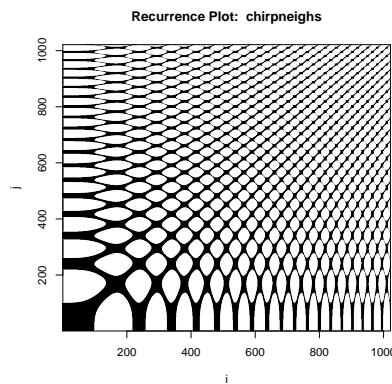


STATISTICAL DATA ANALYSIS: RECURRENCE PLOT

GÜNTHER SAWITZKI



CONTENTS

1.	Setup	1
1.1.	Local Bottleneck	3
2.	Test Cases	4
3.	Takens' Recurrence States	6
4.	Recurrence Plots	11
5.	Recurrence Quantification Analysis	11
6.	Applied Recurrence Plots	12
6.1.	Sinus	12
6.2.	Uniform random	13
6.3.	Chirp Signal	14
7.	Case Study: Geyser data	16
7.1.	Geyser Eruptions	16
7.2.	Geyser Eruptions: Comparison by Dimension	25
7.3.	Geyser Waiting	25
7.4.	Geyser - linearized	29
7.5.	Geyser Eruptions linearized	30
8.	Case Study: HRV data	37
8.1.	RHRV: Comparison by Dimension	39
8.2.	Hart Rate Variation	43
8.3.	RHRV Variation: Comparison by Dimension	44
	References	50

Date: 2013-11.

Key words and phrases. data analysis, distribution diagnostics, recurrence plot.

This waste book is a companion to "G. Sawitzki: Statistical Data Analysis"

Typeset, with minor revisions: August 7, 2014 from svn/cvs Revision : 1.10

gs@statlab.uni-heidelberg.de .

1. SETUP

Input

```
save.RNGseed <- 87149 #.Random.seed
save.RNGkind <- RNGkind()
# save.RNGseed
save.RNGkind
```

Output

```
[1] "Mersenne-Twister" "Inversion"
```

Input

```
set.seed(save.RNGseed, save.RNGkind[1])
```

Input

```
laptme <- function(){
  return(round(structure(proc.time() - chunk.time.start, class = "proc_time")[3],3))
  chunk.time.start <- proc.time()
}
```

Input

```
# install.packages("sintro", repos="http://r-forge.r-project.org", type="source")
library(sintro)
```

We use

Input

```
library(nonlineartseries)
```

ToDo: the takens state plot may be critically affected by outliers. Find a good rescaling.

ToDo: improve choice of alpha

statepairs

Usage.

```
statepairs(states, main, rank = FALSE, nooverlap = FALSE, alpha)
```

Arguments.

states	A matrix: states by dimension.
main	Optional: the main header.
rank	An experimental variant. If rank , the values are rank transformed.
nooverlap	An experimental variant. If nooverlap , the cases are subsampled by dimension.

`alpha` The alpha value used for plotting. The current choice is $\sqrt{\frac{1}{nrstates}}$.

```
statepairs <- function(states, main, rank=FALSE, nooverlap= FALSE, alpha){
  n <- dim(states)[1]; dim <- dim(states)[2]
  if (missing(main)) {
    main <- paste("Takens states:", deparse(substitute(states)), "\n",
    "n=", n, " dim=", dim) }

  if (nooverlap) {states <- states[ seq(1,n, by=dim),]}

  main <- paste(main, " no overlap")}

  if (missing(alpha)) {alpha <- 1/sqrt(dim(states)[1])}
  #cat("alpha=",alpha)

  if (rank) {states <- apply(states, 2, rank, ties.method="random")
  main <- paste(main, " ranked")}
  pairs(states, main=main,
  col=rgb(0,0,0, alpha), pch=19)
  #title(main=main, outer=TRUE, line=-2, cex.main=0.8)
}
```

1.1. Local Bottleneck. To allow experimental implementations, functions from `nonlinearTseries` are aliased here.

```
local.buildTakens <- buildTakens
```

```
local.findAllNeighbours <- nonlinearTseries:::findAllNeighbours
```

```
#non-sparse variant
```

```
Input
```

```
#local.recurrencePlotAux <- nonlinearTseries:::recurrencePlotAux
local.recurrencePlotAux = function(neigs, dim=NULL, lag=NULL, radius=NULL){

  # just for reference. This function is inlined
  neighbourListNeighbourMatrix = function(){
    neigs.matrix = Diagonal(ntakens)
    for (i in 1:ntakens){
      if (length(neigs[[i]])>0){
        for (j in neigs[[i]]){
          neigs.matrix[i,j] = 1
        }
      }
    }
    return (neigs.matrix)
  }

  ntakens=length(neigs)
  neigs.matrix <- matrix(nrow=ntakens, ncol=ntakens)
  #neighbourListNeighbourMatrix()
  #neigs.matrix = Diagonal(ntakens)
  for (i in 1:ntakens){
    neigs.matrix[i,i] = 1 # do we want the diagonal fixed to 1
```

minor cosmetics added to recurrence-PlotAux

ToDo: propagate parameters from `buildTakens` and `findAllNeighbours` in a slot of the result, instead of using explicit parameters in `recurrencePlotAux`.

```

    if (length(neighs[[i]])>0){
      for (j in neighs[[i]]){
        neighs.matrix[i,j] = 1
      }
    }

main <- paste("Recurrence Plot: ",
             deparse(substitute(neighs))
             )
more <- NULL

#use compones of neights if available
if (!is.null(dim)) more <- paste(more, " dim:",dim)
if (!is.null(lag)) more <- paste(more, " lag:",lag)
if (!is.null(radius)) more <- paste(more, " radius:",radius)

if (!is.null(more)) main <- paste(main,"\\n",more)

# need no print because it is not a trellis object!!
#print(
  image(x=1:ntakens, y=1:ntakens,
        z=neighs.matrix,xlab="i", ylab="j",
        col="black",
        #xlim=c(1,ntakens), ylim=c(1,ntakens),
        useRaster=TRUE,  #? is this safe??
        main=main
        )
  #
  )
}

}

```

ToDo: improve feedback for data structures in *non-linearTseries*

2. TEST CASES

We set up a small series of test signals.

For convenience, some source code from other libraries is included to make this self-contained.

As a global constant, we set up the length of the series to be used for test signals.

Input

```

#nsignal <- 256
nsignal <- 1024
system.time.start <- proc.time()

```

For signal representation, we use a common layout.

Input

```

plotsignal <- function (signal) {
  #! alpha level should depend on expected number of overlaps

  par(mfrow=c(1,2))
  plot(signal,
       main="",col=rgb(0,0,1,0.3), pch=20, xlab="index" )

```

```

plot(signal, type="l",
      main="", xlab="index", col=rgb(0,0,0,0.4))
points(signal, col=rgb(0,0,1,0.3), pch=20 )
main=deparse(substitute(signal))
title(main=main, outer=TRUE, line=-2, cex.main=1.2)

}

```

Input

```

sin10 <- function(n=nsignal) {sin( (1:n)/n* 2*pi*10)}
plotsignal(sin10())

```

See Figure figure 1 on the facing page.

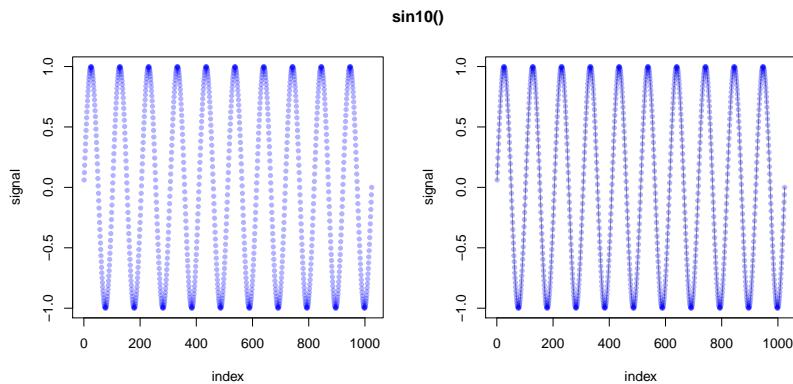


FIGURE 1. Test case: sin10. Signal and linear interpolation.

Input

```

unif <- function(n=nsignal) {runif(n)}
xunif<-unif()
plotsignal(xunif)

```

See Figure figure 2 on the next page,

Input

```

chirp <- function(n=nsignal) {
# this is copied from library(signal)
signal.chirp <- function(t, f0 = 0, t1 = 1, f1 = 100,
                           form = c("linear", "quadratic", "logarithmic"), phase = 0){

form <- match.arg(form)
phase <- 2*pi*phase/360

switch(form,
      "linear" = {
        a <- pi*(f1 - f0)/t1
        b <- 2*pi*f0
        cos(a*t^2 + b*t + phase)
      },
      "quadratic" = {

```

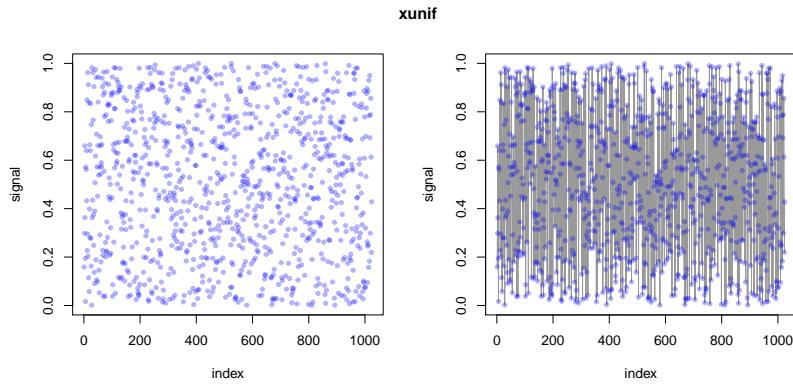


FIGURE 2. Test case: unif - uniform random numbers. Signal and linear interpolation.

```

a <- (2/3*pi*(f1-f0)/t1/t1)
b <- 2*pi*f0
cos(a*t^3 + b*t + phase)
},
"logarithmic" = {
  a <- 2*pi * t1 / log(f1 - f0)
  b <- 2*pi * f0
  x <- (f1-f0)^(1/t1)
  cos(a*x^t + b*t + phase)
})
}

signal.chirp(seq(0, 0.6, len=nsignal))
}
plotsignal(chirp())

```

See Figure figure 3,

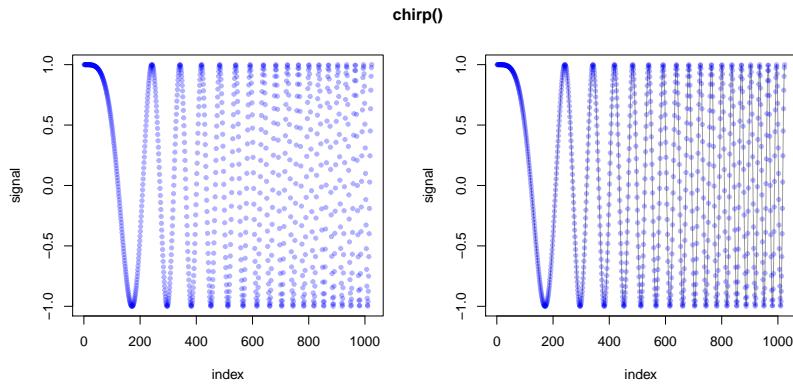


FIGURE 3. Test case: chirp signal. Signal and linear interpolation.

To Do: include
doppler waveslim

3. TAKENS' RECURRENCE STATES

Recurrence plots have been introduced in an attempt to understand near periodic in hydrodynamics. On the one hand, and extended theory on dynamical systems was available,

covering deterministic models. A fundamental concept is that at a certain time a system is in some state, and developing from this. Defining the proper state space is a critical step in modelling.

The other toolkit is that of stochastics processes, in particular Markov models. Classical time series assumes stationarity, and this is obviously not the way to go. A fundamental idea for Markov models is that the system state is seen in a temporal context: you have a Markov process, if you can define a (non-anticipating) state that has sufficient information for prediction: given this state, the future is independent from the past.

Recurrence, coming back to some state, is often a key to understand a near periodic system.

Hydrodynamics is a challenging problem. Understanding planetary motion is a historical challenge, and may be useful as an illustration.

As a simple illustration, let $x = (x_i)$ be a sequence, maybe near periodic. For now, think of i as a time index.

Recurrence plots have two steps. The first was a bold step by Floris Takens. If you do not know the state space of a system, for a choice of “dimension” d , take the sequence of d tuples taken from your data to define the states.

$$u_i = (x_i, \dots, x_{i+d})$$

This is Takens’ delay embedding state (re)construction Takens [1981].

As a mere technical refinement: you may know that your data are a flattened representation of t dimensional data. So you take

$$u_i = (x_i, \dots, x_{i+d*m}).$$

This may be a relict of FORTRAN times, where it was common to flatten two-dimensional structures by case. We ignore this detail here and take $m = 1$.

Conceptually, you define states by observed histories. For classical Markov setup, the state is defined by the previous information x_{i-1} , but for more complex situations you may have to step back in the past. Finding the appropriate d is the challenge. So it may be appropriate to view the Takens states as a family, indexed by the time scope d . The rest is structural information how to arrange items.

ToDo: add support for higher dimensional signals

Of course it is possible to compress information here, sorting states and removing duplicates. Keeping the original definition as the advantage that we have the index i , so that u_i is the state at index position i .

But the states may have an inherent structure, which we may take into account or ignore. Since for this example, we are just in 4-dimensional space, marginal scatterplots may give enough information.

Input

```
sintakens <- local.buildTakens(time.series=sin10(),
                                embedding.dim=4, time.lag=1)
statepairs(sintakens)
```

See Figure figure 4.

Input

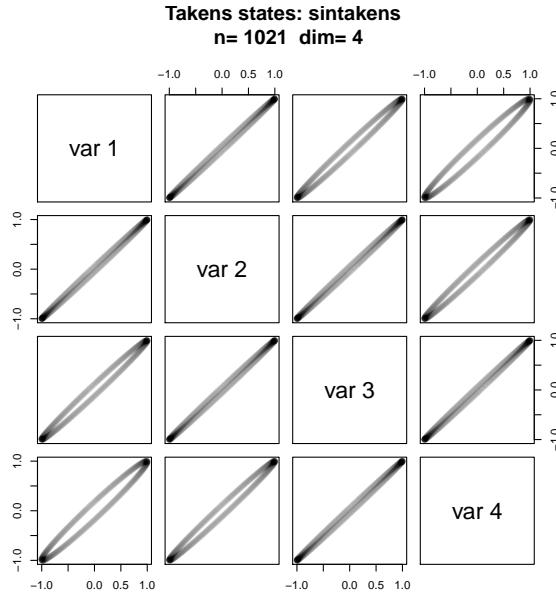


FIGURE 4. Test case: sinus. Note that marginal views of 1-dimensional circles in d space may appear as ellipses. Time used: 0.616 sec.

```
statepairs(sintakens, nooverlap=TRUE)
```

See Figure figure 5.

