Cross Recurrence Plot Toolbox for Matlab

Transdisciplinary Concepts and Methods
Potsdam Institute for Climate Impact Research (PIK)

Reference Manual

Version 5.22, Release 33

Potsdam Institute for Climate Impact Research
P I K

General Information

How to get: The toolbox is freely available in the WorldWideWeb:
http://tocsy.pik-potsdam.de

How to install: After downloading the installation script `install.m`, simply change into the folder with this file and call the command `install` from the Matlab command line. The toolbox will be automatically created and added to the `startup.m` file.

How to deinstall: Just call the command `crpclean` from the Matlab command line. This will remove all files of the CRP toolbox from the filesystem and its entry from the Matlab startup file.

How to work: This toolbox was designed for Matlab, thus one needs to install Matlab before using this toolbox.

How to contact:

Norbert Marwan Telegraphenberg building A56, room 352 Potsdam Institute for Climate Impact Research (PIK) 14473 Potsdam Germany	**Visitors**
Norbert Marwan Potsdam Institute for Climate Impact Research (PIK) P. O. Box 60 12 03 14412 Potsdam Germany	**SnailMail**
+49 331 288 2614	**Phone**
+49 331 288 20738	**Fax**
marwan@pik-potsdam.de	**eMail**

Thanks: This toolbox has been developed in the project *Nonlinear Phase and Correlation Analysis of Palaeomagnetic and Palaeoclimatic Records* under the framework of the Priority Programme *Geomagnetic variations: Spatio-temporal variations, processes and impacts on the system Earth* of the German Science Foundation (Deutsche Forschungsgemeinschaft).

Remarks: The toolbox is still under development. We can not give any warranty for anything related with our programmes. Please send error messages or comments to our contact address.

Future releases: We plan to extend the toolbox to methods of phase analysis and graphical models. Send any wishes to our contact address.

About this document: This manual was prepared with LaTeX 2ε and using the modified refman-package.
Build: November 25, 2021

General Information

Warning: Any uncritical application of the methods included in this toolbox can yield to pitfalls. The users of these programmes are urged to inform theirself by the basics of nonlinear dynamics and the problems which can occur therein.

We give no warranties for the results obtained with the toolbox.

Theoretical Background: The toolbox programmes are based on modern methods of nonlinear data analysis. The main focus lies on (cross) recurrence plots and their quantification. (Cross) Recurrence plots are briefly defined as

$$\mathbf{R}_{i,j}^{m,\varepsilon_i} = \Theta\left(\varepsilon_i - \left\|\vec{x}_i - \vec{x}_j\right\|\right) \quad \text{or} \quad \mathbf{CR}_{i,j}^{m,\varepsilon_i} = \Theta\left(\varepsilon_i - \left\|\vec{x}_i - \vec{y}_j\right\|\right),$$

where ε is a predefined threshold and $\vec{x}_i$ $\vec{y}_i$ are phase space trajectories in an m-dimension phase space (Eckmann and Ruelle, 1987; Marwan and Kurths, 2002). These trajectories can be reconstructed from single time series u_i by using a time delay τ (Takens, 1981) $\hat{\vec{x}}_i = (u_i, u_{i+\tau}, \ldots, u_{i+(m-1)\tau})^T$. The base of a recurrence plot is the distance matrix

$$\mathbf{D}_{i,j}^{m,\varepsilon_i} = \left\|\vec{x}_i - \vec{x}_j\right\|.$$

Another kind of recurrence plot is based on an order pattern representation of the data and is called order patterns recurrence plot (Groth, 2004). There are $m!$ order patterns π_i, for example for $m = 2$ we have

$$\pi_i = \begin{cases} 0 & \text{for } u_i < u_{i+\tau} \\ 1 & \text{for } u_i > u_{i+\tau}. \end{cases}$$

An order pattern recurrence plot is then defined as

$$\mathbf{R}_{i,j}^{m,\varepsilon_i} = \delta\left(\pi_i^x, \pi_j^y\right),$$

and should not be confused with the order matrix

$$\mathbf{O}_{i,j}^{m,\varepsilon_i} = \Theta\left(\left\|u_i - u_j\right\|\right) = \begin{cases} 0 & \text{for } u_i < u_j \\ 1 & \text{for } u_i > u_j. \end{cases}$$

The definition of the order matrix and order patterns recurrence plot can, of course, extended to the bivariate case analogous to the cross recurrence plot.

Several quantification approaches can be applied; the most common are **recurrence rate** (Marwan et al., 2007)

$$RR = \frac{1}{N^2} \sum_{i,j=1}^{N} \mathbf{R}_{i,j}^{m,\varepsilon},$$

determinism

$$DET = \frac{\sum_{l=l_{min}}^{N} l\, P^{\varepsilon}(l)}{\sum_{i,j}^{N} \mathbf{R}_{i,j}^{m,\varepsilon}},$$

(where $P^{\varepsilon}(l) = \{l_i;\ i = 1 \ldots N_l\}$ is the frequency distribution of the lengths l of diagonal structures and N_l is the absolute number of diagonal lines);

General Information

Lmax and **divergence**

$$L_{max} = \max\left(\{l_i;\ i = 1 \dots N_l\}\right) \quad \text{respective} \quad DIV = \frac{1}{L_{max}},$$

entropy

$$ENTR = -\sum_{l=l_{min}}^{N} p(l) \ln p(l) \quad \text{with} \quad p(l) = \frac{P^{\varepsilon}(l)}{\sum_{l=l_{min}}^{N} P^{\varepsilon}(l)},$$

laminarity (Marwan et al., 2002)

$$LAM = \frac{\sum_{v=v_{min}}^{N} v P^{\varepsilon}(v)}{\sum_{v=1}^{N} v P^{\varepsilon}(v)},$$

(where $P^{\varepsilon}(v) = \{v_i;\ i = 1 \dots N_v\}$ denotes the frequency distribution of the lengths l of vertical structures)

trapping time

$$TT = \frac{\sum_{v=v_{min}}^{N} v P^{\varepsilon}(v)}{\sum_{v=v_{min}}^{N} P^{\varepsilon}(v)},$$

recurrence times of first type (Gao and Cai, 2000)

$$T_j^1 = \left|\left\{i, j : \vec{x}_i, \vec{x}_j \in \mathcal{R}_i\right\}\right|,$$

recurrence times of second type

$$T_j^2 = \left|\left\{i, j : \vec{x}_i, \vec{x}_j \in \mathcal{R}_i;\ \vec{x}_{j-1} \notin \mathcal{R}_i\right\}\right|$$

(where $\mathcal{R}_i$ are the recurrence points which belong to the state $\vec{x}_i$).

Further quantifiers are based on complex network theory, as **clustering coefficient** (Marwan et al., 2009)

$$C = \sum_{i=1}^{N} \frac{\sum_{j,k=1}^{N} \mathbf{R}_{i,j}^{m,\varepsilon} \mathbf{R}_{j,k}^{m,\varepsilon} \mathbf{R}_{k,i}^{m,\varepsilon}}{RR_i}$$

with $RR_i = \sum_{j=1}^{N} \mathbf{R}_{i,j}^{m,\varepsilon}$ the local recurrence rate, or

transitivity

$$C = \frac{\sum_{i,j,k=1}^{N} \mathbf{R}_{i,j}^{m,\varepsilon} \mathbf{R}_{j,k}^{m,\varepsilon} \mathbf{R}_{k,i}^{m,\varepsilon}}{\sum_{i,j,k=1}^{N} \mathbf{R}_{i,j}^{m,\varepsilon} \mathbf{R}_{k,i}^{m,\varepsilon}}.$$

Above definitions are for the entire recurrence plot (or for squared windows in it, reveiling some time dependencies). But most of these measures can be quantified for each diagonal line (parallel to the main diagonal) as well, which is even interesting for cross recurrence plots, for example

General Information

$$RR_k = \frac{1}{N-k} \sum_{j-i=k} \mathbf{CR}_{i,j}^{m,\varepsilon} = \frac{1}{N-k} \sum_{l=1}^{N-k} l\, P_k^{\varepsilon}(l)$$

is the recurrence rate of the kth diagonal line in the cross recurrence plot ($P_k^{\varepsilon}(l) = \{l_i;\ i = 1 \ldots N_l\}$ defines the frequency distribution of diagonal line lengths for the kth diagonal line where $k = j - i$ in $\mathbf{CR}_{i,j}^{m,\varepsilon}$).

Moreover, dynamical invariants can be estimated by using recurrence plots. At the moment, they are not yet included in this toolbox. For more details see Marwan et al. (2007).

The following literature is highly recommended to get introduced into nonlinear dynamics, recurrence plots and related methods and to avoid wrong usage or misinterpretation.

1. ECKMANN, J.-P., Ruelle, D.: Ergodic theory of chaos and strange attractors, Review of Modern Physics, 57(3), 1985, 617–656.
2. GAO, J., Cai, H.: On the structures and quantification of recurrence plots, Physics Letters A, 270, 2000, 75–87. DOI:10.1016/S0375-9601(00)00304-2
3. GROTH, A.: Visualization of coupling in time series by order recurrence plots, Physical Review E, 72(4), 046220 (2005). DOI:10.1103/PhysRevE.72.046220
4. MARWAN, N., Kurths, J.: Nonlinear analysis of bivariate data with cross recurrence plots, Physics Letters A, 302, 2002, 299–307. DOI:10.1016/S0375-9601(02)01170-2
5. MARWAN, N., Wessel, N., Meyerfeldt, U., Schirdewan, A., Kurths, J.: Recurrence Plot Based Measures of Complexity and its Application to Heart Rate Variability Data, Physical Review E, 66(2), 2002, 026702. DOI:10.1103/PhysRevE.66.026702
6. MARWAN, N., Romano, M. C., Thiel, M., Kurths, J.: Recurrence Plots for the Analysis of Complex Systems, Physics Reports, 438(5–6), 2007, 237–329. DOI:10.1016/j.physrep.2006.11.001
7. MARWAN, N., Donges, J. F., Zou, Y., Donner, R. V., Kurths, J.: Complex network approach for recurrence analysis of time series, Phys. Lett. A, 373(46), 2009. DOI:10.1016/j.physleta.2009.09.042
8. THIEL, M., Romano, M. C., Kurths, J.: Influence of observational noise on the recurrence quantification analysis, Physica D, 17(3), 2002, 138–152. DOI:10.1016/S0167-2789(02)00586-9
9. TRULLA, L. L., Giuliani, A., Zbilut, J. P., Webber Jr., C. L.: Recurrence quantification analysis of the logistic equation with transients, Physics Letters A, 223, 1996, 255–260. DOI:10.1016/S0375-9601(96)00741-4
10. ZBILUT, J. P., Marwan, N.: The Wiener-Khinchin theorem and recurrence quantification, Phys. Lett. A, 372, 2008. DOI:10.1016/j.physleta.2008.09.027

General Information

GNU General Public License

PREAMBLE

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

General Information

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

    c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

    These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

    Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

    In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

    The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

    If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

    NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ace

Purpose Finds optimal transformation and maximal correlation.

