

# **TSTOOL User Manual**

## **Version 1.11**

Christian Merkwirth  
Ulrich Parlitz  
Immo Wedekind  
Werner Lauterborn

April 12, 2002

# Contents

<b>1</b>	<b>At a glance</b>	<b>5</b>
<b>2</b>	<b>Download and Installation</b>	<b>7</b>
2.1	Installation . . . . .	7
2.1.1	Windows . . . . .	7
2.1.2	Unix . . . . .	7
2.1.3	Global installation . . . . .	8
2.2	First Steps . . . . .	8
2.3	Pitfalls . . . . .	8
2.4	Copyright notice . . . . .	8
<b>3</b>	<b>First Steps</b>	<b>9</b>
3.1	Example analysis of a time-series from a chaotic Colpitts oscillator . . . . .	9
<b>4</b>	<b>Nearest Neighbors Searching</b>	<b>13</b>
4.1	Definition . . . . .	13
4.2	Approximate nearest neighbors searching . . . . .	13
4.3	Range searching . . . . .	14
4.4	Matlab mex-functions . . . . .	14
4.4.1	nn_prepare . . . . .	14
4.4.2	nn_search . . . . .	14
4.4.3	range_search . . . . .	15
4.5	Example session . . . . .	16
<b>5</b>	<b>Handling the Graphical User Interface</b>	<b>19</b>
5.1	Filelist . . . . .	19
5.2	Figure . . . . .	20
5.3	Menus . . . . .	20
5.3.1	Signal . . . . .	20
5.3.2	Methods I . . . . .	21
5.3.3	Methods II . . . . .	22

5.3.4	Utilities . . . . .	23
5.3.5	Modify . . . . .	24
5.3.6	Macro . . . . .	25
5.3.7	Options . . . . .	25
5.3.8	Help . . . . .	26
5.3.9	View . . . . .	26
<b>6</b>	<b>Mex-Function Reference</b>	<b>27</b>
6.1	<code>akimaspline</code> - Cubic spline interpolation using Akima splines . . . . .	27
6.2	<code>amutual</code> - compute auto mutual information function . . . . .	28
6.3	<code>baker</code> - Generate Baker time-series . . . . .	28
6.4	<code>boxcount</code> - Classical boxcounting algorithm . . . . .	29
6.5	<code>cao</code> - Determine minimum embedding dimension by Cao's method . . . . .	29
6.6	<code>chaosys</code> - integrate dynamical system given by a set of ordinary differential equations	30
6.7	<code>corrsum</code> - Computation of the correlation sum . . . . .	32
6.8	<code>corrsum2</code> - Computation of the correlation sum . . . . .	33
6.9	<code>fnearneigh</code> - Fast nearest neighbor search . . . . .	34
6.10	<code>gendimest</code> - Estimate generalized dimension spectrum . . . . .	35
6.11	<code>henon</code> - Generate henon time-series . . . . .	36
6.12	<code>largelyap</code> - Compute separation of nearby trajectories . . . . .	36
6.13	<code>nn_prepare</code> - Do nearest neighbor preprocessing . . . . .	37
6.14	<code>nn_search</code> . . . . .	37
6.15	<code>predict</code> . . . . .	38
6.16	<code>range_search</code> . . . . .	39
6.17	<code>return_time</code> . . . . .	39
6.18	<code>takens_estimator</code> . . . . .	40
6.19	<code>tentmap</code> - Generate tentmap time-series . . . . .	41
6.20	Class <code>signal</code> . . . . .	43
6.20.1	Overview . . . . .	43
6.20.2	Attributes . . . . .	43
6.20.3	Member functions . . . . .	43
6.21	Class <code>description</code> . . . . .	66
6.21.1	Overview . . . . .	66
6.21.2	Attributes . . . . .	67
6.21.3	Member functions . . . . .	67
6.22	Class <code>core</code> . . . . .	70
6.22.1	Overview . . . . .	70
6.22.2	Attributes . . . . .	70

6.22.3	Member functions . . . . .	71
6.23	Class <code>achse</code> . . . . .	79
6.23.1	Overview . . . . .	79
6.23.2	Attributes . . . . .	79
6.23.3	Member functions . . . . .	79
6.24	Class <code>unit</code> . . . . .	82
6.24.1	Overview . . . . .	82
6.24.2	Attributes . . . . .	82
6.24.3	Member functions . . . . .	82
6.25	Class <code>list</code> . . . . .	84
6.25.1	Overview . . . . .	84
6.25.2	Attributes . . . . .	84
6.25.3	Member functions . . . . .	84
<b>7</b>	<b>Frequently asked questions</b>	<b>87</b>
7.1	Questions . . . . .	87
7.2	Answers . . . . .	88
7.2.1	Introduction and general information . . . . .	88
7.2.2	Installation of TSTOOL . . . . .	88
7.2.3	Working with TSTOOL . . . . .	89
7.2.4	Extending TSTOOL . . . . .	91
7.2.5	Miscellaneous questions . . . . .	91
7.2.6	Frequently encountered errors . . . . .	92

# Chapter 1

## At a glance

### What is TSTOOL ?

TSTOOL is a software package for signal processing with emphasis on nonlinear time-series analysis.

### Objectives

- Implement existing algorithms for nonlinear time-series analysis
- Develop new methods for specific data analysis problems
- Create an expandable platform for signal processing

### Implementation

The package is written partly in *Matlab* and partly in **C++**.

Advantages of *Matlab* are :

- Reduced code development time
- Extensive collection of intrinsic mathematical functions
- Excellent graphic capabilities
- High portability from *Unix* to *Windows NT* and other platforms

**C++** is used for computationally demanding algorithms.

### Graphical user interface

A graphical user interface (*GUI*) gives access to the underlying signal processing commands. Parameters for the commands are set via dialog windows.



## Chapter 2

# Download and Installation

### 2.1 Installation

Unpack the compressed TSTOOL distribution into a directory, e.g `C:\Program Files` on Windows, `/usr/local` on Unix.

This can be done with an unpacking tool like *Winzip* if you are working with Windows, or `gzip -dc filename.tgz | tar -xvf -` if you are working with Unix.

After unpacking TSTOOL you get a new directory named `OpenTSTOOL` witch should now contain:

- `startup.m` - the matlab startup script that calls `settpath.m`
- `settpath.m` - a script that does path settings
- `tstoolbox` - the directory that contains all TSTOOL functions, mex files etc.
- `mex-dev` - Source code of the C++ parts of TSTOOL
- `Doc` - HTML/PDF Documentation
- `gpl.txt` - Gnu General Public License

There are several methods to install TSTOOL.

#### 2.1.1 Windows

The simplest way is to change to the `OpenTSTOOL` directory and run Matlab. Matlab will execute the `startup.m` file and so the path settings will be done correctly. Under Windows you can generate an 'TSTOOL' icon (a reference to the matlab binary) with the working path as `OpenTSTOOL`. After double-click to this new icon simply type `tstool`.

You can also use the *global* installation method (2.1.3) under Windows if you dont want to change your working directory to the `OpenTSTOOL`-directory.

#### 2.1.2 Unix

Under unix-like multi-user environments there's an other possibility to install `tstool`. Edit or create a file `~/matlab/startup.m` and enter the following lines into it:

```
path('<full path to the OpenTSTOOL-Dir>',path);
settpath('<full path to the OpenTSTOOL-Dir>');
```

Now you can invoke matlab everywhere and have access to TSTOOL.

### 2.1.3 Global installation

Last but not least if all users of a network wide matlab installation should have access to TSTOOL, edit the file `toolbox/local/matlabrc.m` in the network wide matlab installation and insert the few lines above (2.1.2) into it if you have permission to do so. Otherwise ask your system administrator.

## 2.2 First Steps

1. Start Matlab
  - Windows: Double click the TSTOOL icon that you placed onto your desktop
  - Unix: (installed as 2.1.1), change to the OpenTSTOOL directory and run `matlab`
  - Unix: (installed as 2.1.2) simply run `matlab`
2. Enter `tsdemo` on the Matlab console. This should start a short demo script.
3. Enter `tsdemo2` on the Matlab console. This should run a second script that shows an analysis of a chaotic signal. The reference output of the analysis can be found here.
4. Enter `tstool` to start the graphical user interface for the TSTOOL package.

## 2.3 Pitfalls

See also the FAQ (frequently asked questions) 7!

1. When using *Winzip*, enable *Use path information* to make sure that subdirectories are created.
2. If you issue the command "clear all" at the Matlab console, TSTOOL will no longer work properly. As a remedy, simply change to the TSTOOL base directory and issue "settpath".
3. TSTOOL will not work with Matlab version prior to **5.2**!
4. Matlab 5.2 needs a patch from *The Mathworks* to work properly under **Windows 98**
5. It's not a good idea to place the TSTOOL distribution into the Matlab directory. We obtained reports about strange bugs occuring when the TSTOOL distribution is extracted into the directory where the Matlab system is installed.
6. Under Irix there are some version conflicts between Matlab 5.2/5.3 and 6. The extension of the mex-files has changed. Normally this is no problem because the old extension will also work but TSTOOL have own mex-file directories for each system (Sun, Linux, SGI, Windows) named with this mex-file extension. So the path settings `settpath` do can be incorrect for the Irix system. In such case please replace the following line in the `settpath.m` file:

```
path(fullfile(TSTOOLpath, fullfile('mex', mexext)), path);
```

with this line:

```
path(fullfile(TSTOOLpath, fullfile('mex', 'mexsg64')), path);
```

## 2.4 Copyright notice

TSTOOL falls under the Gnu General Public License. See `gpl.txt` in the `OpenTSTOOL` directory or <http://www.physik3.gwdg.de/tstool/gpl.txt>.



## Chapter 3

# First Steps

### 3.1 Example analysis of a time-series from a chaotic Colpitts oscillator

In this section we briefly demonstrate basic steps for analysing a chaotic time series. The methods used will be explained in more detail in the following sections.

```
>> s = signal('colpitts.dat','ascii')
      s = signal object

      Dlens : 6001
      X-Axis 1 : |

      Name : colpitts
      Type :

      Attributes of data values :
      |

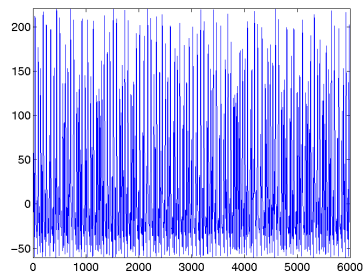
      Comment :

      History :
17-Aug-1999 15:08:24 : Imported from ASCII file 'colpitts.dat'
```

By entering the above command line, the overloaded constructor for class `signal` was called. Giving a filename as argument tells the constructor to load the datafile and convert it into a signal object. The datafile `'colpitts.dat'` contains a time-series generated by an electrical Colpitts circuit that oscillates chaotically.

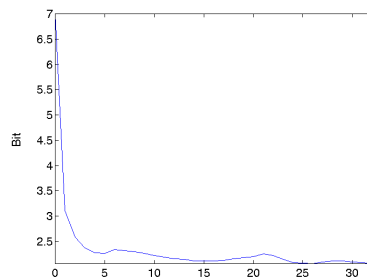
To plot signal `s`, just issue the following command :

```
view(s);
```



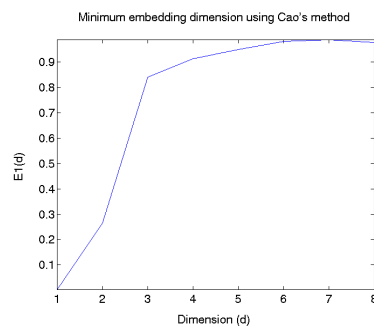
Lets find a good choice for a delay-time by using the first minimum of the auto mutual information function

```
a = amutual(s,32);
view(a);
```



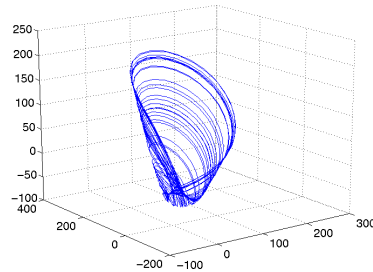
the first minimum of the auto mutual information can be found at four. Now we need to know the minimal embedding dimension for the colpitts signal. We use Cao's method with a delay time of four, a maximal dimension of eight, three nearest neighbors and 1000 reference points.

```
c = cao(s,8,4,3,1000);
view(c);
```



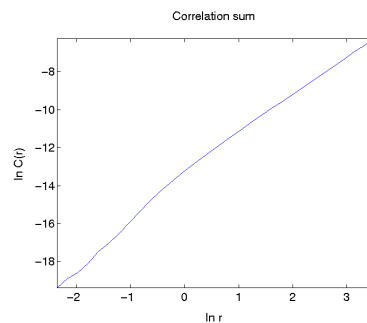
There's a kink in the graph produced by Cao's method at three. So now do a time-delay reconstruction of the Colpitts signal with embedding dimension 3 and delay 4.

```
e = embed(s, 3, 4);
view(e);
```



What's the correlation dimension of the reconstructed data set ? First let's take a look at the scaling of the correlation sum versus the radius (as log-log plot).

```
view(corrsum(e, -1, 0.05, 40, 32));
```



Next, we use the Takens estimator for the correlation dimension. It needs basically the same input arguments as the function `corrdim2`.

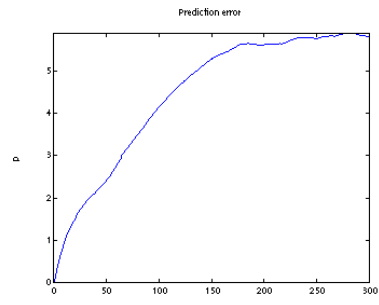
```
>> takens_estimator(e, -1, 0.05, 40)
```

```
ans =
```

```
1.9483
```

And what about its largest Lyapunov exponent ? To estimate the largest Lyapunov exponent, we take a look at the scaling of the prediction error.

```
view(largelyap(e, 1000, 300, 40, 2));
```



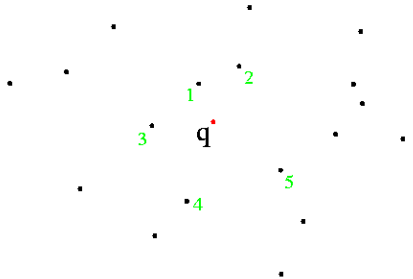
## Chapter 4

# Nearest Neighbors Searching

An integral part of a majority of methods for nonlinear time series analysis is searching for nearest neighbors. The performance of these methods depends strongly of the performance of the employed nearest neighbor algorithm. Thus, choosing an efficient nearest neighbor algorithm should be done very carefully.

### 4.1 Definition

Definition : A set  $\mathbf{P}$  of data points in  $D$ -dimensional space is given. Then we define the nearest neighbor to some reference point  $\mathbf{q}$  (also called *query point*) to be the point of data set  $\mathbf{P}$  that has the smallest distance to  $\mathbf{q}$  (we don't issue the problem of ambiguity at this point). The more general task of finding more than one nearest neighbor is called *k nearest neighbors problem*. In general, the reference point  $\mathbf{q}$  is an arbitrarily located point, but it is also possible that  $\mathbf{q}$  is itself a member of data set  $\mathbf{P}$  (as illustrated in the figure, where five neighbors to  $\mathbf{q}$  (excluding self match) are found).



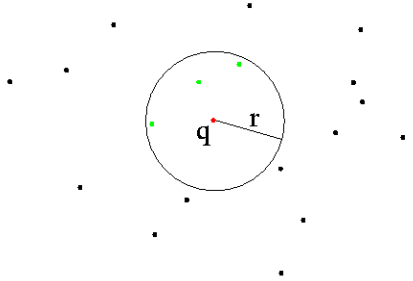
### 4.2 Approximate nearest neighbors searching

Approximate nearest neighbors algorithms report neighbors to the query point  $\mathbf{q}$  with distances possibly greater than the true nearest neighbors distances. The maximal allowed relative error *epsilon* is given as a parameter to the algorithm. For *epsilon*=0, the approximate search returns the true (exact) nearest neighbor(s).

Computing exact nearest neighbors for data set with *fractal dimension* much higher than 6 seems to be a very time-consuming task. Few algorithms seem to perform significantly better than a brute-force computation of all distances. However, it has been shown that by computing nearest neighbors approximately, it is possible to achieve significantly faster execution times with relatively small actual errors in the reported distances.

## 4.3 Range searching

In the task of *range searching*, we ask for all points of data set  $\mathbf{P}$  that have distance  $\mathbf{r}$  or less from the query point  $\mathbf{q}$ . Sometimes *range searching* is called a *fixed size approach*, while *k nearest neighbors searching* is called a *fixed mass approach*.



## 4.4 Matlab mex-functions

### 4.4.1 nn\_prepare

`nn_prepare` does the preprocessing for a given data set *pointset*. The returned data structure *atria* contains preprocessing information that is necessary to use `nn_search` or `range_search`.

Preprocessing and searching is divided into different mex-files to give the user the possibility to *re-use* the preprocessing data (contained in *atria*) when doing multiple searches on the same point set. However, as soon as the underlying point set is changed or modified, one has to recompute *atria* for the changed point set.

**Syntax:**

- `atria = nn_prepare(pointset)`
- `atria = nn_prepare(pointset, metric)`
- `atria = nn_prepare(pointset, metric, clustersize)`

**Input arguments:**

- **pointset** - a  $\mathbf{N}$  by  $\mathbf{D}$  double matrix containing the coordinates of the point set, organized as  $\mathbf{N}$  points of dimension  $\mathbf{D}$
- **metric** - (optional) either 'euclidian' or 'maximum' (default is 'euclidian')
- **clustersize** - (optional) threshold for clustering algorithm, defaults to 64

### 4.4.2 nn\_search

`nn_search` does exact or approximate k-nearest neighbor queries to one or more query points. These query points can be given explicitly or taken from the data set of points (see below).

Before one can use `nn_search`, one has to call `nn_prepare` to compute the preprocessing information. However, as long as the input point set isn't modified, the preprocessing information is valid and can be re-used for multiple calls to `nn_search` or `range_search`.

**Syntax:**

- `[index, distance] = nn_search(pointset, atria, query_points, k)`
- `[index, distance] = nn_search(pointset, atria, query_points, k, epsilon)`
- `[index, distance] = nn_search(pointset, atria, query_indices, k, exclude)`
- `[index, distance] = nn_search(pointset, atria, query_indices, k, exclude, epsilon)`

#### Input arguments:

- **pointset** - a **N** by **D** double matrix containing the coordinates of the point set, organized as **N** points of dimension **D**
- **atria** - output of `nn_prepare` for pointset
- **query\_points** - a **R** by **D** double matrix containing the coordinates of the query points, organized as **R** points of dimension **D**
- **query\_indices** - query points are taken out of the pointset, `query_indices` is a vector of length **R** which contains the indices of the query points (indices may vary from 1 to N)
- **k** - number of nearest neighbors to compute
- **epsilon** - (optional) relative error for approximate nearest neighbors queries, defaults to 0 (= exact search)
- **exclude** - in case the query points are taken out of the pointset, `exclude` specifies a range of indices which are omitted from search. For example if the index of the query point is 124 and `exclude` is set to 3, points with indices 121 to 127 are omitted from search. Using `exclude = 0` means: exclude self-matches

#### Output arguments:

- **index** - a matrix of size **R** by **k** which contains the indices of the nearest neighbors. Each row of **index** contains **k** indices of the nearest neighbors to the corresponding query point.
- **distance** - a matrix of size **R** by **k** which contains the distances of the nearest neighbors to the corresponding query points, sorted in increasing order.

### 4.4.3 range\_search

The routine `range_search` does a range search to one or more query points. These query points can be given explicitly or taken from the data set of points (see below).