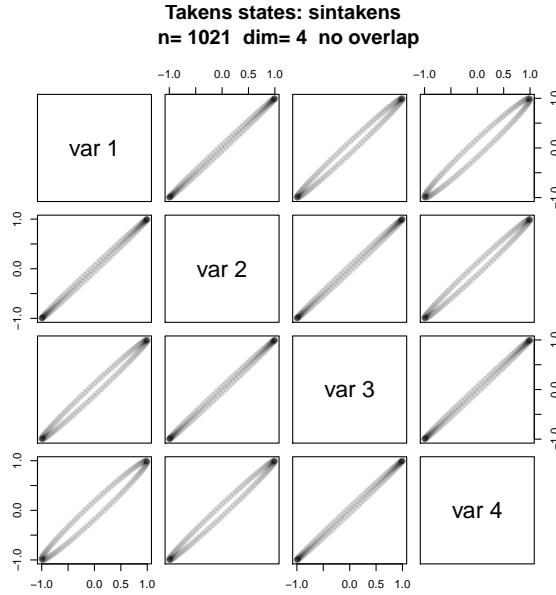


FIGURE 5. Test case: sinus. Note that marginal views of 1-dimensional circles in d space may appear as ellipses. Time used: 0.869 sec.

Input

```
uniftakens <- local.buildTakens( time.series=xunif,
                                  embedding.dim=4, time.lag=1)
statepairs(uniftakens)
```

See Figure figure 6 on the next page.

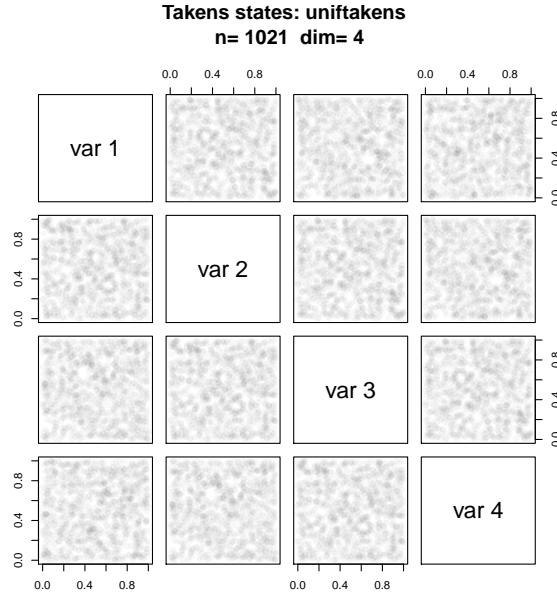


FIGURE 6. Test case: uniform random numbers. Time used: 0.654 sec.

Input

```
statepairs(uniftakens, nooverlap=TRUE)
```

See Figure figure 7 on the facing page.

Input

```
chirptakens <- local.buildTakens( time.series=chirp(),
                                   embedding.dim=4, time.lag=1)
statepairs(chirptakens)
```

See Figure figure 8 on page 10

Input

```
statepairs(chirptakens, nooverlap=TRUE)
```

See Figure figure 9 on page 10

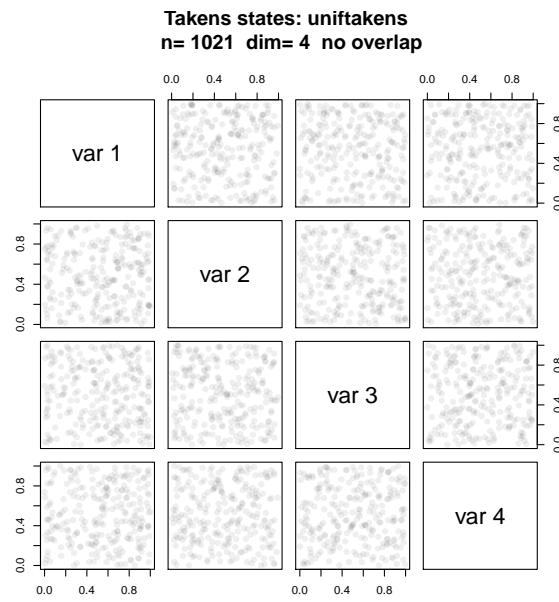


FIGURE 7. Test case: uniform random numbers. Time used: 0.938 sec.

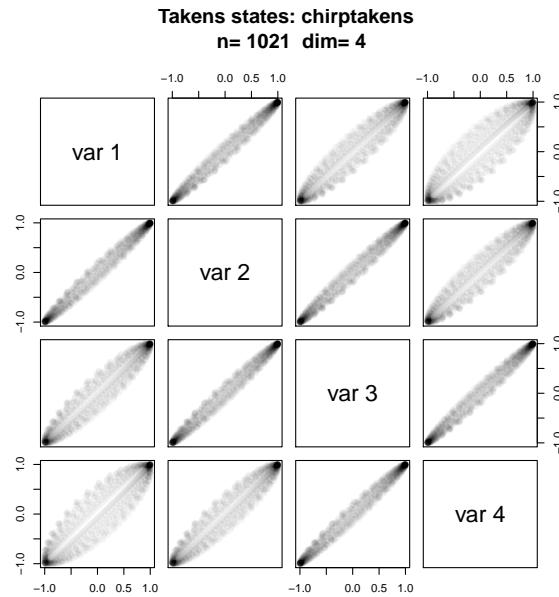


FIGURE 8. Test case: chirp signal. Time used: 0.637 sec.

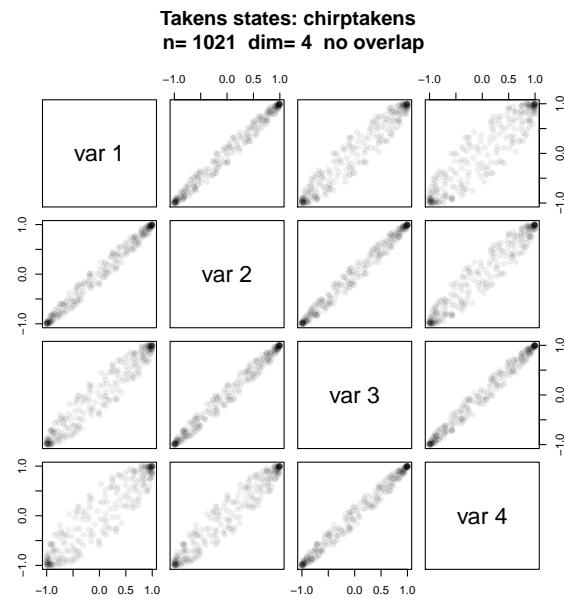


FIGURE 9. Test case: chirp signal. Time used: 0.912 sec.

ToDo: consider dimension-adjusted radius

The next step, taken in Eckmann *et al.* [1987] was to use a two dimensional display. Take a scatterplot with the Taken's states a marginal. Take a sliding window of your process data, and for each i , find the “distance” of u_i from and to any of the collected states. If the distance is below some chosen threshold, mark the point (i, j) for which $u(j)$ is in the ball of radius $r(i)$ centred at $u(i)$.

The original publication Eckmann *et al.* [1987] actually used a nearest neighbourhood environment to cover about 10 data points.

The construction has considerable arbitrary choices. The critical radius may depend on the point i . In practical applications, using a constant radius is a common first step. Using a dichotomous marking was what presumably was necessary when the idea was introduced. With todays technology, we can allow a markup on a finer scale, as has been seen in Orion-1.

ToDo: support distance instead of 0/1 indicators

We can gain additional freedom by using a correlation view: instead of looking from one axis, we can walk along the diagonal, using two reference axis.

Helpful hints how to interpret recurrence plots are in “Recurrence Plots At A Glance” <<http://www.recurrence-plot.tk/glance.php>>.

5. RECURRENCE QUANTIFICATION ANALYSIS

While visual inspection is the prime way to assess recurrence plots, quantification of some aspects revealed of the plot may be helpful. A collection of indices is provided by a recurrence quantification analysis (RQA) Zbilut and Webber [2006], Webber Jr and Zbilut [2005].

See Table table 1 on the next page.

This is a hack to report RQA information. $dim = NULL$ is added to align calling with other functions.

<pre>showrqa <- function(takens, dim=NULL, radius, do.hist = TRUE) { xxrqa <- rqa(takens=takens, radius=radius) cat(paste(deparse(substitute(takens)), " n=", dim(takens)[1], " Dim:", dim(takens)[2], "\n")) cat(paste("Radius:", radius, " Recurrence coverage REC:", round(xxrqa[1]\$REC, 3), "\n")) cat(paste("Determinism:", round(xxrqa\$DET,3), " Laminarity:",round(xxrqa\$LAM,3), "\n")) cat(paste("DIV:", round(xxrqa\$DIV,3), "\n")) cat(paste("Trend:", round(xxrqa\$TREND,3), " Entropy:",round(xxrqa\$ENTR,3), "\n")) cat(paste("Diagonal lines max:", round(xxrqa\$Lmax,3), " Mean:",round(xxrqa\$Lmean,3), " Mean off main:",round(xxrqa\$LmeanWithoutMain,3), "\n")) cat(paste("Vertical lines max:", round(xxrqa\$Vmax,3), " Mean:",round(xxrqa\$Vmean,3), "\n")) # str(xxrqa[4:12]) }</pre>	<i>Input</i>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------

TABLE 1. Recurrence Quantification Analysis (RQA)

<i>REC</i>	Recurrence. Percentage of recurrence points in a recurrence Plot.
<i>DET</i>	Determinism. Percentage of recurrence points that form diagonal lines.
<i>LAM</i>	Percentage of recurrent points that form vertical lines.
<i>RATIO</i>	Ratio between <i>DET</i> and <i>RR</i> .
<i>Lmax</i>	Length of the longest diagonal line.
<i>Lmean</i>	Mean length of the diagonal lines.
<i>DIV</i>	The main diagonal is not taken into account.
<i>Vmax</i>	Inverse of <i>Lmax</i> .
<i>Vmean</i>	Longest vertical line.
	Average length of the vertical lines.
<i>TREND</i>	This parameter is also referred to as the Trapping time.
<i>ENTR</i>	Shannon entropy of the diagonal line lengths distribution
<i>diagonalHistogram</i>	Trend of the number of recurrent points depending on the distance to the main diagonal
<i>recurrenceRate</i>	Histogram of the length of the diagonals. Number of recurrent points depending on the distance to the main diagonal.

```

oldpar <- par(mfrow=c(2,1))
if (do.hist){
  barplot(xxrqa$diagonalHistogram,
    main=paste(deparse(substitute(takens)), "Diagonal",
    "\n n=", dim(takens)[1], " Dim:",
    dim(takens)[2], " Radius: ",radius))
  barplot(xxrqa$recurrenceRate,
    main=paste(deparse(substitute(takens)), "Recurrence Rate",
    "\n n=", dim(takens)[1], " Dim:",
    dim(takens)[2], " Radius: ",radius))
}
par(oldpar)
invisible(xxrqa)
}

```

6. APPLIED RECURRENCE PLOTS

6.1. Sinus.

Input

```

sin10neighs<-local.findAllNeighbours(sintakens, radius=0.2)
save(sin10neighs, file="sin10neighs.Rdata")

load(file="sin10neighs.RData")
local.recurrencePlotAux(sin10neighs, dim=2, radius=0.2)

```

See Figure figure 10 on the facing page.

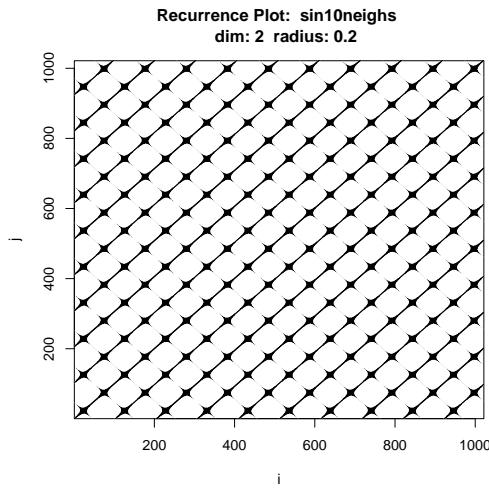


FIGURE 10. Recurrence Plot. Test case: sinus curves. Time used: 1.333 sec.

Input

```
showrqa(sintakens, radius=0.2)
```

Output

```
sintakens n= 1021 Dim: 4
Radius: 0.2 Recurrence coverage REC: 0.136
Determinism: 0.96 Laminarity: 0.982
DIV: 0.001
Trend: 0 Entropy: 2.74
Diagonal lines max: 1020 Mean: 14.263 Mean off main: 14.157
Vertical lines max: 25 Mean: 10.251
```

See Figure figure 11.

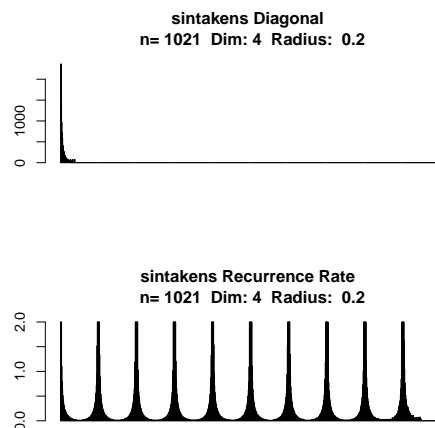


FIGURE 11. Recurrence Plot RQA. Test case: sinus curves. Time used: 0.41 sec.

6.2. Uniform random.

Input

```
load(file="unifneighs.RData")
local.recurrencePlotAux(unifneighs, radius=0.2)
```

See Figure figure 12.

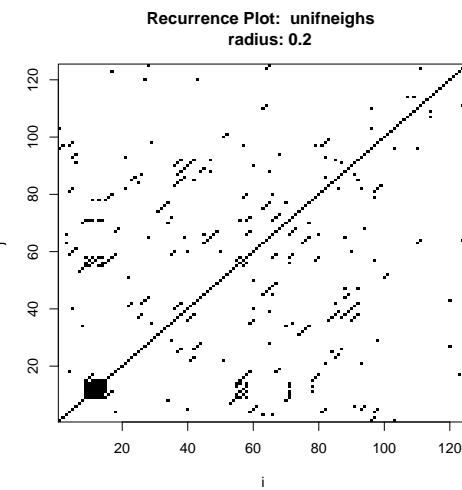


FIGURE 12. Recurrence Plot. Test case: uniform random numbers. Time used: 0.121 sec.

6.3. Chirp Signal.

Input

```
chirpneighs<-local.findAllNeighbours(chirptakens, radius=0.6)#0.4
save(chirpneighs, file="chirpneighs.RData")
```

Input

```
load(file="chirpneighs.RData")
local.recurrencePlotAux(chirpneighs)
```

See Figure figure 13 on the next page.