Syntax
```
mcor=ace(x,y,[w,ii,oi])
[theta, phi]=ace(x,y,[,w,ii,oi])
[theta, phi, mcor]=ace(x,y,[,w,ii,oi])
[theta, phi, mcor, i, o, imax, omax]=ace(x,y,[,w,ii,oi])
ace(...)]
```

Description Estimates the optimal transformations of the system `theta(x)=phi(x)` and computes the resulting maximal correlation `mcor`, where `x` is a one-column vector and `y` can be a multi-column vector.

`[theta, phi, mcor, i, o, imax, omax]=ace(x,y [,w,ii,oi])` estimates the optimal transformations `theta`, `phi` and the maximal correlation `mcor` and outputs the number of inner iterations `i`, break-up number of inner inner iterations, number of outer iterations `o` and break-up number of outer inner iterations. If the algorithm doesn't converge, the number of iterations will be negative signed.

Without output arguments, `ace` plots the optimal transformations `theta` and `phi`.

Parameters `w` is the half-length of the boxcar window, `ii` is the maximal number of inner iterations, `oi` is the minimal number of outer iterations.

Examples
```
x=(-1:.002:1)+.3*rand(1,1001);
y=(-1:.002:1).^2+.3*rand(1,1001);
corrcoef(x,y)
ace(y,x)
```

See Also `mcf`

References
Breiman, L., Friedman, J. H.: Estimating Optimal Transformations for Multiple regression and Correlation, J. Am. Stat. Assoc., 80(391), 1985.

Voss, H., Kurths, J.: Reconstruction of nonlinear time delay models from data by the use of optimal transformations, Phys. Lett. A, 234, 1997.

adjust

Purpose Adjusts two two-column vectors.

Syntax `[x, y]=adjust(a,b,flag)`

Description Adjusts the two-column vectors `a` and `b` to the same time scale (in first column), whereby using the `flag`, the following methods for adjustment can be choosen:

`0`	– (default) adjustment by cutting.
`1`	– adjustment by cubic interpolating.
`-1`	– adjustment by cubic interpolating and forced length (given by `A`).
`2`	– gap filling by histogram estimation (experimental status).
`3`	– gap filling by AR(p) estimation (experimental status).

Except for `flag=0`, `x` and `y` will have the same length.

Examples
```
x=(1:110)';
y1=x(11:end); y1(:,2)=sin(x(11:end)/10);
y2=x(1:100)/2; y2(:,2)=sin(x(1:100)/5);
[z1 z2]=adjust(y1,y2);
```

arfit

Purpose AR parameter estimation via Yule-Walker method.

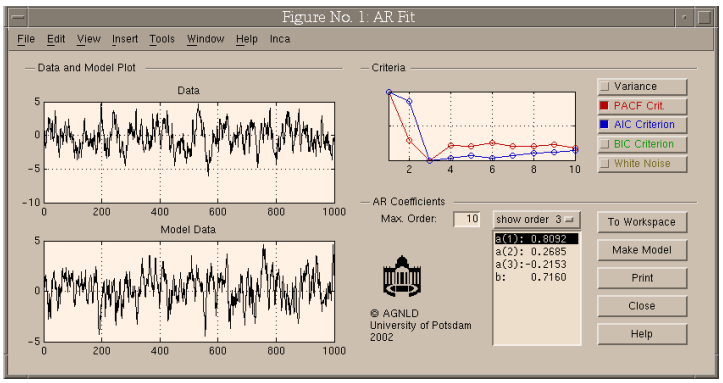
Syntax
```
arfit(x,p)
a=arfit(x,p)
[a y]=arfit(x,p)
```

Description `arfit(x,p)` opens a GUI for AR coefficients estimation for the vector `x` using the Yule-Walker method. The coefficients and order selection criterias for all orders until the maximal order `p` will be solved. The coefficients are solved by the Levinson- Durbin algorithmus. The criteria are normalized in order to show them in the same plot.

`a=arfit(x,p)` returns the vector `a` of length `(p+1)` of the AR coefficients and the noise level for the corresponding AR model of the model order `p`. The GUI is suppressed.

`[a y]=arfit(x,p)` returns the vector `y` of a realization of the resulting AR model. The GUI is suppressed.

Example
```
x=rand(3,1);
a=[.8 .3 -.25 .9]';
for i=4:1000,
  x(i)=sum(a(1:3).*x(i-1:-1:i-3))+a(end)*randn;
end
arfit(x,10)
```



corrgram

Purpose Calculate windowed cross correlation between two signals.

Syntax
```
c = corrgram(a,b,maxlag,window,noverlap,method)
[c,l,t] = corrgram(...)
c = corrgram(a,b)
corrgram(a,b)
```

Description `c = corrgram(a,b,maxlag,window,noverlap)` calculates the windowed cross correlation between the signals in vector `a` and vector `b`. `corrgram` splits the signals into overlapping segments and forms the columns of `c` with their cross correlation values up to maximum lag specified by scalar `maxlag`. Each column of `c` contains the cross correlation function between the short-term, time-localized signals `a` and `b`. Time increases linearly across the columns of `c`, from left to right. Lag increases linearly down the rows, starting at `-maxlag`. If lengths of `a` and `b` differ, the shorter signal is filled with zeros. If `n` is the length of the signals, `c` is a matrix with `2*maxlag+1` rows and

```
k = fix((n-noverlap)/(window-noverlap))
```

columns.

`c = corrgram(...,method)` using either Pearson correlation ('pearson', default), Spearman's rank correlation ('spearman'), or Kendall's Tau ('kendall').

`[c,l,t] = corrgram(...)` returns a column of lag `L` and one of time `T` at which the correlation coefficients are computed. `L` has length equal to the number of rows of `c`, `T` has length `k`.

`c = corrgram(a,b)` calculates windowed cross correlation using defeault settings; the defeaults are `maxlag = floor(0.1n)`, `window = floor(0.1*n)` and `noverlap = 0`. You can tell `corrgram` to use the defeault for any parameter by leaving it off or using `[]` for that parameter, e.g. `corrgram(a,b,[],1000)`.

`corrgram(a,b)` with no output arguments plots the windowed cross correlation using the current figure.

Example
```
x = cos(0:.01:10*pi)';
y = sin(0:.01:10*pi)' + .5 * randn(length(x),1);
corrgram(x,y)
```

`corrcoeff, corr, xcorr, migram`

crp

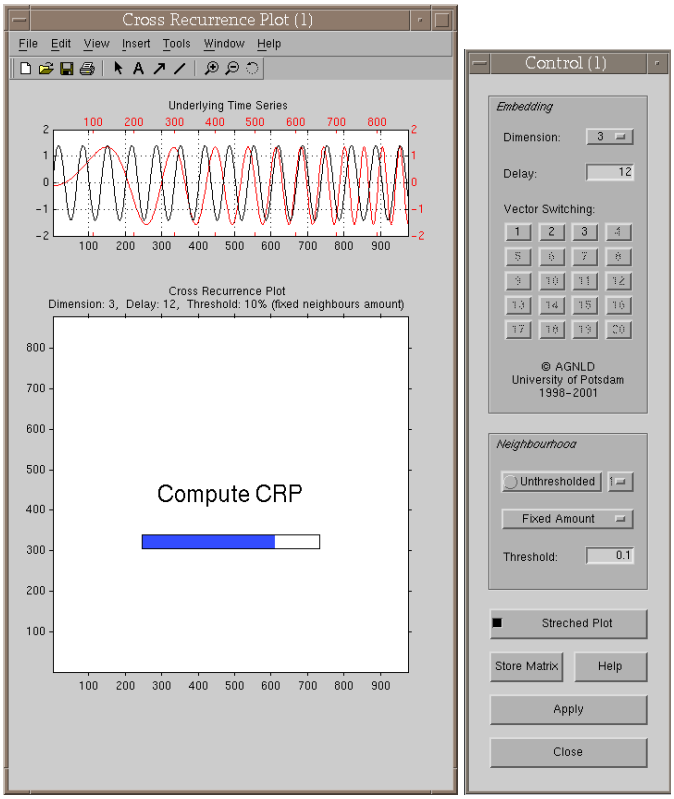
Purpose Creates a cross recurrence plot/ recurrence plot.

Syntax
```
crp(x)
crp(x,y)
crp(x,m,t,e)
r=crp(x,[],m,t,e)
r=crp(x,m,t,e,'param1','param2',...)
r=crp(x,y,m,'param1')
```

Description Creates a cross recurrence plot/ recurrence plot, order patterns recurrence plot as well as a distance matrix/ order matrix. Results can be stored into the workspace.

Allows to change the parameters interactively by using a GUI.

The source-data `x` and test-data `y` can be one- or a two-coloumn vectors (then, in the first column have to be the time); if the test-data `y` is not specified, a simple (auto) recurrence plot is created.



Parameters Dimension `m`, delay `t` and the size of neighbourhood `e` are the first three numbers after the data series; further parameters can be used to switch between various methods of finding the neighbours of the phasespace trajectory, to suppress the normalization of the data and to suppress the GUI (useful in order to use this programme by other programmes).

Methods of finding the neighbours/ of plot.

`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm.
`'minnorm'`	– Minimum norm.
`'nrmnorm'`	– Euclidean norm between normalized vectors (all vectors have the length one).
`'rr'`	– Maximum norm, fixed recurrence rate.
`'fan'`	– Fixed amount of nearest neighbours.
`'inter'`	– Interdependent neighbours.
`'omatrix'`	– Order matrix.
`'opattern'`	– Order patterns recurrence plot.
`'distance'`	– Distance coded matrix (global CRP, Euclidean norm).

Normalization of the data series.

`'normalize'`	– Normalization of the data.
`'nonormalize'`	– No normalization of the data.

Suppressing the GUI.

`'gui'`	– Creates the GUI and the output plot.
`'nogui'`	– Suppresses the GUI and the output plot.
`'silent'`	– Suppresses all output.

Parameters not needed to be specified.

Limitations For higher speed in output the whole matrix of the recurrence plot is in the work space – this limits the application of long data series. However, with a little Matlab script, long data series can be handled too (cf. Examples).