Before one can use `range_search`, one has to call `nn_prepare` to compute the preprocessing information. However, as long as the input point set isn't modified, the preprocessing information is valid and can be re-used for multiple calls to `nn_search` or `range_search`.

#### Syntax:

- `[count, neighbors] = range_search(pointset, atria, query_points, r)`
- `[count, neighbors] = range_search(pointset, atria, query_indices, r, exclude)`

#### Input arguments:

- **pointset** - a **N** by **D** double matrix containing the coordinates of the point set, organized as **N** points of dimension **D**

- **query\_points** - a **R** by **D** double matrix containing the coordinates of the query points, organized as **R** points of dimension **D**
- **query\_indices** - query points are taken out of the pointset, query\_indices is a vector of length **R** which contains the indices of the query points
- **r** - range or search radius ( $r > 0$ )
- **exclude** - in case the query points are taken out of the pointset, exclude specifies a range of indices which are omitted from search. For example if the index of the query point is 124 and exclude is set to 3, points with indices 121 to 127 are omitted from search. Using exclude = 0 means: exclude self-matches

#### Output arguments:

- **count** - a vector of length **R** contains the number of points within distance r to the corresponding query point
- **neighbors** - a Matlab cell structure of size **R** by **2** which contains vectors of indices and vectors of distances to the neighbors for each given query point. This output argument can not be stored in a standard Matlab matrix because the number of neighbors within distance r is not the same for all query points. The vectors of indices and distances for one query point have exactly the length that is given in count. The values in the distances vectors are **not** sorted.

## 4.5 Example session

```
% create a 3-dimensional data set with 100000 points

pointset = rand(100000, 3);

% do the preprocessing for this point set

atria = nn_prepare(pointset, 'euclidian');

% now search for 2 (exact) nearest neighbors, using points 1 to
% 10 as query points, excluding self-matches

[index, distance] = nn_search(pointset, atria, 1:10, 2, 0)

index =

    5618    96574
   38209    84549
   54991    60397
   38429    59732
    4114    76991
   72121     452
   13678    59332
   26022    16718
   86042    38436
   24830    44434

distance =
```



```

0.0101    0.0175
0.0078    0.0134
0.0132    0.0167
0.0050    0.0223
0.0087    0.0097
0.0124    0.0189
0.0129    0.0168
0.0046    0.0110
0.0101    0.0103
0.0156    0.0177

% now do a range search for radius 0.0224, using points 1 to 10 as
% query points, excluding self-matches

[count, neighbors] = range_search(pointset, atria, 1:10, 0.0224, 0)

count =

     4
    10
     7
     2
     5
     6
     2
     4
     7
     5

neighbors =

    [1x4 double]    [1x4 double]
    [1x10 double]   [1x10 double]
    [1x7 double]    [1x7 double]
    [1x2 double]    [1x2 double]
    [1x5 double]    [1x5 double]
    [1x6 double]    [1x6 double]
    [1x2 double]    [1x2 double]
    [1x4 double]    [1x4 double]
    [1x7 double]    [1x7 double]
    [1x5 double]    [1x5 double]

% let's see the indices of the points that are within range to the first query point
neighbors{1,1}

ans =

    56921    97100    96574    5618

% let's see the corresponding distances of the points that are
% within range to the first query point

neighbors{1,2}

```

ans =

0.0176    0.0186    0.0175    0.0101

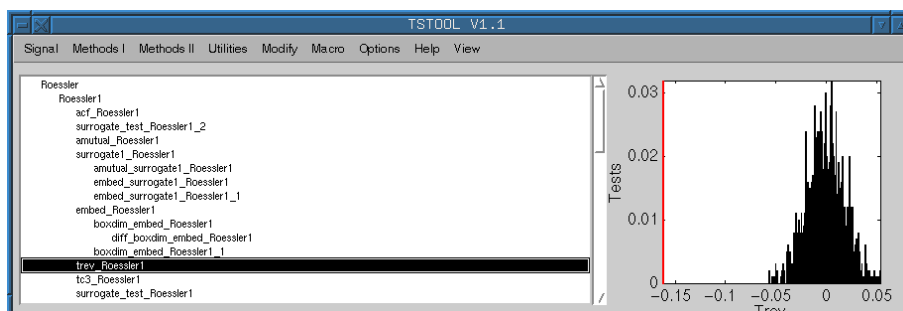
## Chapter 5

# Handling the Graphical User Interface

With the Graphical User Interface (GUI) most of TSTOOLs methods are available without coping with the command line syntax of every function.

To invoke the GUI simply type `tstool` at the matlab command prompt. If you invoke the GUI for the first time, you get informed that the GUI will generate a directory for temporary files. This will reside at `OpenTSTOOL/datafiles` on Windows systems and at `~/.tstool` on Unix systems.

First of all, how does the GUI looks like?



There are three parts:

- Filelist
- Figure
- Menubar

### 5.1 Filelist

Every loaded or generated signal shows its name in an own line, they arranged hierarchically. The data of the signals is stored in separated files in the directory generated at the first run.

To process a special method, click on one of the signals in the filelist an choose the method from the menubar.

## 5.2 Figure

This figure can show you the signal you have choosen from the filelist. Normaly this feature is switched off because of the time consumption especialy for large signals. To switch on this feature choose the menu item `OPTION-INSTANT VIEW-SMALL WINDOW`.

## 5.3 Menus

### 5.3.1 Signal

All menu items in this menu have something to do with the filelist and the storage of the signals.

- **LOAD**

This menu item simply loads data into the GUI (more precise: load a signal and save it to the temporary data directory). A line in the fileview will be added with its filename.

- **SAVE**

Write the marked signal in the fileview to disk.

- **IMPORT FILE FROM**

Generate a signal from foreign formats like `ASCII`, `Matlab Vector`, `Soundfiles` etc. See `signal` class constructor reference (6.20.3.66) for more information.

- **EXPORT FILE FROM**

Write the marked signal in the fileview to disk in a foreign format. See `signal/write` reference (6.20.3.83) for more information.

- **GENERATE**

This menu item can generate signals. See `mex/chaosys` reference (6.6).

- **AUDIO PLAYBACK**

Plays a scalar signal as audio with the matlab function `soundsc`. If there is no sampling rate set in the signal 8KHz will be used.

- **RESCAN**

Normally all files with the correct extension in the temporary directory are displayed in the filelist. For some reason it is possible that a signal is displayed in the filelist but doesnt exist (e. g. an other process deleted this file) or some files missing in the filelist (e. g. if you simply copy a signal file in the temporary directory without using the `LOAD` menu item). In such situation use the `RESCAN` menu item to let `TSTOOL` look up all files again correctly. This can take some time on slow machines or large signal files. Simply restarting the GUI will not make a rescan!

- **REMOVE ENTRY**

If you want to remove a filelist entry use this menu item or simply type `Ctrl-d`. The selected filelist entry will disappear and the corresponding temporary file will be deleted.

- **SHOW**

A signal stores many information about the data. This information can be displayed by this menu item.

- **EDIT**

- **DESIRED TYPE OF PLOT**

Here you can choose the type of plot `TSTOOL` should use for your signal. See `signal/view` reference (6.20.3.82) for additional information.

- DESCRIPTIVE PARAMETERS
- AXES LABELS
- COMMENT TEXT

### 5.3.2 Methods I

In this menu all methods with scalar input are grouped. Most of this methods invokes the underlying TSTOOL-function directly and do not need addition explanations. To enter the parameters a dialog box will be opened.

For some parameters there is a checkbox at the right side named '**in units**'. Normally this parameter is entered in the units of samples. When you switch on the checkbox you can also enter this parameters in units of the first axis.

- RECONSTRUCTION
  - TIME-DELAY VECTORS  
see `signal/embed` (6.20.3.21)
  - MINIMUM EMBEDDING DIMENSION (CAO's)  
see `signal/cao` (6.20.3.9)
- SPECTRAL
  - FFT  
see `signal/fft` (6.20.3.22)
  - PERIODOGRAM  
see `signal/spec` (6.20.3.68)
  - SPECTROGRAM  
see `signal/spec2` (6.20.3.69)
  - SCALOGRAM  
see `signal/scalogram` (6.20.3.62)
- DERIVATIVE/INTEGRATION
  - INTEGRATE  
see `signal/int` (6.20.3.33)
  - DIFFERENTIATE  
see `signal/diff` (6.20.3.18)
- CORRELATION AND MORE
  - AUTO CORRELATION  
see `signal/acf` (6.20.3.2)
  - AUTO MUTUAL INFORMATION  
see `signal/amutual` (6.20.3.4)
- FILTER
  - MOVING AVERAGE  
see `signal/movav` (6.20.3.44)
  - MEDIAN FILTER  
see `signal/medianfilt` (6.20.3.40)
  - MULTIREOLUTION ANALYSIS  
see `signal/mutlires` (6.20.3.45)

- SURROGATE DATA GENERATION
  - PERMUTATION OF SAMPLES  
see `signal/surrogate3` (6.20.3.74)
  - THEILER ALG. I  
see `signal/surrogate1` (6.20.3.72)
  - THEILER ALG. II  
see `signal/surrogate2` (6.20.3.73)
- SURROGATE DATA TEST
  - TIME REVERSIBILITY  
see `signal/trev` (6.20.3.80)
  - HIGHER ORDER MOMENTS  
see `signal/tc3` (6.20.3.78)
  - FUNCTION  
see `signal/surrogate_test` (6.20.3.75)
- PREDICTION
  - LOCAL CONSTANT  
see `signal/predict2` (6.20.3.55)
- MISC
  - SQUARED MAGNITUDE  
see `signal/power` (6.20.3.53)
  - ABSOLUTE VALUE  
see `signal/abs` (6.20.3.1)
  - DECIBEL VALUE  
see `signal/db` (6.20.3.16)
  - HISTOGRAM  
see `signal/histo` (6.20.3.30)

### 5.3.3 Methods II

In this menu all methods with multivariate input signals are grouped.

- DECOMPOSITIONS
  - PCA (KARHUNEN-LOEVE)  
see `signal/pca` (6.20.3.49)
  - ARCHETYPAL ANALYSIS  
see `signal/arch` (6.20.3.7)
- LYAPUNOV EXPONENTS
  - LARGEST  
see `signal/largelyap` (6.20.3.36)
- FRACTAL DIMENSIONS
  - BOX COUNTING APPROACH

- \* CAPACITY DIMENSION D0  
see `signal/boxdim` (6.20.3.8)
- \* INFORMATION DIMENSION D1  
see `signal/infodim` (6.20.3.31)
- \* CORRELATION DIMENSION D2  
see `signal/corrdim` (6.20.3.11)
- CORRELATION SUM APPROACH
  - \* CORRELATION SUM D2 (GPA LIKE APPROACH)  
see `signal/corrsum` (6.20.3.12)
  - \* CORRELATION DIMENSION D2 (FIXED NUMBER OF PAIRS)  
see `signal/corrsum2` (6.20.3.13)
  - \* TAKENS ESTIMATOR D2  
see `signal/takens_estimator` (6.20.3.77)
- NEAREST NEIGHBOR ALGORITHMS
  - \* INFORMATION DIMENSION D1 (NNK)  
see `signal/infodim2` (6.20.3.32)
  - \* FRACTAL DIMENSION SPECTRUM  
see `signal/fracdims` (6.20.3.27)
- PERIODICITY
  - RETURN TIMES  
see `signal/return_time` (6.20.3.58)
  - RECIPROCAL LOCAL DENSITY  
see `signal/localdensity` (6.20.3.38)
- MODELING
  - POLYNOM SELECTION  
see `util/pauswahl`
- POINCARÉ SECTION  
see `signal/poincare` (6.20.3.52)
- PREDICTION
  - LOCAL CONSTANT  
see `signal/predict` (6.20.3.54)

### 5.3.4 Utilities

Some useful information about the signals can be retrieved by functions in this menu.

- MINIMUM  
see `signal/min` (6.20.3.42)
- MAXIMUM  
see `signal/max` (6.20.3.39)
- FIRST LOCAL MINIMUM  
see `signal/firstmin` (6.20.3.25)
- FIRST LOCAL MAXIMUM  
see `signal/firstmax` (6.20.3.24)

- FIRST ZERO CROSSING  
see `signal/firstzero` (6.20.3.26)
- MEAN  
see `mean` (Matlab reference)
- STANDARD DEVIATION  
see `std` (Matlab reference)
- RMS  
root mean square
- COMPARE TWO SIGNALS  
Only the data values are compared. See `core/compare` (6.22.3.3)

### 5.3.5 Modify

- CUT  
see `signal/cut` (6.20.3.15)
- SWAP DIMENSIONS  
see `signal/swap` (6.20.3.76)
- REVERSE  
see `signal/reverse` (6.20.3.59)
- INTERPOLATIONS
  - CUBIC SPLINE  
see `signal/upsample` (6.20.3.81)
  - AKIMA SPLINE  
see `signal/upsample` (6.20.3.81)
  - FFT BASED  
see `signal/upsample` (6.20.3.81)
- Normalize
  - CENTER AROUND ZERO  
see `signal/center` (6.20.3.10)
  - SCALE BY FACTOR  
see `signal/scale` (6.20.3.61)
  - FIT TO INTERVAL  
see `signal/norm1` (6.20.3.47)
  - CENTER AND DIVIDE BY STD  
see `signal/norm2` (6.20.3.48)
  - REMOVE TREND  
see `signal/trend` (6.20.3.79)
  - TRANSFORM TO RANG VALUES  
see `signal/rang` (6.20.3.56)
- SPLIT MULTICHANNEL SIGNAL  
Splits up a  $n$ -channel signal in  $n$  signals by using `signal/cut` (6.20.3.15).



- ADD TWO SIGNALS  
see `signal/plus` (6.20.3.51)
- DIFFERENCE OF TWO SIGNALS  
see `signal/minus` (6.20.3.43)
- MERGE TWO SIGNALS  
see `signal/merge` (6.20.3.41)

### 5.3.6 Macro

TSTOOL records the processed commands for every signal. So TSTOOL knows how this signal is modified and can generate a matlab script with processes the same commands to arbitrary signal.

The generated scripts will be saved in the directory `scripts` inside the directory for temporary files.

- CREATE MACRO FROM SIGNAL  
Generate script named `macro.m`.
- SHOW/EDIT MACRO
- RENAME MACRO  
Renamed macros will be displayed at the end of this menu after restart of TSTOOL.
- APPLY MACRO TO SIGNAL  
Invoke `macro.m` with the selected signal.
- APPLY MACRO TO ALL  
Invoke `macro.m` with every loaded signal.

After the separation line a list of all `*.m`-Files in the directory `scripts` is shown. Selecting one of these menus apply the corresponding script to the actual selected signal.

### 5.3.7 Options

Here some settings can be edited. All these settings will be saved in the file `tstool.mat` in the temporary data directory.

- PARAMETERS
  - RECONSTRUCTION PARAMETERS  
The default settings for the Reconstruction dialog box can be edited here (see menu METHODS I - RECONSTRUCTION - TIME DELAY VECTORS 5.3.2)
  - DEFAULT WINDOW TYPE  
Used by FFT etc.
- FILE AND DIRECTORY OPTIONS  
The actual search directory for LOAD and SAVE can be edited here. Also the default file extension can be altered.
- INSTANT VIEW
  - SMALL WINDOW  
If you switch on this feature, the small figure at the right of the filelist will show you the selected signal immediately after selection.
  - LARGE WINDOW  
Every time a new signal is generate (e.g. applying a command to an existing signal) e new figure window will be opened displaying it.

### 5.3.8 Help

The menu `USAGE` will start your web browser with the HTML-Version of this manual shipped with the `OpenTSTOOL` distribution. Run `'help docopt'` at the matlab command prompt to configure your web browser correctly.

After the separation line you can view the matlab command line help for every method of the `signal` class. The same information is in the HTML- and in the PDF-Version of the manual in class reference section 6.20.

### 5.3.9 View

This menu invokes a new figure window viewing the selected signal using the `signal/view` command (6.20.3.82). You can also simply type `Ctrl-v`.

## Chapter 6

# Mex-Function Reference

Parts of TSTOOL's functionality are coded in mex-files. All TSTOOL mex-files are located in the directory `tstoolbox/mex`. It is possible to use these mex-files independently of the full TSTOOL installation.

### 6.1 `akimaspline` - Cubic spline interpolation using Akima splines

Compared to Matlab's built-in cubic spline, Akima spline interpolation better copes with discontinuities in a time series.

#### Syntax:

- `yy = akimaspline(x,y,xx)`

#### Input arguments:

- `x`, `y` - vectors describing knot data, see Matlab's original spline function
- `xx` - vector of positions at which the spline is evaluated

#### Output arguments:

- `yy` - evaluated function values

#### Example:

```
x = 1:100;
y = floor((x + rand(1, 100))/ pi);
xx = 1:0.1:100;
yy = akimaspline(x, y, xx);
plot(x,y, xx, yy, 'r')
```

## 6.2 amutual - compute auto mutual information function

Fast, but crude auto mutual information of a scalar timeseries for the timelags from zero to maxtau. The input time series should be much longer than maximal timelag maxtau. The algorithm uses equidistant histogram boxes, so results are bad in a mathematical sense. However, a fast algorithm based on ternary search trees to store only nonempty boxes is used.

**Syntax:**

- `a = amutual(ts, maxtau, partitions)`

**Input arguments:**

- `ts` - vector holding time series data
- `maxtau` - maximal time lag
- `partitions` - number of partitions for the one-dimensional histogram

**Output arguments:**

- `a` - vector of length maxtau+1, holding auto mutual information

## 6.3 baker - Generate Baker time-series

Generate time-series from the iterated Baker map [150].

**Syntax:**

- `x = baker(length, [eta l1 l2 x0 y0])`

**Input arguments:**

- `length` - number of samples to generate
- `[eta l1 l2 x0 y0]` - vector of parameters and initial conditions

**Output arguments:**

- `x` - time series

**Example:**

```
x = baker(2000, [0.6 0.25 0.4 rand(1,1) rand(1,1)]);  
plot(x(1:end-1,2), x(2:end,2), 'r')
```

## 6.4 boxcount - Classical boxcounting algorithm

boxcount is a fast algorithm that partitions a data set of points into equally spaced and sized boxes. The algorithm is based on Robert Sedgewick's *Ternary Search Trees* [149] which offer a fast and efficient way to create and search a multidimensional histogram. Empty boxes require no storage space, therefore the maximum number of boxes (and memory) used can not exceed the number of points in the data set, regardless of the data set's dimension and the number of partitions per axis.

During processing, data values are scaled to be within the range [0,1]. All columns of the input matrix are scaled by the same factor, so no skewing is introduced into the point set.

**Syntax:**

- `[a,b,c] = boxcount(point_set, partitions)`

**Input arguments:**

- **pointset** - a N by D double matrix containing the coordinates of the point set, organized as N points of dimension D. D is limited to 128.
- **partitions** - number of partitions per axis, limited to 16384. For convenience, if a vector is given, boxcount will iterate over all values of this vector.

**Output arguments:**

- **a** - vector of size D with:  $\log_2(\text{sum}(\text{Number of nonempty boxes}))$
- **b** - vector of size D with:  $\text{sum}(p * \log_2(p))$ , where p is the relative frequency of points falling into a box
- **c** - vector of size D with:  $\log_2(\text{sum}(p*p))$ , where p is the relative frequency of points falling into a box

**Example:**

```
p = rand(50000, 4);
p = p - min(min(p));
p = p ./ max(max(p));
[a,b,c] = boxcount(p, 16)
```

## 6.5 cao - Determine minimum embedding dimension by Cao's method

This mex-file applies Cao's method [38] to the input data set. If the data set contains points of dimension D, it computes E and E\* for a data set of dimension 1 (taken from the first column of the input data set), then for a data set of dimension 2 (taken from the first two columns) up to a dimension of D. Optionally, this algorithm extends Cao's method in a straightforward manner to use more than one nearest neighbors.

