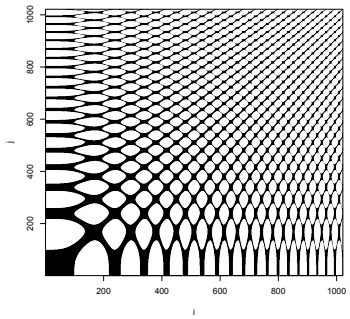
STATISTICAL DATA ANALYSIS: RECURRENCE PLOT

GÜNTHER SAWITZKI



Contents

Date: 2013-11 revised:28.05.2017.
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: June 1, 2017 from svn/cvs *Revision* : 250
gs@statlab.uni-heidelberg.de .

1. Background

1.1. **Takens' Recurrence States.** Recurrence plots have been introduced in an attempt to understand near periodic behaviour in hydrodynamics, in particular the transition to turbulence. 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. A classical field is the movement of celestial bodies.

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.

$$u_i = (x_i, \ldots, x_{i+(d-1)})$$

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 m dimensional data. So you take

$$u_i = (x_i, x_{i+m} \ldots, x_{i+(d-1)*m}).$$

ToDo: add support for higher dimensional signals

This may be a relict of FORTRAN times, where it was common to flatten two-dimensional structures by case.

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.

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.

Takens' states are vectors in M dimensions. There are standard statistical techniques to visualise aspects of M dimensional vectors, at least for not too high dimension. One is the draftsman's plot, a scatterplot matrix by marginals. For Takens states, this is

implemented as *statepairs()*. The other is coplots, a variant of scatterplot matrix by marginals, conditioned by one or two additional variables. For Takens states, this is implemented as *statecoplots()*

To display the Takens state space, we us a variant of pairs().

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

By convention, the states are defined using overlapping sliding windows. This imposes considerable dependence between the states: one state is the shifted previous states, with only the end sub-state replaced. As an option, the states can be subsampled, using only non-overlapping ranges.

The Takens states may be stationary, that is asymptotically the states starting at i do not depend on i. In this case, the first row (or column) contains all information, and pairs plot form an inclusion sequence by. In general, we will use state plots in 4 or 8 dimensions, where the limits are suggested by the print area.

The visual impression of recurrence plots are strongly affected by the coverage, controlled by the radius used to determine neighbourhoods. So far, this parameter is chosen visually, and comparison needs some care.

ToDo: consider dimension-adjusted radius

1.2. **Recurrence Plots.** 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>`.

1.3. **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 1.

TABLE 1. Recurrence Quantification Analysis (RQA)

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

2. R Setup

Input

```
save.RNGseed <- 87149 #.Random.seed
save.RNGkind <- RNGkind()
set.seed(save.RNGseed, save.RNGkind[1])
save.RNGkind
```

Output

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

Input

```
options(warn=1)
```

Input

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

Input

```
if (!require("sintro")) {
# install.packages("sintro",repos="http://r-forge.r-project.org",type="source")
library(sintro)
}
if (!require("nonlinearTseries")) {
install.packages("nonlinearTseries")
library(nonlinearTseries)
}
```

2.0.1. *Takens States.* Takens states are represented as a matrix, one state per row. The number of colums is the imbedding dimension. If present, the `time.lag` attribute is the lag parameter, `id` an identification string for the basic data set.

ToDo: improve choice of alpha

Input

```
alpha=0.5
```

`statepairs` *Show marginal scatterplots of Takens states*

Usage.

```
statepairs(states, main,
range  = NULL,
rank = FALSE, nooverlap = FALSE,
col,...)
```

Arguments.

states	A matrix: Takens states by dimension, one state per row.
main	Optional: the main header.
range	Optional: an interval selecting values to be displayd.
rank	An experimental variant. If *rank*, the values are rank transformed.
nooverlap	An experimental variant. If *nooverlap*, the cases are subsampled by dimension.
col	The current choice is $rgb(0,0,0,\alpha = \sqrt{\frac{1}{nr\ states}})$ as a default.

ToDo: colour by time

Input

```
statepairs <- function(states, main,
        rank=FALSE, nooverlap= FALSE, range=NULL,
        col = rgb(1,0,0, 1/sqrt(dim(states)[1])),  ...){
                n <- dim(states)[1]; dim <-  dim(states)[2]
        time.lag <- attr(states,"time.lag");
        if (is.null(time.lag)) time.lag <- 1

        if (missing(main)) {
                stateid <- attr(states, "id")
                if (is.null(stateid)) stateid <- deparse(substitute(states))
                main <- paste("Takens states:", stateid, "\n",
                        "n:", n," dim:", dim)
                if (time.lag != 1) main <- paste(main, " time lag:", time.lag)
                }

        if (nooverlap) {states <- states[ seq(1,n, by=dim),]
        main <- paste(main," no overlap")}

        if (!is.null(range)) { states[states[]< range[1]] <- NA;
                states[states[]> range[2]] <- NA
                main <- paste(main," trimmed")}

        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, ...)
        col=        col, pch=19, ...)
        #title(main=main, outer=TRUE, line=-2, cex.main=0.8)
}
```

Assuming the index is time, the *statecoplot()* is layed out to show the highest index as response, i.e. $(x_{t-1}, x_t | x_{t-2}, x_{t-3})$.

ToDo: extend for low dimensions, extend parameters

statecoplot *Show conditioning marginal scatterplots of Takens states*

Usage.

```
statecoplot(states, main,
range  = NULL,
rank = FALSE, nooverlap = FALSE,
 col= rgb(0, 0, 0, \alpha = \sqrt{\frac{1}{nr\ states}})$,
number = c(5,5))
```

Arguments.

`states`	A matrix: Takens states by dimension, one state per row.
`main`	Optional: the main header.
`range`	Optional: an interval selecting values to be displayd.
`rank`	An experimental variant. If `rank`, the values are rank transformed.
`nooverlap`	An experimental variant. If `nooverlap`, the cases are subsampled by dimension.
`alpha`	
`col`	The current choice is $rgb(0,0,0,\alpha = \sqrt{\frac{1}{nr\ states}})$, as a default.
`number`	integer; the number of conditioning intervals, for a and b, possibly of length 2.

Input

```
statecoplot <- function(states, main,
                rank = FALSE, nooverlap = FALSE, range = NULL,
                col = rgb(1,0,0, 1/sqrt(dim(states)[1])),
                number = c(5,5), ...){
        n <- dim(states)[1]; dim <-  dim(states)[2]
        time.lag <- attr(states,"time.lag")
        if (is.null(time.lag)) time.lag <- 1
        if (missing(main)) {
        stateid <- attr(states, "id")
        if (is.null(stateid)) stateid <- deparse(substitute(states))
        main <- paste("Takens states:", stateid, "\n",
                "n=", n," dim=", dim)
        if (time.lag != 1) main <- paste(main, " time lag=", time.lag)
                }

        if (nooverlap) {states <- states[ seq(1,n, by=dim),]
        main <- paste(main," no overlap")}


        if (!is.null(range)) {
        states[states[]< range[1]] <- NA;
        states[states[]> range[2]] <- NA
        main <- paste(main," trimmed")}

        if (rank) {states <- apply(states, 2, rank, ties.method="random")
        main <- paste(main," ranked")}

        coplot((states[,4]~states[,3]|states[,1]+ states[,2]),
        number=number, main=main,
        col=rgb(0,0,0, alpha), pch=19, ...)
        #title(main=main, outer=TRUE, line=-2, cex.main=0.8)
}
```

2.0.2. *Local Bottleneck: Recurrence Plots.* To allow experimental implementations, functions from `nonlinearTseries` are aliased here.

Input

```
local.buildTakens <- function (time.series,
        embedding.dim, time.lag=1,
        id=deparse(substitute(time.series)))
{
        takens <-
                nonlinearTseries:::buildTakens(time.series, embedding.dim, time.lag)
```

```
        attr(takens,"time.lag") <- time.lag
        attr(takens,"embedding.dim") <- embedding.dim
        attr(takens,"id") <- id
        return(takens)
}
```

Input

```
local.findAllNeighbours <- function (takens, radius, number.boxes = NULL)
{
        allneighs <-
                nonlinearTseries:::findAllNeighbours(takens, radius, number.boxes = NULL)
        mostattributes(allneighs) <- attributes(takens)
        attr(allneighs, "radius") <- radius
        return(allneighs)
}
```

minor cosmetics added to recurrencePlotAux

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

Input

```
#non-sparse variant
#local.recurrencePlotAux <- nonlinearTseries:::recurrencePlotAux
local.recurrencePlotAux = function(neighs, dim=NULL, lag=NULL, radius=NULL){

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

  ntakens=length(neighs)
  neighs.matrix <- matrix(nrow=ntakens, ncol=ntakens)
  #neighbourListNeighbourMatrix()
    #neighs.matrix = Diagonal(ntakens)
          for (i in 1:ntakens){
                  neighs.matrix[i,i] = 1 # do we want the diagonal fixed to 1
            if (length(neighs[[i]])>0){
              for (j in neighs[[i]]){
                neighs.matrix[i,j] = 1
              }
            }
          }

#! clean up. only one main id should be presentes
        main <- paste("Recurrence Plot: ",
                      deparse(substitute(neighs))
                      )
        id <-  attr(neighs, "id"); if (!is.null(id)) main <- paste(main," id:",id)

        more <- NULL

        more <- paste(more," n:",length(neighs))

        #use components of neights if available
```

```
        embedding.dim <- attr(neighs, "embedding.dim")
        if (is.null(embedding.dim)) embedding.dim <- dim
        if (!is.null(dim)) {
                if (embedding.dim != dim)
                        warning(paste("Embedding dim:", embedding.dim,
                " does not match dim argument=", dim))
                more <- paste(more," dim:",dim)
        }


        attrradius <- attr(neighs, "radius")
        if (!is.null(lag)) more <- paste(more," lag:",lag)
        if (is.null(radius)) radius <- attrradius
        if (!is.null(radius)) {
                more <- paste(more," radius:",radius)
                if (!is.null(attrradius) && (attrradius != radius))
                        warning(paste("Radius attribute:", attrradius,
                        " does not match radius argument=", radius))
                }
        # if (!is.null(attrradius)) more <- paste(more," radius attr:",attrradius)
        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 *nonlinearTseries*

2.0.3. *Recurrence Plots: RQA Information.* This is a hack to report RQA information. $dim = NULL$ is added to align calling with other functions.

Needs improvement.

ToDo: improve to a full *show* method for class *rqa*.

Input