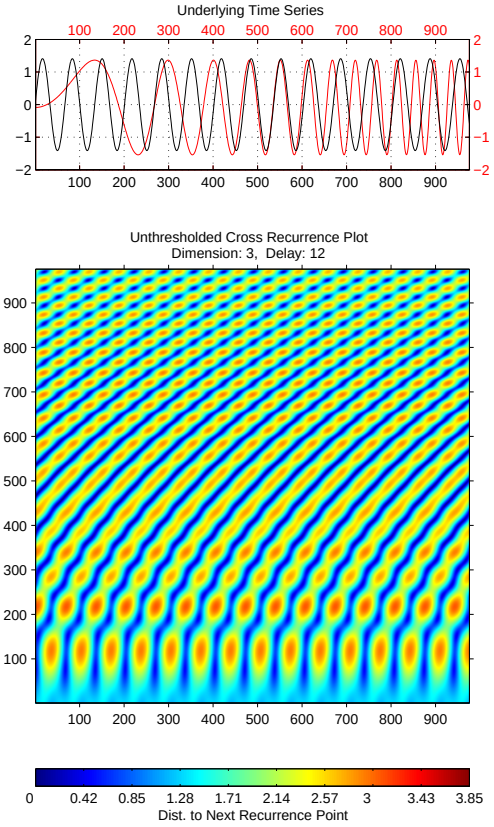
Examples

```
a=sin((1:1000)*2*pi/67);
crp(a,'nonorm','euclidean')

X=crp(a,2,50,.1,'nogui');
spy(double(X))

b=sin(.01*([1:1000]*2*pi/67).^2);
crp(a,b,3,12,'distance')
```

crp



For computing RPs/ CRPs of long data series, use a similar script as in the following. The data length is finally limited by the used platform performance. The examples also illustrate the capabality of using the programme in a script. The first example uses sparse matrices (when calculating a recurrence plot of only one time series, replace x2 by x1):

```
%% some parameter settings and create an example time series
m=3; t=20; e=.5; w=300;
x1=sin((1:5000)/40)'; x2=sin((1:7000)/80)';

clear Y, Y=spalloc(length(x2)-(m-1)*t,length(x1)-(m-1)*t,1);
k=0; h1=waitbar(0,'Compute sub CRPs - Please be patient.');
Nx=length(x1)-(m-1)*t; Ny=length(x2)-(m-1)*t;
ax=ceil(Nx/w); ay=ceil(Ny/w);
Nx2=floor(Nx/ax); Ny2=floor(Ny/ay);

%% compute single CRPs and fill the sparse matrix
for i=1:Nx2:Nx-Nx2;
  for j=1:Ny2:Ny-Ny2, k=k+1; waitbar(k/(Nx*Ny/(Nx2*Ny2)))
    X2=crp(x1(i:i+Nx2+(m-1)*t),x2(j:j+Ny2+(m-1)*t),m,t,e,...
           'nonorm','max','silent');
    X=sparse(double(X2));
    Y(j:j+Ny2-1,i:i+Nx2-1)=X(1:Ny2,1:Nx2);
  end
end
close(h1)

spy(Y)
```

The second example writes single RPs/ CRPs to the hard disk (when calculating a recurrence plot of only one time series, replace x2 by x1):

```
%% some parameter settings and create an example time series
m=3; t=20; e=.5; w=300;
x1=sin((1:5000)/40)'; x2=sin((1:7000)/80)';
Nx=length(x1); Ny=length(x2);

%% compute single CRPs and write them to the hard disk
b1=zeros((m-1)*t+ceil(length(x1)/w)*w,1);
b1(1:length(x1))=x1;
b2=zeros((m-1)*t+ceil(length(x2)/w)*w,1);
b2(1:length(x2))=x2;
h=waitbar(0,'Compute sub CRPs - Please be patient.')
for i=1:w:length(b1)-w-1, waitbar(i/((length(b1)-w-1)))
  for j=1:w:length(b2)-w-1,j
    X=crp(b1(i:i+w+(m-1)*t-1),b2(j:j+w+(m-1)*t-1),m,t,e,...
          'max','silent','nonorm');
    i2=num2str((i+w-1)/w);j2=num2str((j+w-1)/w);
    filename=['CRP_',i2,'_',j2,'.tif'];
    imwrite(X,filename,'tif')
  end
end, close(h)

%% read single CRPs and unify them
xmax=(i+w-1)/w; ymax=(j+w-1)/w;
Y = zeros(length(b1),length(b2));
h=waitbar(0,'Read sub CRPs - Please be patient.');
for i=1:xmax,waitbar(i/xmax)
  for j=1:ymax,
    i2=num2str(i);j2=num2str(j);
    filename=['CRP_',i2,'_',j2,'.tif'];
    X=imread(filename,'tif');
    Y(i*w-(w-1):i*w,j*w-(w-1):j*w)=(X)';
  end
end, close(h)

Y(Nx+1:end,:)=[]; Y(:,Ny+1:end)=[];
spy(double(Y))
```

See Also `crp2`, `crp_big`, `crqa`

crp2

Purpose Creates a cross recurrence plot/ recurrence plot and computes the line of synchronization.

Syntax
```
crp2(x)
crp2(x,y)
crp2(x,m,t,e)
r=crp2(x,[],m,t,e)
r=crp2(x,m,t,e,'param1','param2',...)
r=crp2(x,y,m,'param1')
```

Description Creates a cross recurrence plot/ recurrence plot, order patterns recurrence plot as well as a distance matrix from the embedding vectors `x` and `y`. Results can be stored into the workspace. Further it is possible to estimate the line of synchronization (LOS) in order to get the nonparametric time-relationship between the two considered systems.

Allows to change the parameters interactively by using a GUI.

The embedding dimension `m` is given by the size of the `n` $\times$ `m` matrix `x` and `y`; if the matrix `y` is not specified, a simple (auto) recurrence plot is created.

Parameters Additionally dimension `m`, delay `t` and the size of neighbourhood `e` are the first three numbers after the data series; further parameters can be used to switch between various methods of finding the neighbours of the phasespace trajectory, to suppress the normalization of the data and to suppress the GUI (useful in order to use this programme by other programmes).

Methods of finding the neighbours.

`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm.
`'minnorm'`	– Minimum norm.
`'nrmnorm'`	– Euclidean norm between normalized vectors (all vectors have the length one).
`'rr'`	– Maximum norm, fixed recurrence rate.
`'fan'`	– Fixed amount of nearest neighbours.
`'omatrix'`	– Order matrix (disabled).
`'opattern'`	– Order patterns recurrence plot.
`'distance'`	– Distance coded matrix (global CRP, Euclidean norm).

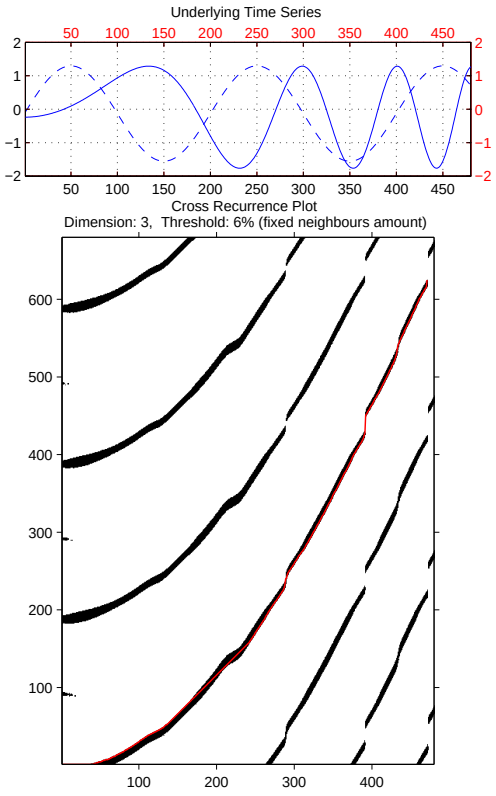
Normalization of the data series.
'normalize' – Normalization of the data.
'nonormalize' – No normalization of the data.
Suppressing the GUI.
'gui' – Creates the GUI and the output plot.
'nogui' – Suppresses the GUI and the output plot.
'silent' – Suppresses all output.

Parameters not needed to be specified.

Limitations For higher speed in output the whole matrix of the recurrence plot is in the work space – this limits the application of long data series. However, a solution for using long data you can find under the description for `crp`.

Examples
```
a=sin((1:1000)*2*pi/200);    % pendulum's location vector
b=cos((1:1000)*2*pi/200);    % pendulum's velocity vector
crp2(a(1:500),b(1:500),'nonorm','euclidean')

b=sin(.01*([1:1000]*2*pi/67).^2);
crp2(b(1:500),a(1:700),3,10,.06,'fan')
```



See Also `crp, crp_big and trackplot`

References Marwan, N., Thiel, M., Nowaczyk, N.: Cross Recurrence Plot Based Synchronization of Time Series, Nonlin. Proc. Geophys., 9, 2002.

crp_big

Purpose Creates a cross recurrence plot/ recurrence plot.

Syntax
```
crp_big(x)
crp_big(x,y)
crp_big(x,m,t,e)
r=crp(x,[],m,t,e)
r=crp(x,m,t,e,'param1','param2',...)
r=crp(x,y,m,'param1')
```

Description Creates a cross recurrence plot/ recurrence plot, order patterns recurrence plot as well as a distance matrix/ order matrix. In contrast to CRP, long data series can be used. Results can be stored into the workspace.

Allows to change the parameters interactively by using a GUI.

The source-data `x` and test-data `y` can be one- or a two-coloumn vectors (then, in the first column have to be the time); if the test-data `y` is not specified, a simple (auto) recurrence plot is created.

Parameters Dimension `m`, delay `t` and the size of neighbourhood `e` are the first three numbers after the data series; further parameters can be used to switch between various methods of finding the neighbours of the phasespace trajectory, to suppress the normalization of the data and to suppress the GUI (useful in order to use this programme by other programmes).

Methods of finding the neighbours/ of plot.

`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm.
`'minnorm'`	– Minimum norm.
`'nrmnorm'`	– Euclidean norm between normalized vectors (all vectors have the length one).
`'rr'`	– Maximum norm, fixed recurrence rate.
`'fan'`	– Fixed amount of nearest neighbours.
`'inter'`	– Interdependent neighbours.
`'omatrix'`	– Order matrix.
`'opattern'`	– Order patterns recurrence plot.
`'distance'`	– Distance coded matrix (global CRP, Euclidean norm).

Normalization of the data series.

`'normalize'`	– Normalization of the data.
`'nonormalize'`	– No normalization of the data.

Suppressing the GUI.

`'gui'`	– Creates the GUI and the output plot.
`'nogui'`	– Suppresses the GUI and the output plot.
`'silent'`	– Suppresses all output.

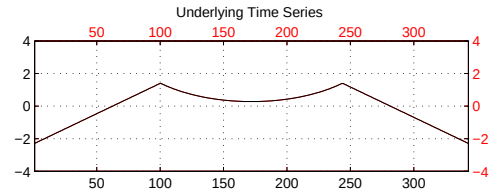
Parameters not needed to be specified.

