname{tr}[I] \\
&= -\frac{1}{2} \sum_{i=1}^{N} N_i d \log(2\pi) + N_i \hat{\mu}_i^\top \hat{\mu}_i + (N_i - 1) d.
\end{aligned}$$

The log-likelihood under H_A is given by,

$$\begin{aligned}
\log L_{H_A}(\mathcal{X}) &= \sum_{i=1}^{N} \sum_{x \in \mathcal{T}_i} \log p_{\mathcal{N}}(x; \hat{\mu}_i, \hat{\Sigma}_i) \\
&= -\frac{1}{2} \sum_{i=1}^{N} N_i d \log(2\pi) + N_i \log\det \hat{\Sigma}_i + \sum_{x \in \mathcal{T}_i} (x - \hat{\mu}_i)^\top \hat{\Sigma}_i^{-1} (x - \hat{\mu}_i) \\
&= -\frac{1}{2} \sum_{i=1}^{N} N_i d \log(2\pi) + N_i \log\det \hat{\Sigma}_i + \operatorname{tr}\left[\sum_{x \in \mathcal{T}_i} (x - \hat{\mu}_i)(x - \hat{\mu}_i)^\top \hat{\Sigma}_i^{-1} \right] \\
&= -\frac{1}{2} \sum_{i=1}^{N} N_i d \log(2\pi) + N_i \log\det \hat{\Sigma}_i + \operatorname{tr}\left[(N_i - 1) \hat{\Sigma}_i \hat{\Sigma}_i^{-1} \right] \\
&= -\frac{1}{2} \sum_{i=1}^{N} N_i d \log(2\pi) + N_i \log\det \hat{\Sigma}_i + (N_i - 1) d.
\end{aligned}$$

Thus the test statistic, and the SSA objective function, is given by

$$\begin{aligned}
\Lambda(\mathcal{X}) &= -2(\log L_{H_0} - \log L_{H_A}) \\
&= \sum_{i=1}^{N} N_i \left(-\log\det \hat{\Sigma}_i + \hat{\mu}_i^\top \hat{\mu}_i \right).
\end{aligned}$$

B Measuring the Error of SSA

In the case where the true solution (the mixing matrix A) is known, e.g. in simulations on synthetic data, we may want to measure the accuracy of the demixing estimated by the SSA algorithm. Since only the non-stationary subspace (spanned by the columns of $A^{\mathfrak{n}}$) is identifiable, we measure the deviation between the true and the estimated $\mathfrak{n}$-space. To that end, we define a *subspace error* based on the *principle angles* between two subspaces. Let U and V be two linear subspaces of the vector space $\mathcal{V}$ with $\dim U \leq \dim V$. The first principle angle θ_1 is defined as the smallest angle between any two vectors $u \in U$ and $v \in V$,

$$\theta_1 = \min \left\{ \frac{u^\top v}{\|u\| \|v\|} \middle| \, u \in U, v \in V \right\} = \angle(u_1, v_1),$$

where the vectors u_1 and v_1 that minimize the angle are called the principle vectors. The remaining principle angles $\theta_2, \ldots, \theta_{\dim U}$ are found recursively in the orthogonal complement of the previous principle vectors,

$$\theta_{k+1} = \min \left\{ \frac{u^\top v}{\|u\| \|v\|} \middle| \, u \in U, u \perp u_1, \ldots, u_k, v \in V, v \perp v_1, \ldots, v_k \right\}.$$

Thus the principle vectors in each subspace are mutually orthogonal. If a principle vector θ_k is zero then U and V share a common subspace and $u_k, v_k \in U \cap V$.

Hence the dimension of the intersection of U and V is equivalent to the number of zero principle angles. The principle angles and vectors can be found using the singular value decomposition (SVD). Let the columns of the matrices U' and V' be orthonormal bases for the spaces U and V respectively. From the singular value decomposition $\hat{U}\Sigma\hat{V}^\top = (U')^\top V'$ we obtain the principle angles from the singular values, $\cos\theta_k = \Sigma_{kk}$ and the principle vectors are the columns of $\hat{U}$ and $\hat{V}$.

In order to quantify the difference between the true and the estimated stationary subspace (spanned by the columns of $\hat{A}^{\mathtt{n}}$), we define the subspace error given the principle angles $\theta_1, \ldots, \theta_{D-d}$ as

$$\mathcal{E}(\hat{A}^{\mathtt{n}}, A^{\mathtt{n}}) = \frac{1}{D-d} \sum_{i=1}^{D-d} \sin^2(\theta_i).$$

This error is zero when the true subspaces are identical and one if they are orthogonal to each other. The subspace error can be interpreted in terms of the sources. It is the percentage of the signal power which a non-stationary sources loses if it is projected from the true onto the estimated non-stationary sources. Conversely, for the estimated stationary sources $\hat{s}^{\mathtt{s}}(t) = \hat{P}^{\mathtt{s}} x(t)$ this means that $\mathcal{E}(\hat{A}^{\mathtt{n}}, A^{\mathtt{n}})$ is the percentage of non-stationary signal power that they contain.

The SSA Toolbox comes with a Matlab function `subspace_error` that computes the subspace error between two subspaces spanned by the columns of two matrices.

References

[1] Paul von Bünau, Frank C. Meinecke, Franz C. Király, and Klaus-Robert Müller. Finding stationary subspaces in multivariate time series. *Phys. Rev. Lett.*, 103(21):214101, Nov 2009.