```
showrqa <- function(takens, dim=NULL, radius,
                digits=3,
                do.hist = TRUE, rm.hist = TRUE,
                log = TRUE ,...)
{
        xxrqa <- rqa(takens=takens,radius=radius)
        xxrqa$radius <- radius
        id <- attr(takens,"id")
        if (is.null(id)) id <- deparse(substitute(takens))
        xxrqa$id<- id
        xxrqa$time.lag <- attr(takens,"time.lag")
        cat(id, " n:", dim(takens)[1], " Dim:", dim(takens)[2], "\n")
        xxrqa$n <- dim(takens)[1]
        xxrqa$dim <- dim(takens)[2]
        #str(xxrqa)
        #str(digits)
        #browser()
```

```
        cat(paste("Radius:", radius,
                " Recurrence coverage REC:", round(xxrqa$REC, digits),
                " log(REC)/log(R):",
                round(log(xxrqa$REC)/log(radius), digits), "\n"))
        xxrqa$logratio <- log(xxrqa$REC)/log(radius)
        cat(paste("Determinism:", round(xxrqa$DET, digits),
                " Laminarity:",round(xxrqa$LAM, digits), "\n"))
        cat(paste("DIV:", round(xxrqa$DIV, digits), "\n"))
        cat(paste("Trend:", round(xxrqa$TREND, digits),
                " Entropy:",round(xxrqa$ENTR, digits), "\n"))
        cat(paste("Diagonal lines max:", round(xxrqa$Lmax, digits),
                 " Mean:",round(xxrqa$Lmean, digits),
                " Mean off main:",round(xxrqa$LmeanWithoutMain, digits), "\n"))
        cat(paste("Vertical lines max:", round(xxrqa$Vmax, digits),
                " Mean:",round(xxrqa$Vmean, digits), "\n"))
                # str(xxrqa[4:12])

        if (do.hist){
        if (log==TRUE) log<-"y"
        oldpar <- par(mfrow=c(2,1))

        xxrqa$diagonalHistogram[xxrqa$diagonalHistogram==0] <- NA # hack for log zero counts
        dh<- xxrqa$diagonalHistogram
        #if (log=="y")  {dh <- dh+1}

        id <- attr(takens,"id"); if (is.null(id) ) id <- deparse(substitute(takens))
        pars <- paste("\n n=", dim(takens)[1], " Dim:",
                dim(takens)[2])
        lag<- attr(takens,"time.lag")
        if (!is.null(lag) && (lag !=1) ) pars <- paste(pars, " Lag:", lag)
        plot(dh, type="h", main=paste( id, " Diagonal:",
                pars," Radius: ",radius, " REC:", round(xxrqa$REC, digits)),
                xlab="length of diagonals",
                log = log, ...)

        xxrqa$recurrenceRate[xxrqa$recurrenceRate==0] <- NA # hack for log zero counts
        drR<- xxrqa$recurrenceRate
        #if (log=="y")  {drR <- drR+1}

        id <- attr(takens,"id"); if (is.null(id) ) id <- deparse(substitute(takens))
        pars <- paste("\n n=", dim(takens)[1], " Dim:",
                dim(takens)[2])
        lag<- attr(takens,"time.lag")
        if (!is.null(lag) && (lag !=1) ) pars <- paste(pars, " Lag:", lag)
        plot(drR, type="h",
                main=paste( id," Recurrence Rate",
                pars," Radius: ",radius, " REC:", round(xxrqa$REC, digits)),
                xlab="distance to diagonal",
                #ylim=c(1,3),
                log = log, ...)
        par(oldpar)
                }

        if (rm.hist) { xxrqa$recurrenceRate <- NULL; xxrqa$diagonalHistogram <- NULL }
        invisible(xxrqa)
}
```

3. Test Signals

We set up a small series of test signals. Some synthetic test signals are introduced here. More test cases used later are the Geyser data set (Section 7 on page 55) and two examples of heart rate data sets (Section 9 on page 81 and Section 10 on page 103) from *library(rhrv)*. The Geyser data set gives an example of a bi-variate point process. The HRV data sets give real live point orocess examples

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
#nsignal <- 4096
system.time.start <- proc.time()
```

For signal representation, we use a common layout.

Input

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

        if (missing(ylab)) { ylab <- deparse(substitute(signal)) }

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

        plot(signal, type = "l",
                main = "",  xlab = "index", ylab = ylab,
                col = rgb(0, 0, 0, 0.4))
        points(signal,
                col = rgb(0, 0, 1, 0.3), pch = 20)
        if (missing(main)) { main = deparse(substitute(signal)) }
        title(main = paste("signal and linear interpolation\n",main),
                outer = TRUE, line = -2, cex.main = 1.2)
}
```

3.1. **Sinus.**

Input

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

See Figure 1 on the following page.

3.2. **Uniform Random Numbers.**

Input

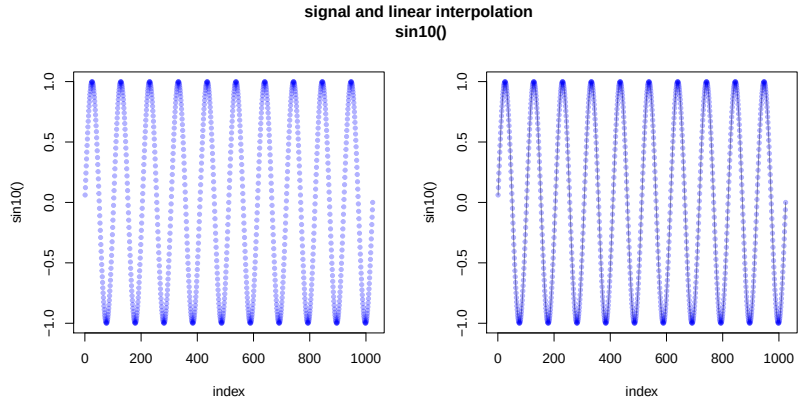


FIGURE 1. Test case: sin10. Signal and linear interpolation.

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

See Figure 2,

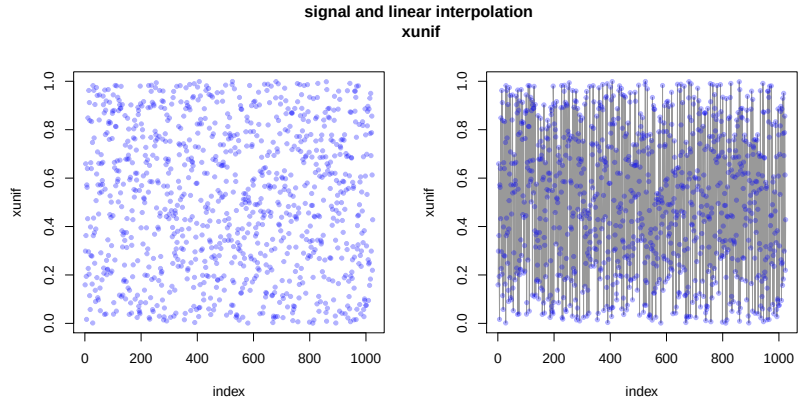


FIGURE 2. Test case: unif - uniform random numbers. Signal and linear interpolation.

3.3. **Chirp Signal.**

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" = {
```

```
        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 3,

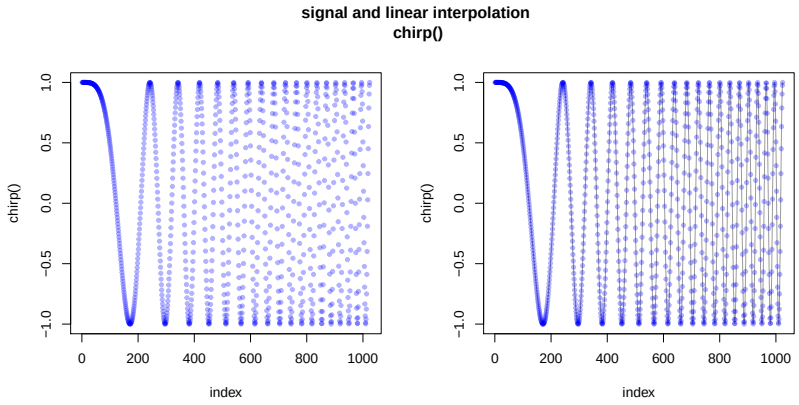


FIGURE 3. Test case: chirp signal. Signal and linear interpolation.

3.4. **Doppler signal.**

Input

```
doppler <- function(n=nsignal) {

dopplersignal  <- function(x) { sqrt(x*(1-x))* sin((2.1*pi)/(x+0.05))}
dopplersignal((1:nsignal)/nsignal)
}
plotsignal(doppler())
```

See Figure 4 on the next page,

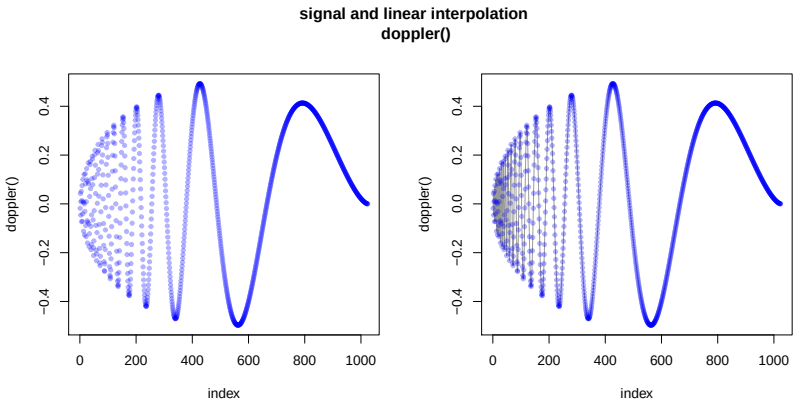


FIGURE 4. Test case: Doppler signal. Signal and linear interpolation.

4. nonlinearTseries Quick-Start

Input

```
#suppressMessages(library('nonlinearTseries'))
library('plot3D')
# by default, the simulation creates a RGL plot of the system's phase space
```

4.1. **Sinus.**

Input

```
sinN <- sin10()
oldpar <-  par(mfrow=c(1,2))
# taudelay estimation based on the autocorrelation function
tau.acf = timeLag(sinN, technique = "acf", do.plot = T,
main=paste("acf\n",deparse(substitute(x))))
cat("tau.acf:",tau.acf)
```

Output