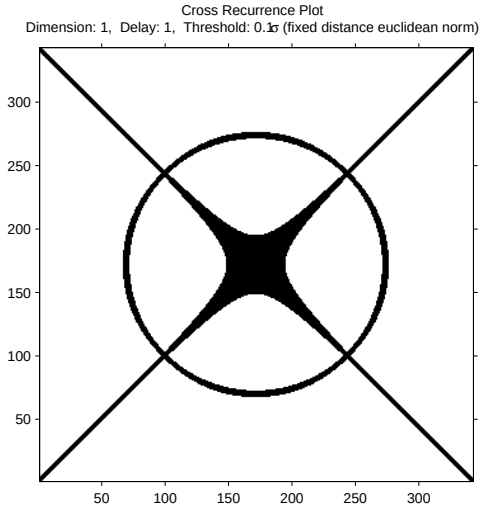
Limitations In contrast to `crp` and `crp2`, this command allows to work with longer data series. The algorithm computes the CRP piecewise. However, the possibility to store the CRP in the workspace limits the length of data series again. However, a solution for using long data you can find under the description for `crp`.

crp_big

Examples

```
a=sqrt(100^2-(-71:71).^2); b=1:100;
b(101:100+length(a))=-(a)+170;
b(end+1:end+100)=100:-1:1;
crp_big(b,1,1,.1,'euclidean')
```





See Also `crp, crp2 and crqa`

crqa

Purpose Computes and plots the CRQA measures.

Syntax
```
crqa(x)
crqa(x,y)
y=crqa(x,y,m,t,e,w,ws)
y=crqa(x,y,m,t,e,w,ws,lmin,vmin)
y=crqa(x,y,m,t,e,w,ws,lmin,vmin,tw)
y=crqa(x,y,m,t,e,[],'param1','param2',...)
```

Description Recurrence quantification analysis of cross-recurrence with the first vector `x` and the second `y`.

The input vectors can be multi-column vectors, where each column will be used as a component of the phase-space vector. However, if the first column is monotonically increasing, it will be used as an time scale for plotting.

Parameters `CRQA(...)` without any output arguments opens a GUI for interactively control the CRQA. If an output is specified with using the option `'gui'`, then the output will contain the figure handle.

Dimension `m`, delay `t`, the size of neighbourhood `e`, the window size `w` and the shift value `ws` are the first five numbers after the data series; if `w=[]` then the whole plot will be calculated. The minimal length of diagonal and vertical structures can be specified with `lmin` and `vmin` respectively (default is 2).

As the last numeric parameter, the size of the Theiler window `tw` can be specified (default is 1). This window excludes the recurrence points parallel to the main diagonal from the analysis. The application of the Theiler window is useful only for recurrence plots. In cross recurrence plots, the size of the Theiler window will be set automatically to zero.

Further parameters can be used to switch between various methods of finding the neighbours of the phasespace trajectory, to suppress the normalization of the data, to apply a correction schema for border lines, and to suppress the GUI (useful in order to use this programme by other programmes).

Methods of finding the neighbours.

`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm.
`'minnorm'`	– Minimum norm.
`'nrmnorm'`	– Euclidean norm between normalized vectors (all vectors have the length one).
`'rr'`	– Maximum norm, fixed recurrence rate.
`'fan'`	– Fixed amount of nearest neighbours.
`'inter'`	– Interdependent neighbours.
`'omatrix'`	– Order matrix.
`'opattern'`	– Order patterns recurrence plot.

Normalization of the data series.

`'normalize'`	– Normalization of the data.
`'nonormalize'`	– No normalization of the data.

Methods regarding considering border lines.

`'all'`	– (Default) Considers all individual lengths of border lines (most commonly used approach until 2021)
`'censi'`	– Correction schema for border lines as proposed by Censi et al. 2004, in which the length of the longest border line is used for all border lines (recommended for cyclical signals).
`'kelo'`	– Correction schema for border lines using KEep LOngest diagonal line (KELO), in which only the longest border line (in each triangle) of the RP is considered but all other border lines are discarded.

Suppressing the GUI.

`'gui'`	– Creates the GUI.
`'nogui'`	– Suppresses the GUI.
`'silent'`	– Suppresses all output.

Output

`y(:,1)`	– Recurrence rate.
`y(:,2)`	– Determinism.
`y(:,3)`	– Averaged diagonal length.
`y(:,4)`	– Length of longest diagonal line.
`y(:,5)`	– Entropy of diagonal length.
`y(:,6)`	– Laminarity.
`y(:,7)`	– Trapping time.
`y(:,8)`	– Length of longest vertical line.
`y(:,9)`	– Recurrence time of 1st type.
`y(:,10)`	– Recurrence time of 2nd type.
`y(:,11)`	– Recurrence period density entropy.
`y(:,12)`	– Clustering coefficient.
`y(:,13)`	– Transitivity.

Parameters not needed to be specified.

The window of length w is applied on the data and not on the RP, i. e. the RP will have smaller size than the window, thus $w-(m-1)\tau$. If we consider the data window at time $i \dots i+w$, the corresponding RQA measures are assigned to time i. Therefore, if you see a beginning of a transition in the plot of the RQA measures at time i, this transition will probably happen at time $i+w-(m-1)\tau$.

Limitations For higher speed in output the whole matrix of the recurrence plot is in the work space – this limits the application of long data series. However, a solution for using long data you can find under the description for `crp`.

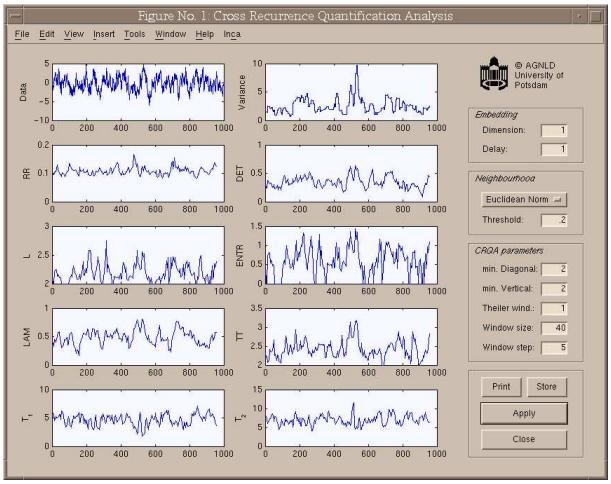
Warning The RQA measures may differ from those of the RQA programmes by Charles Webber Jr. For compatibility use a Theiler window of size one and ensure that the data are normalized before by the same distance which is used in the RQA programmes; e. g. normalize with the maximal phase space diameter, which can be estimated with the programme `pss`:

```
RQA=crqa(100*x/pss(x,dim,lag,'euclidean'),...
         dim,lag,e,[],[],l_min,v_min,1,...
         'euclidean','nonormalize','silent')
```

crqa

Examples

```
a=randn(300,1);
crqa(a,1,1,.2,40,2,'euc')
```




```
N=300; w=40; ws=2;
a=3.4:.6/(N-1):4;
b=.5; for i=2:N, b(i)=a(i)*b(i-1)*(1-b(i-1));end
y=crqa(b,3,2,.1,w,ws);
subplot(2,1,1), plot(a,b,'.','markersize',.1)
title('logistic map'), axis([3.4 4 0 1])
subplot(2,1,2), plot(a(1:ws:N-w),y(1:ws:N-w,1))
ylabel('recurrence rate'), axis([3.4 4 0 1])
```

See Also `crqad, crp, dl, tt, pss`

References

Marwan, N., Romano, M. C., Thiel, M., Kurths, J.: Recurrence Plots for the Analysis of Complex Systems, Phys. Rep., 438, 2007.

Little, M., McSharry, P., Roberts, S., Costello, D., Moroz, I.: Exploiting Nonlinear Recurrence and Fractal Scaling Properties for Voice Disorder Detection, Biomed. Eng. Online, 6, 2007.

Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., Hwang, D.-U.: Complex networks: Structures and dynamics, Phys. Rep., 424, 2006.

Marwan, N., Donges, J. F., Zou, Y., Donner, R. V., Kurths, J.: Complex network approach for recurrence analysis of time series, Phys. Lett. A, 373(46), 2009.

Kraemer, K. H., Marwan, N.: Border effect corrections for diagonal line based recurrence quantification analysis measures, Phys. Lett. A, 383, 2019.

crqad

Purpose Computes and plots the diagonalwise CRQA measures.

Syntax
```
crqad(x)
crqad(x,y)
y=crqad(x,y,m,t,e,w,lmin)
y=crqad(x,y,m,t,e,[],'param1','param2',...)
```

Description Recurrence quantification analysis of diagonals in the cross recurrence plot of the vectors `x` and `y` as well as `x` and `-y` for the diagonals within the range `[-w,w]` around the main diagonal. The output is a structure (see below).

Parameters Dimension `m`, delay `t`, the size of neighbourhood `e` and the window size `w` are the first five numbers after the data series; if `w=[]` then the whole plot will be calculated. Variable `lmin` sets the minimal length of what should be considered to be a diagonal line. Further parameters can be used to switch between various methods of finding the neighbours of the phasespace trajectory, to suppress the normalization of the data and to suppress the GUI (useful in order to use this programme by other programmes). The minimal length of diagonal and vertical structures can be setted only in the GUI.

Methods of finding the neighbours.

`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm.
`'minnorm'`	– Minimum norm.
`'nrmnorm'`	– Euclidean norm between normalized vectors (all vectors have the length one).
`'rr'`	– Maximum norm, fixed recurrence rate.
`'fan'`	– Fixed amount of nearest neighbours.
`'inter'`	– Interdependent neighbours.
`'omatrix'`	– Order matrix.
`'opattern'`	– Order patterns recurrence plot.

Normalization of the data series.

`'normalize'`	– Normalization of the data.
`'nonormalize'`	– No normalization of the data.

Suppressing the GUI.

`'gui'`	– Creates the GUI.
`'nogui'`	– Suppresses the GUI.
`'silent'`	– Suppresses all output.

Output

`y.RRp`	– Recurrence rate `(x,y)`.
`y.RRp`	– Recurrence rate `(x,-y)`.
`y.DETp`	– Determinism `(x,y)`.
`y.DETm`	– Determinism `(x,-y)`.
`y.Lp`	– Averaged diagonal length `(x,y)`.
`y.Lm`	– Averaged diagonal length `(x,-y)`.

Parameters not needed to be specified.

crqad

Limitations For higher speed in output the whole matrix of the recurrence plot is in the work space – this limits the application of long data series. However, a solution for using long data you can find under the description for `crp`.

Examples
```
a=sin(0:.1:80)+randn(1,801);
b=sin(0:.1:80)+randn(1,801);
crqad(a,b,3,15,.1,100,'fan')
```

See Also `crqad_big, crqa, crp, dl, tt`

References Marwan, N., Kurths, J.: Nonlinear analysis of bivariate data with cross recurrence plots, Phys. Lett. A, 302, 2002.

crqad_big

Purpose Computes and plots the diagonalwise CRQA measures of long data series.