Input

```
showrqa(chirptakens, radius=0.6)
```

Output

```
chirptakens n= 1021 Dim: 4
Radius: 0.6 Recurrence coverage REC: 0.341
Determinism: 0.988 Laminarity: 0.998
DIV: 0.001
Trend: 0 Entropy: 3.254
Diagonal lines max: 1020 Mean: 12.496 Mean off main: 12.46
Vertical lines max: 125 Mean: 14.721
```

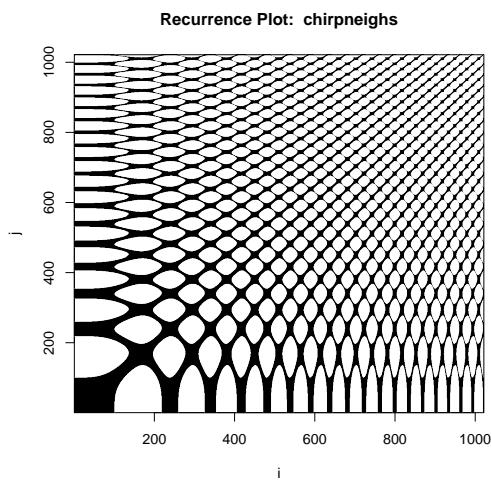


FIGURE 13. Recurrence Plot. Test case: chirp signal. Time used: 2.263 sec.

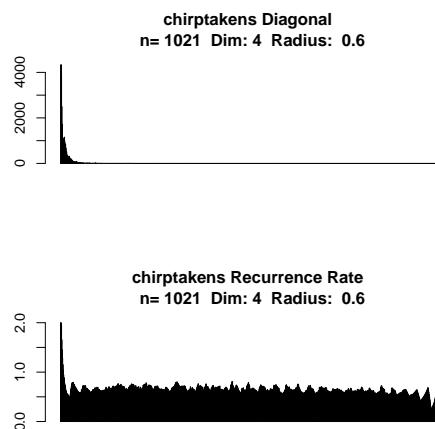


FIGURE 14. Recurrence Plot RQA. Test case: chirp signal. Time used: 2.882 sec.

See Figure figure 14 on the facing page.

7. CASE STUDY: GEYSER DATA

This is a classical data set with a two dimensional structure, *duration* and *waiting*.

```
library(MASS)
data(geyser)
```

Input

7.1. Geyser Eruptions.

```
plot(signal(geyser$duration)
```

Input

See Figure figure 15,

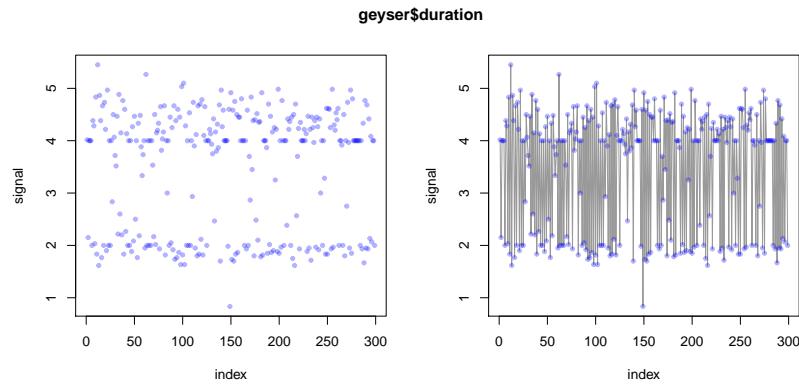


FIGURE 15. Example case: Old Faithful Geyser eruptions. Signal and linear interpolation.

```
eruptionstakens4 <-
local.buildTakens( time.series=geyser$duration,
embedding.dim=4, time.lag=1)
statepairs(eruptionstakens4)
```

See Figure figure 16 on the facing page

```
statepairs(eruptionstakens4, nooverlap=TRUE)
```

See Figure figure 17 on page 18

```
eruptionsneighs4<-local.findAllNeighbours(eruptionstakens4, radius=0.8)
save(eruptionsneighs4, file="eruptionsneighs4.RData")
```

```
load(file="eruptionsneighs4.RData")
local.recurrencePlotAux(eruptionsneighs4)
```

ToDo: double check:
MASS:::geyser should be used, not ***faithful***
ToDo: Geyser extended to two-dimensional data in ***geyserlin***. Check.

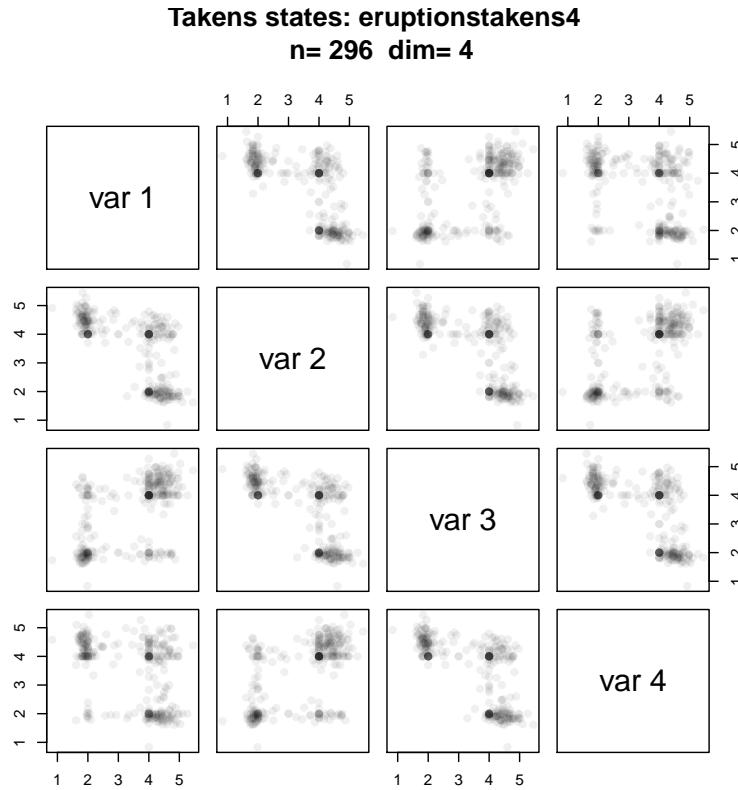


FIGURE 16. Example case: Old Faithful Geyser eruptions. Time used: 0.246 sec.

See Figure figure 18 on the following page.

Input

```
showrqa(eruptionstakens4, radius=0.8)
```

Output

```
eruptionstakens4 n= 296 Dim: 4
Radius: 0.8 Recurrence coverage REC: 0.112
Determinism: 0.903 Laminarity: 0.076
DIV: 0.05
Trend: 0 Entropy: 1.779
Diagonal lines max: 20 Mean: 3.919 Mean off main: 3.79
Vertical lines max: 5 Mean: 3.041
```

See Figure figure 19 on page 19.

7.1.1. Geyser eruptions. Dim=2.

Input

```
eruptionstakens2 <-
  local.buildTakens(time.series=geyser$duration,
                    embedding.dim=2, time.lag=1)
statepairs(eruptionstakens2)
```

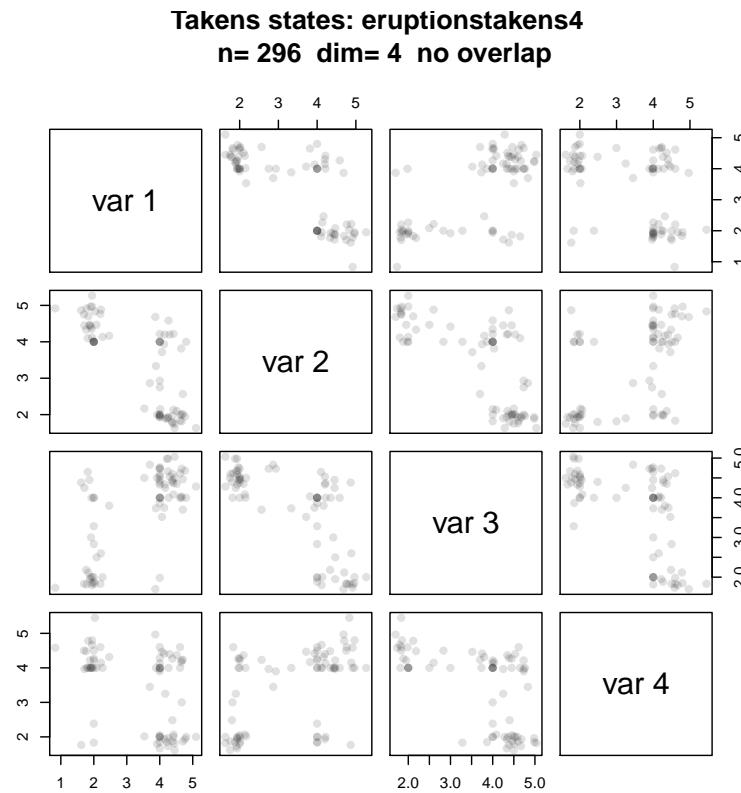


FIGURE 17. Example case: Old Faithful Geyser eruptions. Time used: 0.432 sec.

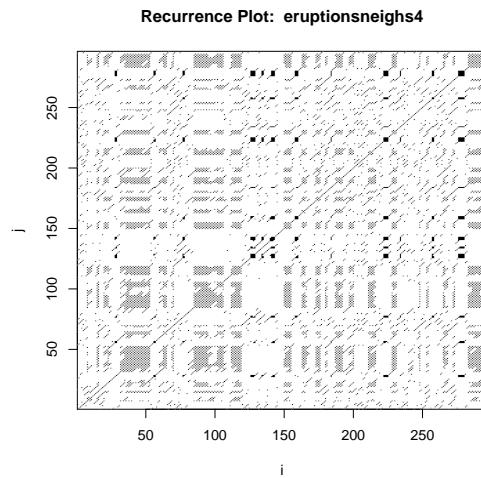


FIGURE 18. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.169 sec.

See Figure figure 20 on the facing page

Input
`statepairs(eruptionstakens2, nooverlap=TRUE)`

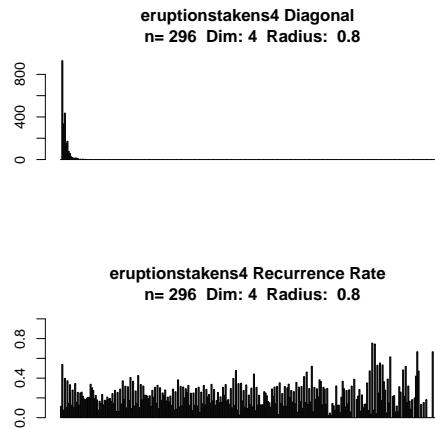


FIGURE 19. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.357 sec.

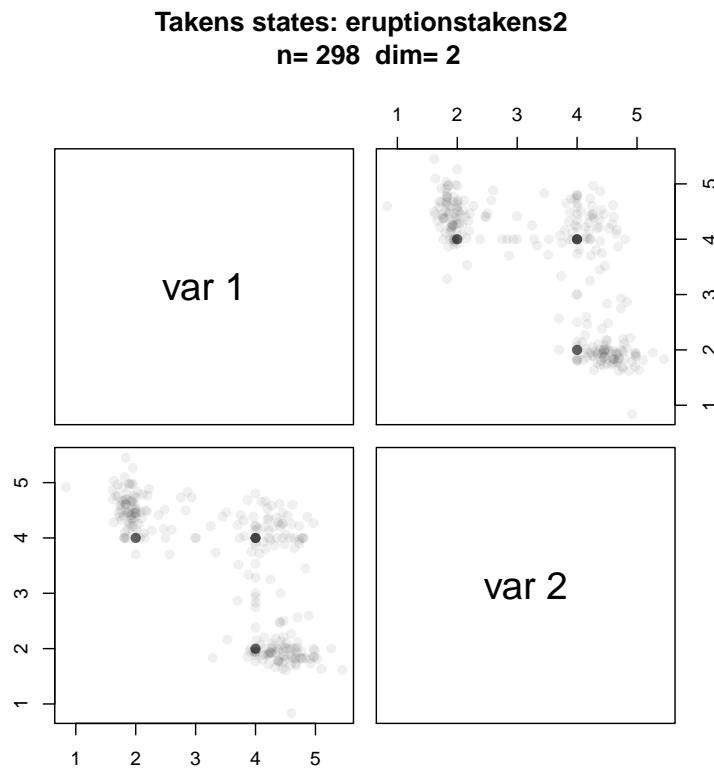


FIGURE 20. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.107 sec.

See Figure figure 21 on the next page

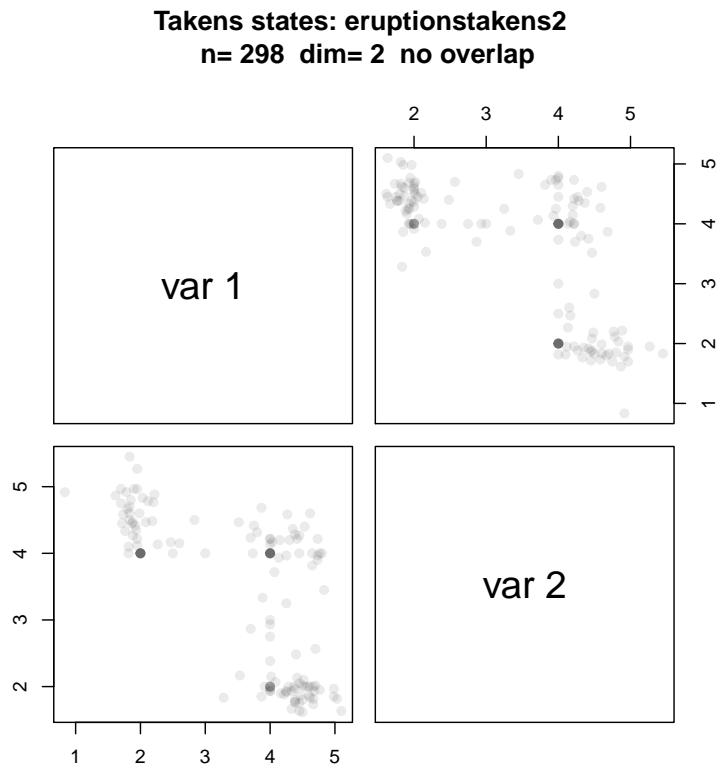


FIGURE 21. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.229 sec.

```
eruptionsneighs2<-local.findAllNeighbours(eruptionstakens2,
                                             radius=0.8)
save(eruptionsneighs2, file="eruptionsneighs2.RData")
```

Input

```
load(file="eruptionsneighs2.RData")
local.recurrencePlotAux(eruptionsneighs2)
```

See Figure figure 22 on the facing page.

Input

```
showrqa(eruptionstakens2, radius=0.8)
```

Output

```
eruptionstakens2 n= 298 Dim: 2
Radius: 0.8 Recurrence coverage REC: 0.274
Determinism: 0.892 Laminarity: 0.205
DIV: 0.045
Trend: 0 Entropy: 1.691
Diagonal lines max: 22 Mean: 3.644 Mean off main: 3.595
Vertical lines max: 7 Mean: 3.457
```

See Figure figure 23 on the next page.