```
tau.acf: 20
```

Input

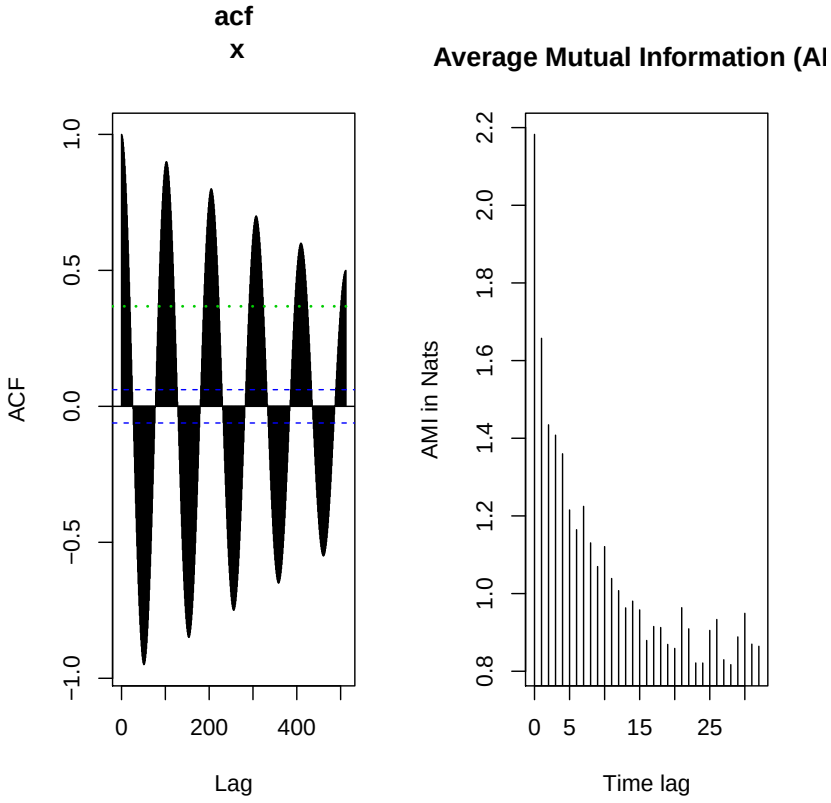
```
# taudelay estimation based on the mutual information function
tau.ami=NA
try(tau.ami <- timeLag(sinN, technique = "ami", do.plot = T,
main=paste("ami\n",deparse(substitute(x)))))
cat("tau.ami:",tau.ami)
```

Output

```
tau.ami: NA
```

Input

```
par(oldpar)
```



Note: fails. See Rnw source code.

4.2. **Uniform Random Numbers.**

Input

```
unifN <- unif()
oldpar <-  par(mfrow=c(1,2))
# taudelay estimation based on the autocorrelation function
tau.acf = timeLag(unifN, technique = "acf", do.plot = T,
main=paste("acf\n",deparse(substitute(x))))
cat("tau.acf:",tau.acf)
```

Output

```
tau.acf: 1
```

Input

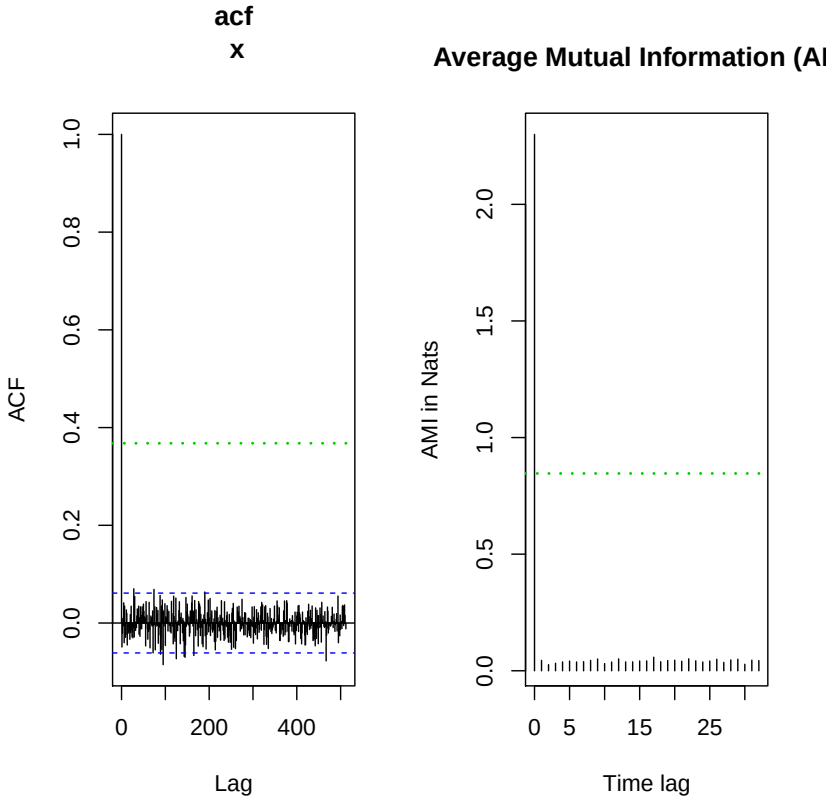
```
# taudelay estimation based on the mutual information function
tau.ami = timeLag(unifN, technique = "ami", do.plot = T,
main=paste("ami\n",deparse(substitute(x))))
cat("tau.ami:",tau.ami)
```

Output

```
tau.ami: 1
```

Input

```
par(oldpar)
```



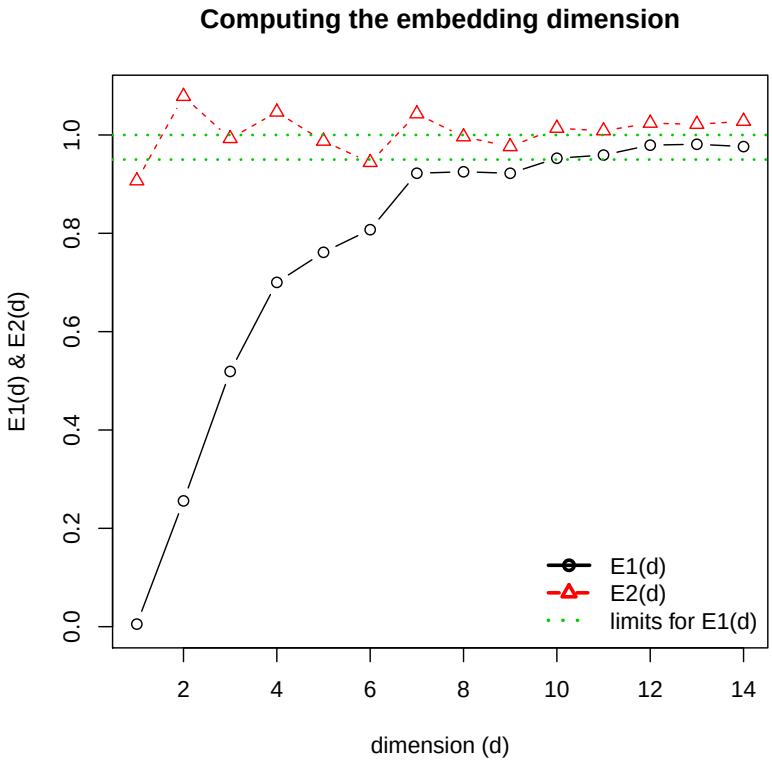
Input

```
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(unifN, time.lag = tau.ami,
max.embedding.dim = 15)}  else  {
emb.dim = estimateEmbeddingDim(unifN, time.lag = tau.acf,
max.embedding.dim = 15)}
cat("Estimated embedding dim:", emb.dim)
```

Output

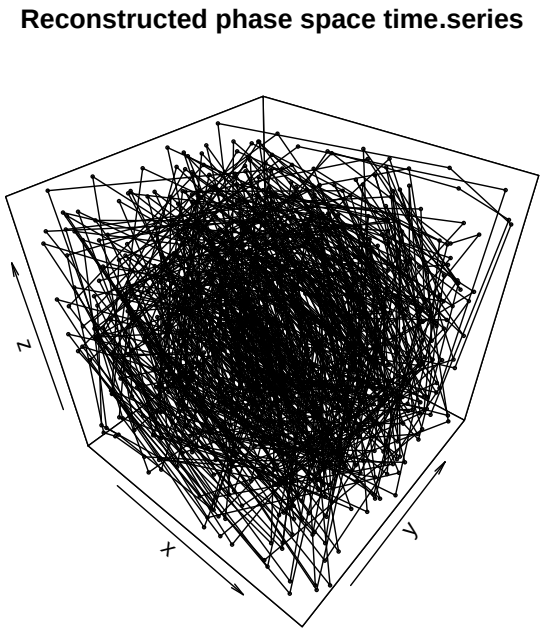
```
Estimated embedding dim: 10
```

Input



Input

```
tak = buildTakens(unifN,embedding.dim = emb.dim, time.lag = tau.ami)
scatter3D(tak[,1], tak[,2], tak[,3],
main = paste("Reconstructed phase space",deparse(substitute(time.series))),
col = 1, type="o",cex = 0.3)
```



4.3. **Chirp.**

Input

```
chirpN <- chirp()
oldpar <-  par(mfrow=c(1,2))
# taudelay estimation based on the autocorrelation function
tau.acf = timeLag(chirpN, technique = "acf", do.plot = T,
main=paste("acf\n",deparse(substitute(x))))
cat("tau.acf:",tau.acf)
```

Output