**Syntax:**

- `[E, E*] = cao(pointset, query_indices, k)`

**Input arguments:**

- **pointset** - a N by D double matrix containing the coordinates of the point set, organized as N points of dimension D
- **query\_indices** - query points are taken out of the pointset, query\_indices is a vector of length R which contains the indices of the query points (indices may vary from 1 to N)
- **k** - number of nearest neighbors to compute. Cao's method can be extended to use more than only the first nearest neighbor (k=1).

#### Output arguments:

- **E** and **E\*** are vectors of size D. Please refer the Cao's article [38] for a precise description of their meaning.

## 6.6 chaosys - integrate dynamical system given by a set of ordinary differential equations

**chaosys** gives the user the possibility to compute time series data for a couple of dynamical systems, among which are Lorenz, Chua, Roessler etc. This routine is not meant as a replacement for Matlab's suite of functions for solving ODEs, but as a fast way to generate some data sets to evaluate the processing capabilities of TSTOOL. The integration is done by an ODE solver using an Adams Pece scheme with local extrapolation [151]. It is at least faster than Matlab's native ODE solver. However, it is not possible to extend the set of systems without recompiling **chaosys**.

#### Syntax:

- `x = chaosys(length, stepwidth, initial_conditions, mode, parameters)`

#### Input arguments:

- **length** - number of samples to generate
- **stepwidth** - integration step size
- **initial\_conditions** - vector of initial conditions
- **mode**:
  - 0: Lorenz
  - 1: Generalized Chua : Double Scroll
  - 2: Generalized Chua : Five Scroll
  - 3: Duffing
  - 4: Roessler
  - 5: Toda Oscillator
  - 6: Van der Pol Oscillator
  - 7: Pendulum

For an exact definition of the ODE systems, please refer to this header file.

- **parameters** - vector of systems parameters. The order of the parameters is exactly the same as in the constructors of the DGL subclasses in the above file.

#### Output arguments:

- **x** contains the output of the integration, organized as matrix of size samples by dim, where dim is the number of ODEs that define the system

**Example:**

```
x = chaosys(20000, 0.025, [0.1 -0.1 0.02], 0);
plot(x(:,1));
```

**Definitions of the ODEs:**

The **parameters** of the odes are a vector of [a,b,...].

**Lorenz:**

$$\begin{aligned}\frac{dy_1}{dt} &= a(y_1 - y_2) \\ \frac{dy_2}{dt} &= by_1 - y_2 - y_1y_3 \\ \frac{dy_3}{dt} &= y_1y_2 + cy_3\end{aligned}$$

**Generalized Chua:**

$$\begin{aligned}\frac{dy_1}{dt} &= a(y_1 - by_2) \\ \frac{dy_2}{dt} &= by_1 - y_2 + y_3 \\ \frac{dy_3}{dt} &= -cy_2\end{aligned}$$

**Duffing:**

$$\begin{aligned}\frac{dy_1}{dt} &= y_2 \\ \frac{dy_2}{dt} &= -y_1 - y_1^3 - by_2 + a \cos y_3 \\ \frac{dy_3}{dt} &= c\end{aligned}$$

**Rössler:**

$$\begin{aligned}\frac{dy_1}{dt} &= -y_2 - y_3 \\ \frac{dy_2}{dt} &= -y_1 + ay_2 \\ \frac{dy_3}{dt} &= b + y_3(y_1 - c)\end{aligned}$$

**Toda oscillator:**

$$\begin{aligned}\frac{dy_1}{dt} &= 1 + a \sin(bt) - by_2 - \exp y_1 \\ \frac{dy_2}{dt} &= y_1\end{aligned}$$

**van der Pol oscillator:**

$$\begin{aligned}\frac{dy_1}{dt} &= a \sin(bt) - c(by_2^2 - 1) - d^2y_1 \\ \frac{dy_2}{dt} &= y_1\end{aligned}$$

pendulum:

$$\begin{aligned}\frac{dy_1}{dt} &= a \sin(bt) - cby_2 - d \sin y_1 \\ \frac{dy_2}{dt} &= y_1\end{aligned}$$

## 6.7 corrsum - Computation of the correlation sum

The topics correlation sum and correlation dimension estimation can also be found here.

**Syntax:**

- `[c, d] = corrsum(pointset, query_indices, range, exclude)`
- `[c, d] = corrsum(pointset, query_indices, range, exclude, bins)`
- `[c, d] = corrsum(atria, pointset, query_indices, range, exclude)`
- `[c, d] = corrsum(atria, pointset, query_indices, range, exclude, bins)`

**Input arguments:**

- **atria** - output of `nn_prepare` for `pointset` (optional) (cf. Section 6.13)
- **pointset** - a `N` by `D` double matrix containing the coordinates of the point set, organized as `N` points of dimension `D`
- **query\_indices** - query points are taken out of the `pointset`, `query_indices` is a vector of length `R` which contains the indices of the query points (indices may vary from 1 to `N`)
- **range** - search range, may be given in one of two ways
  - If only a single value is given, this value is taken as maximal search radius relative to the attractor diameter ( $0 < \text{relative\_range} < 1$ ). The minimal search radius is determined automatically by searching for the minimal interpoint distance in the data set.
  - If a vector of length two is given, the values are interpreted as absolute minimal and maximal search radius.
- **exclude** - in case the query points are taken out of the `pointset`, `exclude` specifies a range of indices which are omitted from search. For example if the index of the query point is 124 and `exclude` is set to 3, points with indices 121 to 127 are omitted from search. Using `exclude = 0` means: exclude self-matches
- **bins** - number of distance values at which the correlation sum is evaluated, defaults to 32

**Output arguments:**

- **c** - vector of correlation sums, `length(c) = bins`
- **d** - vector of the corresponding distances at which the correlation sums (stored in `c`) were computed. `d` is exponentially spaced, `length(c) = bins`

**Example:**

```
x = chaosys(25000, 0.025, [0.1 -0.1 0.02], 0); % generate data from Lorenz system
x = x(5001:end,:); % discard first 5000 samples due to transient
% now compute correlation sum up to five percent of attractor diameter
[c,d] = corrsum(x, randref(1,20000, 1000), 0.05, 0);
loglog(d,c) % and show the result as log-log plot
```



## 6.8 corrsum2 - Computation of the correlation sum

This is an extended version of the correlation sum algorithm. It tries to accelerate the computation of the correlation sum by using a different number of reference points at each length scale. For large length scales, only a few number of reference points will be used since for this scale, quite a lot of neighbors will fall within this range (and also the search time will be high). The smaller the length scale, the more reference points are used. The algorithm tries to keep the number of pairs found within each range roughly constant at `Npairs` to ensure a good statistic even for the smallest length scales. However, the number of reference points actually used may be limited to be within `[Nref_min Nref_max]` to give at least some control to the user. All reference points are chosen randomly from the data set without reoccurrences of the same index.

### Syntax:

- `[c, d, e, f, g] = corrsum(pointset, Npairs, range, exclude)`
- `[c, d, e, f, g] = corrsum(pointset, Npairs, range, exclude, bins)`
- `[c, d, e, f, g] = corrsum(pointset, Npairs, range, exclude, bins, opt_flag)`
- `[c, d, e, f, g] = corrsum(atria, pointset, Npairs, range, exclude)`
- `[c, d, e, f, g] = corrsum(atria, pointset, Npairs, range, exclude, bins)`
- `[c, d, e, f, g] = corrsum(atria, pointset, Npairs, range, exclude, bins, opt_flag)`

### Input arguments:

- **atria** - output of `nn_prepare` for `pointset` (optional) (cf. Section 6.13)
- **pointset** - a `N` by `D` double matrix containing the coordinates of the point set, organized as `N` points of dimension `D`
- **Npairs** - Number of pairs to find within each length scale. The algorithm will adapt the number of reference points while computing the correlation sum. Reference points are chosen randomly from the `pointset`. Optionally, a vector of the form `[Npairs Nref_min Nref_max]` may be given. For no length scale less than `Nref_min` reference points will be used. Additionally, not more than `Nref_max` reference points will be used at all.
- **range** - search range, may be given in one of two ways
  - If only a single value is given, this value is taken as maximal search radius relative to attractor diameter ( $0 < \text{relative\_range} < 1$ ). The minimal search radius is determined automatically by searching for the minimal interpoint distance in the data set.
  - If a vector of length two is given, the values are interpreted as absolute minimal and maximal search radius.
- **exclude** - in case the query points are taken out of the `pointset`, `exclude` specifies a range of indices which are omitted from search. E.g. if the index of the query point is 124 and `exclude` is set to 3, points with indices 121 to 127 are omitted from search. `exclude = 0` means : exclude self-matches
- **bins** - number of distance values at which the correlation sum is evaluated, defaults to 32
- **opt\_flag** - optional flag to control the algorithm:
  - 0 - Use euclidian distance, be verbose, don't allow to count a pair of points twice
  - 1 - Use maximum distance, be verbose, don't allow to count a pair of points twice

- 2 - Use euclidian distance, be verbose, allow to count a pair of points twice
- 3 - Use maximum distance, be verbose, allow to count a pair of points twice
- 4 - Use euclidian distance, be silent, don't allow to count a pair of points twice
- 5 - Use maximum distance, be silent, don't allow to count a pair of points twice
- 6 - Use euclidian distance, be silent, allow to count a pair of points twice
- 7 - Use maximum distance, be silent, allow to count a pair of points twice

If the preprocessing output `atria` is given, the type of metric used to create this overrides the settings by `opt_flag`.

#### Output arguments:

- `c` - vector of correlation sums, `length(c) = bins`
- `d` - vector of the corresponding distances at which the correlation sums (stored in `c`) where computed. `d` is exponentially spaced, `length(c) = bins`
- `e` - vector of the number of pairs found within this range, `length(e) = bins`
- `f` - vector of the number of total pairs that were tested, `length(f) = bins`
- `g` - vector containing the indices of the reference points actually used by the algorithm.

#### Example:

```
x = chaosys(25000, 0.025, [0.1 -0.1 0.02], 0);
x = x(5001:end,:); % discard first 5000 samples due to transient
% now compute correlation sum up to five percent of attractor diameter
[c,d] = corrsum2(x,[1000 100 2000], 0.05, 200);
loglog(d,c) % and show the result as log-log plot
```

## 6.9 fnearneigh - Fast nearest neighbor search

`fnearneigh` is based on the advanced triangle inequality algorithm ATRIA. However, it does not support approximate queries. The functionality of `fnearneigh` is almost the same as that of `nn_search` (cf. Section 6.14), so `fnearneigh` might become obsolete in future versions of TSTOOL.

#### Syntax:

- `[index, distance] = fnearneigh(pointset, query_points, k)`
- `[index, distance] = fnearneigh(pointset, query_indices, k, exclude)`

#### Input arguments:

- `pointset` - a `N` by `D` double matrix containing the coordinates of the point set, organized as `N` points of dimension `D`
- `query_points` - a `R` by `D` double matrix containing the coordinates of the query points, organized as `R` points of dimension `D`
- `query_indices` - query points are taken out of the `pointset`, `query_indices` is a vector of length `R` which contains the indices of the query points (indices may vary from 1 to `N`)
- `k` - number of nearest neighbors to be determined

- **exclude** - in case the query points are taken out of the pointset, exclude specifies a range of indices which are omitted from search. For example if the index of the query point is 124 and exclude is set to 3, points with indices 121 to 127 are omitted from search. Using exclude = 0 means: exclude self-matches

#### Output arguments:

- **index** - a matrix of size R by k which contains the indices of the nearest neighbors. Each row of index contains k indices of the nearest neighbors to the corresponding query point.
- **distance** - a matrix of size R by k which contains the distances of the nearest neighbors to the corresponding query points, sorted in increasing order.

## 6.10 gendimest - Estimate generalized dimension spectrum

The Renyi dimension spectrum of a points set can be estimated using information about the distribution of the interpoint distances. Since we are interested in the scaling behaviour of the Renyi information for small distances, we don't need to compute all interpoint distances, the distances to k nearest neighbors for each reference point are sufficient [150].

Robust estimation is used instead of mean square error fitting.

#### Syntax:

- `[dimensions, moments] = gendimest(dists, gammas, kmin_low, kmin_high, kmax)`

#### Input arguments:

- **dists** - a matrix of size R by k which contains distances from reference points to their k nearest neighbors, sorted in increasing order. This matrix can be obtained by calling `nn_search` (cf. Section 6.14) or `fnearest` (cf. Section 6.9) on the point set whose dimension spectrum is to be investigated.
- **gammas** - vector of the moment orders
- **kmin\_low** - first kmin,  $1 \leq kmin\_low$
- **kmin\_high** - last kmin,  $kmin\_low \leq kmin\_high < kmax$
- **kmax** - highest neighbor order up to which,  $kmax \leq k$

#### Output arguments:

- **dimensions** - matrix of size `length(gammas)` by `kmin_upper-kmin_lower+1`, holding the dimension estimates
- **moments** (optional) - matrix of size k by `length(gammas)`, storing the computed moments of the neighbor distances

#### Example:

```
x = chaosys(25000, 0.025, [0.1 -0.1 0.02], 0); % generate data from Lorenz system
x = x(5001:end,:); % discard first 5000 samples due to transient
[nn, dist] = fnearest(x, randref(1, 20000, 1000), 128, 0);
gammas = -5:0.5:5;
gedims = gendimest(dist, gammas, 8, 8, 128);
plot(1-gammas./gedims', gedims)
xlabel('q');ylabel('D_q');title('Renyi dimension')
```

## 6.11 henon - Generate henon time-series

Generate time series by iterating the henon map.

**Syntax:**

- `x = henon(length, [a b xo yo])`

**Input arguments:**

- `length` - number of samples to generate
- `[a b xo yo]` - vector of parameters and initial conditions

**Output arguments:**

- `x` - vector of size D

**Example:**

```
x = henon(500, [-1.4 0.3 0.2 0.12]);  
plot(x(:,1), x(:,2), 'r');
```

## 6.12 largelyap - Compute separation of nearby trajectories

`largelyap` is an algorithm very similar to the Wolf algorithm [90], it computes the average exponential growth of the distance of neighboring orbits via the prediction error. The increase of the prediction error vs the prediction time allows an estimation of the largest lyapunov exponent.

**Syntax:**

- `x = largelyap(pointset, query_indices, taumax, k exclude)`
- `x = largelyap(atria, pointset, query_indices, taumax, k exclude)`

**Input arguments:**

- `atria` - output of `nn_prepare` for `pointset` (optional) (cf. Section 6.13)
- `pointset` - a N by D double matrix containing the coordinates of the point set, organized as N points of dimension D
- `query_indices` - query points are taken out of the `pointset`, `query_indices` is a vector of length R which contains the indices of the query points (indices may vary from 1 to N)
- `taumax` - maximal time shift
- `k` - number of nearest neighbors to compute
- `exclude` - in case the query points are taken out of the `pointset`, `exclude` specifies a range of indices which are omitted from search. For example if the index of the query point is 124 and `exclude` is set to 3, points with indices 121 to 127 are omitted from search. Using `exclude = 0` means: exclude self-matches

**Output arguments:**

- `x` - vector of length `taumax+1`,  $x(\tau) = 1/N_{\text{ref}} * \sum (\log_2(\text{dist}(\text{reference point} + \tau, \text{nearest neighbor} + \tau) / \text{dist}(\text{reference point}, \text{nearest neighbor})))$   
[146]

## 6.13 nn\_prepare - Do nearest neighbor preprocessing

The intention of this mex-file was to reduce the computational overhead of preprocessing for nearest neighbor or range searching. With `nn_prepare` it is possible to do the preprocessing for a given point set only once and save the created tree structure into a Matlab variable. This Matlab variable, usually called *atria*, can then be used for repeated neighbor searches on the same point set. Most mex-files that rely on nearest neighbor or range search offer the possibility to use this variable *atria* as optional input argument. However, if the underlying point set is altered in any way, the preprocessing has to be repeated for the new point set. If the preprocessing output does not belong to the given point set, wrong results or program termination may occur.

### Syntax:

- `atria = nn_prepare(pointset)`
- `atria = nn_prepare(pointset, metric)`
- `atria = nn_prepare(pointset, metric, clustersize)`

### Input arguments:

- `pointset` - a N by D double matrix containing the coordinates of the point set, organized as N points of dimension D
- `metric` - (optional) either 'euclidian' or 'maximum' (default is 'euclidian')
- `clustersize` - (optional) threshold for clustering algorithm, defaults to 64

### Example:

```
pointset = rand(40000, 3);
atria = nn_prepare(pointset);
[c, d] = corrsum(atria, pointset, 1:17:40000, 0.05, 0);
plot(log(d), log(c))
D = takens_estimator(atria, pointset, 1:17:40000, 0.05, 0)
```

## 6.14 nn\_search

### Syntax:

- `[index, distance] = nn_search(pointset, atria, query_points, k)`
- `[index, distance] = nn_search(pointset, atria, query_points, k, epsilon)`
- `[index, distance] = nn_search(pointset, atria, query_indices, k, exclude)`
- `[index, distance] = nn_search(pointset, atria, query_indices, k, exclude, epsilon)`

### Input arguments:

- `pointset` - a N by D double matrix containing the coordinates of the point set, organized as N points of dimension D
- `atria` - output of (cf. Section 6.13) `nn_prepare` for pointset

- **query\_points** - a R by D double matrix containing the coordinates of the query points, organized as R points of dimension D
- **query\_indices** - query points are taken out of the pointset, **query\_indices** is a vector of length R which contains the indices of the query points (indices may vary from 1 to N)
- **k** - number of nearest neighbors to be determined
- **epsilon** - (optional) relative error for approximate nearest neighbors queries, defaults to 0 (= exact search)
- **exclude** - in case the query points are taken out of the pointset, **exclude** specifies a range of indices which are omitted from search. For example if the index of the query point is 124 and **exclude** is set to 3, points with indices 121 to 127 are omitted from search. Using **exclude** = 0 means: exclude self-matches

#### Output arguments:

- **index** - a matrix of size R by k which contains the indices of the nearest neighbors. Each row of **index** contains k indices of the nearest neighbors to the corresponding query point.
- **distance** - a matrix of size R by k which contains the distances of the nearest neighbors to the corresponding query points, sorted in increasing order.

## 6.15 predict

State space based prediction using nearest neighbors. The algorithm computes one or more nearest neighbors to an initial state vector. The images of the nearest neighbors are used to estimate the image of the initial state vector. The next iteration uses the previously computed image as new initial state vector [145].

#### Syntax:

- `x = predict(pointset, length, k, stepsize, mode)`

#### Input arguments:

- **pointset** - a N by D double matrix containing the coordinates of the point set, organized as N points of dimension D
- **length** - number of iterations (length of prediction)
- **k** - number of nearest neighbors
- **stepsize** - prediction stepsize, usually one
- **mode** - (optional) method to estimate image of initial state vector
  - 0 - direct prediction, no weight is applied to neighbors
  - 1 - direct prediction, biquadratic weight is applied to neighbors
  - 2 - integrated prediction, no weight is applied to neighbors
  - 3 - integrated prediction, biquadratic weight is applied to neighbors

#### Output arguments:

- **x** - data set as double matrix, size **length** by D

## 6.16 range\_search

**Syntax:**

- `[count, neighbors] = range_search(pointset, atria, query_points, r)`
- `[count, neighbors] = range_search(pointset, atria, query_indices, r, exclude)`

**Input arguments:**

- **pointset** - a N by D double matrix containing the coordinates of the point set, organized as N points of dimension D
- **atria** - output of (cf. Section 6.13) `nn_prepare` for pointset
- **query\_points** - a R by D double matrix containing the coordinates of the query points, organized as R points of dimension D
- **query\_indices** - query points are taken out of the pointset, **query\_indices** is a vector of length R which contains the indices of the query points
- **r** - range or search radius ( $r > 0$ )
- **exclude** - in case the query points are taken out of the pointset, **exclude** specifies a range of indices which are omitted from search. For example if the index of the query point is 124 and **exclude** is set to 3, points with indices 121 to 127 are omitted from search. Using **exclude** = 0 means: exclude self-matches

**Output arguments:**

- **count** - a vector of length R contains the number of points within distance r to the corresponding query point
- **neighbors** - a Matlab cell structure of size R by 2 which contains vectors of indices and vectors of distances to the neighbors for each given query point. This output argument can not be stored in a standard Matlab matrix because the number of neighbors within distance r is not the same for all query points. The vectors of indices and distances for one query point have exactly the length that is given in **count**. The values in the distances vectors are **not** sorted..