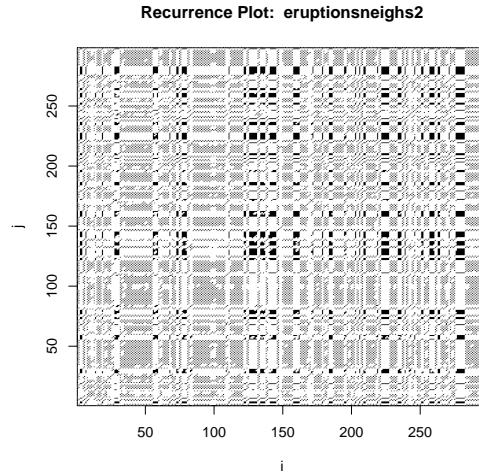


FIGURE 22. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.236 sec.

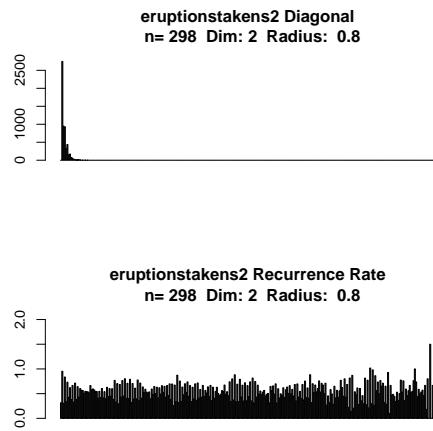


FIGURE 23. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.435 sec.

7.1.2. Geyser eruptions. Dim=6.

Input

```
eruptionstakens6 <- local.buildTakens( time.series=geyser$duration,
                                         embedding.dim=6,time.lag=1)
statepairs(eruptionstakens6)
```

See Figure figure 24 on the following page.

Input

```
statepairs(eruptionstakens6, nooverlap=TRUE)
```

See Figure figure 25 on page 23.

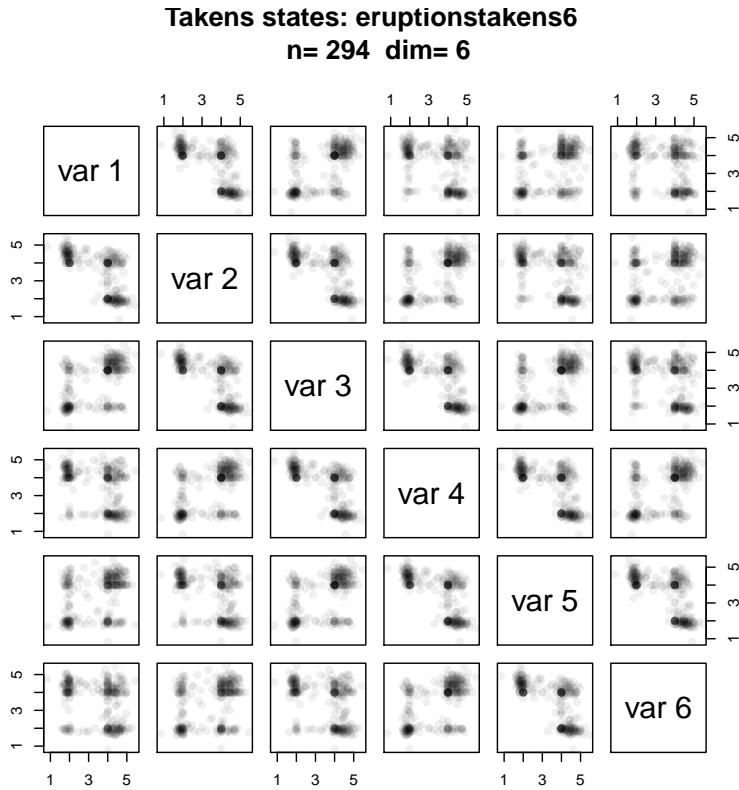


FIGURE 24. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.489 sec.

Input

```
eruptionsneighs6<-local.findAllNeighbours(eruptionstakens6,
                                             radius=0.8)
save(eruptionsneighs6, file="eruptionsneighs6.RData")
```

Input

```
load(file="eruptionsneighs6.RData")
local.recurrencePlotAux(eruptionsneighs6)
```

See Figure figure 26 on the facing page.

Input

```
showrqa(eruptionstakens6, radius=0.8)
```

Output

```
eruptionstakens6 n= 294 Dim: 6
Radius: 0.8 Recurrence coverage REC: 0.05
Determinism: 0.923 Laminarity: 0.016
DIV: 0.056
Trend: 0 Entropy: 1.748
Diagonal lines max: 18 Mean: 4.022 Mean off main: 3.73
Vertical lines max: 3 Mean: 2.29
```

See Figure figure 27 on page 24.

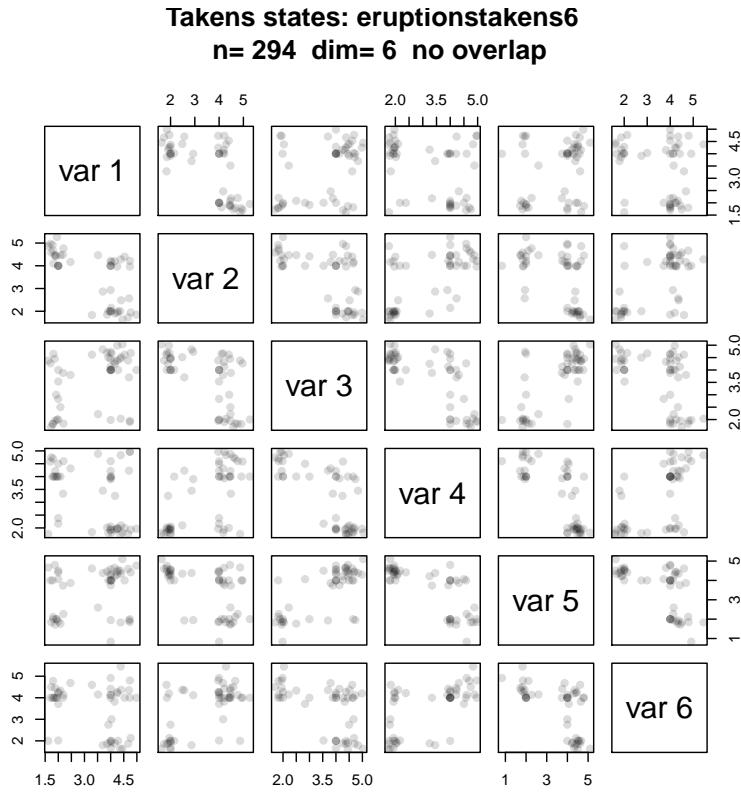


FIGURE 25. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.736 sec.

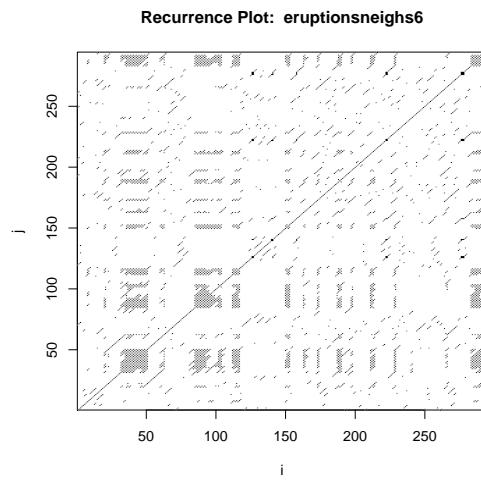


FIGURE 26. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.122 sec.

7.1.3. Geyser eruptions. Dim=8.

Input

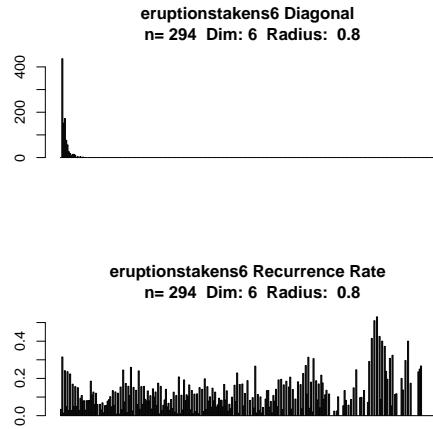


FIGURE 27. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.276 sec.

```
eruptionstakens8 <- local.buildTakens( time.series=geyser$duration,
                                         embedding.dim=8,time.lag=1)
statepairs(eruptionstakens8)
```

See Figure figure 28 on the facing page.

Input
statepairs(eruptionstakens8, nooverlap=TRUE)

See Figure figure 29 on page 26.

Input
eruptionsneighs8<-local.findAllNeighbours(eruptionstakens8,
 radius=0.8)
save(eruptionsneighs8, file="eruptionsneighs8.RData")

Input
load(file="eruptionsneighs8.RData")
local.recurrencePlotAux(eruptionsneighs8)

See Figure figure 30 on page 26.

Input
showrqa(eruptionstakens8, radius=0.8)

Output
eruptionstakens8 n= 292 Dim: 8
Radius: 0.8 Recurrence coverage REC: 0.024
Determinism: 0.924 Laminarity: 0
DIV: 0.062
Trend: 0 Entropy: 1.8
Diagonal lines max: 16 Mean: 4.583 Mean off main: 3.871
Vertical lines max: 0 Mean: 0

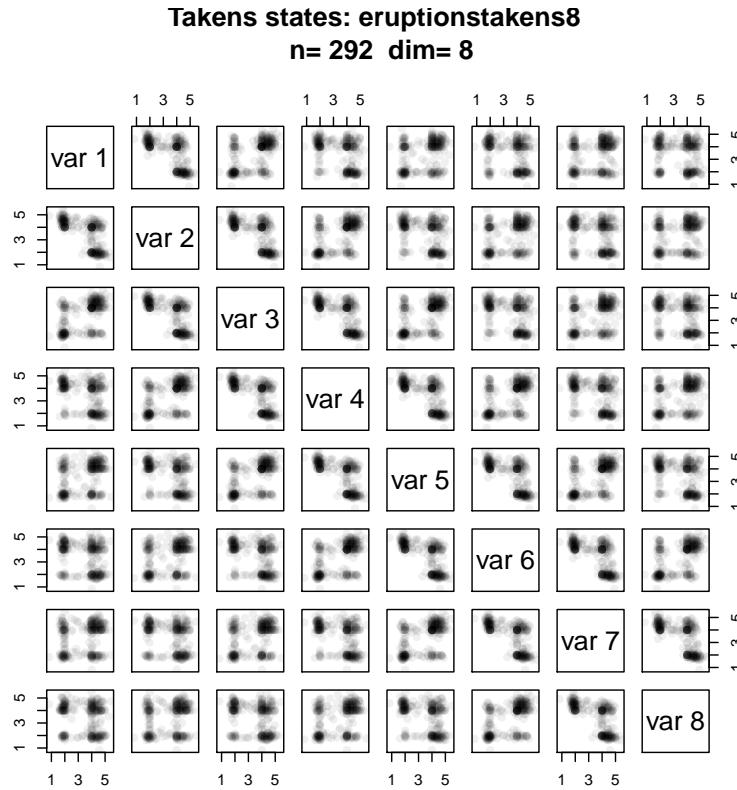


FIGURE 28. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.818 sec.

See Figure figure 31 on page 27.

7.2. Geyser Eruptions: Comparison by Dimension. For comparison, recurrence plots for the Geyser data with varying dimension are in Figure figure 32 on page 27

7.3. Geyser Waiting.

Input

```
plotsignal(geyser$waiting)
```

See Figure figure 33 on page 28.

Input

```
waitingtakens <-  
local.buildTakens( time.series=geyser$waiting,  
                   embedding.dim=4, time.lag=4)  
statepairs(waitingtakens)
```

See Figure figure 34 on page 28.

Input

```
waitingneighs<-local.findAllNeighbours(waitingtakens, radius=16)  
save(waitingneighs, file="waitingneighs.Rdata")
```

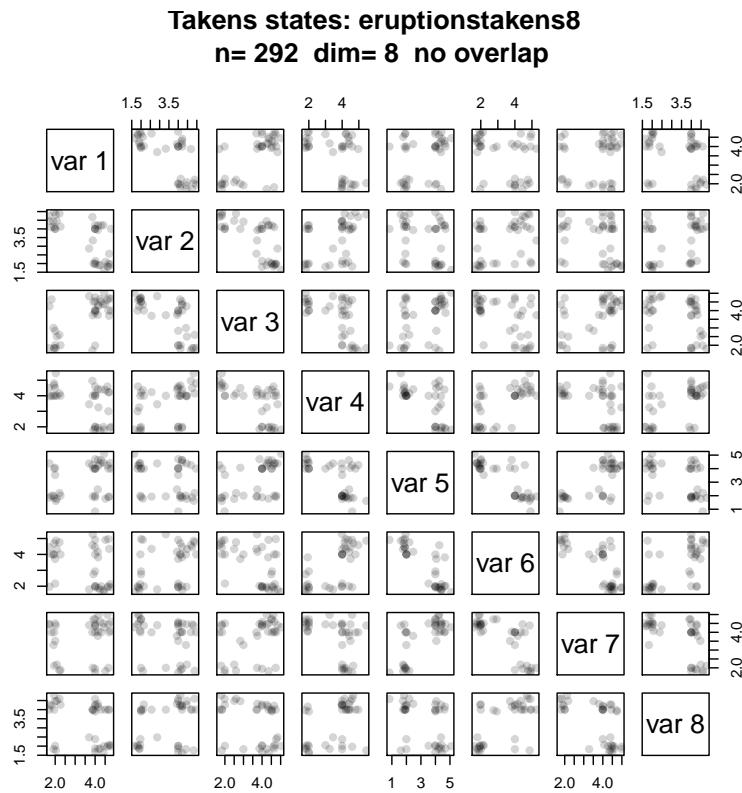


FIGURE 29. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 1.135 sec.

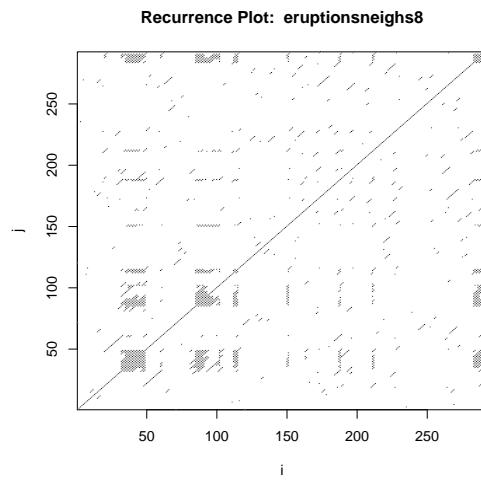


FIGURE 30. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.129 sec.

Input

```
showrqa(waitingtakens, radius=16)
```

Output

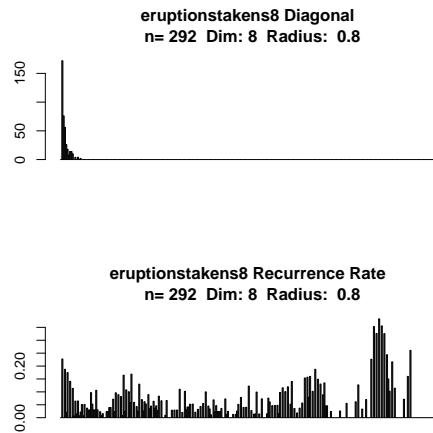


FIGURE 31. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.291 sec.