```
tau.acf: 11
```

Input

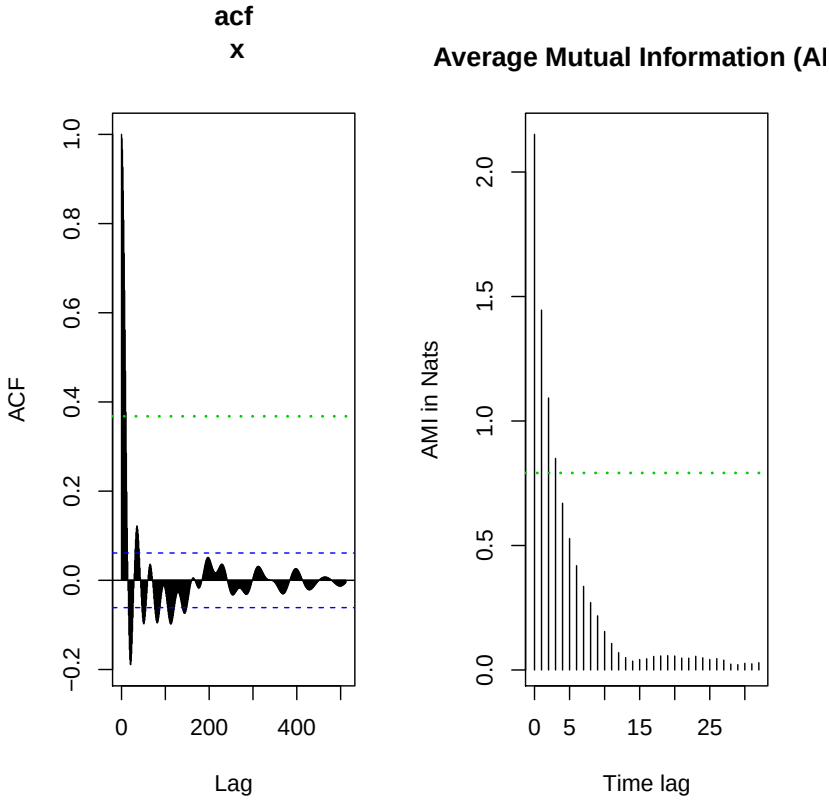
```
# taudelay estimation based on the mutual information function
tau.ami = timeLag(chirpN, technique = "ami", do.plot = T,
main=paste("ami\n",deparse(substitute(x))))
cat("tau.ami:",tau.ami)
```

Output

```
tau.ami: 4
```

Input

```
par(oldpar)
```



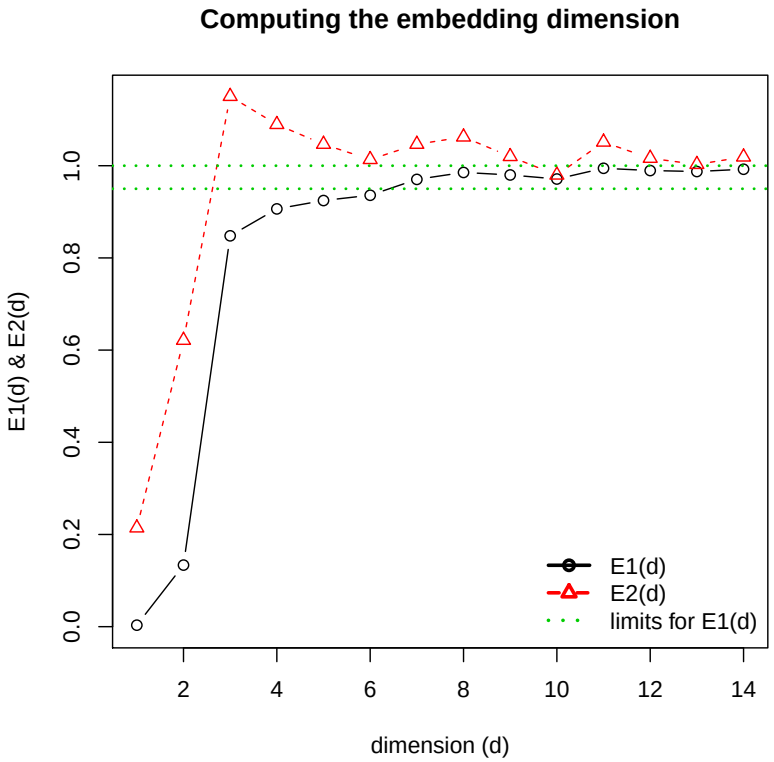
Input

```
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(chirpN, time.lag = tau.ami,
max.embedding.dim = 15)}  else  {
emb.dim = estimateEmbeddingDim(chirpN, time.lag = tau.acf,
max.embedding.dim = 15)}
cat("Estimated embedding dim:", emb.dim)
```

Output

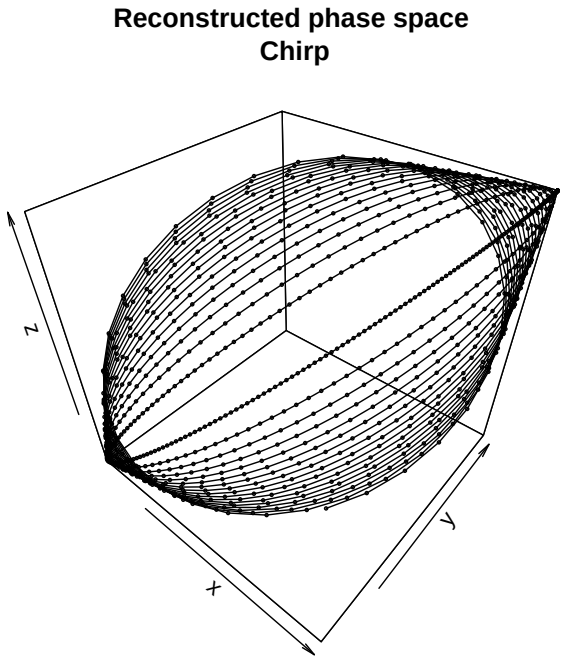
```
Estimated embedding dim: 7
```

Input



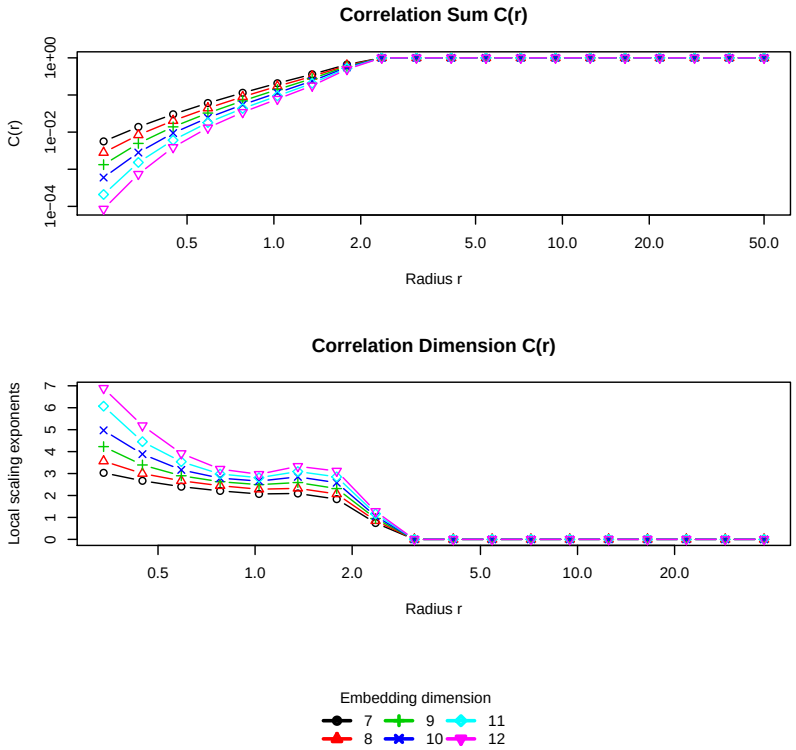
Input

```
tak = buildTakens(chirpN,embedding.dim = emb.dim, time.lag = tau.ami)
scatter3D(tak[,1], tak[,2], tak[,3],
main = paste("Reconstructed phase space\n","Chirp"),
col = 1, type="o",cex = 0.3)
```



Input

```
cd = corrDim(chirpN,
min.embedding.dim = emb.dim,
max.embedding.dim = emb.dim + 5,
time.lag = tau.ami,
min.radius = 0.001, max.radius = 50,
n.points.radius = 40,
do.plot=FALSE)
plot(cd)
```



Input

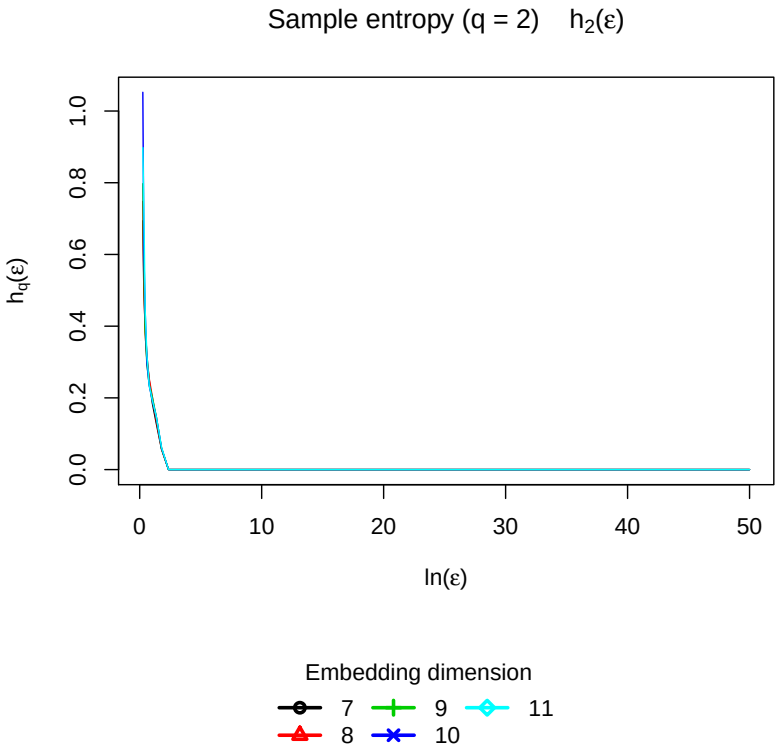
```
oldpar <- par(mfrow=c(1,2))
se = sampleEntropy(cd, do.plot =T)
se.est = estimate(se, do.plot = T,
regression.range = c(8,15))
cat("Sample entropy estimate: ", mean(se.est), "\n")
```

Output