Syntax
```
crqad_big(x)
crqad_big(x,y)
y=crqad_big(x,y,m,t,e,w,lmin)
y=crqad_big(x,y,m,t,e,[],'param1','param2',...)
```

Description Recurrence quantification analysis of diagonals in the cross recurrence plot of the vectors `x` and `y` as well as `x` and `-y` for the diagonals within the range `[-w,w]` around the main diagonal. The output is a structure (see below).

Parameters Dimension `m`, delay `t`, the size of neighbourhood `e` and the window size `w` are the first five numbers after the data series; if `w=[]` then the whole plot will be calculated. Variable `lmin` sets the minimal length of what should be considered to be a diagonal line. Further parameters can be used to switch between various methods of finding the neighbours of the phasespace trajectory, to suppress the normalization of the data and to suppress the GUI (useful in order to use this programme by other programmes). The minimal length of diagonal and vertical structures can be setted only in the GUI.

Methods of finding the neighbours.

`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm.
`'minnorm'`	– Minimum norm.

Normalization of the data series.

`'normalize'`	– Normalization of the data.
`'nonormalize'`	– No normalization of the data.

Suppressing the GUI.

`'gui'`	– Creates the GUI.
`'nogui'`	– Suppresses the GUI.
`'silent'`	– Suppresses all output.

Output

`y.RRp`	– Recurrence rate `(x,y)`.
`y.RRp`	– Recurrence rate `(x,-y)`.
`y.DETp`	– Determinism `(x,y)`.
`y.DETm`	– Determinism `(x,-y)`.
`y.Lp`	– Averaged diagonal length `(x,y)`.
`y.Lm`	– Averaged diagonal length `(x,-y)`.

Parameters not needed to be specified.

crqad_big

Limitations In contrast to `crqad`, only maximum, Euclidean and minimum norm are available.

Examples
```
a=sin(0:.1:800)+randn(1,8001);
b=sin(0:.1:800)+randn(1,8001);
crqad_big(a,b,3,15,.1,50,'euc')
```

See Also `crqa, crqad, crp, dl, tt`

References Marwan, N., Kurths, J.: Nonlinear analysis of bivariate data with cross recurrence plots, Phys. Lett. A, 302, 2002.

dl

Purpose Mean of the diagonal line lengths and their distribution.

Syntax
```
a=dl(x)
[a b]=dl(x)
[a b]=dl(x,'param1','param2')
```

Description Computes the mean `a` and the lengths of the found diagonal lines, stored in vector `b`. In order to get the histogramme of the line lengths, simply call `hist(b,[1 max(b)])`.

Different methods for considering border lines (lines starting and ending at a border of the RP) can be used.

Methods regarding considering border lines.

- `'all'` – (Default) Considers all individual lengths of border lines (most commonly used approach until 2021)
- `'censi'` – Correction schema for border lines as proposed by Censi et al. 2004, in which the length of the longest border line is used for all border lines (recommended for cyclical signals).
- `'kelo'` – Correction schema for border lines using KEep LOngest diagonal line (KELO), in which only the longest border line (in each triangle) of the RP is considered but all other border lines are discarded.
- `'semi'` – Relaxing the definition of border lines: not only lines starting AND ending at a border of the RP, but also semi border lines, which are lines that start or end at a border of the RP but have the corresponding ending or starting not at the border, are count. Has only effect for `'censi'` or `'kelo'` method.

Remark In Censi et al. 2004, the length of the LOI was considered to be the longest borderline. Here we use a modification by excluding the LOI from the set of borderlines. This usually results in a shorter length of the border lines than in the original Censi approach. But this would allow us to use this correction schema also for non-cyclical signals without strange effects.

Examples
```
a = sin(linspace(0,5*2*pi,1000));
X = crp(a,2,50,.2,'nonorm','nogui');
[l1 l_dist1] = dl(X,'all'); % considering all border lines
[l2 l_dist2] = dl(X,'censi'); % apply Censi correction for border lines
[l2 l_dist2] = dl(X,'kelo'); % apply KELO correction for border lines
subplot(3,1,1)
hist(l_dist1,200)
title(sprintf('considering all border lines, l=%.1f',l1))
subplot(3,1,2)
nexttile
hist(l_dist2,200)
title(sprintf('Censi correction, l=%.1f',l2))
subplot(3,1,3)
nexttile
hist(l_dist3,200)
title(sprintf('KELO correction, l=%.1f',l3))
```

See Also `crqa, tt`

References Censi, F., et al.: Proposed corrections for the quantification of coupling patterns by recurrence plots, IEEE Trans. Biomed. Eng., 51, 2004.

Kraemer, K. H., Marwan, N.: Border effect corrections for diagonal line based recurrence quantification analysis measures, Phys. Lett. A, 383, 2019.

embedding

Purpose Create embedding vector.

Syntax `y=embed(x,m,t)`

Description Creates embedding vector `y` from time series `x` using a time delay embedding with dimension `m` and time delay `t`. The resulting embedding vector has length $N - t(m-1)$, where N is the length of the original time series.

Examples
```
x = sin(0:0.1:10*2*pi);
y = embed(x,2,16);
plot(y(:,1),y(:,2))
```

entropy

Purpose Entropy of a distribution.

Syntax `e=entropy(h)`

Description Computes the entropy of the distribution `h`.

Examples
```
x=randn(100,1);
h=hist(x);
entropy(h')
```

fnn

Purpose Find the optimal embedding dimension by means of false nearest neighbours.

Syntax
```
y=fnn(x)
y=fnn(x,m)
y=fnn(x,m,t)
y=fnn(x,m,t,r,s)
fnn(...)
fnn(...,param)
```

Description Computes the amount `y` of false nearest neighbours (FNN) as a function of the embedding dimension. The optimal embedding is then chosen as the one where the amount of FNNs almost vanishes.

`fnn(...)` without any output arguments opens a GUI for interactively changing the parameters.

By using the GUI, the FNN can be stored into the workspace.

`fnn` without any arguments calls a demo (the same as the example below).

Parameters The parameters maximal dimension `m` (defeault 10), delay `t` (defeault 1), neighbourhood criterion `r` (defeault 10), size of the neighbourhood `s` (defeault Inf) and maximal number of random samples `n` (defeault `length(x)` if the data length is smaller than 500, else 200) are optional.

Additional parameters according to the GUI.

`'gui'`	– Creates the GUI.
`'nogui'`	– Suppresses the GUI.
`'silent'`	– Suppresses all output.

Parameters not needed to be specified.

Examples
```
x=sin(0:.2:8*pi)'+.1*randn(126,1);
fnn(x,10,[],5)
```

See Also `phasespace, pss, mi`

References Kennel, M. B., Brown, R., Abarbanel, H. D. I.: Determining embedding dimension for phase-space reconstruction using a geometrical construction, Phys. Rev. A, 45, 1992.

hist2

Purpose Creates a two dimensional histogram.

Syntax
```
p=hist2(x)
p=hist2(x,y)
p=hist2(x,k,l)
[p,j]=hist2(...)
hist2(...)
hist2(..,'gui')
```

Description `p=hist2(x)` bins the two-dimensional density of `x(i)` and `x(i+1)` into a 10x10 equally spaced matrix and returns it in `p`.

`p=hist2(x,y)` bins the two-dimensional density of x_i and y_i into a 10x10 equally spaced matrix and returns it in `p`.

`p=hist2(x,k,l)`, where `k` and `l` are scalars, uses `k` bins and a lag `l`.

`[p,j]=hist2(...)` returns the matrix `p` and the two-column vector `j` containing the two-dimensional density matrix and the bin location for `x` (and `y`).

`hist2(...)` without any output arguments produces a histogram plot.

`hist2(...,'gui')` creates a GUI for interactively changing of the parameters.

Examples
```
x=randn(10000,1);
hist2(x)
```

See Also `histn`, `mi`

histn

Purpose Creates a multi-dimensional histogram.

Syntax
```
p=histn(x)
p=histn(x1,...,xN)
p=histn([x,...,xN])
p=histn(x,l)
[p,j]=histn(...)
histn(...)
```

Description `p=histn(x)` bins the two-dimensional density of `x(i)` and `x(i+1)` into a 10x10 equally spaced matrix and returns it in `p` (this is similar to `hist2`).

`p=histn(x1,x2,...,xN)` or `p=hist2([x1,x2,...,xN])` bins the `N`-dimensional density of x_i into a 10x10 equally spaced matrix and returns it in `p`. Since both variants of input the arguments can be combined, the usage of various multi-column vectors is possible; the dimension is the sum of the number of all vectors' columns.

`p=histn(x,l)`, where `l` is a scalars, uses a lag `l`.

`p=histn(x,k,l)`, where `k` and `l` are scalars, uses `k` bins and a lag `l`.

`[p,j]=histn(...)` returns the `N`-dimensional matrix `p` and the two-column vector `j` containing the `N`-dimensional density matrix and the bin location for `x1,...,xN`.

`histn(...)` without any output arguments produces a histogram plot.

Examples
```
x=randn(10000,3);
histn(x)
```

See Also `hist2`, `mi`

jrp

Purpose Creates a joint recurrence plot.

Syntax
```
jrp(x)
jrp(x,y)
jrp(x,m,t,e)
r=jrp(x,[],m,t,e)
r=jrp(x,m,t,e,'param1','param2',...)
r=jrp(x,y,m,'param1')
```

Description Creates a simple joint recurrence plot of maximal two data series, based on different norms or recurrence plots. Embedding parameters will be the same for both systems. Results can be stored into the workspace.

Allows to change the parameters interactively by using a GUI.

The source-data `x` and test-data `y` can be one- or a two-coloumn vectors (then, in the first column have to be the time); if the test-data `y` is not specified, a simple (auto) recurrence plot is created.

Parameters Dimension `m`, delay `t` and the size of neighbourhood `e` are the first three numbers after the data series; further parameters can be used to switch between various methods of finding the neighbours of the phasespace trajectory, to suppress the normalization of the data and to suppress the GUI (useful in order to use this programme by other programmes).

Methods of finding the neighbours/ of plot.

`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm.
`'minnorm'`	– Minimum norm.
`'nrmnorm'`	– Euclidean norm between normalized vectors (all vectors have the length one).
`'rr'`	– Maximum norm, fixed recurrence rate.
`'fan'`	– Fixed amount of nearest neighbours.
`'inter'`	– Interdependent neighbours.
`'omatrix'`	– Order matrix.
`'opattern'`	– Order patterns recurrence plot.
`'distance'`	– Distance coded matrix (global JRP, Euclidean norm).

Normalization of the data series.

`'normalize'`	– Normalization of the data.
`'nonormalize'`	– No normalization of the data.

Suppressing the GUI.

`'gui'`	– Creates the GUI and the output plot.
`'nogui'`	– Suppresses the GUI and the output plot.
`'silent'`	– Suppresses all output.

Parameters not needed to be specified.

jrp

Limitations For higher speed in output the whole matrix of the recurrence plot is in the work space – this limits the application of long data series. However, with a little Matlab script, long data series can be handled too (cf. Examples for `crp`).