## 6.17 return\_time

**return\_time** may be used to find hidden periodicity in multivariate data, e.g. embedded time series data. It computes a histogram of *return times*. For any given reference point, **return\_time** calculates the time span until the time series returns to that location in phase space (by means of nearest neighbors). A histogram of these time spans is computed. Strong peaks in this histogram might be a sign of periodicity in the data.

**Syntax:**

- `r = return_time(pointset, query_indices, k, max_time, exclude)`
- `r = return_time(atria, pointset, query_indices, k, max_time, exclude)`

**Input arguments:**

- **atria** - output of **nn\_prepare** for pointset (optional) (cf. Section 6.13)
- **pointset** - a  $N$  by  $D$  double matrix containing the coordinates of the point set, organized as  $N$  points of dimension  $D$
- **query\_indices** - query points are taken out of the pointset, **query\_indices** is a vector of length  $R$  which contains the indices of the query points (indices may vary from 1 to  $N$ )
- **k** - number of nearest neighbors to be determined
- **max\_time** - integer scalar, gives an upper limit for return times that should be considered.
- **exclude** - in case the query points are taken out of the pointset, **exclude** specifies a range of indices which are omitted from search. For example if the index of the query point is 124 and **exclude** is set to 3, points with indices 121 to 127 are omitted from search. Using **exclude** = 0 means: exclude self-matches

#### Output arguments:

- **r** - vector of length **max\_time**, containing the histogram of return times

## 6.18 takens\_estimator

#### Syntax:

- `D = takens_estimator(pointset, query_indices, relative_range, exclude)`
- `D = takens_estimator(atria, pointset, query_indices, relative_range, exclude)`

#### Input arguments:

- **atria** - output of **nn\_prepare** for pointset (optional) (cf. Section 6.13)
- **pointset** - a  $N$  by  $D$  double matrix containing the coordinates of the point set, organized as  $N$  points of dimension  $D$
- **query\_indices** - query points are taken out of the pointset, **query\_indices** is a vector of length  $R$  which contains the indices of the query points (indices may vary from 1 to  $N$ )
- **relative\_range** - search radius, relative to attractor diameter ( $0 < \text{relative\_range} < 1$ )
- **exclude** - in case the query points are taken out of the pointset, **exclude** specifies a range of indices which are omitted from search. For examples if the index of the query point is 124 and **exclude** is set to 3, points with indices 121 to 127 are omitted from search. Using **exclude** = 0 means: exclude self-matches

#### Output arguments:

- **D** - scalar value, estimation of correlation dimension



## 6.19 tentmap - Generate tentmap time-series

Generate samples of the generalized iterated tentmap.

**Syntax:**

- `x = tentmap(length, [h e s x0])`

**Input arguments:**

- `length` - number of samples to generate
- `[h e s x0]` - vector of parameters and initial conditions

**Output arguments:**

- `x` - time series

**Example:**

```
x = tentmap(500, [0 1 0.97 rand(1,1)]);  
plot(x)  
plot(x(1:end-1), x(2:end), '.'
```



## 6.20 Class signal

### 6.20.1 Overview

Class `signal` is TSTOOL's main class. Objects of this type model real world signals. A signal does not only store the pure sample values, it holds much more information like axes, units of sample values or the axes units, and even more descriptive information like labels, command lines and a processing history.

The majority of functions in the `tstoolbox` take a signal as input argument and return a processed signal as output. This allows for combining or *chaining* of several processing steps in order to get the desired output.

### 6.20.2 Attributes

- `xaxes` *cellarray* of at least one object of type *achse*
- `core` object of type *core* (cf. Section 6.22)
- `description` object of type *description* (cf. Section 6.21)

### 6.20.3 Member functions

#### 6.20.3.1 `abs`

**Syntax:**

- `abs(s)`

Take absolut value of all data values of signal `s`. If sample values are complex, `abs(s)` returns the complex modulus (magnitude) of each sample.

#### 6.20.3.2 `acf`

**Syntax:**

- `acf(s, len)`

**Input arguments:**

- `len` -length of the fft (*optional*)

Autocorrelation function for real scalar signals, using fft (of length `len`). If `len` is omitted a default value is calculated. The maximum of the calculated length is 128.

#### 6.20.3.3 `acp`

**Syntax:**

- `acp(s, tau, past, maxdelay, maxdim, nref)`

**Input arguments:**

- **tau** - proper delay time for **s**
- **past** - number of samples to exclude before and after reference index (to avoid correlation effects)
- **maxdelay** - maximal delay (should be much smaller than the length of **s**) (*optional*)
- **maxdim** - maximal dimension to use (*optional*)
- **nref** - number of reference points (*optional*)

Auto crossprediction function for real scalar signals for increasing dimension. The default value for **maxdelay** is 25% of the input signal's length. The default for **maxdim** is 8 and for **nref** it is 10% of the input signal's length.

#### 6.20.3.4 amutual

**Syntax:**

- **amutual(s, maxtau, bins)**

**Input arguments:**

- **maxtau** - maximal delay (should be much smaller than the length of **s**) (*optional*)
- **bins** - number of bins used for histogram calculation (*optional*)

Auto mutual information function for real scalar signals, can be used to determine a proper delay time for time-delay reconstruction. The default value for **maxtau** is 25% of the input signal's length. The default number of bins is 128.

$$I = \sum P(A, B) \log_2 \frac{P(A, B)}{P(A)P(B)}$$

#### 6.20.3.5 amutual2

**Syntax:**

- **amutual2(s, len)**

**Input arguments:**

- **len** - maximal lag

Auto mutual information (average) function for real scalar signals using 128 equidistant partitions.

#### 6.20.3.6 analyze

**Syntax:**

- **analyze(s, maxdim)**

**Input arguments:**

- **maxdim** - analyze will not use a dimension higher than this limit

Try to do a automatic analysis procedure of a time series. The time series is embedded using the first zero of the auto mutual information function for the delay time.

### 6.20.3.7 arch

**Syntax:**

- `[rs, archetypes]=arch(s, na, mode='normalized')`

**Input arguments:**

- `na` - number of generated archetypes
- `mode` - mode can be one of the following : `'normalized'` , `'mean'`, `'raw'` (*optional*)

Archetypal analysis of column orientated data set:

- each row of data is one 'observation', e.g. the sample values of all channels in a multichannel measurement at one point in time
- in mode `'normalized'` each column of data is centered by removing its mean and then normalized by dividing through its standard deviation before the covariance matrix is calculated
- in mode `'mean'` only the mean of every column of data is removed
- in mode `'raw'` no preprocessing is applied to data

Default value for mode is `'normalized'`.

### 6.20.3.8 boxdim

**Syntax:**

- `rs = boxdim(s, bins)`

**Input arguments:**

- `s` - data points (row vectors)
- `bins` - maximal number of partition per axis (*optional*)

Compute the boxcounting (capacity) dimension of a time-delay reconstructed timeseries `s` for dimensions from 1 to D, where D is the dimension of the input vectors using boxcounting approach. The default number of bins is 100.

### 6.20.3.9 cao

**Syntax:**

- `[E1, E2] = cao(s, maxdim, tau, NNR, Nref)`

**Input arguments:**

- `s` - scalar input signal
- `maxdim` - maximal dimension
- `tau` - delay time
- `NNR` - number of nearest neighbor to use
- `Nref` - number of reference points (-1 means: use all points)

Estimate minimum embedding dimension using Cao's method.

The second output argument, `E2`, can be used to distinguish between deterministic and random data.

#### 6.20.3.10 center

##### Syntax:

- `center(s)`

Center signal by removing it's mean.

#### 6.20.3.11 corrdim

##### Syntax:

- `rs = corrdim(s, bins)`

##### Input arguments:

- `s` - data points (row vectors)
- `bins` - maximal number of partition per axis (*optional*)

Compute the correlation dimension of a time-delay reconstructed timeseries `s` for dimensions from 1 to `D`, where `D` is the dimension of the input vectors using boxcounting approach. The default number of `bins` is 100.

#### 6.20.3.12 corrsum

##### Syntax:

- `rs = corrsum(s, n, range, past, bins)`

##### Input arguments:

- `n` - number of randomly chosen reference points (`n == -1` means: use all points)
- `range` - maximal relative search radius (relative to attractor size) 0..1
- `past` - number of samples to exclude before and after each reference index
- `bins` - number of bins (*optional*)

Compute scaling of correlation sum for time-delay reconstructed timeseries `s` (Grassberger-Proccacia Algorithm), using fast nearest neighbor search. Default number of `bins` is 20.

#### 6.20.3.13 corrsum2

##### Syntax:

- `rs = corrsum2(s, npairs, range, past, bins)`

##### Input arguments:

- `npairs` - number of pairs per bins
- `range` - maximal relative search radius (relative to attractor size) 0..1
- `past` - number of samples to exclude before and after each reference index
- `bins` - number of bins (*optional*), defaults to 32

Compute scaling of correlation sum for time-delay reconstructed timeseries `s` (Grassberger-Proccacia Algorithm), using fast nearest neighbor search.

### 6.20.3.14 crosscorrdim

#### Syntax:

- `rs = crosscorrdim(s, s2, n, range, past, bins)`

#### Input arguments:

- `n` - number of randomly chosen reference points (`n == -1` means : use all points)
- `range` - maximal relative search radius (relative to size of data set `s2`) 0..1
- `past` - number of samples to exclude before and after each reference index
- `bins` - number of bins (*optional*)

Compute scaling of cross-correlation sum for time-delay reconstructed timeseries `s` against signal `s2` (with same dimension as `s`), using fast nearest neighbor search. Reference points are taken out of signal `s`, while neighbors are searched in `s2`. The default number of `bins` is 32.

### 6.20.3.15 cut

#### Syntax:

- `rs = cut(s, dim, start, stop)`

#### Input arguments:

- `dim` - dimension along which the signal is cutted
- `start` - position where to start the cut
- `stop` - position where to stop (*optional*)

Cut a part of the signal. If `stop` is omitted only the data at `start` is cutted.

### 6.20.3.16 db

#### Syntax:

- `db(s, dbmin)`

Compute decibel values of signal relative to a reference value that is determined by the signal's `yunit` values below `dbmin` are set to `dbmin`. If `dbmin` is omitted it is set to -120.

### 6.20.3.17 delaytime

#### Syntax:

- `tau = delaytime(s, maxdelay, past)`

#### Input arguments:

- `maxdelay` - maximal delay time
- `past` - ?

Compute optimal delaytime for a scalar timeseries with method of Parlitz and Wichard.

### 6.20.3.18 diff

#### Syntax:

- `diff(s, nth)`

Compute the `nth` numerical derivative along dimension 1. `s` has be to sampled equidistantly.

### 6.20.3.19 dimensions

#### Syntax:

- `[bc,in,co] = dimensions(s, bins)`

#### Input arguments:

- `s` - data points (row vectors)
- `bins` - maximal number of partition per axis, default is 100

#### Output arguments:

- `bc` - scaling of boxes with partititon sizes ( $\log_2 - \log_2$ )
- `in` - scaling of information with partititon sizes ( $\log_2 - \log_2$ )
- `co` - scaling of correlation with partititon sizes ( $\log_2 - \log_2$ )

Compute boxcounting, information and correlation dimension of a time-delay reconstructed timeseries `s` for dimensions from 1 to `D`, where `D` is the dimension of the input vectors using boxcounting approach.

Scale data to be within 0 and 1. Give a sortiment of (integer) partitionsizes with almost exponential behaviour.

### 6.20.3.20 display

### 6.20.3.21 embed

#### Syntax:

- `emb = embed(s, dim, delay, shift, windowtype)`

#### Input arguments:

- `dim` - embedding dimension
- `delay` - time delay (*optional*)
- `shift` - shift for two sequent time delay vectors (*optional*)
- `windowtype` - type of window (*optional*)

#### Output arguments:

- `emb` - `n` by `dim` array, each row contains the coordinates of one point

Embeds signal `s` with embedding dimension `dim` and delay `delay` (in samples). `s` must be a scalar time series. The default values for `dim` and `delay` are equal to one. The default value for `windowtype` is 'Rect', which is currently the only possible value.



#### 6.20.3.22 `fft`

**Syntax:**

- `f = fft(s)`

**Output arguments:**

- `f` - `n` by 2 array, the first column contains the magnitudes, the second one the phases.

Fourier transform of scalar signal `s`.

#### 6.20.3.23 `filterbank`

**Syntax:**

- `filterbank(s, depth, filterlen)`

Filter scalar signal `s` into  $2^{\text{depth}}$  bands of equal bandwidth, using maximally flat filters.

#### 6.20.3.24 `firstmax`

**Syntax:**

- `[xpos, unit] = firstmax(s)`

Give information about first local maximum of scalar signal `s`.

#### 6.20.3.25 `firstmin`

**Syntax:**

- `[xpos, unit] = firstmin(s)`

Give information about first local minimum of scalar signal `s`.

#### 6.20.3.26 `firstzero`

**Syntax:**

- `[xpos, unit] = firstzero(s)`

Give information about first zero of scalar signal `s`, using linear interpolation.

### 6.20.3.27 `fracdims`

#### Syntax:

- `rs = fracdims(s, kmin, kmax, Nref, gstart, gend, past, steps)`
- `rs = fracdims(s, kmin, kmax, Nref, gstart, gend, past)`
- `rs = fracdims(s, kmin, kmax, Nref, gstart, gend)`

#### Input arguments:

- `kmin` - minimal number of neighbors for each reference point
- `kmax` - maximal number of neighbors for each reference point
- `Nref` - number of randomly chosen reference points (`n == -1` means : use all points)
- `gstart` - starting value for moments
- `gend` - end value for moments
- `past` - (optional) number of samples to exclude before and after each reference index, default is 0
- `steps` - (optional) number of moments to calculate, default is 32

Compute fractal dimension spectrum  $D(q)$  using moments of neighbor distances for time-delay reconstructed timeseries `s`.

Do the main job - computing nearest neighbors for reference points.

### 6.20.3.28 `getaxis`

#### Syntax:

- `a = getaxis(s, dim)`

Get one of the current `axes`.

### 6.20.3.29 `gmi`

#### Syntax:

- `gmi(s, D, eps, NNR, len, Nref)`

#### Input arguments:

- `D` -
- `eps` -
- `NNR` -
- `len` -
- `Nref` -

Generalized mutual information function for a scalar time series

### 6.20.3.30 histo

#### Syntax:

- `histo(s, partitions)`

Histogram function using equidistantly spaced partitions.

### 6.20.3.31 infodim

#### Syntax:

- `rs = infodim(s, bins)`

#### Input arguments:

- `s` - data points (row vectors)
- `bins` - maximal number of partition per axis, default is 100

Compute the information dimension of a time-delay reconstructed timeseries `s` for dimensions from 1 to `D`, where `D` is the dimension of the input vectors. Using boxcounting approach. Scale data to be within 0 and 1. Give a sortiment of (integer) partitionsizes with almost exponential behaviour.

### 6.20.3.32 infodim2

#### Syntax:

- `rs = infodim2(s, n, kmax, past)`

#### Input arguments:

- `n` - number of randomly chosen reference points (`n == -1` means : use all points)
- `kmax` - maximal number of neighbors for each reference point
- `past` - number of samples to exclude before and after each reference index

Compute scaling of moments of the nearest neighbor distances for time-delay reconstructed timeseries `s`. This can be used to calculate information dimension `D1`.

Numerically compute first derivative of  $\log \gamma(k)$  after  $k$ .

### 6.20.3.33 int

#### Syntax:

- `int(s)`

Numerical integration along dimension 1 signal `s` has to be sampled equidistantly.

#### 6.20.3.34 intspikeint

##### Syntax:

- `rs = intspikeint(s)`

Compute the interspike intervalls for a spiked scalar timeseries, using transformation on ranked values.

#### 6.20.3.35 intspikint

##### Syntax:

- `rs = intspikeint(s)`

Compute the interspike intervalls for a spiked scalar timeseries, using transformation on ranked values.

#### 6.20.3.36 largelyap

##### Syntax:

- `rs = largelyap(s, n, stepsahead, past, nnr)`

##### Input arguments:

- `n` - number of randomly chosen reference points (-1 means: use all points)
- `stepsahead` - maximal length of prediction in samples
- `past` - exclude
- `nnr` - number of nearest neighbours (*optional*)

##### Output arguments:

- `rs` -

Compute the largest lyapunov exponent of a time-delay reconstructed timeseries `s`, using formula (1.5. of Nonlinear Time-Series Analysis, Ulrich Parlitz 1998 [146]).

#### 6.20.3.37 level\_adaption

##### Syntax:

- `level_adaption(s, timeconstants, dynamic_limit, threshold)`

Each channel of signal `s` is independently divided by a scaling factor that adapts to the current level of the samples in this channel. The adaption process is simulated using a cascade of feedback loops (Püschel 1998) which consists of low pass filters with time constants given as second argument to this function. The number of time constants given determines the number of feedback loops that are used.

Higher values for time constants will result in slower adaption speed. Short time changes in the signal will be transmitted almost linearly. In each feedback loop, a nonlinear compressing characteristic (see Stefan Münkner 1993) limits the signal values to be within `[-dynamic_limit dynamic_limit]`. A low value for `dynamic_limit` will introduce nonlinear distortions to the signal.

To prevent the feedback loops from adapting to a zero level (in case all input values are zero), a tiny threshold is given as 4th argument. The scaling factors will not shrink below this threshold.

### 6.20.3.38 localdensity

#### Syntax:

- `rs = localdensity(s, n, past)`

#### Input arguments:

- `n` - number of nearest neighbour to compute
- `past` - a nearest neighbour is only valid if it is at least `past` timesteps away from the reference point `past = 1` means: use all points but `ref_point` itself

Uses accelerated searching, distances are calculated with euclidian norm.

### 6.20.3.39 max

#### Syntax:

- `[maximum, yunit, xpos, xunit] = max(s)`

Give information about maximum of scalar signal `s`.

#### Example:

```
disp('maximum of signal : ')
disp(['y = ' num2str(m) ' ' label(yunit(s))]);
disp(['x = ' num2str(xpos) ' ' label(a)]);
```

### 6.20.3.40 medianfilt

#### Syntax:

- `rs = medianfilt(s, len)`

Moving median filter of width `len` samples for a scalar time series (`len` should be odd).

### 6.20.3.41 merge

#### Syntax:

- `merge(signal1, signal2, dB)`
- `merge(signal1, signal2)`

#### Input arguments:

- `signal1, signal2` - Signals
- `dB` - energy ratio, (optional, default = 0)

Merges signal `s1` and `s2` into a new signal with energy ration `dB` (in decibel) a positive value of `dB` increases the amount of `signal1` in the resulting signal.

#### 6.20.3.42 min

##### Syntax:

- `[minimum, yunit, xpos, xunit] = min(s)`

Give information about minimum of scalar signal **s**.