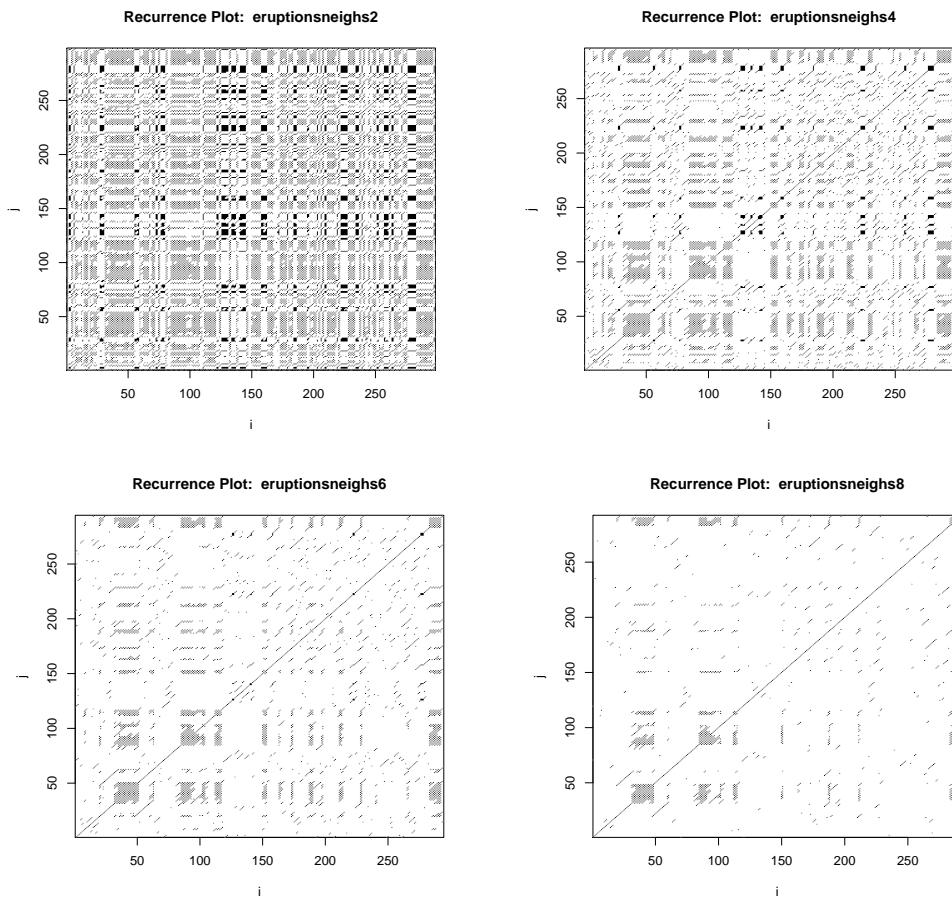


FIGURE 32. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=2, 4, 6, 8.

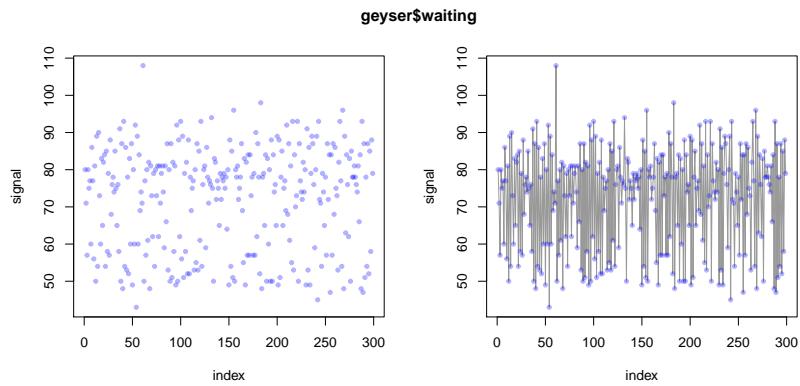


FIGURE 33. Example case: Old Faithful Geyser waiting. Signal and linear interpolation. Time used: 0.504 sec.

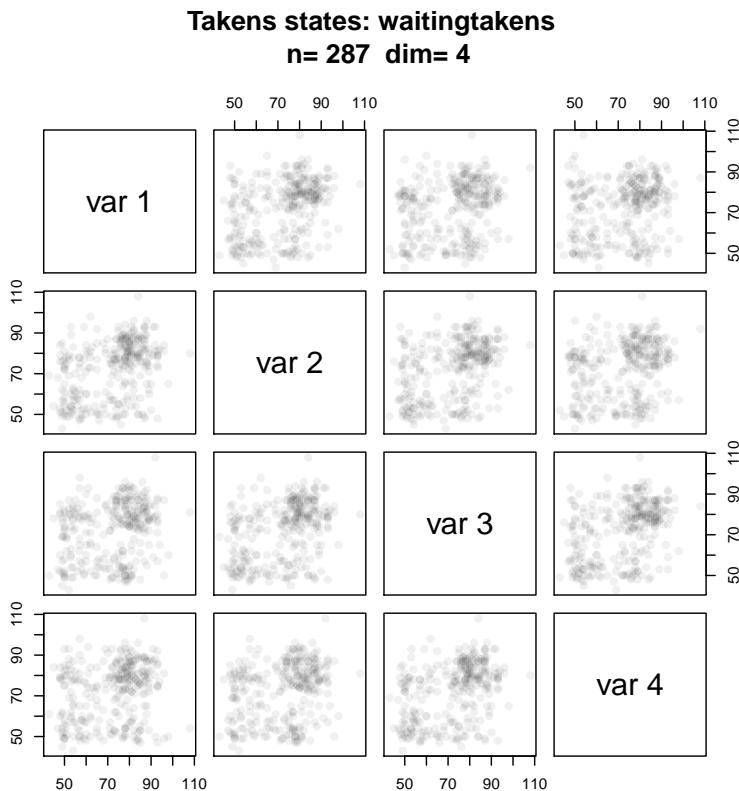


FIGURE 34. Example case: Old Faithful Geyser waiting. Time used: 0.256 sec.

```

waitingtakens n= 287 Dim: 4
Radius: 16 Recurrence coverage REC: 0.137
Determinism: 0.382 Laminarity: 0.053
DIV: 0.053
Trend: 0 Entropy: 1.002
Diagonal lines max: 19 Mean: 2.878 Mean off main: 2.688
Vertical lines max: 3 Mean: 2.315

```

See Figure figure 35.

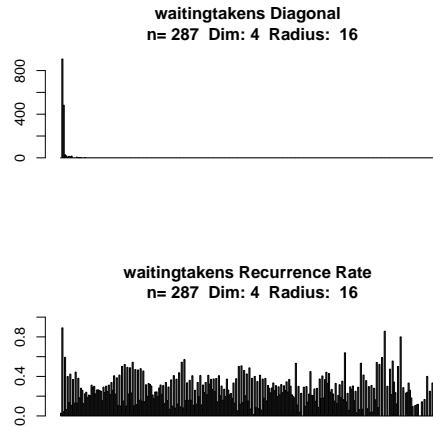


FIGURE 35. Recurrence Plot RQA. Example case: Old Faithful Geyser waiting. Time used: 0.156 sec.

Input

```
load(file="waitingneighs.RData")
local.recurrencePlotAux(waitingneighs)
```

See Figure ?? on page ??.

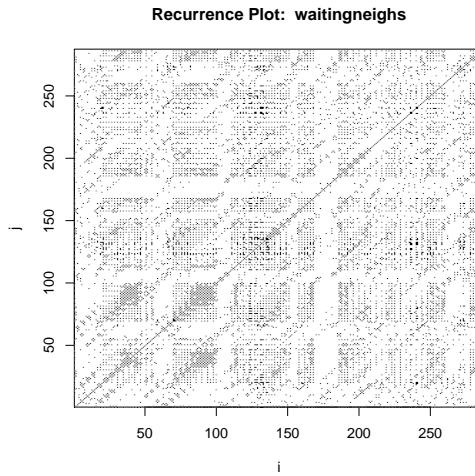


FIGURE 36. Recurrence Plot. Example case: Old Faithful Geyser waiting. Time used: 0.356 sec.

7.4. Geyser - linearized. So far, *nonlinearTseries* only handles multivariate data by FORTRAN conventions, using a lag parameter.

As a hack, we transform the data to FORTRAN conventions.

Input

```
geyserlin <- t(geyser)
dim(geyserlin)<-NULL
dimnames(geyserlin)<-NULL
```

Now duration and waiting are mixed. A $lag = 2$ separates the dimension again. The Taken states iterate over the index, giving alternating a duration and waiting state.

7.5. Geyser Eruptions linearized.

Input

```
plotsignal(geyserlin)
```

See Figure figure 37.

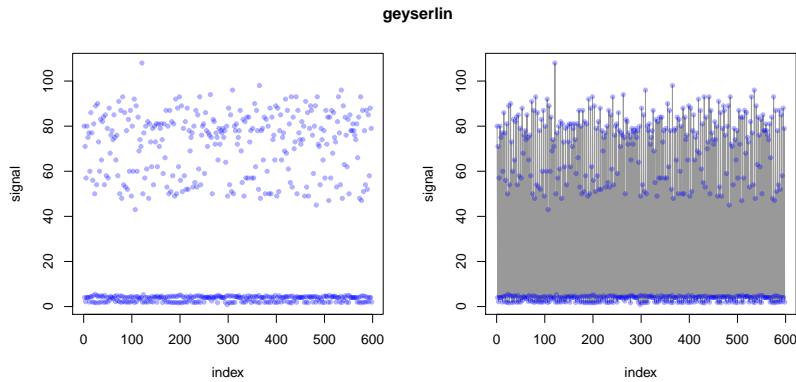


FIGURE 37. Example case: Old Faithful Geyser eruptions. Signal and linear interpolation.

Input

```
gleruptionstakens4 <-
  local.buildTakens( time.series=geyserlin,
                     embedding.dim=4, time.lag=2)
statepairs(gleruptionstakens4)
```

See Figure figure 38 on the facing page.

Input

```
eruptionsneighs4<-local.findAllNeighbours(gleruptionstakens4, radius=0.8)
save(eruptionsneighs4, file="eruptionsneighs4.RData")
```

Input

```
showrqa(gleruptionstakens4, radius=0.8)
```

Output

```
gleruptionstakens4 n= 592 Dim: 4
Radius: 0.8 Recurrence coverage REC: 0.029
Determinism: 0.059 Laminarity: 0
DIV: Inf
Trend: 0 Entropy: 0
```

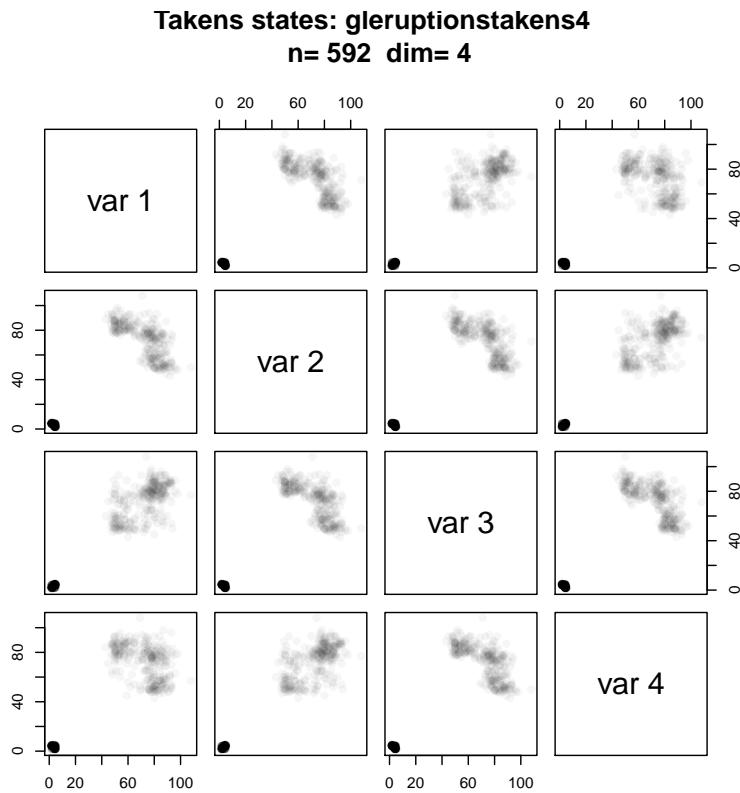


FIGURE 38. Example case: Old Faithful Geyser eruptions. Time used: 0.402 sec.

```
Diagonal lines max: 0 Mean: 592 Mean off main: NaN
Vertical lines max: 0 Mean: 0
```

See Figure figure 39.

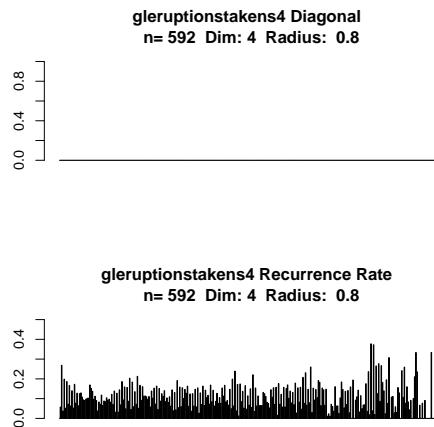


FIGURE 39. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.257 sec.

Input

```
load(file="eruptionsneighs4.RData")
local.recurrencePlotAux(eruptionsneighs4)
```

See Figure figure 40.

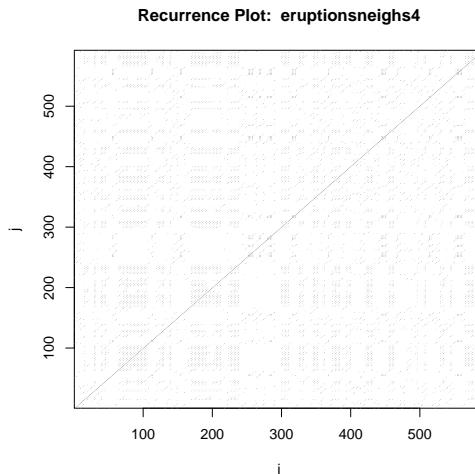


FIGURE 40. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.584 sec.

7.5.1. Geyser eruptions - linearized. Dim=2.

Input

```
gleruptionstakens2 <-
  local.buildTakens(time.series=geyserlin,
                    embedding.dim=2, time.lag=2)
statepairs(gleruptionstakens2)
```

See Figure figure 41 on the next page.

Input

```
eruptionsneighs2<-local.findAllNeighbours(gleruptionstakens2, radius=0.8)
save(eruptionsneighs2, file="eruptionsneighs2.RData")
```

Input

```
load(file="eruptionsneighs2.RData")
local.recurrencePlotAux(eruptionsneighs2)
```

See Figure figure 42 on the facing page.