Examples
```
a=sin((1:1000)*2*pi/67);
b=sin(.01*([1:1000]*2*pi/67).^2);
jrp(a,b,3,12,'fan')
```

See Also `crp, jrqa`

References Romano, M., Thiel, M., Kurths, J., von Bloh, W.: Multivariate Recurrence Plots, Phys. Lett. A , 330, 2004.

jrqa

Purpose Computes and plots the JRQA measures.

Syntax
```
jrqa(x)
jrqa(x,y)
y=jrqa(x,y,m,t,e,w,ws)
y=jrqa(x,y,m,t,e,w,ws,lmin,vmin)
y=jrqa(x,y,m,t,e,w,ws,lmin,vmin,tw)
y=jrqa(x,y,m,t,e,[],'param1','param2',...)
```

Description Recurrence quantification analysis of joint-recurrence plots with the first vector `x` and the second `y`.

The input vectors can be multi-column vectors, where each column will be used as a component of the phase-space vector. However, if the first column is monotonically increasing, it will be used as an time scale for plotting.

Parameters `JRQA(...)` without any output arguments opens a GUI for interactively control the JRQA. If an output is specified with using the option `'gui'`, then the output will contain the figure handle.

Dimension `m`, delay `t`, the size of neighbourhood `e`, the window size `w` and the shift value `ws` are the first five numbers after the data series; if `w=[]` then the whole plot will be calculated. The minimal length of diagonal and vertical structures can be specified with `lmin` and `vmin` respectively (default is 2).

As the last numeric parameter, the size of the Theiler window `tw` can be specified (default is 1). This window excludes the recurrence points parallel to the main diagonal from the analysis.

Further parameters can be used to switch between various methods of finding the neighbours of the phasespace trajectory, to suppress the normalization of the data and to suppress the GUI (useful in order to use this programme by other programmes).

Methods of finding the neighbours.

`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm.
`'minnorm'`	– Minimum norm.
`'nrmnorm'`	– Euclidean norm between normalized vectors (all vectors have the length one).
`'rr'`	– Maximum norm, fixed recurrence rate.
`'fan'`	– Fixed amount of nearest neighbours.
`'inter'`	– Interdependent neighbours.
`'omatrix'`	– Order matrix.
`'opattern'`	– Order patterns recurrence plot.

Normalization of the data series.

`'normalize'`	– Normalization of the data.
`'nonormalize'`	– No normalization of the data.

Suppressing the GUI.

`'gui'`	– Creates the GUI.
`'nogui'`	– Suppresses the GUI.
`'silent'`	– Suppresses all output.

Output

`y(:,1)`	– Recurrence rate.
`y(:,2)`	– Determinism.
`y(:,3)`	– Averaged diagonal length.
`y(:,4)`	– Length of longest diagonal line.
`y(:,5)`	– Entropy of diagonal length.
`y(:,6)`	– Laminarity.
`y(:,7)`	– Trapping time.
`y(:,8)`	– Length of longest vertical line.
`y(:,9)`	– Recurrence time of 1st type.
`y(:,10)`	– Recurrence time of 2nd type.

Parameters not needed to be specified.

Limitations For higher speed in output the whole matrix of the recurrence plot is in the work space – this limits the application of long data series. However, a solution for using long data you can find under the description for `crp`.

Examples

```
N=500; w=40; ws=10;
b=.4; a=.6; mu=.8:-0.7/N:.1;

% two mutually coupled logistic maps
for i=2:N,
      a(i)=3.6*a(i-1)*(1-a(i-1));
      b(i)=4*b(i-1)*(1-b(i-1))-mu(i)*a(i);
end

% coupling is obtained by higher RR and DET values
jrqa(a,b,1,1,.2,w,ws);
```

See Also `crqa, jrp, crp`

References

Trulla, L. L., Giuliani, A., Zbilut, J. P., Webber Jr., C. L.: Recurrence quantification analysis of the logistic equation with transients, Phys. Lett. A, 223, 1996.

Marwan, N., Wessel, N., Meyerfeldt, U., Schirdewan, A., Kurths, J.: Recurrence Plot Based Measures of Complexity and its Application to Heart Rate Variability Data, Phys. Rev. E, 66(2), 2002.

Romano, M., Thiel, M., Kurths, J., von Bloh, W.: Multivariate Recurrence Plots, Phys. Lett. A , 330, 2004.

mcf

Purpose Plots the maximal correlation function.

Syntax
```
mcf(x,y [,w,t])
mcor=mcf(x,y [,w,t])
[time, mcor]=mcf(x,y [,w,t])
```

Description Without any output arguments, `mcf` plots the maximal correlation function up to the maximal lag of `t` and by using a boxcar window size of `2 w + 1`. Else, the maximal correlation function is stored in the vector `mcor` and its time scale in the vector `time`. If `w=[]`, the default boxcar window size is 11.

Examples
```
x=sin(0:.05:10)+.5*randn(1,201);
y=cos(0:.05:10);
mcf(x,y,[],20)
```

See Also `ace`

References Breiman, L., Friedman, J. H.: Estimating Optimal Transformations for Multiple regression and Correltaion, J. Am. Stat. Assoc., Vol. 80, No. 391, 1985.

Voss, H., Kurths, J.: Reconstruction of nonlinear time delay models from data by the use of optimal transformations, Phys. Lett. A, 234, 1997.

mgui

Purpose GUI for data analysis programmes.

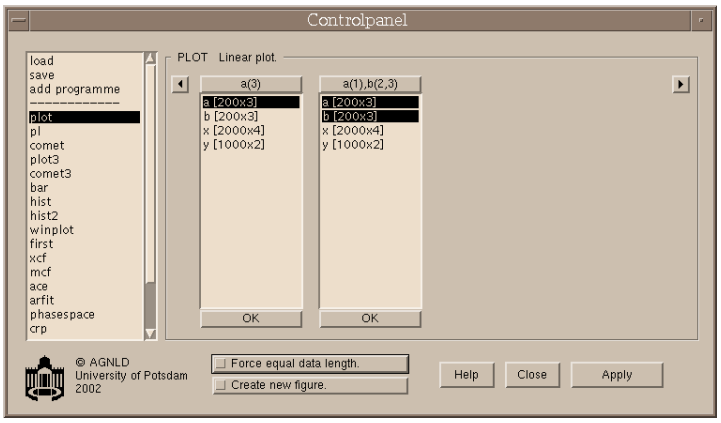
Syntax `mgui`

Description `mgui` starts a GUI and supplies Matlab programmes for their easy application to data which are in the Matlab workspace.

The presented programmes are stored in the file `mgui.rc` where own programmes can be added. Just include a line with the name of the programme and the minimal and maximal number of arguments, divided by a blank space or tabulator as a separator, e.g.

```
plot 1 4
```

If the embedded programmes provide an output, then this output will be stored in the variable `ans` in the Matlab workspace.



mi

Purpose Histogram based mutual information.

Syntax
```
i=mi(x)
i=mi(x1,...,xN)
i=mi(x,...,xN,l)
i=mi(x,...,xN,k,l)
[i s]=mi(...)
mi(...)
mi(...,'param')
```

Description Computes the mutual information between the vectors `x1,...,xN`. The auto mutual information can be computed by using only one vector. The arguments can be multi-column vectors. The result `i` will be a N×N matrix.

`[i s]=mi(...)` computes the mutual information and the standard error (only for one and two arguments).

`mi(...)` without any output arguments opens a GUI for interactively changing the parameters.

By using the GUI, the mutual information can be stored into the workspace. If their standard error is available, they will be appended to the mutual information matrix as the last two columns (the stored matrix will have the size 2×4).

`mi` without any arguments calls a demo (the same as the example below).

Parameters The parameters numbers of bins `k` and maximal lag `l` are optional. If the number of bins is not set, an amount of 10 will be used.

Additional parameters according to the GUI.

`'gui'`	– Creates the GUI.
`'nogui'`	– Suppresses the GUI.
`'silent'`	– Suppresses all output.

Parameters not needed to be specified.

Examples
```
x=sin(0:.2:8*pi)'+.1*randn(126,1);
mi(x,10,40)
```

Remark Please note that the mutual information derived with `mi` slightly differs from the results derived with `migram`. The reason is that `mi` also considers estimation errors.

See Also `hist2, histn, entropy`

References Roulston, M. S.: Estimating the errors on measured entropy and mutual information, Physica D, 125, 1999.

migram

Purpose Calculate windowed mutual information between two signals.

Syntax
```
i = migram(a,b,maxlag,window,noverlap,nbins)
[i,l,t] = migram(...)
i = migram(a,b)
migram(a,b)
```

Description `i = migram(a,b,maxlag,window,noverlap)` calculates the windowed mutual information between the signals in vector `a` and vector `b`. `migram` splits the signals into overlapping segments and forms the columns of `i` with their mutual information values up to maximum lag specified by scalar `maxlag`. Each column of `i` contains the mutual information function between the short-term, time-localized signals `a` and `b`. Time increases linearly across the columns of `i`, from left to right. Lag increases linearly down the rows, starting at `maxlag`. If lengths of `a` and `b` differ, the shorter signal is filled with zeros. If `n` is the length of the signals, `i` is a matrix with `2*maxlag+1` rows and

```
k = fix((n-noverlap)/(window-noverlap))
```

columns.

`i = migram(a,b,maxlag,window,noverlap,nbins)` calculates the mutual information based on histograms with the number of bins `nbins`.

`i = migram(...,'norm')` calculates the renormalised mutual information, which is $i/\log(n_{\text{bins}})$ and ensures a value range $[0 \ldots 1]$.

`[i,l,t] = migram(...)` returns a column of lag `l` and one of time `t` at which the mutual information is computed. `l` has length equal to the number of rows of `i`, `t` has length `k`.

`i = migram(a,b)` calculates windowed mutual information using defeault settings; the defeaults are `maxlag = floor(0.1*n)`, `window = floor(0.1*n)`, `noverlap = 0` and `nbins = 10`. You can tell `migram` to use the defeault for any parameter by leaving it off or using `[]` for that parameter, e.g. `migram(a,b,[],1000)`.

`migram(a,b)` with no output arguments plots the mutual information using the current figure.

Example
```
x = cos(0:.01:10*pi)';
y = sin(0:.01:10*pi)' + .5 * randn(length(x),1);
migram(x,y)
```

Remark Please note that the mutual information derived with `mi` slightly differs from the results derived with `migram`. The reason is that `mi` also considers estimation errors.

See Also `mi, corrgram`

normalize

Purpose Normalizes data.

Syntax `y=normalize(x)`

Description Normalizes the matrix `x` to zero-mean and standard deviation of one ($y = (x - \langle x \rangle)/\sigma_x$).

Examples
```
x=randn(100,1);
std(x), mean(x)
std(normalize(x)), mean(normalize(x))
```

phasespace

Purpose Computes phase space size.