```
Sample entropy estimate:  0
```

Input

```
par(oldpar)
```



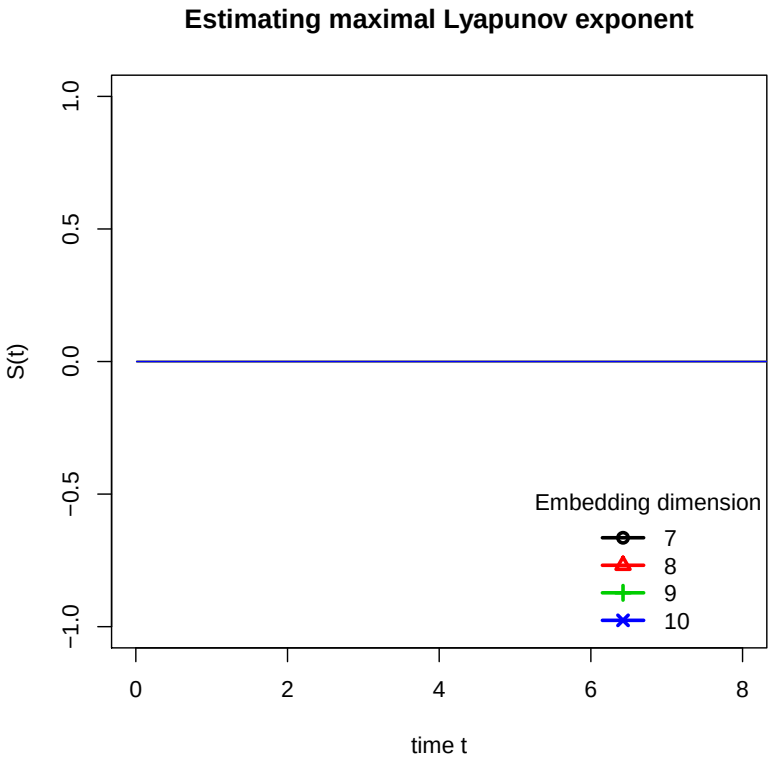
Input

```
#sampling.period = diff(lor$time)[1]
ml = maxLyapunov(chirpN,
sampling.period=0.01,
min.embedding.dim = emb.dim,
max.embedding.dim = emb.dim + 3,
time.lag = tau.ami,
radius=1,
max.time.steps=1000,
do.plot=FALSE)
plot(ml,type="l", xlim = c(0,8))
ml.est = estimate(ml, regression.range = c(0,3),
do.plot = T,type="l")
#cat("expected: 0.906 estimate:", ml.est,"\n")
cat("estimate:", ml.est,"\n")
```

Output

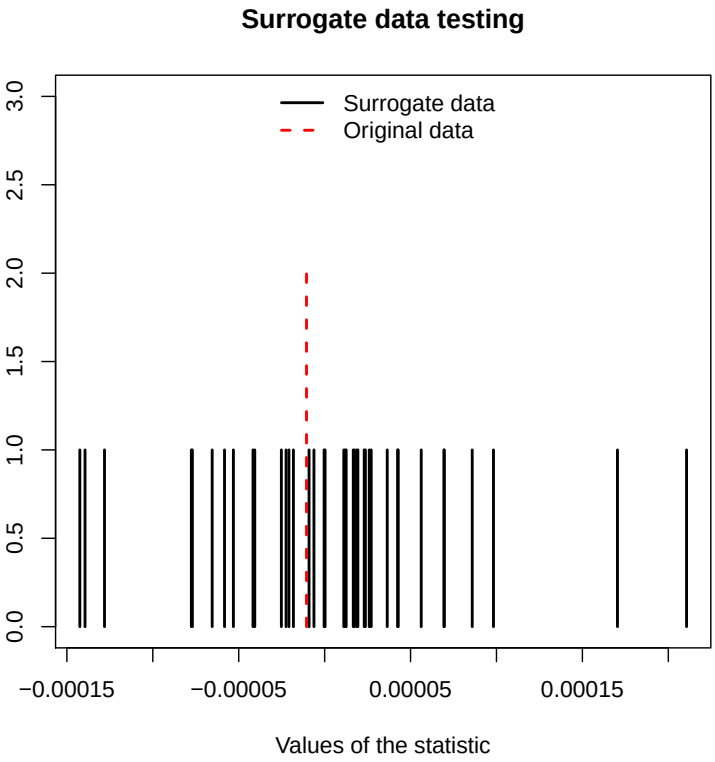
```
estimate: 0
```

Input



Input

```
st = surrogateTest(chirpN,significance = 0.05,one.sided = F,
FUN = timeAsymmetry, do.plot=F)
## Computing statistics
##
## Null Hypothesis: Data comes from a linear stochastic process
## Reject Null hypothesis:
## Original data's stat is significant larger than surrogates' stats
plot(st)
```


Surrogate data testing
Surrogate data
Original data
0.0
0.5
1.0
1.5
2.0
2.5
3.0
−0.00015
−0.00005
0.00005
0.00015
Values of the statistic

4.4. **Doppler.**

Input

```
dopplerN <- doppler()
oldpar <-  par(mfrow=c(1,2))
# taudelay estimation based on the autocorrelation function
tau.acf = timeLag(dopplerN, technique = "acf", do.plot = T,
main=paste("acf\n",deparse(substitute(x))))
cat("tau.acf:",tau.acf)
```

Output

```
tau.acf: 45
```

Input

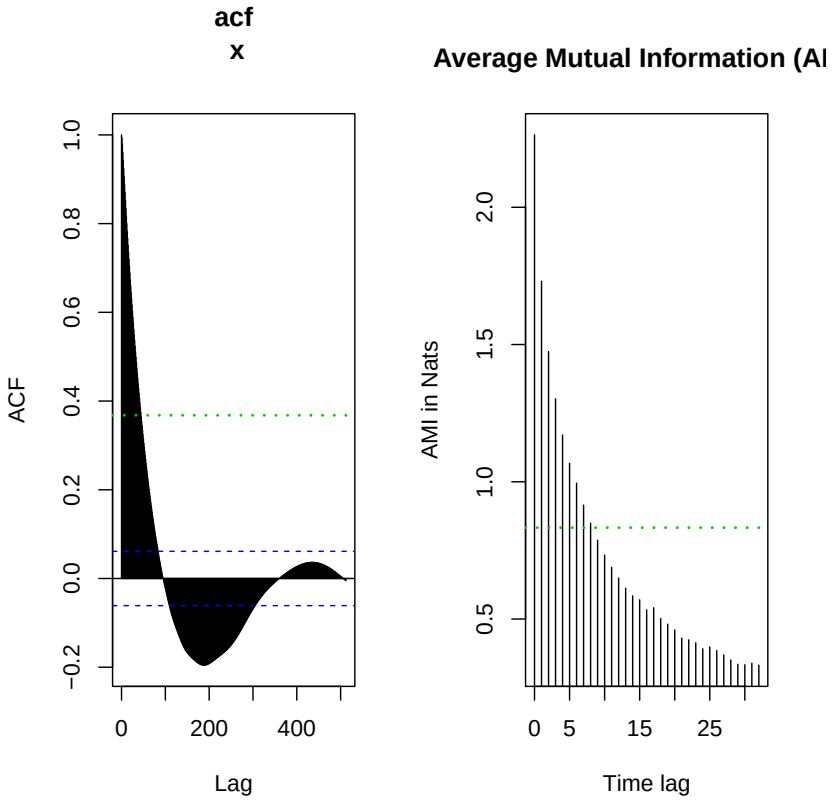
```
# taudelay estimation based on the mutual information function
tau.ami = timeLag(dopplerN, technique = "ami", do.plot = T,
main=paste("ami\n",deparse(substitute(x))))
cat("tau.ami:",tau.ami)
```

Output