7.5.2. Geyser eruptions - linearized. Dim=6.

Input

```
gleruptionstakens6 <- local.buildTakens( time.series=geyserlin,
                                             embedding.dim=6,time.lag=2)
statepairs(gleruptionstakens6)
```

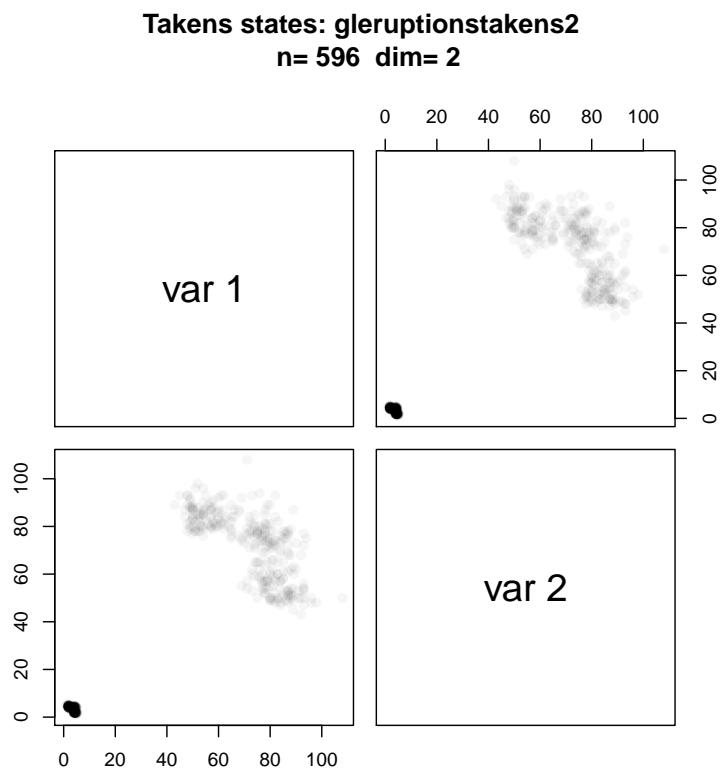


FIGURE 41. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.148 sec.

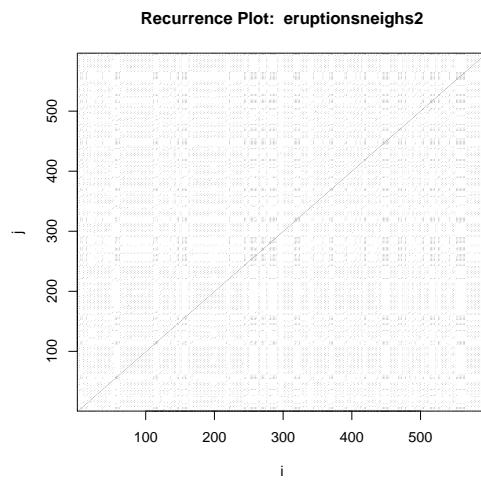


FIGURE 42. Recurrence Plot. Example case: Old Faithful Geyser eruptions linearized. Dim=2. Time used: 0.373 sec.

See Figure figure 43 on the next page

Input

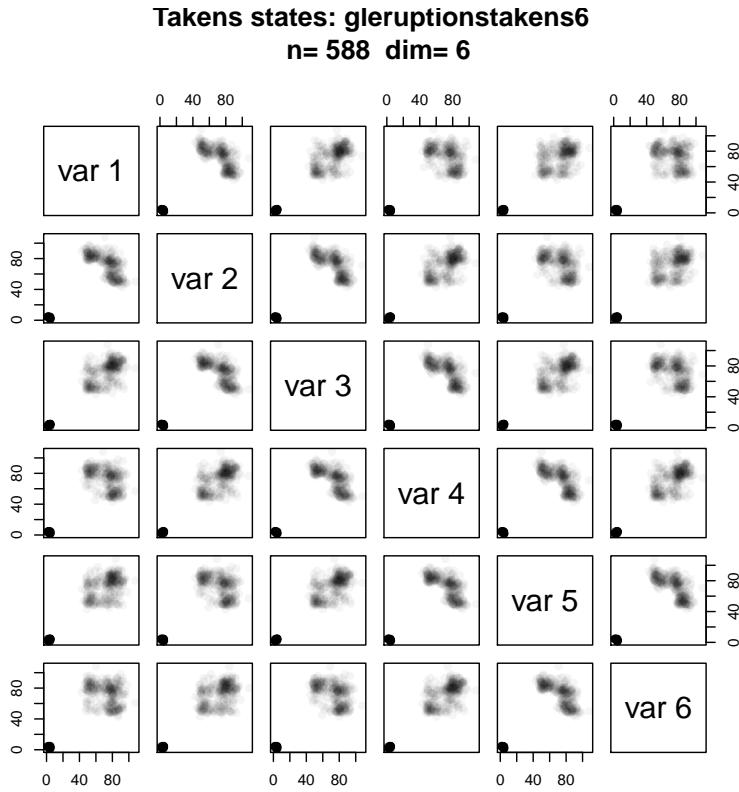


FIGURE 43. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.836 sec.

```
eruptionsneighs6<-local.findAllNeighbours(gleruptionstakens6, radius=0.8)
save(eruptionsneighs6, file="eruptionsneighs6.RData")
```

Input

```
load(file="eruptionsneighs6.RData")
local.recurrencePlotAux(eruptionsneighs6)
```

See Figure figure 44 on the facing page.

7.5.3. Geyser eruptions - linearized. Dim=8.

Input

```
gleruptionstakens8 <- local.buildTakens(time.series=geyserlin,
    embedding.dim=8,time.lag=2)
statepairs(gleruptionstakens8)
```

See Figure figure 45 on the next page

Input

```
eruptionsneighs8<-local.findAllNeighbours(gleruptionstakens8,
    radius=0.8)
save(eruptionsneighs8, file="eruptionsneighs8.RData")
```

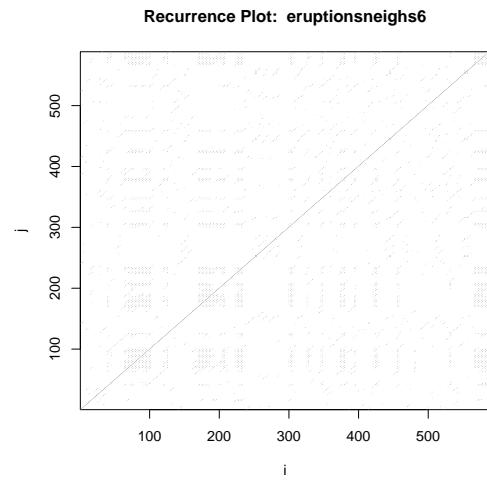


FIGURE 44. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.24 sec.

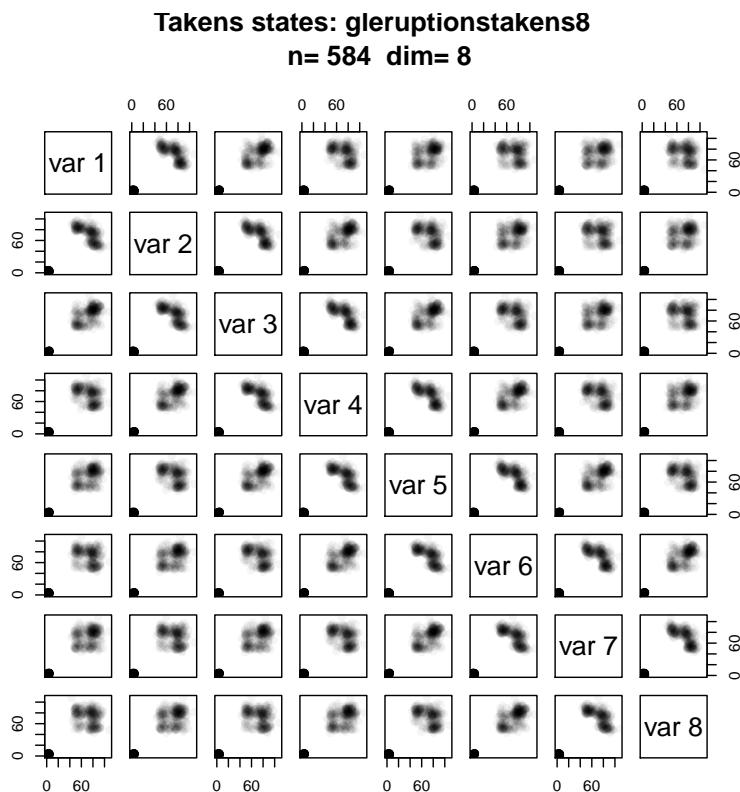


FIGURE 45. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 1.473 sec.

Input

```
load(file="eruptionsneighs8.RData")
local.recurrencePlotAux(eruptionsneighs8)
```

See Figure figure 46

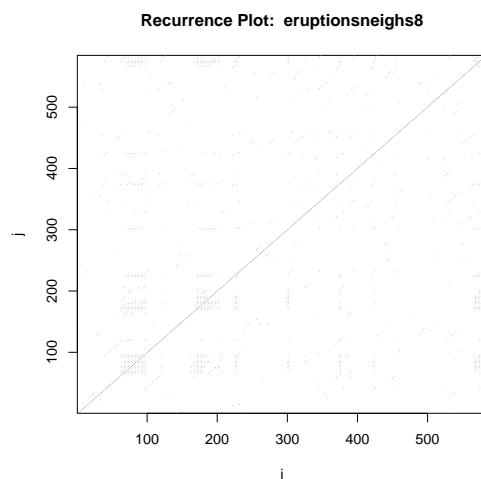


FIGURE 46. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.225 sec.

Only 1024 data points used in this section

8. CASE STUDY: HRV DATA

Input

```
library(RHRV)
load("/users/gs/projects/rforge/rhrv/pkg/data/HRVData.rda")
load("/users/gs/projects/rforge/rhrv/pkg/data/HRVProcessedData.rda")
#####
### code chunk number 1: creation
#####
hrv.data = CreateHRVData()
hrv.data = SetVerbose(hrv.data, TRUE )
#####
### code chunk number 3: loading
#####
hrv.data = LoadBeatAscii(hrv.data, "example.beats",
  RecordPath = "/users/gs/projects/rforge/rhrv/tutorial/beatsFolder")
```

Output

```
** Loading beats positions for record: example.beats **
Path: /users/gs/projects/rforge/rhrv/tutorial/beatsFolder
Scale: 1
Date: 01/01/1900
Time: 00:00:00
Number of beats: 17360
```

Input

```
# RecordPath = "beatsFolder")

#####
### code chunk number 4: deriving
#####
hrv.data = BuildNIHR(hrv.data)
```

Output

```
** Calculating non-interpolated heart rate **
Number of beats: 17360
```

Input

```
plotsignal(hrv.data$Beat$RR)
```

ToDo: We have outliers at approximately $2 \times RR$. Could this be an artefact of preprocessing, filtering out too many impulses?

See Figure figure 47 on the next page.

Input

```
hrvRRTakens4 <- local.buildTakens( time.series=hrv.data$Beat$RR[1:nsignal],
  embedding.dim=4, time.lag=1)
statepairs(hrvRRTakens4)
```

See Figure figure 48 on the following page.

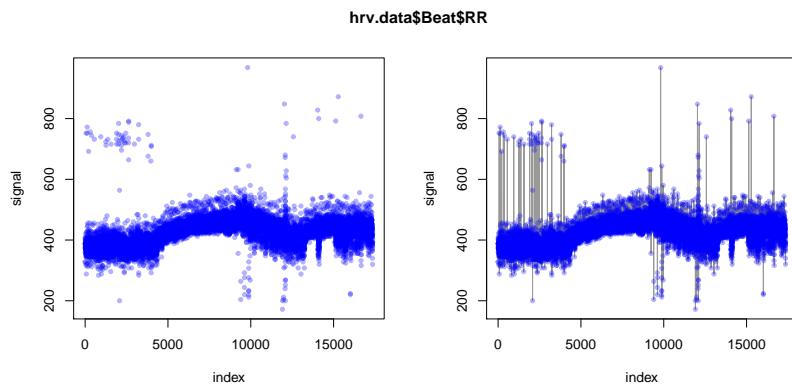


FIGURE 47. RHRV tutorial example.beats. Signal and linear interpolation.

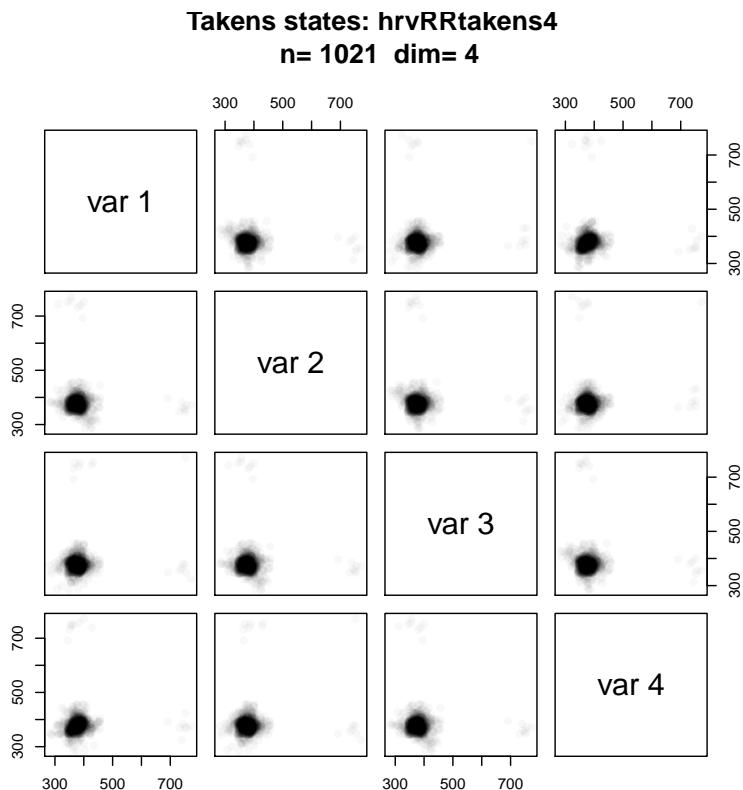


FIGURE 48. RHRV tutorial example.beats. Time used: 0.526 sec.

Input
`statepairs(hrvRRtakens4, rank=TRUE)`

See Figure figure 49 on the next page.