##### Example:

```
disp('minimum of signal : ')
disp(['y = ' num2str(m) ' ' label(yunit(s))]);
disp(['x = ' num2str(xpos) ' ' label(a)]);
```

#### 6.20.3.43 minus

##### Syntax:

- `rs=minus(s, offset)`
- `rs=minus(s1,s2)`

##### Input arguments:

- **s**, **s1**, **s2** - signal object
- **offset** - scalar value

Calculate difference of signals **s1** and **s2** or subtract a scalar value from **s**.

#### 6.20.3.44 movav

##### Syntax:

- `rs = movav(s, len, windowtype)`
- `rs = movav(s, len)`

Moving average of width **len** (samples) along first dimension.

#### 6.20.3.45 multires

##### Syntax:

- `rs = multires(s) => scale=3`
- `rs = multires(s, scale)`

Multires perform multiresolution analysis. **Y = MULTIRES (X,H,RH,G,RG,SC)** obtains the SC successive details and the low frequency approximation of signal in **X** from a multiresolution scheme. The analysis lowpass filter **H**, synthesis lowpass filter **RH**, analysis highpass filter **G** and synthesis highpass filter **RG** are used to implement the scheme.

Results are given in a **scale+1** channels. The first scale channels are the details corresponding to the scales  $2^1$  to  $2^{\text{scale}}$  the last row contains the approximation at scale  $2^{\text{SC}}$ . The original signal can be restored by summing all the channels of the resulting signal.

#### 6.20.3.46 nearneigh

##### Syntax:

- `rs = nearneigh(s, n) => past=1`
- `rs = nearneigh(s, n, past)`

##### Input arguments:

- `n` - number of nearest neighbour to compute
- `past` - a nearest neighbour is only valid if it is at least `past` timesteps away from the reference point. `past = 1` means: use all points but `ref_point` itself

`n` nearest neighbour algorithm. Find `n` nearest neighbours (in order of increasing distances) to each point in signal `s` uses accelerated searching, distances are calculated with euclidian norm.

#### 6.20.3.47 norm1

##### Syntax:

- `rs=norm1(s) => low=0 , upp=1`
- `rs=norm1(s, low) => upp=1`
- `rs=norm1(s, low, upp)`

Scale and move signal values to be within `[low,upp]`.

#### 6.20.3.48 norm2

##### Syntax:

- `rs=norm2(s)`

Normalize signal by removing it's mean and dividing by the standard deviation.

#### 6.20.3.49 pca

##### Syntax:

- `[rs, eigvals, eigvecs] = pca(s) => mode='normalized' , maxpercent = 95`
- `[rs, eigvals, eigvecs] = pca(s, mode) => maxpercent = 95`
- `[rs, eigvals, eigvecs] = pca(s, mode, maxpercent)`

##### Input arguments:

- each row of data is one 'observation', e.g. the sample values of all channels in a multichannel measurement at one point in time
- `mode` can be one of the following : 'normalized' (default), 'mean', 'raw'

- in mode '**normalized**' each column of data is centered by removing its mean and then normalized by dividing through its standard deviation before the covariance matrix is calculated
- in mode '**mean**' only the mean of every column of data is removed
- in mode '**raw**' no preprocessing is applied to data
- **maxpercent** gives the limit of the accumulated percentage of the resulting eigenvalues, default is 95 %

Principal component analysis of column orientated data set.

#### 6.20.3.50 plosivity

**Syntax:**

- `rs = plosivity(s, blen) => flen=1 , thresh=0, windowtype = 'Rect'`
- `rs = plosivity(s, blen, flen) => thresh=0, windowtype = 'Rect'`
- `rs = plosivity(s, blen, flen, thresh) => windowtype = 'Rect'`
- `rs = plosivity(s, blen, flen, thresh, windowtype)`

Compute plosivity of a spectrogram. See also: `window` for list of possible window types.

#### 6.20.3.51 plus

**Syntax:**

- `rs=plus(s, offset)`
- `rs=plus(s1, s2)`

Add two signals `s1` and `s2` or add a scalar value `offset` to `s`.

#### 6.20.3.52 poincare

**Syntax:**

- `rs=poincare(s, ref)`

Compute Poincare-section of an embedded time series the result is a set of vector points with dimension `DIM-1`, when the input data set of vectors had dimension `DIM`. The projection is done orthogonal to the tangential vector at the vector with index.

#### 6.20.3.53 power

**Syntax:**

- `power(s)`

Calculate squared magnitude of each sample.



### 6.20.3.54 predict

#### Syntax:

- `rs = predict(s, dim, delay, len) => nnr=1`
- `rs = predict(s, dim, delay, len, nnr) => mode=0`
- `rs = predict(s, dim, delay, len, nnr, mode)`

#### Input arguments:

- `dim` - dimension for time-delay reconstruction
- `delay` - delay time (in samples) for time-delay reconstruction
- `len` - length of prediction (number of output values)
- `nnr` - number of nearest neighbors to use (default is one)
- `step` - stepsize (in samples) (default is one)
- `mode`:
  - 0 = Output vectors are the mean of the images of the nearest neighbors
  - 1 = Output vectors are the distance weighted mean of the images of the nearest neighbors
  - 2 = Output vectors are calculated based on the local flow using the mean of the images of the neighbors
  - 3 = Output vectors are calculated based on the local flow using the weighted mean of the images of the neighbors

Local constant iterative prediction for scalar data, using fast nearest neighbor search. Four methods of computing the prediction output are possible.

### 6.20.3.55 predict2

#### Syntax:

- `rs = predict2(s, len, nnr, step, mode)`

#### Input arguments:

- `len` - length of prediction (number of output values)
- `nnr` - number of nearest neighbors to use (default is one)
- `step` - stepsize (in samples) (default is one)
- `mode`:
  - 0 = Output vectors are the mean of the images of the nearest neighbors
  - 1 = Output vectors are the distance weighted mean of the images of the nearest neighbors
  - 2 = Output vectors are calculated based on the local flow using the mean of the images of the neighbors
  - 3 = Output vectors are calculated based on the local flow using the weighted mean of the images of the neighbors

Local constant iterative prediction for phase space data (e.g. data stemming from a time delay reconstruction of a scalar time series), using fast nearest neighbor search. Four methods of computing the prediction output are possible.

#### 6.20.3.56 rang

**Syntax:**

- `rs = rang(s)`

Transform scalar time series to rang values.

#### 6.20.3.57 removeaxis

**Syntax:**

- `s = removeaxis(s, dim)`

Remove axis one of the current **xaxes**. No bound checking for **dim**.

#### 6.20.3.58 return\_time

**Syntax:**

- `rs = return_time(s, nnr, maxT) => past=1`
- `rs = return_time(s, nnr, maxT, past)`
- `rs = return_time(s, nnr, maxT, past, N)`

**Input arguments:**

- **nnr** - number of nearest neighbors
- **maxT** - maximal return time to consider
- **past** - a nearest neighbor is only valid if it is as least past timesteps away from the reference point **past = 1** means: use all points but **tt ref\_point** itself
- **N** - number of reference indices

Compute histogram of return times.

#### 6.20.3.59 reverse

**Syntax:**

- `rs=reverse(s)`

Reverse signal along dimension 1.

#### 6.20.3.60 rms

**Syntax:**

- `rs = rms(s)`

Calculate root mean square value for signal along dimension 1.

### 6.20.3.61 scale

#### Syntax:

- `scale(signal, factor)`

Scale signal by factor `f`.

### 6.20.3.62 scalogram

#### Syntax:

- `rs = scalogram(s) => scalemin=0.1`
- `rs = scalogram(s, scalemin) => scalemax=1`
- `rs = scalogram(s, scalemin, scalemax) => scalestep=0.1`
- `rs = scalogram(s, scalemin, scalemax, scalestep) => mlen=10`
- `rs = scalogram(s, scalemin, scalemax, scalestep, mlen)`

Scalogram of signal `s` using morlet wavelet. See also: `spec2`.

### 6.20.3.63 setaxis

#### Syntax:

- `s = setaxis(s, dim, achse)`

Change one of the current `xaxes`.

### 6.20.3.64 setunit

#### Syntax:

- `s = setunir(s, dim, u)`

Change unit of one of the current `xaxes`.

### 6.20.3.65 shift

#### Syntax:

- `s = shift(s, distance) (dim=1)`
- `s = shift(s, distance, dim)`

shift signal on axis No. `dim` by distance (measured in the unit of the axis) to the right

### 6.20.3.66 signal

#### Syntax:

- `s = signal(array)`  
creates a new signal object from a data array `array` the data inside the object can be retrieved with `x = data(s)`;
- `s = signal(array, achse1, achse2, ...)`  
creates a new signal object from a data array `array`, using `achse1` etc. as xachse entries
- `s = signal(array, unit1, unit2, ...)`  
creates a new signal object from a data array `array`, using `unit1` etc. to create xachse objects
- `s = signal(array, samplerate1, samplerate2, ...)`  
creates a new signal object from a data array `array`, using as xunit `'s'` (second) and scalar `samplerate1` as `samplerate(s)`

A signal object contains signal data, that is a collection of real or complex valued samples. A signal can be one or multi-dimensional. The number of dimensions is the number of axes that are needed to describe the data.

An example for an one-dimensional signal is a one-channel measurement (timeseries), or the power spectrum of a one-channel measurement. An example for a two-dimensional signal is a twelve-channel measurement, with one time axis and a 'channel' axis. Another example for a two-dimensional signal is a short time spectrogram of a time series, where we have a time axis and a frequency axis.

Each axis can have a physical unit(e.g. `'s'` or `'Hz'`), a starting point and a step value. E.g. if a time-series is sampled with 1000 Hz, beginning at 1 min 12 sec, the unit is `'s'`, the starting point is 72 and the step value (delta) is 0.001.

But not only the axes have physical units, also the sample value themselves can have a unit, maybe `'V'` or `'Pa'`, depending on what the sampled data represent (=, `yunit`)

All units are stored as objects of class `'unit'`, all axes are stored as objects of class `'achse'` (this somewhat peculiar name was chosen because of conflicts with reserved matlab keywords `'axis'` and `'axes'`, which otherwise would have been the first choice).

Example for creating a 2-dimensional signal with y-unit set to `'Volt'`, the first dimension's unit is `'second'` (time), the second dimension's unit is `'n'` (Channels).

#### Examples:

- ```
tmp = rand(100, 10);

s = signal(tmp, unit('s'), unit('n'));
s = setyunit(s, unit('V'));
s = addcomment(s, 'Example signal with two dimensions')
```
- Loading from disk  

```
s = signal(filename)
```

  
loads a previously stored signal object
- Importing from other file formats:  

```
ASCII: s = signal('data/spalte1.dat', 'ASCII')
WAVE: s = signal('data/Sounds/hat.wav', 'WAVE')
AU (SUN AUDIO): s = signal('data/Sounds/hat.au', 'AU')
(old) NLD-Format : s = signal('test.nld', 'NLD')
```

### 6.20.3.67 spacing

- `v = spacing(s)` (dim=1)
- `v = spacing(s, dim)`

return spacing values for xaxis nr. dim

### 6.20.3.68 spec

**Syntax:**

- `rs = spec(s)`

compute power spectrum for real valued scalar signals. Multivariate signals are accepted but may produce unwanted results as only the spectrum of the first column is returned.

### 6.20.3.69 spec2

**Syntax:**

- `rs = spec2(s)`

**Input Arguments:**

- `fensterlen` - size of window (*optional*)
- `fenster` - window type (*optional*)
- `vorschub` - shift in samples (*optional*)

spectrogram of signal s using short time fft

**Examples:**

```
view(spec2(sine(10000, 1000, 8000), 512, 'Hanning'))
```

### 6.20.3.70 stts

**Syntax:**

- `rs = stts(s, I)` (J=0, K=1, L=1)
- `rs = stts(s, I, J)` (K=1, L=1)
- `rs = stts(s, I, J, K)` (L=1)
- `rs = stts(s, I, J, K, L)`

**Input Arguments:**

- `s` - input data set of N snapshots of length M, given as N by M matrix
- `I` - number of spatial neighbours
- `J` - number of temporal neighbours (in the past)
- `K` - spatial shift (= spatial delay)
- `L` - temporal delay

Spatiotemporal prediction conforming to U. Parlitz, NONLINEAR TIME-SERIES ANALYSIS Chapter 1.10.2.1.

#### 6.20.3.71 sttseerror

##### Syntax:

- `rs = sttseerror(s1, s2)`

##### Input Arguments:

- `s1` - original signal
- `s2` - predicted signal

compute error function for prediction of spatial-temporal systems  
see U. Parlitz "Nonlinear Time Series Analysis", Section 1.10.2.2 Eq. 1.10

#### 6.20.3.72 surrogate1

##### Syntax:

- `rs = surrogate1(s)`

create surrogate data for a scalar time series by randomizing phases of fourier spectrum  
see : James Theiler et al.'Using Surrogate Data to Detect Nonlinearity in Time Series', APPENDIX : ALGORITHM I

#### 6.20.3.73 surrogate2

##### Syntax:

- `rs = surrogate2(s)`

create surrogate data for a scalar time series  
see : James Theiler et al.'Using Surrogate Data to Detect Nonlinearity in Time Series', APPENDIX : ALGORITHM II

#### 6.20.3.74 surrogate3

##### Syntax:

- `rs = surrogate3(s)`

create surrogate data for a scalar time series by permuting samples randomly

#### 6.20.3.75 surrogate\_test

##### Syntax:

- `rs=surrogate_test(s, ntests, method,func)`

##### Input Arguments:

- `s` - has to be a real, scalar signal

- **ntests** - is the number of surrogate data sets to create
- **method** - method to generate surrogate data sets:
  - 1: surrogate1
  - 2: surrogate2
  - 3: surrogate3
- **func** - string with matlab-code, have to return a signal object with a scalar time series. The data to process is a signal object referred by the qualifier **s** (see example).

#### Output Arguments:

- **rs** is a signal object with a three dimensional time series. The first component is the result of the **func** function applied to the original data set **s**. The second component is the mean of the result of the **func** function applied to the **ntests** surrogate data sets. The third component is the standard deviation. There is a special **plothint** ('surrerrorbar') for the **view** function to show this result in the common way.

**surrogate\_test** runs an automatic surrogate data test task. It generates **ntests** surrogate data sets and performs the **func** function to each set. **func** is a string with matlab-code who returns a signal **s** with a scalar time series.

#### Example:

```
st = surrogate_test(s, 10, 1, 1, 'largelyap(embed(s,3,1,1), 128,20,10);');
```

#### 6.20.3.76 swap

##### Syntax:

- **rs = swap(s)** (exchange dimension 1 and dimension 2)
- **rs = swap(s, dim1, dim2)**

Exchange signal's dimensions (and axes)

#### 6.20.3.77 takens\_estimator

##### Syntax:

- **D2 = takens\_estimator2(s, n, range, past)**

##### Input Arguments:

- **n** - number of randomly chosen reference points (**n == -1** means : use all points)
- **range** - maximal relative search radius (relative to attractor size) 0..1
- **past** - number of samples to exclude before and after each reference index

Takens estimator for correlation dimension

### 6.20.3.78 tc3

#### Syntax:

- `rs = tc3(s,tau,n,method)`

#### Input Arguments:

- `tau` - see explanation below
- `n` - number of surrogate data sets to generate
- `method` - method to generate the surrogate data sets:
  - 1: `surrogate1`
  - 2: `surrogate2`
  - 3: `surrogate3`

#### Output Arguments:

- `rs` is a row vector, returned as signal object. The first item is the  $T_{C3}$  value for the original data set `s`. The following `n` values are the  $T_{C3}$  values for the generated surrogates. There exist a special `plothint` ('surrbar') for the `view` function to show this kind of result in the common way.

This function calculates a special value for the original data set and the `n` generated surrogate data sets. The  $T_{C3}$  value is defined as followed:

$$T_{C3}(\{x_n\}, \tau) = \frac{\langle x_n x_{n-\tau} x_{n-2\tau} \rangle}{|\langle x_n x_{n-\tau} \rangle|^{\frac{3}{2}}}$$

In terms of surrogate data test this is a test statistics for higher order moments. The original `tc3` function is located under `utils/tc3.m` and use simple matlab vectors.

### 6.20.3.79 trend

#### Syntax:

- `rs = trend(s, len)`

trend correction

calculate moving average of width `len` (samples) for a scalar time series (`len` should be odd) and remove the result from the input signal

### 6.20.3.80 trev

#### Syntax:

- `rs = trev(s,tau,n,method)`

#### Input Arguments:

- `tau` - see explanation below
- `n` - number of surrogate data sets to generate



- **method** - method to generate the surrogate data sets:

- 1: `surrogate1`
- 2: `surrogate2`
- 3: `surrogate3`

#### Output Arguments:

- **rs** is a row vector, returned as signal object. The first item is the  $T_{REV}$  value for the original data set **s**. The following **n** values are the  $T_{REV}$  values for the generated surrogates. There exist a special `plothint` ('surrbar') for the `view` function to show this kind of result in the common way.

This function calculates a special value for the original data set and the **n** generated surrogate data sets. The  $T_{REV}$  value is defined as followed:

$$T_{REV}(\{x_n\}, \tau) = \frac{\langle (x_n - x_{n-\tau})^3 \rangle}{\langle (x_n - x_{n-\tau})^2 \rangle^{\frac{3}{2}}}$$

In terms of surrogate data test this is a test statistics for time reversibility. The original `trev` function is located under `utils/trev.m` and use simple matlab vectors.

#### 6.20.3.81 upsample

##### Syntax:

- `rs = upsample(s, factor, method)`

##### Input Arguments:

- **method** may be one of the following :
  - 'fft'
  - 'spline'
  - 'akima'
  - 'nearest'
  - 'linear'
  - 'cubic'
- **s** has be to sampled equidistantly for fft interpolation

Change sample rate of signal **s** by one-dimensional interpolation

#### 6.20.3.82 view

##### Syntax:

- `view(signal)` (fontsize=12)
- `view(signal, fontsize)`
- `view(signal, fontsize, figurehandle)`

Signal viewer that decides from the signal's attributes which kind of plot to produce, using the signal's `plothint` entry to get a hint which kind of plot to produce  
Possible plothints are:

- `'graph'`
- `'bar'`
- `'surrbar'`
- `'surrerrorbar'`
- `'points'`
- `'xyplot'`
- `'xypoints'`
- `'scatter'`
- `'3dcurve'`
- `'3dpoints'`
- `'spectrogram'`
- `'image'`
- `'multigraph'`
- `'multipoints'`
- `'subplotgraph'`

### 6.20.3.83 write

Syntax:

- `write(s, filename)` (writes in TSTOOL's own file format)
- `write(s, filename, 'ASCII')`
- `write(s, filename, 'WAV')` (RIFF WAVE FORMAT)
- `write(s, filename, 'AU')` (SUN AUDIO FORMAT)
- `write(s, filename, 'NLD')` (old NLD FORMAT)
- `write(s, filename, 'SIPP')` (si++ file format)

writes a signal object to file `filename` (uses matlab's file format)

## 6.21 Class description

### 6.21.1 Overview

Class description is the second base class of class `signal` (cf. Section 6.20). An object of type `description` stores all descriptive information belonging to a signal.