```
tau.ami: 9
```

Input

```
par(oldpar)
```



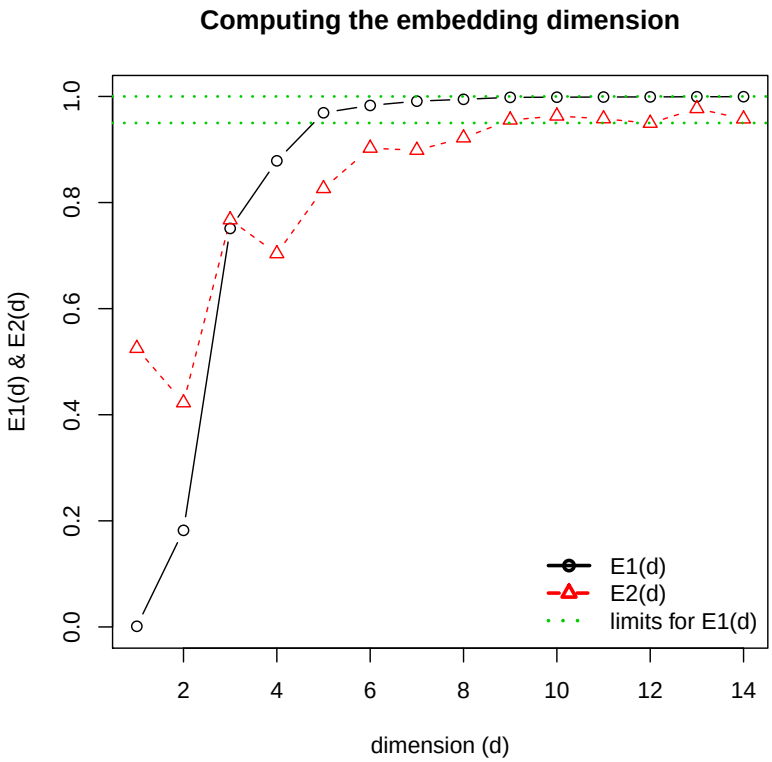
Input

```
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(dopplerN, time.lag = tau.ami,
max.embedding.dim = 15)}  else  {
emb.dim = estimateEmbeddingDim(dopplerN, time.lag = tau.acf,
max.embedding.dim = 15)}
cat("Estimated embedding dim:", emb.dim)
```

Output

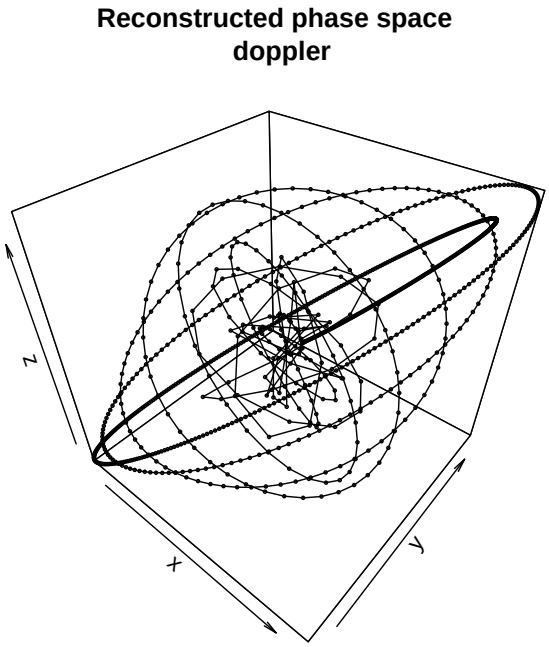
```
Estimated embedding dim: 5
```

Input



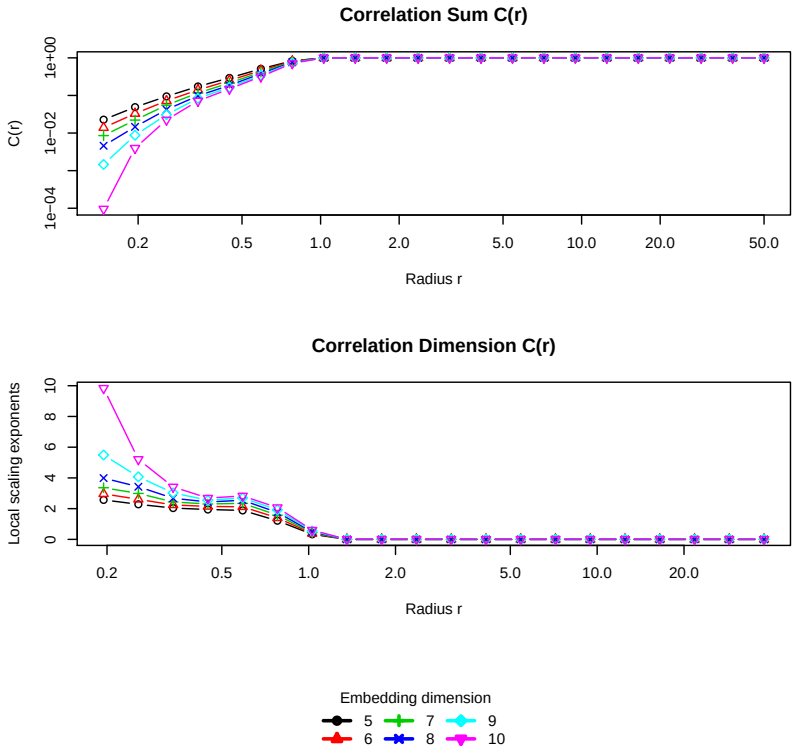
Input

```
tak = buildTakens(dopplerN,embedding.dim = emb.dim, time.lag = tau.ami)
scatter3D(tak[,1], tak[,2], tak[,3],
main = "Reconstructed phase space \n doppler" ,
col = 1, type="o",cex = 0.3)
```



Input

```
cd = corrDim(dopplerN,
min.embedding.dim = emb.dim,
max.embedding.dim = emb.dim + 5,
time.lag = tau.ami,
min.radius = 0.001, max.radius = 50,
n.points.radius = 40,
do.plot=FALSE)
plot(cd)
```



Input

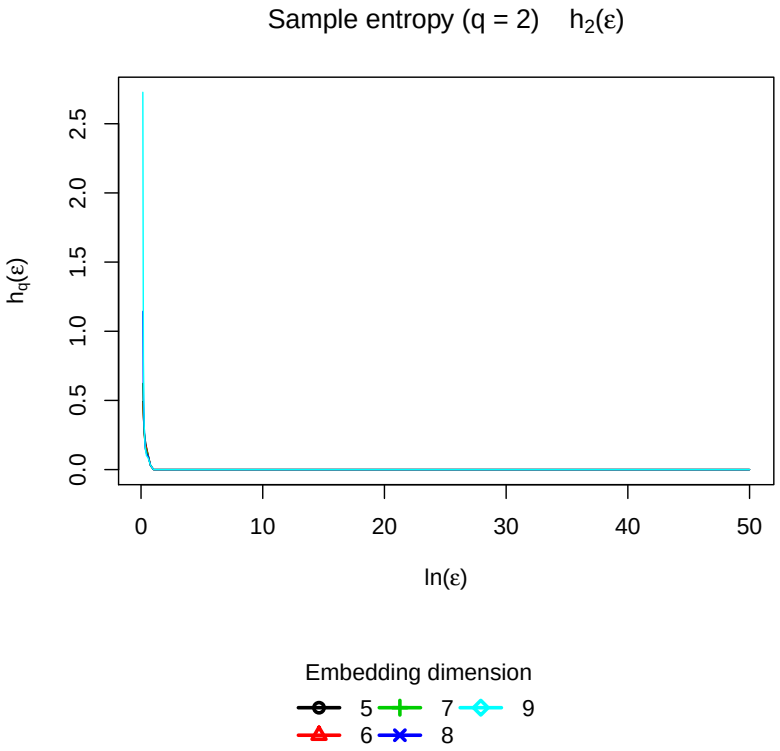
```
oldpar <- par(mfrow=c(1,2))
se = sampleEntropy(cd, do.plot = T)
se.est = estimate(se, do.plot = T,
regression.range = c(8,15))
cat("Sample entropy estimate: ", mean(se.est), "\n")
```

Output

```
Sample entropy estimate:  0
```

Input

```
par(oldpar)
```



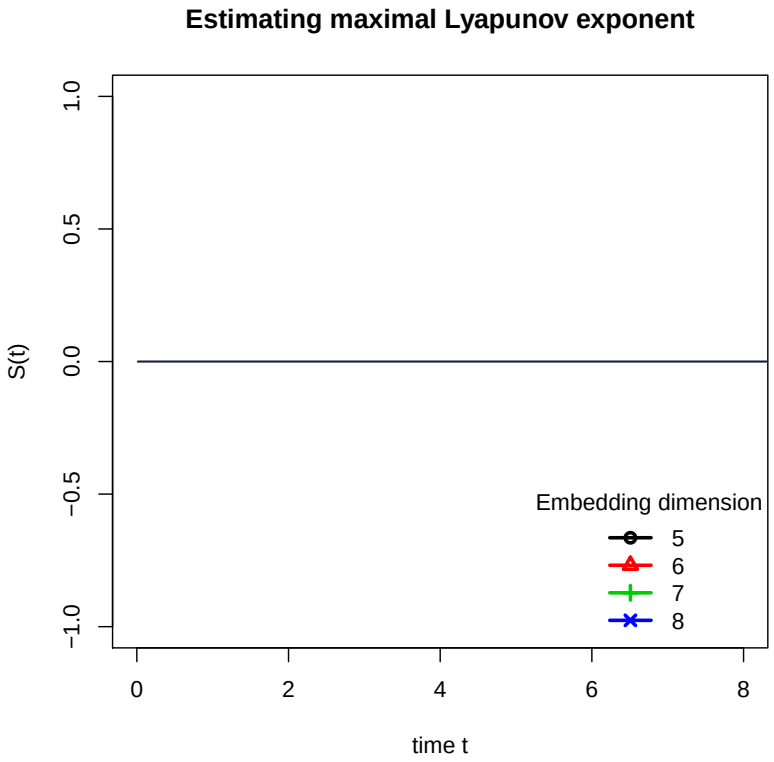
Input

```
#sampling.period = diff(lor$time)[1]
ml = maxLyapunov(dopplerN,
sampling.period=0.01,
min.embedding.dim = emb.dim,
max.embedding.dim = emb.dim + 3,
time.lag = tau.ami,
radius=1,
max.time.steps=1000,
do.plot=FALSE)
plot(ml,type="l", xlim = c(0,8))
ml.est = estimate(ml, regression.range = c(0,3),
do.plot = T,type="l")
#cat("expected: 0.906 estimate:", ml.est,"\n")
cat("estimate:", ml.est,"\n")
```

Output

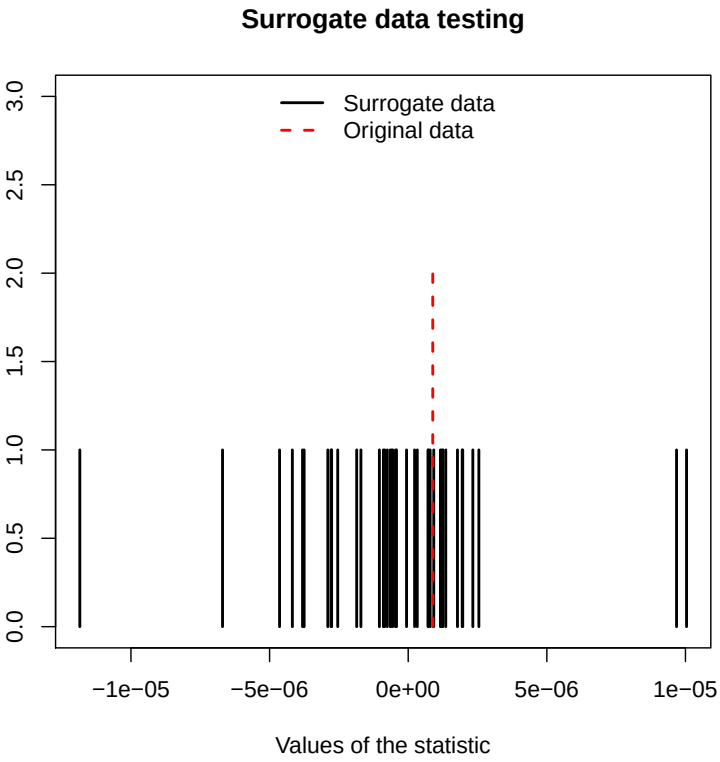
```
estimate: 0
```

Input



Input