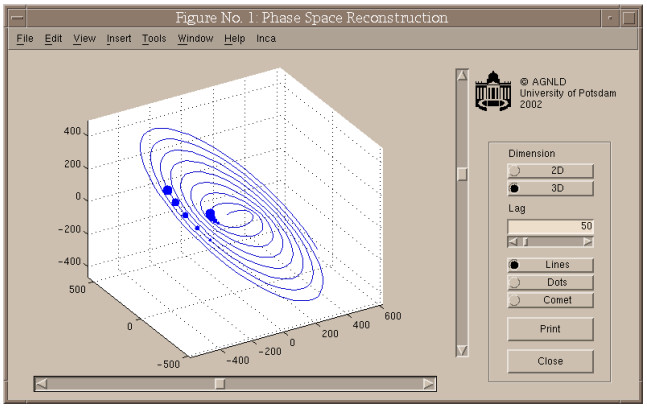
Syntax
```
phasespace(x)
phasespace(x,y)
phasespace(x,y,z)
```

Description Shows the 3D phase space trajectory of the system which is presented by the observation `x`. The phase vectors are a reconstruction by using the time delay method (Takens, 1981). A GUI provides to change the embedding dimension to 2D.

`phasespace(x,y)` or `phasespace(x,y,z)` uses the one-column vectors `x`, `y` (and `z`) as the components of the phase space trajectory. The representation is 2D (3D) only and cannot be switched to the other representation.

`phasespace` without any arguments calls a demo (the same as the example below).

Example
```
phasespace(cos(0:.1:32).*[321:-1:1])
```



See Also `fnn, pss`

References Takens, F.: Detecting Strange Attractors in Turbulence, Lecture Notes in Mathematics, 898, Springer, Berlin, 1981

phasesynchro

Purpose Indicator of phase synchronisation by means of recurrences.

Syntax
```
cpr=phasesynchro(x,y)
cpr=phasesynchro(x,y,m,t,e,w)
cpr=phasesynchro(x,y,m,t,e,w,'param1','param2')
```

Description `cpr=phasesynchro(x,y [,param1,param2,...])` calculates the index of phase synchronisation based on recurrences.

`cpr=phasesynchro(x,y,m,t,e,w)` uses the dimension `m`, delay `t`, the size of neighbourhood `e` and the range `w` of past and future time steps.

If `x` and `y` are multi-column vectors then they will be considered as phase space vectors (`taucrp` can be used for real phase space vectors without embedding).

The call of `phasesynchro` without output arguments plots the tau-recurrence rate and the CPR value in the current figure.

Parameters Dimension `M`, delay `T`, the size of neighbourhood `E` and the range `W` are the first four numbers after the data series; further parameters can be used to switch between various methods of finding the neighbours of the phasespace trajectory and to suppress the normalization of the data.

Methods of finding the neighbours.

`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm.
`'minnorm'`	– Minimum norm.
`'rr'`	– Maximum norm, fixed recurrence rate.
`'fan'`	– Fixed amount of nearest neighbours.

Normalization of the data series.

`'normalize'`	– Normalization of the data.
`'nonormalize'`	– No normalization of the data.

Suppressing the plot.

`'silent'`	– Suppresses the plot of the results.

Parameters not needed to be specified.

Example
```
a = sin((1:1000) * 2 * pi/67);
b = sin((1:1000) * 2 * pi/67) + randn(1,1000);
phasesynchro(a,2,17,'nonorm','euclidean');
```

See Also `crp, crqad`

References Marwan, N., Romano, M. C., Thiel, M., Kurths, J.: Recurrence Plots for the Analysis of Complex Systems, Physics Reports, 438(5-6), 2007.

Romano, M. C., Thiel, M., Kurths, J., Kiss, I. Z., Hudson, J.: Detection of synchronization for non-phase-coherent and non-stationary data, Europhysics Letters, 71(3), 2005.

pss

Purpose Computes phase space size.

Syntax
```
pss(x)
pss(x,m,t)
[y z]=pss(...)
[y z]=pss(...,'param')
```

Description `pss(...)` computes the maximal phase space diameter of embedded data series `x` with the embedding parameters dimension `m` and lag `t`. A norm can be specified with an additional parameter.

`[y z]=pss(...)` computes the maximal `y` and the averaged `z` phase space diameter of embedded data series `x`.

Parameters Parameter for used norm.

`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm (default).
`'minnorm'`	– Minimum norm.

See Also `phasespace, crp, crp2`

recons

Purpose
Reconstruct a time series from a recurrence plot. `y=recons(x)`

Syntax
```
y=recons(x,...,method)
y=recons(x,...,name)
y=recons(x,p,...)
```

Description
`y=recons(x)` reconstructs a time series `y` from the recurrence plot in the matrix `x`.

`y=recons(x,...,method)` specifies the reconstruction method where string `method` can be either 'thiel' or 'hirata', using the method by Marco Thiel (defeault) or Yoshito Hirata.

`y=recons(x,...,name)` reconstructs the time series using the named cumulative distribution function, which can be 'norm' or 'Normal' (defeault), 'unif' or 'Uniform'. This is only necessary for the method by Thiel. The reconstruction from the Hirata will not be scaled.

`y=recons(x,p,...)` reconstructs the time series using the cumulative distribution function given by vector `P` (should be 2nd argument). This is only necessary for the method by Thiel. The reconstruction from the Hirata will not be scaled.

See Also
`crp, crp2, jrp, twinsurr`

References
Thiel, M., Romano, M. C., Kurths, J.: How much information is contained in a recurrence plot?, Phys. Lett. A, 330, 2004.

Hirata, Y., Horai, S., Aihara, K.: Reproduction of distance matrices from recurrence plots and its applications, Eur. Phys. J. ST, 164, 2008.

rpde

Purpose Computes the recurrence time entropy.

Syntax
```
y=rpde(x)
y=rpde(x,...)
```

Description `y=rpde(x)` calculates the normalised entropy `y` of the recurrence time distribution of time series `x`, also known as recurrence period density entropy (RPDE).

Note In contrast to the calculation of RPDE here, in `crqa` a Theiler window is applied to the RP by default, resulting in different RPDE values. For comparison, you should ensure that the Theiler window in `crqa` is set to 0.

Examples
```
a=sin(0:.1:80);
b=sin(0:.1:80) + 0.1 * randn(1,801);
rpde(a,3,15,.1)
rpde(b,3,15,.1)
```

See Also `crqa, tt`

References Little, M., McSharry, P., Roberts, S., Costello, D., Moroz, I.: Exploiting Nonlinear Recurrence and Fractal Scaling Properties for Voice Disorder Detection, Biomed. Eng. Online, 6, 2007.

rrspec

Purpose Tau-recurrence rate spectrum.

Syntax
```
rrspec(x,m,t,e,w,fs,...)
p=rrspec(...)
[p f]=rrspec(...)
```

Description `rrspec(x,m,t,e,w,fs,...)` calculates the tau-recurrence rate spectrum based on a recurrence plot using embedding dimension `m`, embedding delay `t`, recurrence threshold `e`, maximal lag for tau-recurrence `w`, and sampling frequency `fs`. The input arguments are similar to those of the command `taucrp`.

`p = rrspec(...)` returns the tau-recurrence rate spectrum in vector `p`.

`[p f] = rrspec(...)` returns the tau-recurrence rate spectrum in vector `p` and the vector of corresponding frequencies `f`.

Examples
```
fs = 22;
x = sin(2*pi * [0:1/fs:44]);
rrspec(x,2,1,.1,[],fs)
```

See Also `taucrp, rtspec`

References Zbilut, J. P., Marwan, N.: The Wiener-Khinchin theorem and recurrence quantification, Phys. Lett. A, 372, 2008.

rtspec

Purpose Recurrence time spectrum.

Syntax
```
rtspec(x,m,t,e,fs,...)
p=rtspec(...)
[p f]=rtspec(...)
```

Description `rtspec(x,m,t,e,w,fs,...)` calculates the recurrence time spectrum based on a recurrence plot using embedding dimension `m`, embedding delay `t`, recurrence threshold `e`, and sampling frequency `fs`. The input arguments are similar to those of the command `crp`.

`p = rtspec(...)` returns the recurrence time spectrum in vector `p`.

`[p f] = rtspec(...)` returns the recurrence time spectrum in vector `p` and the vector of corresponding frequencies `f`.

Examples
```
fs = 22;
x = sin(2*pi * [0:1/fs:44]);
rtspec(x,2,1,.1,fs)
```

See Also `crp`, `rrspec`

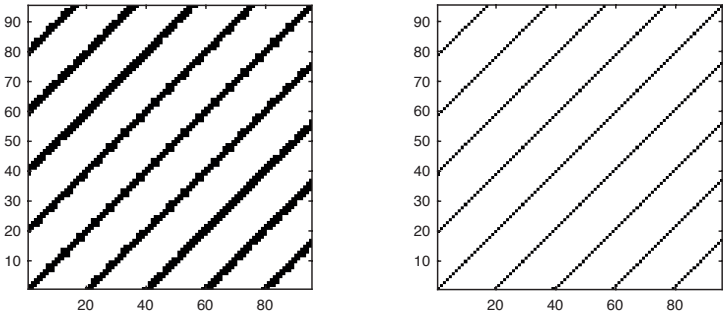
skel_crp

Purpose Removes thickening of diagonal lines in RP.

Syntax `y=skel_crp(x);`

Description `y = skel_crp(x)` creates a new recurrence matrix `y` in which all recurrence points of the recurrence matrix `x` are removed which lead to a thickening of diagonal lines. Slubs, but also block structures are removed in favour of the longest diagonal lines (skeletonization). Whenever a diagonal line (starting with the longest lines contained in the diagonal line length histogram) encounters an adjacent diagonal line, this adjacent line and – recursively – all its consecutive adjacent lines, get deleted.

Examples
```
a = sin(linspace(0,5*2*pi,100));
X = crp(a,2,5,.5,'nonorm','nogui');
Y = skel_crp(X);
nexttile
imagesc(X)
axis xy square
nexttile
imagesc(Y)
axis xy square
colormap([1 1 1; 0 0 0])
```



See Also `crp, dl`

References Kraemer, K. H., Marwan, N.: Border effect corrections for diagonal line based recurrence quantification analysis measures, Phys. Lett. A, 383, 2019.

taucrp

Purpose Creates a close returns plot.

Syntax
```
r=taucrp(x)
r=taucrp(x,m,t,e,w)
r=taucrp(x,y,m,t,e,w)
r=taucrp(x,y,m,t,e,w,'param')
```

Description `r=taucrp(x [,y] [,param1,param2,...])` creates a cross recurrence plot/ recurrence plot `r` for a limited range of past and future states, also known as close returns plot.

`r=taucrp(x,m,t,e,w)` uses the dimension `m`, delay `t`, the size of neighbourhood `e` and the range `w` of past and future time steps.