Input
`hrvRNeighs4 <- local.findAllNeighbours(hrvRRtakens4, radius=16)`
`save(hrvRNeighs4, file="hrvRNeighs4.Rdata")`

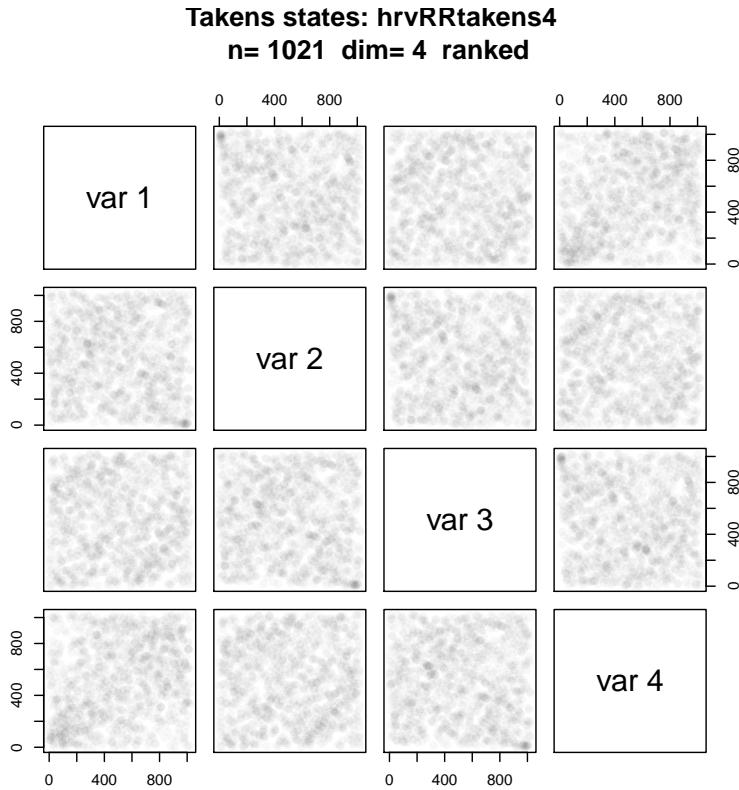


FIGURE 49. RHRV tutorial example.beats. Ranked data. Time used: 1.187 sec.

Time used: 0.109 sec.

Input

```
load(file="hrvRRneighs4.RData")
local.recurrencePlotAux(hrvRRneighs4)
```

See Figure figure 50 on the following page.

8.1. RHRV: Comparison by Dimension.

Input

```
hrvRRtakens2 <- local.buildTakens( time.series=hrv.data$Beat$RR[1:nsignal],
                                       embedding.dim=2,time.lag=1)
hrvRRneighs2 <- local.findAllNeighbours(hrvRRtakens2, radius=16)
save(hrvRRneighs2, file="hrvRRneighs2.Rdata")
```

Time used: 0.123 sec.

Input

```
load(file="hrvRRneighs2.RData")
local.recurrencePlotAux(hrvRRneighs2)
```

See Figure figure 51 on page 42. Time used: 1.692 sec.

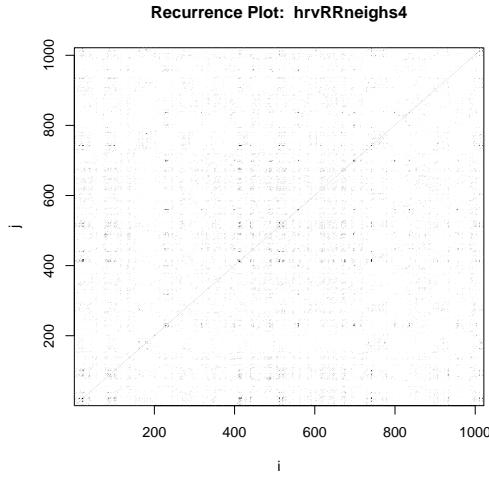


FIGURE 50. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=4. Time used: 0.74 sec.

Input

```
hrvRRtakens6 <- local.buildTakens( time.series=hrv.data$Beat$RR[1:nSignal],
                                     embedding.dim=6, time.lag=1)
hrvRRneighs6 <- local.findAllNeighbours(hrvRRtakens6, radius=16)
save(hrvRRneighs6, file="hrvRRneighs6.Rdata")
```

Time used: 0.178 sec.

Input

```
load(file="hrvRRneighs6.RData")
local.recurrencePlotAux(hrvRRneighs6)
```

Dim=6. Time used: 0.835 sec.

Input

```
hrvRRtakens8 <- local.buildTakens( time.series=hrv.data$Beat$RR[1:nSignal],
                                     embedding.dim=8, time.lag=1)
hrvRRneighs8 <- local.findAllNeighbours(hrvRRtakens8, radius=32)
save(hrvRRneighs8, file="hrvRRneighs8.Rdata")
```

Time used: 0.296 sec.

Input

```
load(file="hrvRRneighs8.RData")
local.recurrencePlotAux(hrvRRneighs8)
```

Dim=8. Time used: 1.061 sec.

Input

```
hrvRRtakens12 <- local.buildTakens( time.series=hrv.data$Beat$RR[1:nSignal],
                                       embedding.dim=2, time.lag=1)
hrvRRneighs12 <- local.findAllNeighbours(hrvRRtakens12, radius=16)
save(hrvRRneighs12, file="hrvRRneighs12.Rdata")
```

Time used: 1.295 sec.

Input

```
load(file="hrvRRneighs12.RData")
local.recurrencePlotAux(hrvRRneighs12)
```

Time used: 1.656 sec.

Input

```
hrvRRTakens16 <- local.buildTakens(
  time.series=hrv.data$Beat$RR[1:nSignal],
  embedding.dim=16, time.lag=1)
hrvRRneighs16 <- local.findAllNeighbours(hrvRRTakens16, radius=32)
save(hrvRRneighs16, file="hrvRRneighs16.Rdata")
```

Time used: 1.953 sec.

Input

```
load(file="hrvRRneighs16.RData")
local.recurrencePlotAux(hrvRRneighs16)
```

Time used: 0.673 sec.

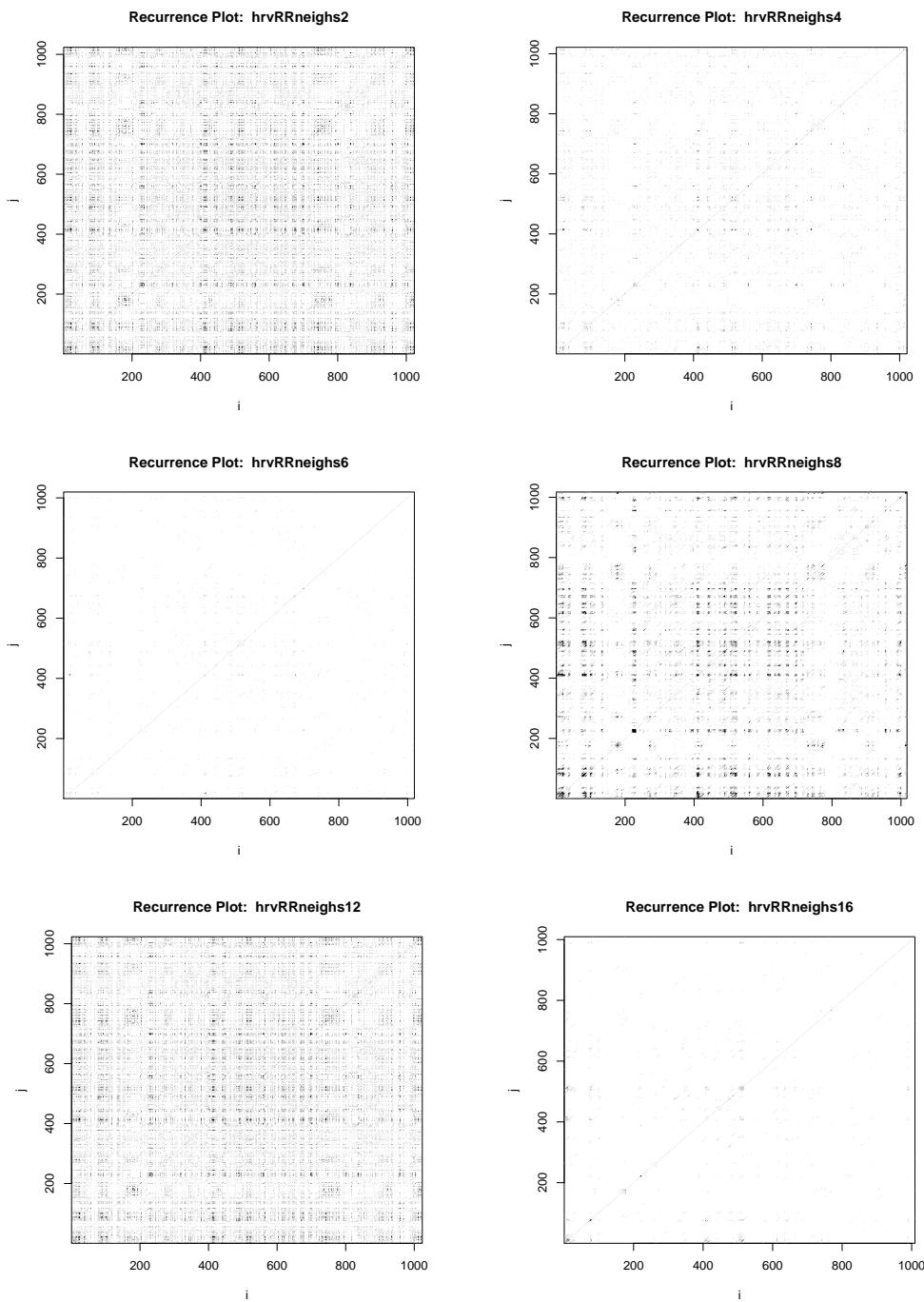


FIGURE 51. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 0.673 sec.

ToDo: Consider using differences

8.2. **Hart Rate Variation.** Since we are not interested in heart rate (or pulse), but in heart rate variation, a proposal is to use scaled differences.

```
Input
# source('/users/gs/projects/rforge/rhrv/pkg/R/BuildNIHR2.R', chdir = TRUE)
BuildNIDHR <-
function(HRVData, verbose=NULL) {
#-----
# Obtains instantaneous heart rate variation from beats positions
# D for difference
#-----
if (!is.null(verbose)) {
    cat(" --- Warning: deprecated argument, using SetVerbose() instead ---\n",
        "     --- See help for more information!! ---\n")
    SetVerbose(HRVData,verbose)
}

if (HRVData$Verbose) {
    cat("** Calculating non-interpolated heart rate differences **\n")
}

if (is.null(HRVData$Beat$Time)) {
    cat(" --- ERROR: Beats positions not present... Impossible to calculate Heart Rate!! ---\n")
    return(HRVData)
}

NBeats=length(HRVData$Beat$Time)
if (HRVData$Verbose) {
    cat(" Number of beats:",NBeats,"\\n");
}

# addition gs
#using NA, not constant extrapolation as else in RHRV
#drr=c(NA,NA,1000.0* diff(HRVData$Beat$Time, lag=1 , differences=2))
HRVData$Beat$dRR=c(NA, NA,
    1000.0*diff(HRVData$Beat$Time, lag=1, differences=2))

HRVData$Beat$avRR=(c(NA,HRVData$Beat$RR[-1])+HRVData$Beat$RR)/2

HRVData$Beat$HRRV <- HRVData$Beat$dRR/HRVData$Beat$avRR

return(HRVData)
}
```

differences for HRV

Input

```
hrv.data <- BuildNIDHR(hrv.data)
```

Output

```
** Calculating non-interpolated heart rate differences **
Number of beats: 17360
```

Input

```
HRRV <- hrv.data$Beat$HRRV
```

These are the displays of the Takens state space we used before, now for HRRV:

`plotsignal(HRRV)`

Input

See Figure figure 52,

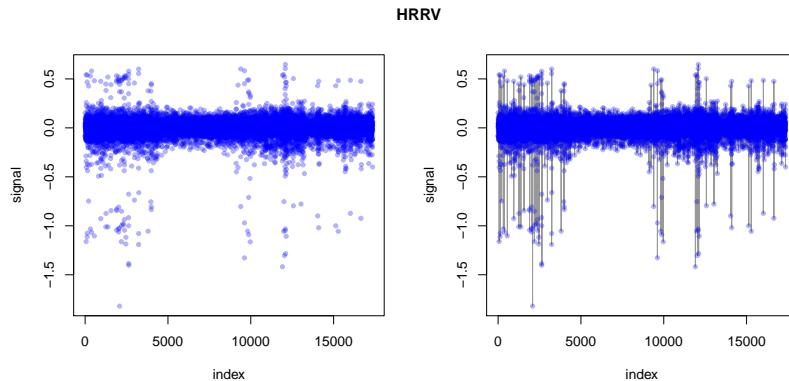


FIGURE 52. RHRV tutorial example.beats. HRRV Signal and linear interpolation.

Only 1024 data points used in this section

`hrvRRVtakens4 <-
local.buildTakens(time.series=HRRV[1:nSignal],
embedding.dim=4,time.lag=1)
statepairs(hrvRRVtakens4)`

See Figure figure 53 on the next page

`statepairs(hrvRRVtakens4, rank=TRUE)`

Input

See Figure figure 54 on page 46

ToDo: findAllNeighbours does not handle NAs

`#use hack: findAllNeighbours does not handle NAs
hrvRRVneighs4 <- local.findAllNeighbours(hrvRRVtakens4[-(1:2),], radius=0.125)
save(hrvRRVneighs4, file="hrvRRVneighs4.Rdata")`

Time used: 0.236 sec.

`load(file="hrvRRVneighs4.RData")
local.recurrencePlotAux(hrvRRVneighs4, dim=4, radius=0.125)`

8.3. RHRV Variation: Comparison by Dimension.

ToDo: check. There seem to be strange artefacts.

We should expect the breathing rhythm, so a time lag in the order of 10 is to be expected.

ToDo: fix default setting for radius.

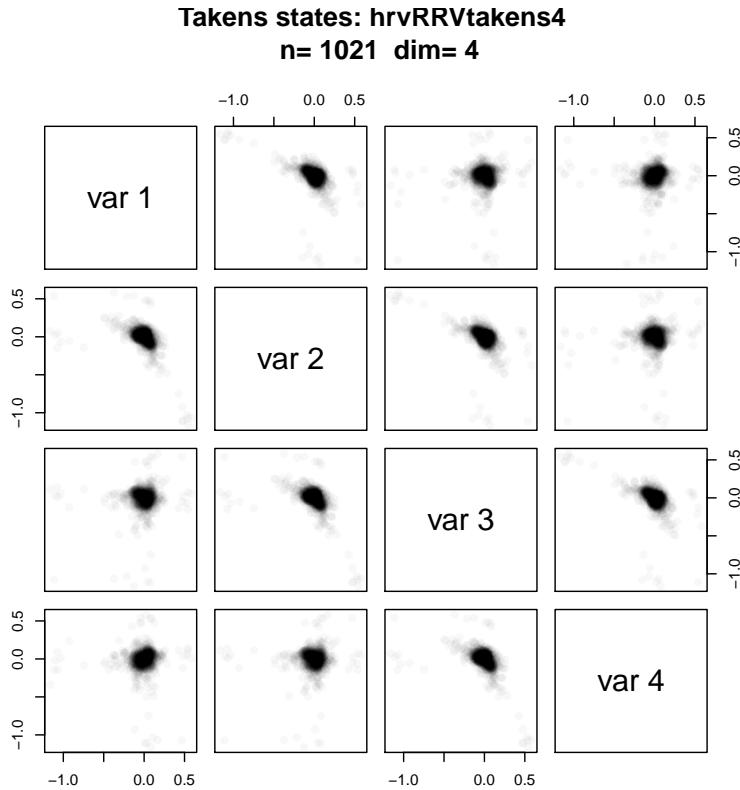


FIGURE 53. RHRV tutorial example.beats. HRRV Time used: 0.667 sec.