## 6.21.2 Attributes

- label - *string*
- name - *string*
- type - *string*
- plothint - *string*
- comment - object of type list (cf. Section 6.25)
- history - object of type list (cf. Section 6.25)
- creator - *string*
- yname - *string*
- yunit - object of type unit (cf. Section 6.24)
- commandlines - object of type list (cf. Section 6.25)
- optparam - *cell array*, may be used to store optional information

## 6.21.3 Member functions

### 6.21.3.1 addcommandlines

adds new commandline to list of commands that have been applied to that signal

**example 1** `addcommandlines(s, 's = spec2(s', 512, 'Hanning' )`) will add `'s = spec2(s, 512, 'Hanning');'` to the list of applied commands

**example 2** `len = 512; text = 'Hanning'; addcommandlines(s, 's = spec2(s', 512, 'Hanning' )`) will add `'s = spec2(s, 512, 'Hanning');'` to the list of applied commands

### 6.21.3.2 addcomment

adds new comment to current list of comments

### 6.21.3.3 addhistory

adds text to current history list always the current time and date is written into the first line

### 6.21.3.4 commandlines

### 6.21.3.5 comment

### 6.21.3.6 creator

### 6.21.3.7 description

description class constructor **Syntax:**

- `d = description()`
- `d = description(name)`

- `d = description(name, type)`
- `d = description(yunit)`

An object of class `description` contains auxiliary descriptive information for a signal, e.g. information about data unit, creator, how the signal should be plotted, a user specified comment text, a processing history and the commandlines that were used to generate this signal

#### 6.21.3.8 display

`description/display`

#### 6.21.3.9 history

`description/history`

#### 6.21.3.10 label

#### 6.21.3.11 makescript

**Syntax:**

- `makescript (signal, scriptfilename)`

creates a Matlab m-file that contains exactly the the processing steps that have been applied to get the input signal. This gives a kind of macro facility for `tstool`.

**Example** signal `s` was calculated through several processing steps from signal `s0` (the raw or original signal) Now `makescript(s, 'foo.m')` will create a Matlab m-file named *foo.m* which, applied to `s0`, will give `s`.

#### 6.21.3.12 merge

**Syntax:**

- `d = merge(d1, d2)`

merge two descriptions

Most items are taken from first description. History is taken from both descriptions. This function may be useful when writing binary operators for class `signal`

#### 6.21.3.13 name

`description/name` **Syntax:**

- `n = name(d)`

Get signal's name

#### 6.21.3.14 newcomment

##### Syntax:

- `d = newcomment(d, string)`
- `d = newcomment(d, list)`

Replace old comment with new comment

#### 6.21.3.15 optparams

##### Syntax:

- `param = optparams(d, nr)`

get optional parameter number `nr`

#### 6.21.3.16 plothint

#### 6.21.3.17 setlabel

##### Syntax:

- `d = setlabel(d, label)`

the label field of a description is used to give a signal some 'tag' which remains constant through various processing steps

e.g. which topic this signal belongs to

#### 6.21.3.18 setname

##### Syntax:

- `d = setname(d, name)`

the name field of a description is used when the signal is loaded from file, it will not be continued through several processing steps

#### 6.21.3.19 setoptparams

##### Syntax:

- `d = setoptparams(d, nr, param)`

set optional parameter number `nr`

#### 6.21.3.20 setplothint

#### 6.21.3.21 settype

##### Syntax:

- `d = settype(d, string)`

Set a new type for signal

### 6.21.3.22 setyname

#### Syntax:

- `d = setyunit(d, string)`

Set signal's y-name

e.g. `d = setyunit(d, 'V')`

### 6.21.3.23 setyunit

#### Syntax:

- `d = setyunit(d, unit)`
- `d = setyunit(d, string)`

Set signal's y-unit

e.g. `d = setyunit(d, 'V')`

### 6.21.3.24 type

return signal type (e.g. 'Correlation function', 'Spectrogram' etc.)

### 6.21.3.25 yname

return name of the measured data (e.g. 'Heartbeat rate', 'Current' etc.)

### 6.21.3.26 yunit

return y-unit of the sampled data values (e.g. Volt, Pa etc.)

## 6.22 Class core

### 6.22.1 Overview

Class core is a base class of class signal (cf. Section 6.20). An object of type core stores the pure sample values of a signal, without any additional descriptive information. The separation of the *numerical* and the descriptive part of a signal simplifies the writing of m-files that work on signals.

### 6.22.2 Attributes

- `data` - *double matrix* (one, two or multidimensional)  
TSTOOL stores a one-dimensional time-series always as a row vector ! Rows correspond to the first xaxis, columns to the second ...
- `dlens` - *double vector*, storing size of data

### 6.22.3 Member functions

#### 6.22.3.1 acf

**Syntax:**

- `acf(cin, m)`

**Input Arguments:**

- `cin` core object
- `m` fft-length

`acf` calculates the autocorrelation function of `cin` via fft of length `m`.

#### 6.22.3.2 amutual2

**Syntax:**

- `amutual(cin, len)`

**Input Arguments:**

- `cin` core object
- `len` maximal lag

`amutual2` calculates the mutual information of a time series against itself, with increasing lag uses equidistant partitioning to compute histograms.

#### 6.22.3.3 compare

**Syntax:**

- `compare(c1,c2, tolerance)`

**Input Arguments:**

- `c1,c2` core object of two signals
- `tolerance` tolerance of the signals's RMS value (default tolerance=1e-6)

`compare` compare two signals whether they have equal values slight differences due to rounding errors are ignored depending on the value of tolerance when signals are found to be not equal, a zero is returned.

#### 6.22.3.4 core

`core` class constructor **Syntax:**

- `c = core(arg)`

**Input Arguments:**

- `arg` double array

A core object contains the pure data part of a signal object.  
Methods: `ndim` `dlens` `data`

### 6.22.3.5 data

#### Syntax:

- `d = data(c, varargin)`
- `c`=core object

#### Input Arguments:

- `varargin` - selector string for data-elements in matlab notation

Return signal's data values

With no extra arguments, `data` returns the data array of a signal object

Another possible call is : `data(signal, ':,:,1:20')`

### 6.22.3.6 db

#### Syntax:

- `cout = db(cin, ref, scf, dbmin)`

#### Input Arguments:

- `cin` - core object
- `ref` - reference value
- `scf` - scaling factor
- `dbmin` - minimal db-value

compute decibel values to reference value `ref` and scaling factor (10 or 20) `scf`

### 6.22.3.7 diff

#### Syntax:

- `cout = diff(cin, nth, delta)`

#### Input Arguments:

- `cin` - core object
- `nth` - number of derivations
- `delta` - time difference between to signal values

`nth` numerical derivative along dimension 1 when `data` was sampled equidistantly with `samplerate = 1/delta`



#### 6.22.3.8 display

**Syntax:**

- `display(c)`

**Input Arguments:**

- `c` - core object

#### 6.22.3.9 dlens

**Syntax:**

- `d=dlens(c, nr)`

**Input Arguments:**

- `c` - core object

returns sizes of dimensions (same as function 'size' under matlab)

#### 6.22.3.10 embed

**Syntax:**

- `cout = embed(cin, dim, delay, shift, windowtype)`

**Input Arguments:**

- `cin` - core object
- `dim` - embed dimension
- `delay` - delay time in samples for time delay vectors
- `shift` - shift in samples for two sequent time delay vectors
- `windowtype` - type of window

Create time delay vectors with dimension `dim`, `delay` is measured in samples

The input must be a scalar time series

The result is a `n` by `dim` array, each row contains the coordinates of one point

#### 6.22.3.11 filterbank

**Syntax:**

- `filterbank(cin,H,G,ORDER,BASIS)`

**Input Arguments:**

- `H` - lowpass filter

- **G** - highpass filter
- **ORDER** - indicates the type of tree:
  - 0 - band sorting according to the filter bank
  - 1 - band sorting according to the frequency decomposition
- **BASIS** - desired subband decomposition

calculates the Wavelet Packet Transform of `cin`. It can be obtained using a selection algorithm function. It may be switched from one format to another using `CHFORMAT`. The different bands are sorted according to `ORDER` and `BASIS`. If `BASIS` is omitted, the output is a matrix with the coefficients obtained from all the wavelet packet basis in the library. Each column in the matrix represents the outputs for a level in the tree. The first column is the original signal. If the length of `X` is not a power of 2, the columns are zero padded to fit the different lengths. Run the script '`BASIS`' for help on the basis format.

See also: `IWPK`, `CHFORMAT`, `PRUNEADD`, `PRUNENON`, `GROWADD`, `GROWNON`.

#### 6.22.3.12 `int`

**Syntax:**

- `cout = int(cin, delta)`

**Input Arguments:**

- `cin` - core object
- `delta` - time period between two data samples

numerical integration along dimension 1 when data was sampled equidistantly with `samplerate = 1/delta`

#### 6.22.3.13 `intermutual`

**Syntax:**

- `intermutual(cin1,cin2,n)`

**Input Arguments:**

- `cin1,cin2` - core objects

Calculates the mutual information of `cin1` and `cin2`.

#### 6.22.3.14 `isempty`

**Syntax:**

- `r = isempty(s)`

**Input Arguments:**

- `s` - core object

test if core contains no (valid) data

### 6.22.3.15 medianfilt

#### Syntax:

- `medianfilt(cin,len)`

#### Input Arguments:

- `cin` - core object

moving median filter

### 6.22.3.16 minus

#### Syntax:

- `minus(c1,c2)`

#### Input Arguments:

- `c1,c2` - core objects

subtract `c2` from each columns of `c1`

### 6.22.3.17 movav

#### Syntax:

- `movav(cin,len)`

#### Input Arguments:

- `cin` - core object
- `len` - average length

moving average

### 6.22.3.18 multires

#### Syntax:

- `multires(cin,h,rh,g,rg,sc)`

#### Input Arguments:

- `cin` - core object

### 6.22.3.19 ndim

**Syntax:**

- `ndim(c)`

**Input Arguments:**

- `c` - core object

return number of dimensions, a scalar value has 0 dimensions

### 6.22.3.20 norm1

**Syntax:**

- `cout = norm1(cin,low,upp)`

**Input Arguments:**

- `cin` - core object
- `low` - column number
- `upp` - column number

normalize each single column of a the core object to be within [low,upp]

### 6.22.3.21 norm2

**Syntax:**

- `cout = norm2(cin)`

**Input Arguments:**

- `cin` - core object

normalize signal by removing it's mean and dividing by the standard deviation

### 6.22.3.22 plus

**Syntax:**

- `plus(c1,c2)`

**Input Arguments:**

- `c1,c2` - core objects

add c2 to each columns of c1

#### 6.22.3.23 rang

**Syntax:**

- `cout = rang(cin)`

**Input Arguments:**

- `cin` - core object

#### 6.22.3.24 rms

**Syntax:**

- `cout = rms(cin)`

**Input Arguments:**

- `cin` - core object

compute root mean square value of each column of `c1`

#### 6.22.3.25 scalogram

**Syntax:**

- `cout = scalogram(cin, smin, smax, sstep, tim)`

#### 6.22.3.26 spec

**Syntax:**

- `cout = spec(cin)`

**Input Arguments:**

- `cin` - core object

compute power spectrum for real valued signals

#### 6.22.3.27 spec2

**Syntax:**

- `cout = spec2(cin, fensterlen, fenster, vorschub)`

**Input Arguments:**

- `cin` - core object
- `fensterlen` - window size
- `fenster` - type of window
- `vorschub` - moving step

spectrogramm of data using short time fft

### 6.22.3.28 surrogate1

#### Syntax:

- `cout = surrogate1(cin)`

#### Input Arguments:

- `cin` - core object

create surrogate data for a scalar time series by randomizing phases of fourier spectrum  
see : James Theiler et al.'Using Surrogate Data to Detect Nonlinearity in Time Series', APPENDIX : ALGORITHM I

### 6.22.3.29 surrogate2

#### Syntax:

- `cout = surrogate2(cin)`

#### Input Arguments:

- `cin` - core object

create surrogate data for a scalar time series  
see : James Theiler et al.'Using Surrogate Data to Detect Nonlinearity in Time Series', APPENDIX : ALGORITHM II

### 6.22.3.30 surrogate3

#### Syntax:

- `cout = surrogate3(cin)`

#### Input Arguments:

- `cin` - core object

create surrogate data for a scalar time series by permuting samples randomly

### 6.22.3.31 uminus

#### Syntax:

- `r = uminus(c)`

#### Input Arguments:

- `c` - core object

negate time series

### 6.22.3.32 vertcat

**Syntax:**

- `r = vertcat(c1,c2)`

**Input Arguments:**

- `c1,c2` - core objects

catenate two timeseries vertically

## 6.23 Class achse

### 6.23.1 Overview

Class `achse` models an axis, e.g. a time axis or a frequency axis. A signal has at least one axis (if it is a one dimensional signal). A multidimensional signal has several `achse` objects. An `achse` object is basically described by an object of class `unit` and the spacing values. The spacing may be linear, logarithmic or arbitrary (in case of non-uniform sampling).

#### 6.23.1.1 Why is class `achse` not called class `axis` ?

The names `axis` and `axes` are already occupied in Matlab. So, `achse`, which is the german translation of `axis`, was used as name for that class.

### 6.23.2 Attributes

- `name` - *string*, name of axis (e.g. `'Time'`)
- `quantity` - *string*
- `unit` - object of type `unit` (cf. Section 6.24)
- `resolution` - *string*, may be `'linear'`, `'logarithmic'` or `'arbitrary'`
- `first` - *double value*, starting value of this axis
- `delta` - *double value*, stepwidth for this axis
- `values` - *double vector*, stores spacing values in case of `'arbitrary'` resolution
- `opt` - *cell array*, may be used to store optional information

### 6.23.3 Member functions

#### 6.23.3.1 achse

`achse` class constructor

**Syntax:**

- `a = achse`  
creates default `achse` object

- `a = achse(axes)`  
copies achse object `axes` into `a`
- `a = achse(unt)`  
creates achse object using unit `unt`, with linear spacing, `first = 0`, `delta = 1`
- `a = achse(vec)`  
creates achse object with arbitrary spacing, using values in `vec` as spacing data
- `a = achse(unt, vec)`  
creates achse object using unit `unt` with arbitrary spacing, using values in `vec` as spacing data
- `a = achse(unt, first, delta)`  
creates achse object with linear spacing, using `delta` and `first`
- `a = achse(unt, first, delta, 'log')`  
creates achse object with logarithmic spacing, using `delta` and `first`

achse used to describe the different dimensions (axes) of a signal object.

#### Example:

- `a = achse(unit('Hz'), 0.01, 10, 'log')`  
creates a logarithmic frequency axis with values 0.01 Hz, 0.1 Hz, 1 Hz, 10 Hz
- `a = achse(label, samplerate)`  
has the same result as  
`a = achse(unit(label), 0, 1/samplerate)`

see also: `delta` `first` `horzcat` `label` `name` `quantity` `resolution` `samplerate` `scale` `setname` `spacing` `unit`

### 6.23.3.2 cut

#### Syntax:

- `a = cut(a, start, stop)`

Cut a part out of achse `a`, beginning from index `start` up to index `stop`. `stop` is only needed in case of arbitrary spacing. `cut` ensures the following:

If `values = spacing(achse1, N)` and `N > n` then  
`values(n:N) == spacing(cut(achse1, n), N+1-n)`

See also: `horzcat`

### 6.23.3.3 delta

### 6.23.3.4 display

### 6.23.3.5 eq

Test if achse `a` and achse `b` are equal. `first` is not (!) taken into account for this test.



**6.23.3.6** first

**6.23.3.7** horzcat

**6.23.3.8** label

**6.23.3.9** name

**6.23.3.10** quantity

**6.23.3.11** resolution

**6.23.3.12** samplerate

Syntax:

- `rate = samplerate(a)`

samplerate returns samplerate of achse object.

**6.23.3.13** scale

Syntax:

- `r = scale(a,f)`

Scale achses delta by factor f.

**6.23.3.14** setdelta

Syntax:

- `a = setdelta(a,f)`

**6.23.3.15** setfirst

Syntax:

- `a = setfirst(a,f)`

**6.23.3.16** setname

Syntax:

- `a = setname(a, newname)`

**6.23.3.17** setunit

Syntax:

- `a = setunit(a,u)`

### 6.23.3.18 setvalues

Syntax:

- `a = setvalues(a, v)`

### 6.23.3.19 spacing

Syntax:

- `v = values(a, len)`

Returns spacing values for linear, logarithmic or arbitrary spacing in case of lin. or log. spacing. `len` values are returned. In case of arbitrary spacing, all stored values are returned.

### 6.23.3.20 unit

## 6.24 Class unit

### 6.24.1 Overview

Objects of class 'unit' try to model physical units. It's possible to multiply or divide objects of this type. A small database is used to find the right label for compound units.

See also : directory *@unit/private*, file *units.mat*

### 6.24.2 Attributes

- `label` - *string*
- `name` - *string*
- `quantity` - *structure*, holding two strings
- `factor` - *double value*
- `exponents` - *vector*
- `dBScale` - *double value*
- `dBRef` - *double value*
- `opt` - *cell array*, may be used to store optional information

### 6.24.3 Member functions

#### 6.24.3.1 char

gives the unit's label (e.g. V for Volt) back.

#### 6.24.3.2 dbref

returns reference value for 0 dB when calculating decibel values from data of this unit.

#### **6.24.3.3 dbscale**

returns scaling value when calculating decibel values from data of this unit. `dpscale` returns either 10 (for power or energy units (e.g. Watt)) or 20 (for all other units (e.g. Volt)).

#### **6.24.3.4 display**

#### **6.24.3.5 double**

gives a row vector which's first element contains the unit's factor and the remaining elements contain the exponents of the SI basic units.

#### **6.24.3.6 eq**

#### **6.24.3.7 exponents**

returns dimension exponents of unit `q`.

#### **6.24.3.8 factor**

returns factor of unit `q`.

#### **6.24.3.9 label**

#### **6.24.3.10 mpower**

**Syntax:**

- `mpower(u,p)`

take unit `u` to power `p`, `p` must be a scalar.

#### **6.24.3.11 mrdivide**

#### **6.24.3.12 mtimes**

#### **6.24.3.13 name**

returns name of unit `q`.

#### **6.24.3.14 quantity**

returns quantity name of unit `q`. If argument which is omitted, the english quantity name will be returned.

### 6.24.3.15 unit

`unit` class constructor

Class `unit` tries to model physical units. A physical unit is mainly described by the exponents of the basic SI units, namely mass, length, time, current, temperature, luminous intensity, mole and plane angle. Each unit belongs to a quantity, e.g. the unit `s` (second) is used when measuring the quantity TIME. Each unit has a name, e.g. 'Ampere', 'Volt', 'Joule', 'hour', and an abbreviation, called label ('A', 'V', 'J', 'h'). Unfortunately, the correspondence between these items is not always bijective. To find corresponding items, a table of units in the file `units.mat` is used.

A unit object can be created with different types of arguments:

- by giving the label: `unit('Hz')` looks up the remaining data (exponents, name, quantity) in the table
- by giving the exponents

Some arithmetic can be done with units:

- units can be multiplied `unit('V') * unit('A') = unit('Watt')`
- or taken to an integer or rational power `unit('m')2`

## 6.25 Class list

### 6.25.1 Overview

Simple list of strings, used in class description (cf. Section 6.21).

### 6.25.2 Attributes

- `data` : *cellarray* of strings
- `len` : double value, counts number of strings in data

### 6.25.3 Member functions

#### 6.25.3.1 append

Syntax:

- `list = append(list, string)`
- `list = append(list, list)`

Add string(s) to existing list.

#### 6.25.3.2 cellstr

`cellstr` return cell array of strings from list `l`.

#### **6.25.3.3 char**

returns a char array from list 1.

#### **6.25.3.4 display**

#### **6.25.3.5 get**

**Syntax:**

- `s = get(l, nr)`

returns string number nr from list 1.

#### **6.25.3.6 length**

**Syntax:**

- `len = length(l)`

returns the number of strings in list 1.

#### **6.25.3.7 list**

**Syntax:**

- `l = list`  
creates empty list
- `l = list('Hello world')`  
create list with one entry, 'Hello world'
- `l = list('Hello', 'My' , 'World')`  
create list with three entries
- `l = list('Hello', 'My' , 'World')`  
create list with three entries

An object of type list contains a list of strings.