```
st = surrogateTest(dopplerN,significance = 0.05,one.sided = F,
FUN = timeAsymmetry, do.plot=F)
## Computing statistics
##
## Null Hypothesis: Data comes from a linear stochastic process
## Reject Null hypothesis:
## Original data's stat is significant larger than surrogates' stats
plot(st)
```


Surrogate data testing
Surrogate data
Original data
0.0
0.5
1.0
1.5
2.0
2.5
3.0
−1e−05
−5e−06
0e+00
5e−06
1e−05
Values of the statistic

5. Takens' States for Test Signals

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

See Figure 5.

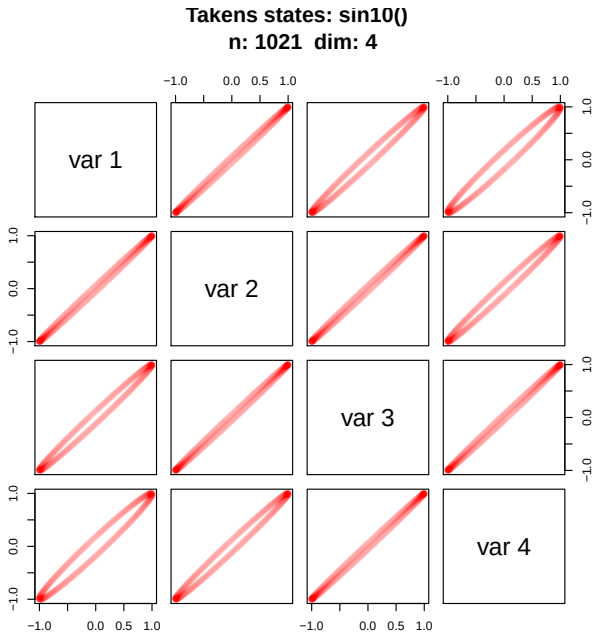


Figure 5. Takens states. Test case: sinus. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 1.943 sec.

The states only catch the local behaviour, where "local" depends on the sampling rate and the variation of the signal. For the sinus signal, we get a better picture if we subsample the signal.

Input
```
sintakenslag16 <- local.buildTakens(time.series=sin10(),
        embedding.dim=4, time.lag=16)
statepairs(sintakenslag16) #4
```

See Figure 6 on the next page.

Input
```
statepairs(sintakens, nooverlap=TRUE) #dim=4
```

See Figure 7 on the facing page.

Input
```
statecoplot(sintakens) #4
```

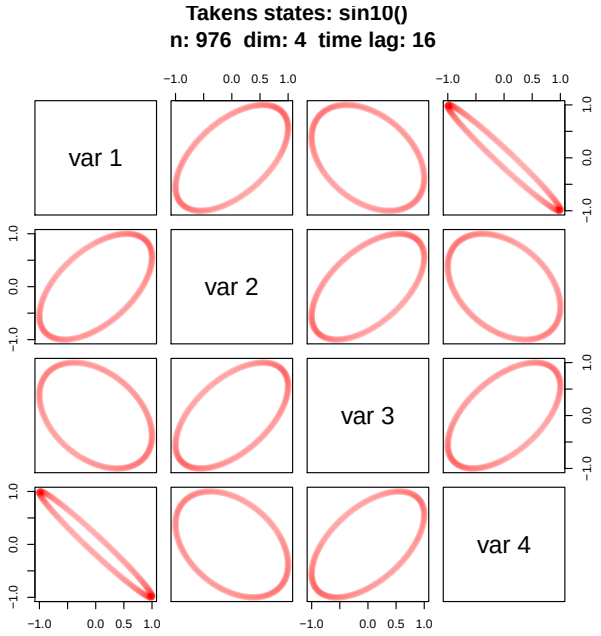


FIGURE 6. Takens states. Test case: sinus at time lag 16. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 1.238 sec.

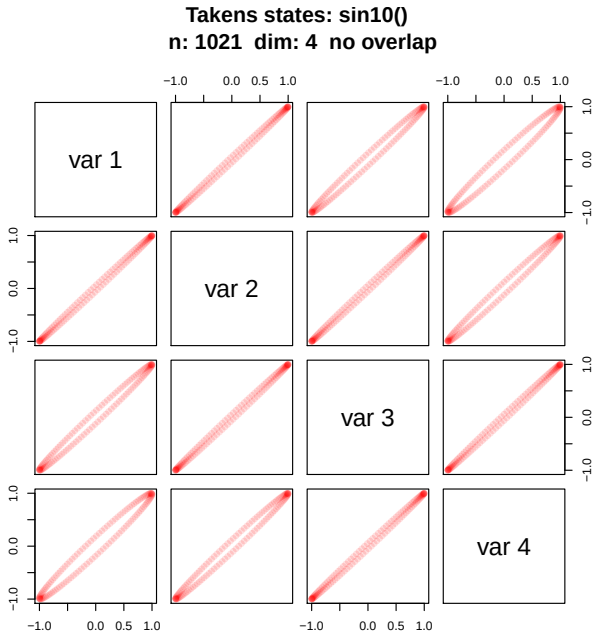


FIGURE 7. Takens states. Test case: sinus, no overlap. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 1.419 sec.

See Figure 8.

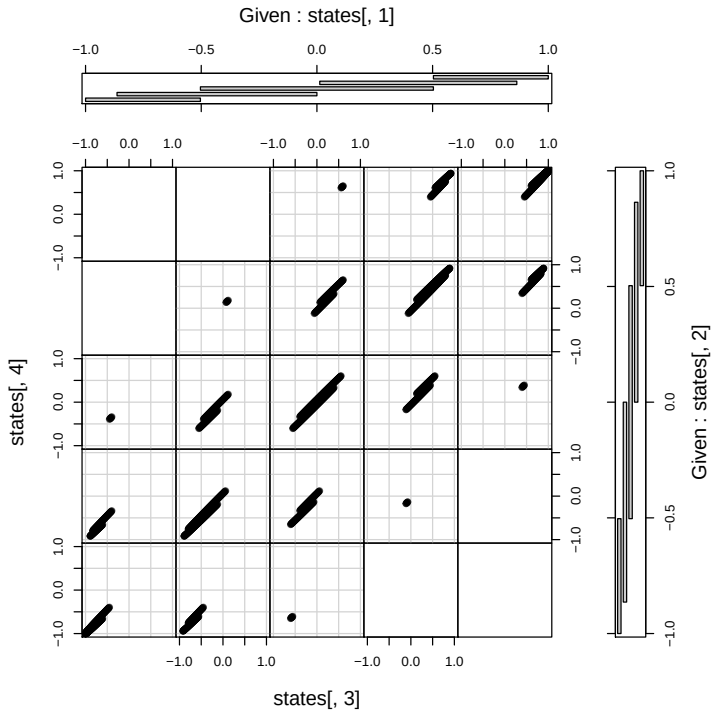


FIGURE 8. State coplot. Test case: sinus. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.264 sec.

Input

```
statecoplot(sintakenslag16) #4
```

See Figure 9 on the facing page.

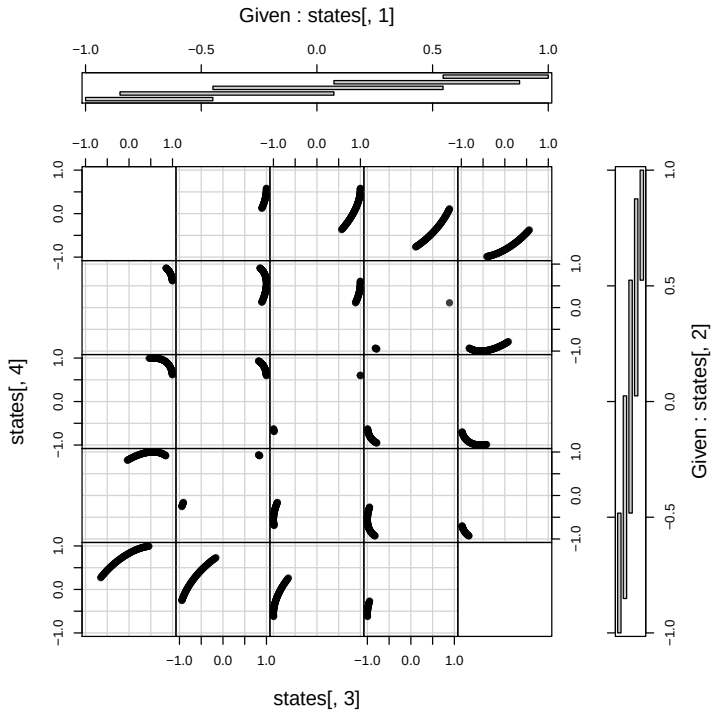


FIGURE 9. State coplot. Test case: sinus, time lag 16. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.529 sec.

Input

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

See Figure 10.

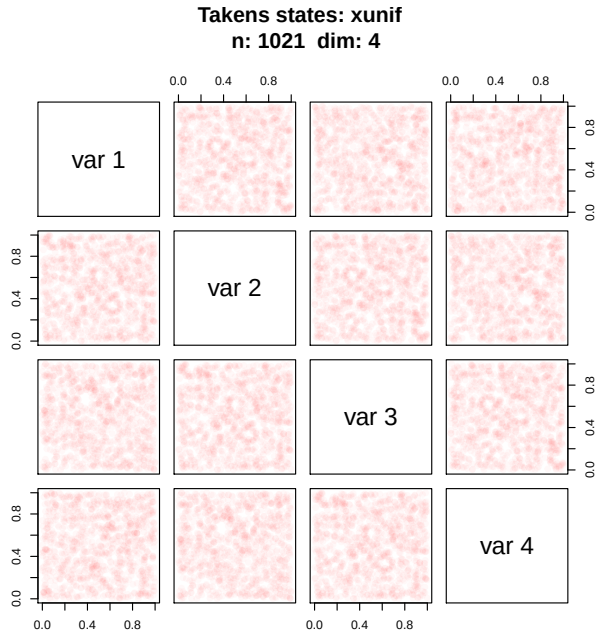


FIGURE 10. Takens states. Test case: uniform random numbers. Time used: 0.979 sec.

Input