Input

```
hrvRRVtakens2 <- local.buildTakens( time.series=HRRV[1:nseries],
                                       embedding.dim=2, time.lag=1)
hrvRRVneighs2 <- local.findAllNeighbours(hrvRRVtakens2[-(1:2), ],
                                             radius=0.125)
save(hrvRRVneighs2, file="hrvRRVneighs2.Rdata")
```

Time used: 0.428 sec.

Input

```
showrqa(hrvRRVtakens2[-(1:2), ], radius=0.125, do.hist=FALSE)
```

Output

```
hrvRRVtakens2[-(1:2), ] n= 1021 Dim: 2
Radius: 0.125 Recurrence coverage REC: 0.515
Determinism: 0.939 Laminarity: 0.81
DIV: 0.027
Trend: 0 Entropy: 2.346
Diagonal lines max: 37 Mean: 5.373 Mean off main: 5.362
Vertical lines max: 48 Mean: 3.998
```

Input

```
load(file="hrvRRVneighs2.RData")
local.recurrencePlotAux(hrvRRVneighs2, dim=2, radius=0.125)
```

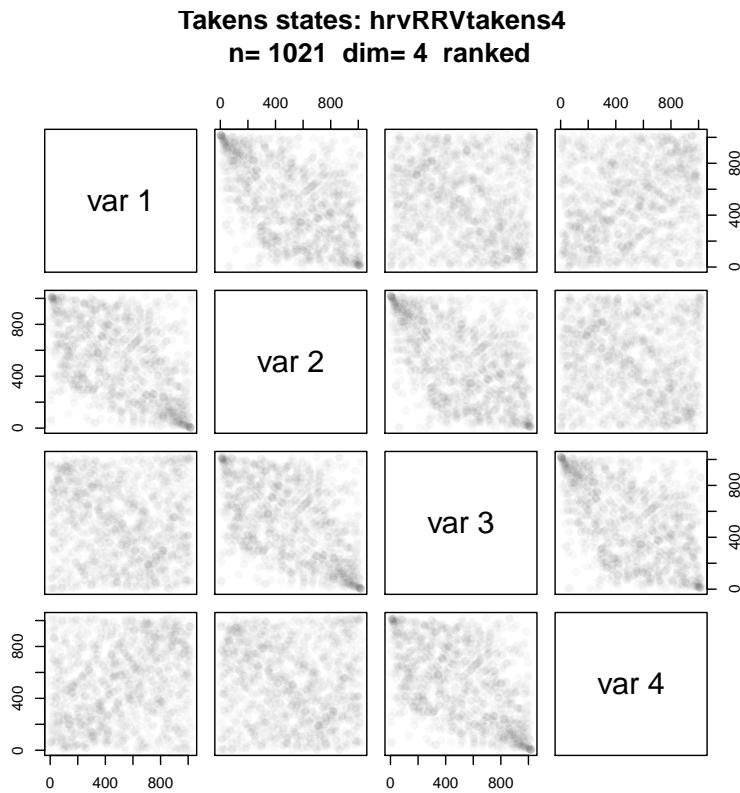


FIGURE 54. RHRV tutorial example.beats. Ranked HRRV data. Time used: 1.326 sec.

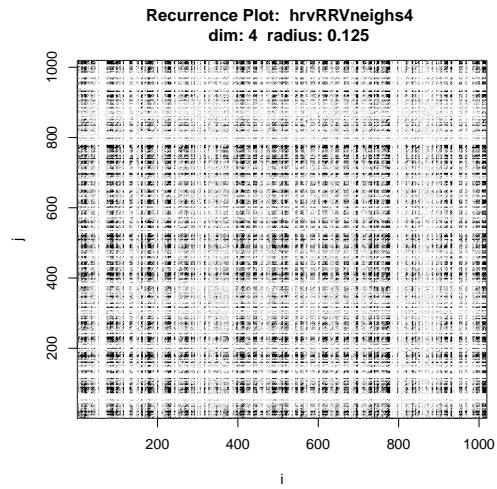


FIGURE 55. Recurrence Plot. Example case: RHRV tutorial example.beats. HRRV Dim=4. Time used: 2.47 sec.

Time used: 3.797 sec.

Input

```
hrvRRVtakens6 <- local.buildTakens( time.series=HRRV[1:nseries],
                                      embedding.dim=6, time.lag=1)
hrvRRVneighs6 <- local.findAllNeighbours(hrvRRVtakens6[-(1:2),], radius=0.125)
save(hrvRRVneighs6, file="hrvRRVneighs6.Rdata")
```

Time used: 0.483 sec.

Input

```
showrqa(hrvRRVtakens6[-(1:2),], radius=0.125, do.hist=FALSE)
```

Output

```
hrvRRVtakens6[-(1:2), ] n= 1017 Dim: 6
Radius: 0.125 Recurrence coverage REC: 0.179
Determinism: 0.943 Laminarity: 0.451
DIV: 0.03
Trend: 0 Entropy: 2.305
Diagonal lines max: 33 Mean: 5.243 Mean off main: 5.213
Vertical lines max: 22 Mean: 2.887
```

Input

```
load(file="hrvRRVneighs6.RData")
local.recurrencePlotAux(hrvRRVneighs6, dim=6, radius=0.125)
```

Dim=6. Time used: 1.792 sec.

Input

```
hrvRRVtakens8 <- local.buildTakens( time.series=HRRV[1:nseries],
                                      embedding.dim=8, time.lag=1)
hrvRRVneighs8 <- local.findAllNeighbours(hrvRRVtakens8[-(1:2),], radius=0.125)
save(hrvRRVneighs8, file="hrvRRVneighs8.Rdata")
```

Time used: 0.342 sec.

Input

```
showrqa(hrvRRVtakens8[-(1:2),], radius=0.125, do.hist=FALSE)
```

Output

```
hrvRRVtakens8[-(1:2), ] n= 1015 Dim: 8
Radius: 0.125 Recurrence coverage REC: 0.105
Determinism: 0.943 Laminarity: 0.337
DIV: 0.032
Trend: 0 Entropy: 2.329
Diagonal lines max: 31 Mean: 5.361 Mean off main: 5.308
Vertical lines max: 17 Mean: 2.715
```

Input

```
load(file="hrvRRVneighs8.RData")
local.recurrencePlotAux(hrvRRVneighs8, dim=8, radius=0.125)
```

Dim=8. Time used: 1.256 sec.

Input

```

hrvRRVtakens12 <-
  local.buildTakens( time.series=HRRV[1:nSignal],
                     embedding.dim=12, time.lag=1)
hrvRRVneighs12 <-
  local.findAllNeighbours(hrvRRVtakens12[-(1:2),], radius=3/16)
save(hrvRRVneighs12, file="hrvRRVneighs12.Rdata")

```

Time used: 1.852 sec.

<i>Input</i>		<i>Output</i>
--------------	--	---------------

```
showrqa(hrvRRVtakens12[-(1:2),], radius=3/16, do.hist=FALSE)
```

<i>Input</i>		<i>Output</i>
--------------	--	---------------

```

hrvRRVtakens12[-(1:2), ] n= 1011 Dim: 12
Radius: 0.1875 Recurrence coverage REC: 0.235
Determinism: 0.988 Laminarity: 0.635
DIV: 0.015
Trend: 0 Entropy: 3.075
Diagonal lines max: 68 Mean: 9.775 Mean off main: 9.733
Vertical lines max: 52 Mean: 3.656

```

<i>Input</i>		<i>Output</i>
--------------	--	---------------

```

load(file="hrvRRVneighs12.RData")
local.recurrencePlotAux(hrvRRVneighs12, dim=12, radius=3/16)

```

Time used: 1.982 sec.

<i>Input</i>		<i>Output</i>
--------------	--	---------------

```

hrvRRVtakens16 <- local.buildTakens( time.series=HRRV[1:nSignal],
                                         embedding.dim=16, time.lag=1)
hrvRRVneighs16 <- local.findAllNeighbours(hrvRRVtakens16[-(1:2),], radius=3/16)
save(hrvRRVneighs16, file="hrvRRVneighs16.Rdata")

```

Time used: 2.42 sec.

<i>Input</i>		<i>Output</i>
--------------	--	---------------

```
showrqa(hrvRRVtakens16[-(1:2),], radius=3/16, do.hist=FALSE)
```

<i>Input</i>		<i>Output</i>
--------------	--	---------------

```

hrvRRVtakens16[-(1:2), ] n= 1007 Dim: 16
Radius: 0.1875 Recurrence coverage REC: 0.147
Determinism: 0.99 Laminarity: 0.527
DIV: 0.016
Trend: 0 Entropy: 3.163
Diagonal lines max: 64 Mean: 10.594 Mean off main: 10.523
Vertical lines max: 40 Mean: 3.114

```

<i>Input</i>		<i>Output</i>
--------------	--	---------------

```

load(file="hrvRRVneighs16.RData")
local.recurrencePlotAux(hrvRRVneighs16, dim=16, radius=3/16)

```

Time used: 1.453 sec.

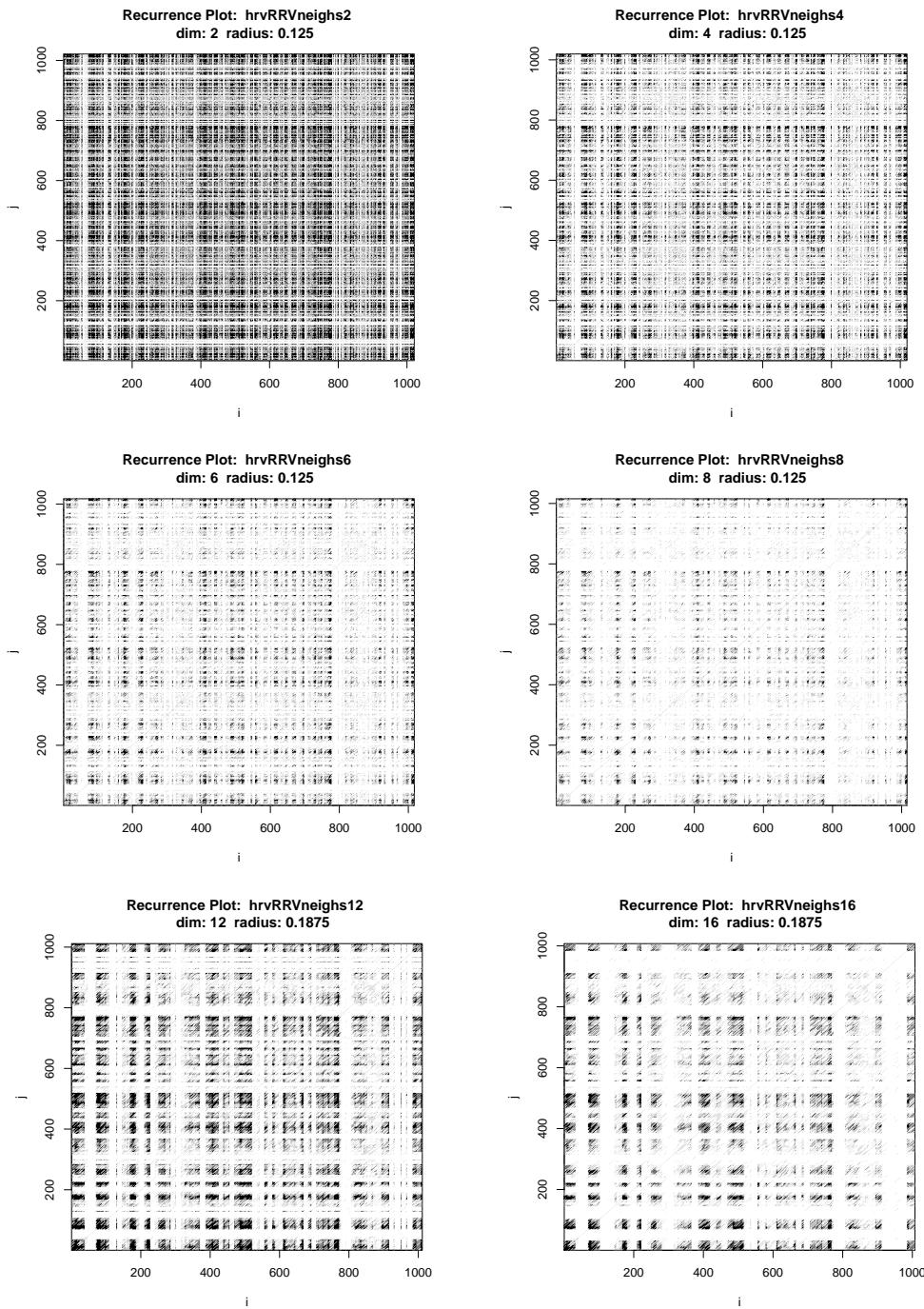


FIGURE 56. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 1.454 sec.

REFERENCES

- Eckmann JP, Kamphorst SO, Ruelle D (1987). “Recurrence plots of dynamical systems.” *Europhys. Lett.*, 4(9), 973–977.
- Takens F (1981). “Detecting strange attractors in turbulence.” In *Dynamical systems and turbulence, Warwick 1980*, pp. 366–381. Springer.
- Webber Jr CL, Zbilut JP (2005). “Recurrence quantification analysis of nonlinear dynamical systems.” *Tutorials in contemporary nonlinear methods for the behavioral sciences*, pp. 26–94.
- Zbilut JP, Webber CL (2006). “Recurrence quantification analysis.” *Wiley encyclopedia of biomedical engineering*. URL <http://onlinelibrary.wiley.com/doi/10.1002/9780471740360.ebs1355/full>.

R session info:

Total Sweave time used: 50.302 sec. at Thu Aug 7 22:31:22 2014.

- R version 3.1.1 (2014-07-10), x86_64-apple-darwin13.1.0
- Locale: en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8
- Base packages: base, datasets, graphics, grDevices, methods, stats, tcltk, utils
- Other packages: leaps 2.9, locfit 1.5-9.1, MASS 7.3-33, Matrix 1.1-4, mgcv 1.8-1, nlme 3.1-117, nonlinearTseries 0.2.1, rgl 0.93.1098, RHRV 4.0, sintro 0.1-3, tkrplot 0.0-23, TSA 1.01, tseries 0.10-32, waveslim 1.7.3
- Loaded via a namespace (and not attached): grid 3.1.1, lattice 0.20-29, quadprog 1.5-5, tools 3.1.1, zoo 1.7-11

L^AT_EX information:

```
textwidth: 5.37607in      linewidth:5.37607in
textheight: 9.21922in
```

Bibliography style: jss

CVS/Svn repository information:

```
$Source: /u/math/j40/cvsroot/lectures/src/dataanalysis/Rnw/recurrence.Rnw,v $
copied to r-forge
HeadURL: svn+ssh://gsawitzki@scm.r-forge.r-project.org/svnroot/rhrv/gs/recurrence.Rnw*
$Revision: 1.10 $
$Date: 2014/08/01 20:33:33 $
$Name:  $
$Author: j40 $
```

E-mail address: gs@statlab.uni-heidelberg.de

GÜNTHER SAWITZKI
 STATLAB HEIDELBERG
 IM NEUENHEIMER FELD 294
 D 69120 HEIDELBERG