#### **6.25.3.8 sort**

sort list 1 in increasing order.



# Chapter 7

## Frequently asked questions

### 7.1 Questions

1. Introduction and general information (cf. Section 7.2.1)
  - What is TSTOOL ? (cf. Section 7.2.1.1)
  - What software is required to run TSTOOL ? (cf. Section 7.2.1.2)
  - On which systems does TSTOOL run ? (cf. Section 7.2.1.3)
  - What about Octave or other Matlab like programming environments ? (cf. Section 7.2.1.4)
2. Installation of TSTOOL
  - All lines in the `OpenTSTOOL/tstoolbox/mex/*.m` are comments, is this right? (cf. Section 7.2.2.1)
  - Where are the precompiled Mex-Files? (cf. Section 7.2.2.2)
  - There are more than one file called e.g. `amutual.m`, why? (cf. Section 7.2.2.3)
  - What does the error message "Attempt to execute SCRIPT ... as a function." mean? (cf. Section 7.2.2.4)
3. Working with TSTOOL (cf. Section 7.2.3)
  - How do I create a signal from my time-series data ? (cf. Section 7.2.3.1)
  - How do I create a signal with logarithmic spacing ? (cf. Section 7.2.3.2)
  - How do I create a signal from non-uniformly sampled data ? (cf. Section 7.2.3.3)
  - How do I change the type of plot that I get with `view` ? (cf. Section 7.2.3.4)
  - What is class 'signal' for ? (cf. Section 7.2.3.5)
  - What is class 'core' for ? (cf. Section 7.2.3.6)
  - What is class 'description' for ? (cf. Section 7.2.3.7)
  - What is class 'achse' for ? (cf. Section 7.2.3.8)
  - Why is class 'achse' called 'achse' and not 'axis' ? (cf. Section 7.2.3.9)
  - What is class 'unit' for ? (cf. Section 7.2.3.10)
  - How is class 'unit' used in TSTOOL ? (cf. Section 7.2.3.11)
4. Extending TSTOOL (cf. Section 7.2.4)
  - How can I write a script to automatize common tasks ? (cf. Section 7.2.4.1)
  - How can I write my own routines for the TSTOOL package ? (cf. Section 7.2.4.2)

5. Miscellaneous questions (cf. Section 7.2.5)

- What's the difference between history and commandlines ? (cf. Section 7.2.5.1)

6. Frequently encountered errors (cf. Section 7.2.6)

- Using a column vector to create a one-dimensional signal (cf. Section 7.2.6.1)
- What does the error message "Attempt to execute SCRIPT ... as a function." mean? (cf. Section 7.2.2.4)

## 7.2 Answers

### 7.2.1 Introduction and general information

#### 7.2.1.1 What is TSTOOL ?

TSTOOL is a software package for nonlinear time series analysis, though it has a lot of features a general signal analysis package would also have.

#### 7.2.1.2 What software is required to run TSTOOL ?

TSTOOL is written in MATLAB, a powerful language for scientific computing, and in **C++**. Therefore you need MATLAB version 5.2 or higher to run `tstool`. Unfortunately, MATLAB is not free software !

#### 7.2.1.3 On which systems does TSTOOL run ?

TSTOOL does work on Windows 95/98/NT and SGI IRIX 6.5. It does not work on all platforms for which MATLAB is available due to the use of *mex-files*, which are functions written in **C** or **C++** that extend MATLAB's set of build-in functions. However, these mex-files must be compiled for every platform individually.

#### 7.2.1.4 What about Octave or other Matlab like programming environments ?

Octave<sup>1</sup> is a freely available language for scientific computing that strongly resembles MATLAB. Unfortunately, Octave is not fully compatible to MATLAB, so TSTOOL does not work with Octave.

TSTOOL makes use of the object oriented features of MATLAB. In the current version of Octave (2.0.14) there's no full support of classes. Even if classes will be supported in future, it's not sure wheter TSTOOL will work properly.

There are several other Matlab like programming environments, e.g. Mideva<sup>2</sup> or Scilab<sup>3</sup>. Up to now, it is not possible to use TSTOOL with these packages.

### 7.2.2 Installation of TSTOOL

#### 7.2.2.1 All lines in the `OpenTSTOOL/tstoolbox/mex/*.m` are comments, is this right?

Yes, this are the comment-texts for the compiled mex functions (e.g. type `help amutual` at the matlab prompt).

---

<sup>1</sup>See URL <http://www.che.wisc.edu/octave/>

<sup>2</sup>See URL <http://www.mathtools.com>

<sup>3</sup>See URL <http://www-rocq.inria.fr/scilab/>



### 7.2.2.2 Where are the precompiled Mex-Files?

- Sun - OpenTSTOOL/tstoolbox/mex/mexsol/\*.mexsol
- SGI - OpenTSTOOL/tstoolbox/mex/mexsg64/\*.mexsg64
- Linux - OpenTSTOOL/tstoolbox/mex/mexglx/\*.mexglx
- Windows - OpenTSTOOL/tstoolbox/mex/dll/\*.dll

### 7.2.2.3 There are more than one file called e.g. amutual.m, why?

For some functions there are up to three versions of the file:

- OpenTSTOOL/tstoolbox/@signal/amutual.m Function that invokes the underlying mex function. It uses signal objects as output and input.
- OpenTSTOOL/tstoolbox/mex/amutual.m Help text for the compiled mex function (e.g. type `help amutual` on the matlab prompt).
- OpenTSTOOL/tstoolbox/mex/mexsol/amutual.mexsol,  
OpenTSTOOL/tstoolbox/mex/mexsg64/amutual.mexsg64,  
OpenTSTOOL/tstoolbox/mex/mexglx/amutual.mexglx,  
OpenTSTOOL/tstoolbox/mex/dll/amutual.dll  
Precompiled mex files for Solaris, SGI, Linux x86 and Windows. Only one of this files may be present on your system, depending on the Version of TSTOOL you have downloaded.

### 7.2.2.4 What does the error message "Attempt to execute SCRIPT ... as a function." mean?

Matlab cannot find the correct mex files for this systems and so it tries to execute the *scripts* OpenTSTOOL/mex/\*.m (which are only the help texts for the mex files). There are many possibilities for this error:

- You downloaded the wrong version of TSTOOL.
- The path setting made by `settpath.m` are not correct. Type `path` at the matlab prompt and look for the path setting for the mex directory (see 7.2.2.3).
- The mex files are not present in the directory noted in 7.2.2.3.

## 7.2.3 Working with TSTOOL

### 7.2.3.1 How do I create a signal from my time-series data ?

Suppose the time-series data is given as the row vector *y*.

```
>> s = signal(y)
>> view(s)
```

If *y* is a column vector, the following syntax must be used:

```
>> s = signal(y')
>> view(s)
```

Suppose the data was recorded with a samplerate of 8 kHz :

```
>> s = signal(y, 8000)
>> view(s)
```

### 7.2.3.2 How do I create a signal with logarithmic spacing ?

Suppose you have data vector `y` whose values were recorded at 3 Hz, 6 Hz, 12 Hz, 24 Hz ...

```
a = achse(unit('Hz'), 3, 2, 'log')
s = signal(y, a)
view(s)
```

### 7.2.3.3 How do I create a signal from non-uniformly sampled data ?

Suppose you have a data vector `y` (of length 4) whose values were recorded at 3 Hz, 5 Hz, 8 Hz, 14.5 Hz

```
a = achse(unit('Hz'), [3 5 8 14.5])
s = signal(y, a)
view(s)
```

### 7.2.3.4 How do I change the type of plot that I get with `view` ?

The way `view` plots a signal depends on the attributes of the signal. It is possible to give `view` a *hint* which type of plot to prefer. This hint can be set with the command `setplothint`. The possible plot types can be obtained by issuing

```
help signal/view
```

at the Matlab prompt. However, if the signal does not support the desired type of plotting (e.g. a one-dimensional time-series can not be visualized as orbit), `view` will use the default plot type for the data.

```
s = signal(rand(1000, 3));
s = setplothint(s, '3dpoints');
view(s)
```

### 7.2.3.5 What is class 'signal' for ?

Class `signal` is TSTOOL's main class. Objects of this type model real world signals. A signal does not only store the pure sample values, it holds much more information like axes, units of sample values or the axes units, and even more descriptive information like labels, command lines and a processing history.

The majority of functions in the `tstoolbox` take a signal as input argument and return a processed signal as output. This allows for combining or *chaining* of several processing steps in order to get the desired output.

### 7.2.3.6 What is class 'core' for ?

Class `core` is a base class of class `signal`. An object of type `core` stores the pure sample values of a signal, without any additional descriptive information. The separation of the *numerical* and the descriptive part of a signal simplifies the writing of m-files that work on signals.

### 7.2.3.7 What is class 'description' for ?

Class `description` is the second base class of class `signal`. An object of type `description` stores all descriptive information belonging to a signal.

#### **7.2.3.8 What is class 'achse' for ?**

Class achse models an axis, e.g. a time axis or a frequency axis. A signal has at least one axis (if it is a one dimensional signal). A multidimensional signal has several achse objects, one for each dimension. An achse object is basically described by an object of class unit and the spacing values. The spacing may be linear, logarithmic or arbitrary (in case of non-uniform sampling).

#### **7.2.3.9 Why is class 'achse' called 'achse' and not 'axis' ?**

The names axis and axes are already occupied in Matlab. So, achse, which is the german translation of axis, was used as name for that class.

#### **7.2.3.10 What is class 'unit' for ?**

Objects of class 'unit' try to model physical units. No one wonders that his computer can multiply real or complex numbers. But in physics or engineering, you also have to multiply or divide physical units, just think of Ohm's law :  $R = U/I$

#### **7.2.3.11 How is class 'unit' used in TSTOOL ?**

Class unit is used as a part of every achse object and as part of a description object. Handling and processing of units is optional for functions that work on signals, because many nonlinear signal analysis functions do not allow consistent handling of units.

### **7.2.4 Extending TSTOOL**

Of course it's possible to extend TSTOOL with some custom functionality or to use parts of TSTOOL in your own m-files, just as with other toolboxes for Matlab.

#### **7.2.4.1 How can I write a script to automatize common tasks ?**

One way to obtain a script is to execute the desired analysis steps with one example signal. The output of this tasks will again be a signal that has stored the syntax of the executed steps in its description. Using the command

```
commandlines(result)
```

will give you this syntax. With copy and paste, it's possible to create a script file from that output.

#### **7.2.4.2 How can I write my own routines for the TSTOOL package ?**

Please refer to the upcoming programming manual for TSTOOL.

### **7.2.5 Miscellaneous questions**

#### **7.2.5.1 What's the difference between history and commandlines ?**

Both attributes of class description are used to record the processing history of a signal. But, while history contains a list human readable entries, commandlines stores the exact syntax of the commands that were applied to the signal.

## 7.2.6 Frequently encountered errors

### 7.2.6.1 Using a column vector to create a one-dimensional signal

TSTOOL stores one-dimensional signals always as row vectors. Giving a column vector will cause unexpected behaviour with most routines that process signals:

```
>> s = signal(sin(0:0.5:100))
s = signal object

Dlens : 1 201
X-Axis 1 : |
X-Axis 2 : |

Name :
Type :

Attributes of data values :
|

Comment :

History :
16-Aug-1999 19:15:01 : Imported from MATLAB workspace
```

Instead, a row vector must be given to create the desired one-dimensional signal:

```
>> s = signal(sin(0:0.5:100)')
s = signal object

Dlens : 201
X-Axis 1 : |

Name :
Type :

Attributes of data values :
|

Comment :

History :
16-Aug-1999 19:16:58 : Imported from MATLAB workspace
```

# Bibliography

- [1] Ott, E. (1993), *Chaos in Dynamical Systems*, Cambridge Cambridge, University Press.
- [2] Thompson, J. M. T., H. B. Stewart (1986), *Nonlinear Dynamics and Chaos*, Chichester, Wiley.
- [3] Schuster, H. G. (1988), *Deterministic Chaos*, 2nd ed., Weinheim, VHC Publishers.
- [4] Bergé, P., Y. Pomeau, C. Vidal (1984), *Order within Chaos: Towards a Deterministic Approach to Turbulence*, New York, John Wiley and Sons.
- [5] Moon, F. C. (1992), *Chaotic and Fractal Dynamics*, New York, John Wiley and Sons.
- [6] Gaponov-Grekhov, A. V., M. I. Rabinovich, (1992), *Nonlinearities in Action – Oscillations, Chaos, Order, Fractals*, Berlin, Springer.
- [7] Lauterborn, W., J. Holzfuss, (1991) , “Acoustic chaos”, *Int. J. Bifurcation and Chaos*, 1, pp.13-26.
- [8] Lauterborn, W., T. Kurz, U. Parlitz, (1997), “Experimental Nonlinear Physics”, *Int. J. Bifurcation and Chaos*, 7, pp.2003-2033.
- [9] Lauterborn, W., U. Parlitz, (1988), “Methods of chaos physics and their application to acoustics”, *J. Acoust. Soc. Am.*, 84, pp.1975-1993.
- [10] Packard, N.H., J.P. Crutchfield, J.D. Farmer, R.S. Shaw (1980), “Geometry from a time series”, *Phys. Rev. Lett.*, 45, pp.712-716.
- [11] Takens, F. (1981), “Detecting strange attractors in turbulence”, in *Dynamical Systems and Turbulence*, eds. Rand, D.A. & Young, L.-S. , Berlin, Springer, pp.366-381.
- [12] Kantz, H., & T. Schreiber (1997), *Nonlinear Time Series Analysis*, Cambridge University Press, Cambridge.
- [13] Abarbanel, H.D.I. (1996), *Analysis of Observed Chaotic Data*, Springer, New York.
- [14] Abarbanel, H.D.I., Brown, R., Sidorowich, J.J., & Tsimring, L.S. (1993), “The analysis of observed chaotic data in physical systems,” *Rev. Mod. Phys.*, 65(4), pp.1331-1392.
- [15] Grassberger, P., Schreiber, T. & Schaffrath, C. (1991), “Nonlinear time sequence analysis,” *Int. J. Bif. Chaos*, 1(3), pp.521-547.
- [16] Sauer, T., Y. Yorke, M. Casdagli (1991), “Embedology”, *J. Stat. Phys.*, 65, pp.579-616.
- [17] Sauer, T., J.A. Yorke (1993), “How many delay coordinates do you need ?”, *Int. J. Bifurcation and Chaos*, 3, pp.737-744.
- [18] Casdagli, M., S. Eubank, J.D. Farmer, J. Gibson (1991), “State space reconstruction in the presence of noise”, *Physica D*, 51, pp.52-98.
- [19] Gibson, J.F., J.D. Farmer, M. Casdagli, S. Eubank (1992), “An analytic approach to practical state space reconstruction” *Physica D*, 57, pp.1-30.
- [20] Broomhead, D.S., G.P. King (1986), “Extracting qualitative dynamics from experimental data”, *Physica D*, 20, pp.217-236.
- [21] Landa, P.S., M.G. Rosenblum (1991), “Time series analysis for system identification and diagnostics”, *Physica D*, 48, pp.232-254.
- [22] Palus, M., I. Dvorak (1992), “Singular-value decomposition in attractor reconstruction: pitfalls and precautions”, *Physica D*, 55, pp.221-234.
- [23] Sauer, T.(1994), “Reconstruction of dynamical systems from interspike intervals”, *Phys. Rev. Lett.*, 72, pp.3811-3814.
- [24] Castro, R., T. Sauer (1997), “Correlation dimension of attractors through interspike intervals”, *Phys. Rev. E*, 55(1), pp.287-290.

- [25] Racicot, D.M., A. Longtin (1997), "Interspike interval attractors from chaotically driven neuron models", *Physica D*, 104, pp.184-204.
- [26] Stark, J., D.S. Broomhead, M.E. Davies, J. Huke (1996), "Takens embedding theorems for forced and stochastic systems", in: Proceedings of the 2nd World Congress of Nonlinear Analysts, Athens, Greece, July 1996.
- [27] Kennedy, M.P. (1994), "Chaos in the Colpitts oscillator", *IEEE Trans. Circuits Syst.*, 41(11), pp.771-774.
- [28] Cenys, A., K. Pyragas (1988), "Estimation of the number of degrees of freedom from chaotic time series", *Phys. Lett. A*, 129, pp.227-230.
- [29] Buzug, Th., T. Reimers, G. Pfister (1990), "Optimal reconstruction of strange Attractors from purely geometrical arguments", *Europhys. Lett.*, 13, pp.605-610.
- [30] Aleksić, Z. (1991), "Estimating the embedding dimension", *Physica D*, 52, pp.362-368.
- [31] Buzug, Th., G. Pfister (1992), "Optimal delay time and embedding dimension for delay-time coordinates by analysis of the global static and local dynamical behavior of strange attractors", *Phys. Rev. A*, 45, pp.7073-7084.
- [32] Gao, J., Z. Zheng (1993), "Local exponential divergence plot and optimal embedding of a chaotic time series", *Phys. Lett. A*, 181, pp.153-158.
- [33] Gao, J., Z. Zheng (1994). "Direct dynamical test for deterministic chaos and optimal embedding of a chaotic time series", *Phys. Rev. E*, 49, pp.3807-3814.
- [34] Huerta, R., C. Santa Cruz, J.R. Dorronsore, V. Lòpez (1995), "Local state-space reconstruction using averaged scalar products of dynamical-system flow vectors", *Europhys. Lett.*, 29, pp.13-18.
- [35] Liebert, W., K. Pawelzik, H.G. Schuster (1991), "Optimal embeddings of chaotic attractors from topological considerations", *Europhys. Lett.*, 14, pp.521-526.
- [36] Kennel, M.B., R. Brown, H.D.I. Abarbanel (1992), "Determining embedding dimension for phase-space reconstruction using a geometrical construction", *Phys. Rev. A*, 45, pp.3403-3411.
- [37] Fredkin, D.R., J.A. Rice (1995), "Method of false nearest neighbors: a cautionary note", *Phys. Rev. E*, 51(4), pp. 2950-2954.
- [38] Cao, L. (1997), "Practical method for determining the minimum embedding dimension of a scalar time series", *Physica D*, 110, pp. 43-50.
- [39] Kember, G., A.C. Fowler (1993), "A correlation function for choosing time delays in phase portrait reconstructions", *Phys. Lett. A*, 179, pp.72-80.
- [40] Rosenstein, M.T., J.J. Collins, C.J. De Luca (1994), "Reconstruction expansion as a geometry-based framework for choosing proper delay times", *Physica D*, 73, pp.82-98.
- [41] Frazer, A.M., H.L. Swinney (1986), "Independent coordinates in strange attractors from mutual information", *Phys. Rev. A*, 33, pp.1134-1140.
- [42] Frazer, A.M. (1989), "Reconstructing attractors from scalar time series: a comparison of singular system and redundancy criteria", *Physica D*, 34, pp.391-404.
- [43] Frazer, A.M. (1989), "Information and entropy in strange attractors", *IEEE Trans. Info. Theory*, 35, pp.245-262.
- [44] Liebert, W., H.G. Schuster (1989), "Proper choice of the time delay for the analysis of chaotic time series", *Phys. Lett. A*, 142, pp.107-111.
- [45] Martinerie, J.M., A.M. Albano, A.I. Mees, P.E. Rapp (1992), "Mutual information, strange attractors, and the optimal estimation of dimension", *Phys. Rev. A*, 45, pp.7058-7064.