If `x` and `y` are multi-column vectors then they will be considered as phase space vectors (`taucrp` can be used for real phase space vectors without embedding).

Parameters Dimension `M`, delay `T`, the size of neighbourhood `E` and the range `W` are the first four numbers after the data series; further parameters can be used to switch between various methods of finding the neighbours of the phasespace trajectory and to suppress the normalization of the data.

Methods of finding the neighbours/ of plot.

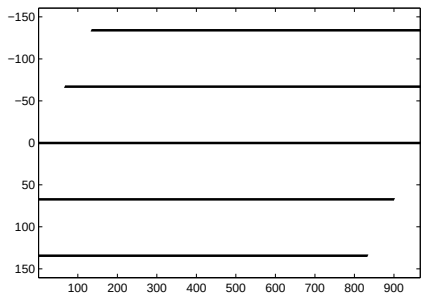
`'maxnorm'`	– Maximum norm.
`'euclidean'`	– Euclidean norm.
`'minnorm'`	– Minimum norm.
`'rr'`	– Maximum norm, fixed recurrence rate.
`'fan'`	– Fixed amount of nearest neighbours.
`'distance'`	– Distance coded matrix (global CRP, Euclidean norm).

Normalization of the data series.

`'normalize'`	– Normalization of the data.
`'nonormalize'`	– No normalization of the data.

Parameters not needed to be specified.

Examples
```
a = sin((1:1000) * 2 * pi/67);
w = 160;
X = taucrp(a,2,17,0.2,w,'nonorm','euclidean');
imagesc(1:size(X,2),-w:w,X), colormap([1 1 1; 0 0 0])
```



taucrp

See Also `crp, crp2, crp_big, jrp, crqa`

References Marwan, N., Romano, M. C., Thiel, M., Kurths, J.: Recurrence Plots for the Analysis of Complex Systems, Physics Reports, 438(5-6), 2007.

trackplot

Purpose Estimates the line of synchronization of a cross recurrence plot.

Syntax
```
trackplot(x)
trackplot(x,dx,dy)
trackplot(x,dx,dy,'param')
a=trackplot(...)
[a b]=trackplot(...)
```

Description `trackplot(x)` estimates the line of synchronization (LOS) in a cross recurrence plot `x`. The resulted path is exported to the workspace variable `t_out`. This command allows the interactive changing of estimation parameters.

`[a b]=trackplot(...)` estimates the LOS and stores it in `a`. The number of recurrence points met by the LOS is stored in `b(1)` and the number of lacks in the LOS is stored in `b(2)`.

Parameters The search of the LOS can be forced with the parameters `dx` and `dy`. An additional flag `param` allows to suppress the GUI (useful in order to use this programme by other programmes).

Suppressing the GUI.

`'gui'`	– Creates the GUI and the output plot.
`'nogui'`	– Suppresses the GUI and the output plot.
`'silent'`	– Suppresses all output.

Examples
```
y=sin([1:900]*2*pi/67)';
y2=sin(.01*([1:900]*2*pi/67).^2)';
x=crp_big(y,y2,3,12,.1,'fan','nogui');
trackplot(x,2,2)
```

See Also `crp2, crp and crp_big`

References Marwan, N., Thiel, M., Nowaczyk, N.: Cross Recurrence Plot Based Synchronization of Time Series, Nonlin. Proc. Geophys. 9, 2002.

trafo

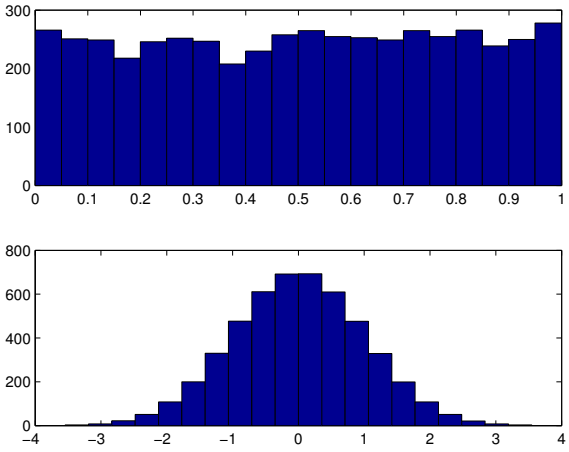
Purpose Transforms data to a desired distribution.

Syntax `y=trafo(x,a)`

Description `y=trafo(x,a)` transforms the data in vector `x` to data `y` of a desired distribution, where

`a=0` – normal distribution (default),
`a=1` – uniform distribution,
`a=2` – exponential distribution.

Example
```
x=rand(5000,1);
subplot(2,1,1), hist(x,20)   % uniformly distributed
y=trafo(x,0);
subplot(2,1,2), hist(y,20)   % normally distributed
```



tt

Purpose Mean trapping time and its distribution.

Syntax
```
a=tt(x)
[a b]=tt(x)
```

Description `a=tt(x)` computes the mean of the length of the vertical line structures in a recurrence plot, so called trapping time `tt`.

`[a b]=tt(x)` computes the tt and the lengths of the found vertical line structures, stored in `b`. In order to get the histogramme of the line lengths, simply call `hist(b,[1 max(b)])`.

See Also `crqa, dl`

twinsurr

Purpose Creates twin surrogates for statistical tests.

Syntax
```
y=twinsurr(x)
y=twinsurr(x,m,t,e,'param1','param2')
y=twinsurr(x,m,t,e,...,n)
```

Description `y=twinsurr(x)` creates twin surrogates `y` based on the vector `x` using recurrences. The matrix `y` contains 100 columns of 100 twin surrogates. If `x` is a $p \times q$ matrix, the resulting surrogate matrix is $p \times 100 \times q$.

`y=twinsurr(x,m,t,e,...)` creates twin surrogates using embedding dimension `m`, delay `t`, recurrence threshold `e`. The input arguments are similar to those of the command `crp`.

`y=twinsurr(x,m,t,e,...,n)` creates `n` surrogates (default is 100).

Example
```
x = rand(3,1);
a = [.8 .3 -.25 .9]';
for i = 4:1000,
    x(i) = sum(a(1:3) .* x(i-1:-1:i-3)) + a(end) * randn;
end
xs = twinsurr(x,1,1,.1,'euc',10);
```

See Also `crp, recons`

References Thiel, M., Romano, M. C., Kurths, J., Rolfs, M., Kliegl, R.: Twin Surrogates to Test for Complex Synchronisation, Europhys. Lett., 75, 2006.

winplot

Purpose Windowed plot.

Syntax

```
winplot(x)
winplot(x,w,ws)
winplot(x,w,ws,flag)
y=winplot(x,'parm')
```

Description `winplot(x [,w,ws])` plots means or variances of the sub-vectors of vector `x`, which have the length `w` and are shifted by the step `ws`. `x` can be a two-column vector, where the first column would be the time-scale.

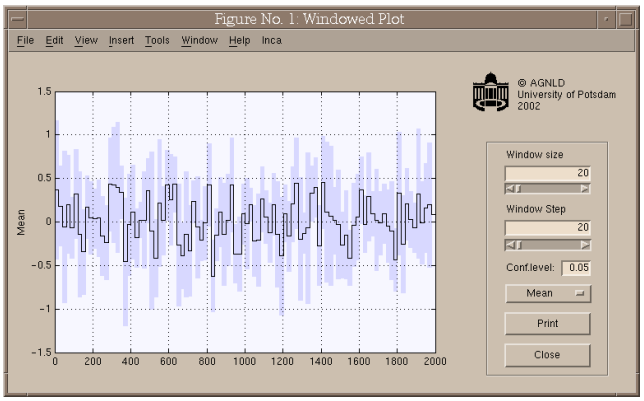
`winplot` without any arguments calls a demo (the same as the example below).

Parameters The optional parameter `flag` can determine the kind of the result, where `flag` can be either a string or a scalar:

`'mean'` or	`1` – Mean (1st moment).
`'var'` or	`2` – Variance (2nd moment).
`'std'` or	`3` – Standard deviation.
`'median'` or	`4` – Median.
`'sqm'` or	`5` – Squared Mean.
`'geo'` or	`6` – Geometric Mean.
`'3rd'` or	`7` – 3rd moment.
`'skw'` or	`8` – Skewness.
`'kur'` or	`9` – Kurtosis.

Example

```
winplot(randn(2000,1),20,20)
```



See Also `plot`

Plugin

Description
A precompiled plugin for the computation of (cross/joint) recurrence plots and their quantification can be used for really long data series (several 10 000 data points). It may accelerate the computation as well.

Usage
Download the corresponding installation script `plugininstall.m` for your system and put it any folder, where Matlab can find it. Call `plugininstall` from the Matlab commandline. You may check if it works with the command `rp_plugin`.

After installation, this plugin is used by the commands `crp`, `crp_big`, `crp2`, `jrp`, `crqa` and `jrqa`, if *Maximum norm*, *Euclidean norm*, *Minimum norm* or *Distance matrix* is used as a neighbourhood criterion. If two data vectors are used (for cross or joint recurrence plots), the plugin will only be used if both data vectors have the same length.

Supported Systems
Currently the following systems are supported:

- True64 OSF1(5.1) on alpha
- HP-UX 11 on HP U9000
- Solaris 5.9 on Sun
- Linux on i686
- Linux on AMD Opteron 64
- Linux on Intel Itanium 2
- Dos/Win on x86

Error Handling

Error Support

If an error occurs, an extensive error report will be supplied in the file `error.log`. Please send us this error report and provide a brief description of what you were doing when this problem occurred. E-mail or FAX this information to us at:

E-mail: marwan@pik-potsdam.de
Fax: ++49 +331 288 20738

Thank you for your assistance.

Error Codes

The following error codes mark the location of the error in the programmes.

code	location in programme
0	ok
1	initialization
2	create crp figure
3	create control gui
4	vectorswitch/ vectorexclude
5	fit dimension display
6	unthresh
7	stretch
81	change colormap
82	change colormap scale
9	store
91	print
101	close all
102	smart close
11	init computation
111	local CRP, fixed distance maximum norm
112	local CRP, fixed distance euclidean norm
113	local CRP, fixed distance minimum norm
114	local CRP, normalized distance euclidean norm
115	local CRP, fixed neigbours amount
116	local CRP, interdependent neigbours method
117	order matrix
117	global CRP
12	show local CRP
13	show global CRP
14	set handles and axes ratios
15	LOS store
16	LOS move
161	LOS move end
17	LOS clear
18	LOS set
19	LOS search
191	looks for the beginning of the diagonal LOS
192	start estimation of the LOS
193	looks for the existence of the next recurrence point
194	determines the coordinates of the next recurrence point
195	determines the local width of the diagonal LOS
196	compute the mean of the diagonal LOS
197	DTW algorithm, seek process

Error Handling

continuation

198	DTW algorithm, fixed points
199	show LOS
20	CRQA computation
30	CRQA plot
90	installation