- [46] Broomhead, D.S., J.P. Huke, M.R. Muldoon (1992), "Linear filters and nonlinear systems", *J. Roy. Stat. Soc.*, B54, pp.373-382.
- [47] Davies, M.E., & K.M. Campbell (1996), "Linear recursive filters and nonlinear dynamics" *Non-linearity*, 9, pp.487-499.
- [48] Kaplan, D., T. Schreiber (1996), "Signal separation by nonlinear projections: The fetal electrocardiogram", *Phys. Rev. E*, 53(5), pp.R4326-R4329.
- [49] Grassberger, P., R. Hegger, H. Kantz, C. Schaffrath, T. Schreiber (1993), "On noise reduction methods for chaotic data" *CHAOS*, 3, pp.127-141.
- [50] Kantz, H., T. Schreiber, I. Hoffmann, T. Buzug, G. Pfister, C.G. Flepp, J. Simonet, R. Badii, E. Brun (1993), "Nonlinear noise reduction: A case study on experimental data", *Phys. Rev. E*, 48, pp.1529-1538.
- [51] Kostelich, E.J., T. Schreiber (1993), "Noise reduction in chaotic time-series data: A survey of common methods", *Phys. Rev. E*, 48, pp.1752-1763.
- [52] Theiler, J., B. Galdrikian, A. Longtin, S. Eubank, J.D. Farmer (1992), "Using surrogate data to detect nonlinearity in time series" in: *Nonlinear Modeling and Forecasting*, eds. M. Casdagli and S. Eubank, SFI Studies in the Sciences of Complexity, Vol.XII (Reading, MA, Addison-Wesley), pp.163-188.
- [53] Theiler, J., S. Eubank, A. Longtin, B. Galdrikian, J.D. Farmer (1992), "Testing for nonlinearity in time series: the method of surrogate data", *Physica D*, 58, pp.77-94.
- [54] Provenzale, A., L.A. Smith, R. Vio, G. Murante (1992), "Distiguishing between low-dimensional dynamics and randomness in measured time series", *Physica D*, 58, pp.31-49.
- [55] Smith, L. (1992), "Identification and prediction of low dimensional dynamics", *Physica D*, 58, pp.50-76.
- [56] Takens, F. (1993), "Detecting nonlinearities in stationary time series", *Int. J. of Bifurcation and Chaos*, 3, pp.241-256.
- [57] Wayland, R., D. Bromley, D. Pickett, A. Passamante (1993), "Recognizing determinism in a time series", *Phys. Rev. Lett.*, 70, pp.580-582.
- [58] Palus, M., V. Albrecht, I. Dvorak (1993), "Information theoretic test for nonlinearity in time series", *Phys. Lett. A*, 175, pp.203-209.
- [59] Kaplan, D. (1994), "Exceptional events as evidence for determinism", *Physica D*, 73, pp.38-48.
- [60] Salvino, L.W., R. Cawley (1994), "Smoothness implies determinism: a method to detect it in time series", *Phys. Rev. Lett.*, 73, pp.1091-1094.
- [61] Savit, R.M. Green (1991), "Time series and dependent variables", *Physica D*, 50, pp.95-116.
- [62] Rapp, P.E., A.M. Albano, I.D. Zimmerman, M.A. Jiménez-Molteni (1994), "Phase-randomized surrogates can produce spurious identifications of non-random structure", *Phys. Lett. A*, 192, pp.27-33.
- [63] Theiler, J. (1995), "On the evidence for low-dimensional chaos in an epileptic electroencephalogram", *Phys. Lett. A*, 196, pp.335-341.
- [64] Schreiber, T., A. Schmitz (1996), "Improved surrogate data for nonlinearity tests", *Phys. Rev. Lett.*, 77(4), pp.635-638.
- [65] Schreiber, T. (1998), "Constrained randomization of time series data", *Phys. Rev. Lett.*, 80(10), pp.2105-2108.
- [66] Judd, K., A. Mees (1995), "On selecting models for nonlinear time series", *Physica D*, 82, pp.426-444.



- [67] Aguirre, L.A., S.A. Billings (1995), "Identification of models for chaotic systems from noisy data: implications for performance and nonlinear filtering", *Physica D*, 85, pp. 239-258.
- [68] Aguirre, L.A., E.M.A.M. Mendes (1996), "Global nonlinear polynomial models: structure, term clustering and fixed points", *Int. J. Bifurc. Chaos*, 6(2), pp.279-294.
- [69] Allie, S., A. Mees, K. Judd, D. Watson (1997), "Reconstructing noisy dynamical systems by triangulation", *Phys. Rev. E*, 55(1), pp. 87-93.
- [70] Szpiro, G.G. (1997), "Forecasting chaotic time series with genetic algorithms", *Phys. Rev. E*, 55(3), pp. 2557-2568.
- [71] Jaeger, L., H. Kantz (1997), "Effective deterministic models for chaotic dynamics perturbed by noise", *Phys. Rev. E*, 55(5), pp. 5234-5247.
- [72] Farmer, J.D., J.J. Sidorowich (1987), "Predicting chaotic time series", *Phys. Rev. Lett.*, 59, pp.845-848.
- [73] Casdagli, M. (1989), "Nonlinear Prediction of chaotic time series", *Physica D*, 35, pp.335-356.
- [74] Brown, R. N.F. Rulkov, E.R. Tracy (1994), "Modeling and synchronizing chaotic systems from time-series data", *Phys. Rev. E*, 49, pp.3784-3800.
- [75] Grassberger, P., I. Procaccia (1983), "On the characterization of strange attractors", *Phys. Rev. Lett.*, 50, pp.346-349.
- [76] Theiler, J. (1986), "Spurious dimension from correlation algorithms applied to limited time-series data", *Phys. Rev. A*, 34, pp.2427-2431.
- [77] Badii, R., A. Politi (1984), "Hausdorff dimension and uniformity factor of strange attractors", *Phys. Rev. Lett.*, 52, pp.1661-1664.
- [78] Badii, R., A. Politi (1985), "Statistical description of chaotic attractors", *J. Stat. Phys.*, 40, pp.725-750.
- [79] Grassberger, P. (1985), "Generalizations of the Hausdorff dimension of fractal measures", *Phys. Lett. A*, 107, pp.101-105.
- [80] Schreiber, T. (1995), "Efficient neighbor searching in nonlinear times series analysis", *Int. J. Bifurcation and Chaos*, 5, pp.349-358.
- [81] Holzfuss, J., G. Mayer-Kress (1986), "An approach to error-estimation in the application of dimension algorithms", in [82], pp.114-122.
- [82] Mayer-Kress, G. (ed.) (1986), *Dimensions and Entropies in Chaotic Systems – Quantification of Complex Behavior*, Berlin, Springer.
- [83] Theiler, J. (1990), "Estimating fractal dimension", *J. Opt. Soc. Am. A*, 7, pp.1055-1073.
- [84] Broggi, G. (1988), "Evaluation of dimensions and entropies of chaotic systems", *J. Opt. Soc. Am. B*, 5, pp.1020-1028.
- [85] Oseledec, V.I. (1968), "A multiplicative ergodic theorem. Lyapunov characteristic numbers for dynamical systems", *Trans. Moscow Math. Soc.*, 19, pp.197-231.
- [86] Benettin, G., L. Galgani, A. Giorgilli, J.-M. Strelcyn (1980), "Lyapunov characteristic exponents for smooth dynamical systems and for hamiltonian systems; a method for computing all of them. Part II: Numerical application " *Meccanica*, 15, pp.21-30.
- [87] Shimada, I., T. Nagashima (1979), "A numerical approach to ergodic problems of dissipative dynamical systems", *Prog. Theor. Phys.*, 61, pp.1605-1616.
- [88] Eckmann, J.-P., D. Ruelle (1985), "Ergodic theory of chaos and strange attractors", *Rev. Mod. Phys.*, 57, pp.617-656.

- [89] Geist, K., U. Parlitz, W. Lauterborn (1990), "Comparison of Different Methods for Computing Lyapunov Exponents", *Prog. Theor. Phys.*, 83, pp.875-893.
- [90] Wolf, A., J.B. Swift, L. Swinney, J.A. Vastano (1985), "Determining Lyapunov exponents from a time series", *Physica D*, 16, pp.285-317.
- [91] Sano, M., Y. Sawada (1985), "Measurement of the Lyapunov spectrum from a chaotic time series", *Phys. Rev. Lett.*, 55, pp.1082-1085.
- [92] Eckmann, J.-P., S.O. Kamphorst, D. Ruelle, S. Ciliberto (1986), "Lyapunov exponents from time series", *Phys. Rev. A*, 34, pp.4971-4979.
- [93] Stoop, R., P.F. Meier (1988), "Evaluation of Lyapunov exponents and scaling functions from time series", *J. Opt. Soc. Am. B*, 5, pp.1037-1045.
- [94] Holzfuss, J., W. Lauterborn (1989), "Lyapunov exponents from a time series of acoustic chaos", *Phys. Rev. A*, 39, pp.2146-2152.
- [95] Stoop, R., J. Parisi (1991), "Calculation of Lyapunov exponents avoiding spurious elements", *Physica D*, 50, pp.89-94.
- [96] Zeng, X., R. Eykholt, R.A. Pielke (1991), "Estimating the Lyapunov-exponent spectrum from short time series of low precision", *Phys. Rev. Lett.*, 66, pp.3229-3232.
- [97] Zeng, X., R.A. Pielke, R. Eykholt (1992), "Extracting Lyapunov exponents from short time series of low precision", *Modern Phys. Lett. B*, 6, pp.55-75.
- [98] Parlitz, U. (1993), "Lyapunov exponents from Chua's circuit", *J. Circuits, Systems and Computers*, 3, pp.507-523.
- [99] Kruegel, Th.M., M. Eiswirth, F.W. Schneider (1993), "Computation of Lyapunov spectra: Effect of interactive noise and application to a chemical oscillator", *Physica D*, 63, pp.117-137.
- [100] Briggs, K. (1990), "An improved method for estimating Lyapunov exponents of chaotic time series", *Phys. Lett. A*, 151, pp.27-32.
- [101] Bryant, P., R. Brown, H.D.I. Abarbanel (1990), "Lyapunov exponents from observed time series", *Phys. Rev. Lett.*, 65, pp.1523-1526.
- [102] Brown, R., P. Bryant, H.D.I. Abarbanel (1991), "Computing the Lyapunov spectrum of a dynamical system from an observed time series", *Phys. Rev. A*, 43, pp.2787-2806.
- [103] Abarbanel, H.D.I., R. Brown, M.B. Kennel (1991), "Lyapunov exponents in chaotic systems: their importance and their evaluation using observed data", *Int. J. Mod. Phys. B*, 5, pp.1347-1375.
- [104] Holzfuss, J., U. Parlitz (1991), "Lyapunov exponents from time series", Proceedings of the Conference *Lyapunov Exponents*, Oberwolfach 1990, eds. L. Arnold, H. Crauel, J.-P. Eckmann, in: *Lecture Notes in Mathematics*, Springer Verlag.
- [105] Parlitz, U. (1992), "Identification of true and spurious Lyapunov exponents from time series", *Int. J. Bifurcation and Chaos*, 2, pp.155-165.
- [106] Kadtke, J.B., J. Brush, J. Holzfuss (1993), "Global dynamical equations and Lyapunov exponents from noisy chaotic time series", *Int. J. Bifurcation Chaos*, 3, pp.607-616.
- [107] Gencay, R., W.D. Dechert (1992), "An algorithm for the  $n$  Lyapunov exponents of an  $n$ -dimensional unknown dynamical system", *Physica D*, 59, pp.142-157.
- [108] Eckmann, J.-P., D. Ruelle (1992), "Fundamental limitations for estimating dimensions and Lyapunov exponents in dynamical systems", *Physica D*, 56, pp.185-187.

- [109] Ellner, S., A.R. Gallant, D. McCaffrey, D. Nychka (1991), "Convergence rates and data requirements for Jacobian-based estimates of Lyapunov exponents from data", *Phys. Lett. A*, 153, pp.357-363.
- [110] Fell, J., J. Röschke, P. Beckmann (1993), "Deterministic chaos and the first positive Lyapunov exponent: a nonlinear analysis of the human electroencephalogram during sleep", *Biol. Cybern.*, 69, pp.139-146.
- [111] Fell, J., P. Beckmann (1994), "Resonance-like phenomena in Lyapunov calculations from data reconstructed by the time-delay method" *Phys. Lett. A*, 190, pp.172-176.
- [112] Sato, S., M. Sano, Y. Sawada (1987), "Practical methods of measuring the generalized dimension and largest Lyapunov exponent in high dimensional chaotic systems", *Prog. Theor. Phys.*, 77, pp.1-5.
- [113] Kurths, J., H. Herzel (1987), "An attractor in solar time series", *Physica D*, 25, pp.165-172.
- [114] Dämmig, M., F. Mitschke (1993), "Estimation of Lyapunov exponents from time series: the stochastic case", *Phys. Lett. A*, 178, pp.385-394.
- [115] Rosenstein, M.T., J.J. Collins, C.J. de Luca (1993), "A practical method for calculating largest Lyapunov exponents from small data sets", *Physica D*, 65, pp.117.
- [116] Kantz, H. (1994), "A robust method to estimate the maximal Lyapunov exponent of a time series", *Phys. Lett. A*, 185, pp.77-87.
- [117] Fujisaka, H., T. Yamada (1993), "Stability theory of synchronized motion in coupled-oscillator systems," *Prog. Theor. Phys.*, 69, pp.32-46.
- [118] Singer, W. (1993), "Synchronization of cortical activity and its putative role in information processing and learning," *Annu. Rev. Physiol.*, 55, pp.349-374.
- [119] Ashwin, P., J. Buescu, I. Stewart (1994), "Bubbling of attractors and synchronisation of chaotic oscillators," *Phys. Lett. A*, 193, pp.126-139.
- [120] Heagy, J.F., T.L. Carroll, L.M. Pecora (1994), "Synchronous chaos in coupled oscillator systems," *Phys. Rev. E*, 50, pp.1874-1885.
- [121] Lai, Y.-C., C. Grebogi (1994), "Synchronization of spatiotemporal chaotic systems by feedback control," *Phys. Rev. E*, 50, pp.1894-1899.
- [122] Collins, J.J., I. Stewart (1994), "A group-theoretic approach to rings of coupled biological oscillators," *Biol. Cybern.*, 71, pp.95-103.
- [123] Lindner, J.F., B.K. Meadows, W.L. Ditto, M.E. Inchiosa, A.R. Bulsara (1995), "Array enhanced stochastic resonance and spatiotemporal synchronization," *Phys. Rev. Lett.*, 75, pp.3-6.
- [124] Braiman, Y., W.L. Ditto, K. Wiesenfeld, M.L. Spano (1995), "Disorder-enhanced synchronization," *Phys. Lett. A* 206, pp.54-60; Braiman, Y., J.F. Lindner, W.L. Ditto (1995), "Taming spatiotemporal chaos with disorder," *Nature*, 378, pp.465-467.
- [125] Pecora, L.M., T.L. Carroll (1990), "Synchronization in chaotic systems," *Phys. Rev. Lett.*, 64, pp.821-824.
- [126] Brown, R., N.F. Rulkov, E.R. Tracy (1994), "Modelling and synchronizing chaotic systems from experimental data," *Phys. Lett. A*, 194, pp.71-76.
- [127] Kocarev, L., U. Parlitz (1995), "General approach for chaotic synchronization with applications to communication", *Phys. Rev. Lett.*, 74(25), pp.5028-5031.
- [128] Parlitz, U., L. Junge, L. Kocarev, (1996), "Synchronization based parameter estimation from time series", *Phys. Rev. E*, 54, pp.6253-6529.

- [129] Parlitz, U., L. Kocarev, T. Stojanovski, H. Preckel (1996), "Encoding messages using chaotic synchronization," *Phys. Rev. E*, 53(5), pp.4351-4361.
- [130] Rulkov, N.F., K.M. Sushchik, L.S. Tsimring, H.D.I. Abarbanel, (1995), "Generalized synchronization of chaos in directionally coupled chaotic systems," *Phys. Rev. E*, 51, pp.980-994.
- [131] Kocarev, L., U. Parlitz (1996), "Generalized synchronization, predictability and equivalence of uni-directionally coupled dynamical systems", *Phys. Rev. Lett.*, 76(11), pp.1816-1819.
- [132] Abarbanel, H.D.I., N.F. Rulkov, M.M. Sushchik (1996), "Generalized synchronization of chaos: The auxiliary system approach", *Phys. Rev. E*, 53(5), pp.4528-4535.
- [133] Parlitz, U., L. Junge, L. Kocarev (1997), "Subharmonic entrainment of unstable period orbits and generalized synchronization", *Phys. Rev. Lett.*, 79(17), pp.3158.
- [134] Parlitz, U., L. Kocarev (1998), "Synchronization of chaotic systems", in: *"Control of Chaos" Handbook* (Ed. H.-G. Schuster), WILEY-VCH.
- [135] Sirovich, L. (1989), 'Chaotic dynamics of coherent structures", *Physica D*, 37, pp. 126-145.
- [136] Rico-Martinez, R., K. Krischer, I.G. Kevrekidis, M.C. Kube, J.L. Hudson, (1992), "Discrete - vs. continuous-time nonlinear signal processing of Cu electrodisolution data," *Chem. Eng. Comm.* 118, pp.25-48.
- [137] Parlitz, U. & G. Mayer-Kress (1995), "Predicting low-dimensional spatiotemporal dynamics using discrete wavelet transforms", *Phys. Rev. E*, 51(4), pp.R2709-R2711.
- [138] H.D.I. Abarbanel, *Analysis of Observed Chaotic Data*, (Springer Verlag, New-York/Berlin/Heidelberg, 1996);
- [139] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman and A. Wu *An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions*, (Proc. of the Fifth Annual ACM-SIAM Symp. on Discrete Algorithms, 1994, pp. 573-582)
- [140] A. Belussi and C. Faloutsos *Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension*, (Conference Proceedings of VLDB, Zurich, Switzerland, Sept. 1995, pp. 299-310)
- [141] S. Berchtold, C. Böhm, D.A. Keim and H.P. Kriegel *A cost model for nearest neighbor search in high-dimensional data space*, (PODS'97, Tuscon, AZ, pp. 78-86)
- [142] P. Grassberger, R. Hegger, H. Kantz, C. Schaffrath and T. Schreiber *On noise reduction methods for chaotic data*, (Chaos, Vol. 3, Nr. 2, 1993, pp. 127-141)
- [143] T.C. Halsey, M.H. Jensen, L.P. Kadanoff, I. Procaccia, and B.I. Shraiman *Fractal measures and their singularities: The characterization of strange sets*, (Phys. Rev. A Vol. 33, Nr. 2, 1986, pp. 1141-1151)
- [144] H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis*, (Cambridge UP, Cambridge, 1997).
- [145] J. McNames *A Nearest Trajectory Strategy for Time Series Prediction*, (Proc. of the International Workshop on Advanced Black-box Techniques for Nonlinear Modeling, 1998, pp. 112-128)
- [146] U. Parlitz *Nonlinear Time-Series Analysis*, in (Nonlinear Modeling - Advanced Black-Box Techniques Eds. J.A.K. Suykens and J. Vandewalle Kluwer Academic Publishers, 1998, pp. 209-239)
- [147] V. Pestov *On the geometry of similarity search : dimensionality curse and contraction of measure*, (Maths and comp. science research report, 99-02, VUW, January 1999, pp. 7), submitted for publication
- [148] T. Schreiber *Efficient neighbor searching in nonlinear time-series analysis*, (Int. J. Bifurcation and Chaos, 5, pp. 349-358)

- [149] R. Sedgewick *Algorithms in C++, Third Edition*, (Addison-Wesley, 1998)
- [150] W. van de Water and P. Schram *Generalized dimensions from near-neighbor information*, (Phys. Rev. A, Vol. 37, Nr. 8, 1988, pp. 3318-3125)
- [151] L. F. Shampine and M. K. Gordon, “The initial value problem”