```
statecoplot(uniftakens) #dim=4
```

See Figure 11 on the facing page.

Input

```
statepairs(uniftakens, nooverlap=TRUE) #dim=4
```

See Figure 12 on the next page.

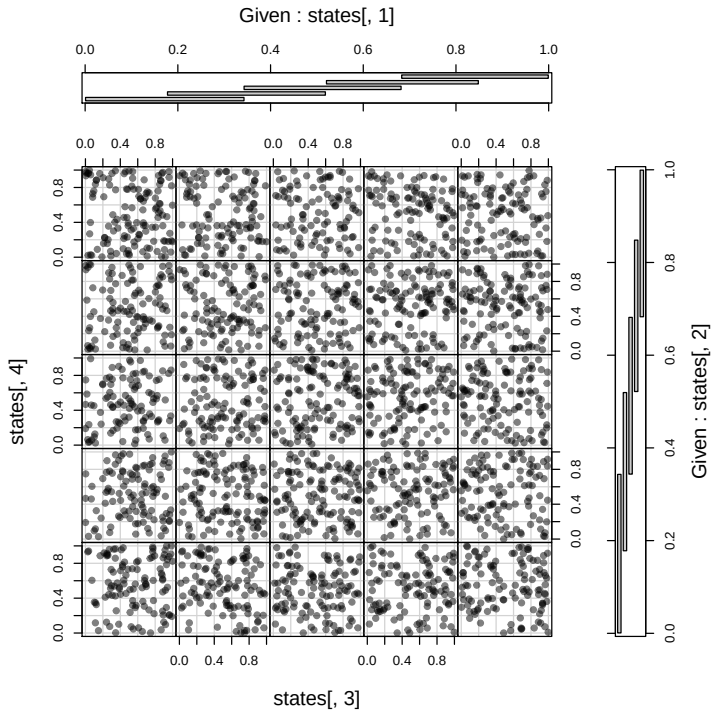


FIGURE 11. State coplot. Test case: uniform random numbers. Time used: 1.212 sec.

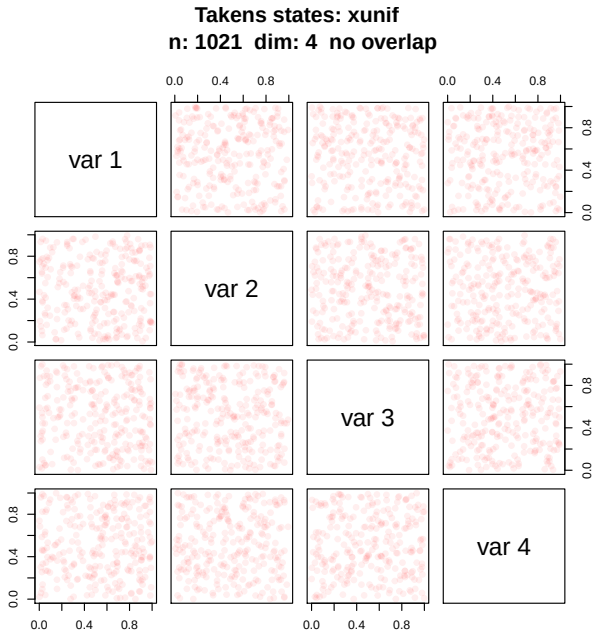


FIGURE 12. Takens states. Test case: uniform random numbers. Time used: 1.43 sec.

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

See Figure 13.

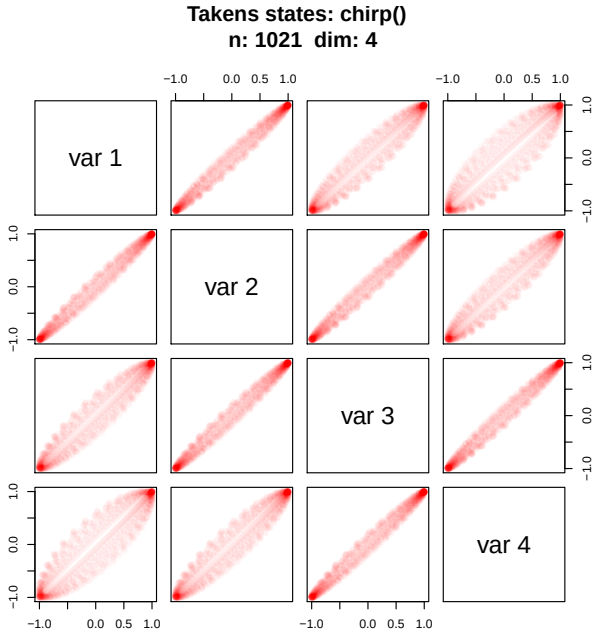


FIGURE 13. Takens states. Test case: chirp signal. Time used: 1.896 sec.

Input
```
statecoplot(chirptakens) #dim=4
```

See Figure 14 on the next page.

Input
```
statepairs(chirptakens, nooverlap=TRUE) #dim=4
```

See Figure 15 on the facing page.

Input
```
dopplertakens <- local.buildTakens(time.series=doppler(),
        embedding.dim=4, time.lag=1)
statepairs(dopplertakens) #4
```

See Figure 16 on page 42.

The states only catch the local behaviour, where “local” depends on the sampling rate and the variation of the signal. For the doppler signal, we get a better picture if we subsample the signal.

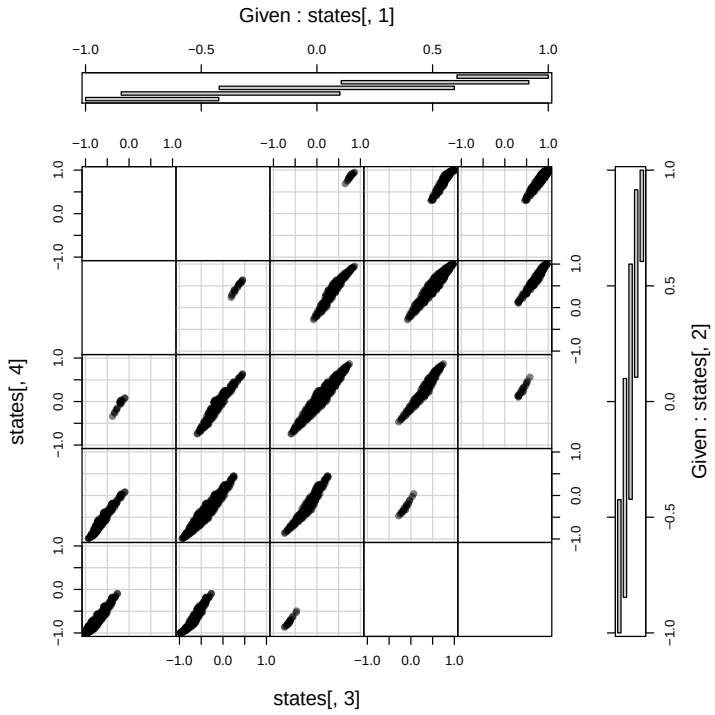


FIGURE 14. State coplot. Test case: chirp signal. Time used: 2.144 sec.

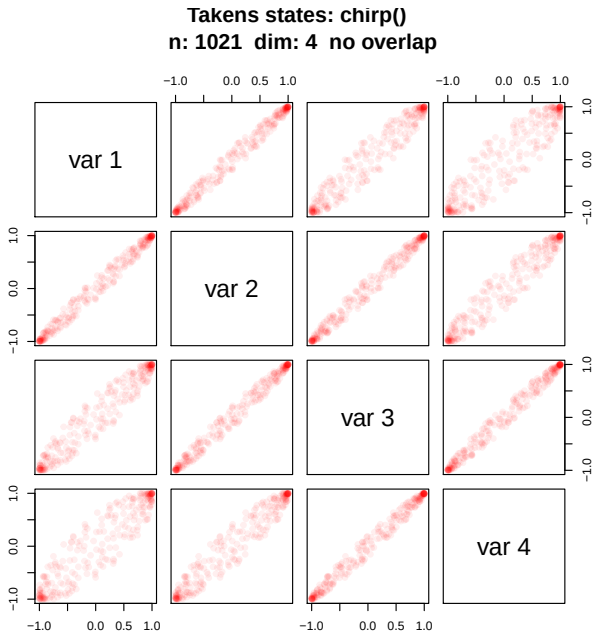


FIGURE 15. Takens states. Test case: chirp signal. Time used: 2.983 sec-

Input

```
dopplertakenslag16 <- local.buildTakens(time.series=doppler(),
        embedding.dim=4, time.lag=16)
statepairs(dopplertakenslag16) #4
```

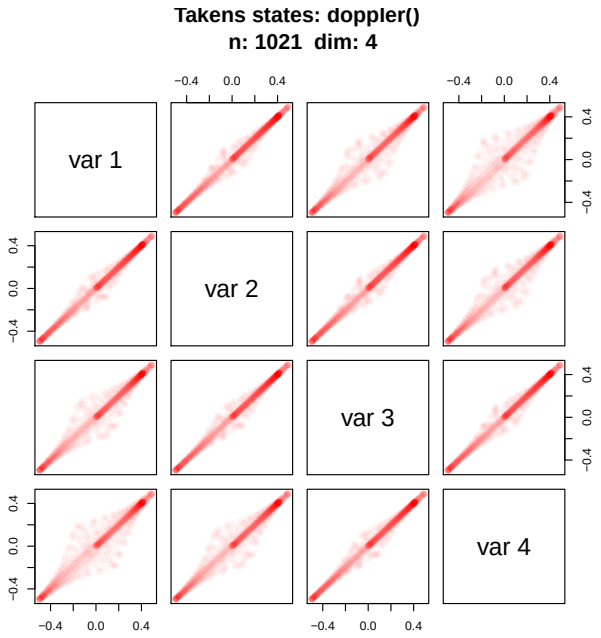


FIGURE 16. Takens states. Test case: Doppler. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 1.266 sec.

See Figure 17.

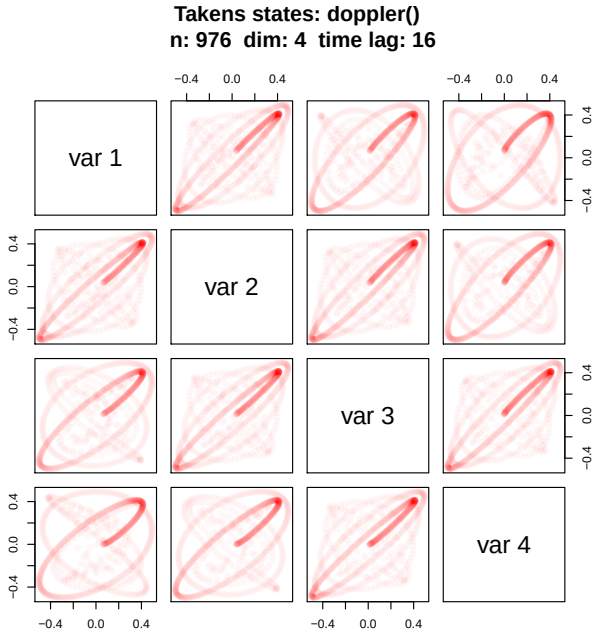


FIGURE 17. Takens states. Test case: doppler at time lag 16. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 1.229 sec.

Input

```
statepairs(dopplertakens, nooverlap=TRUE) #dim=4
```

See Figure 18.

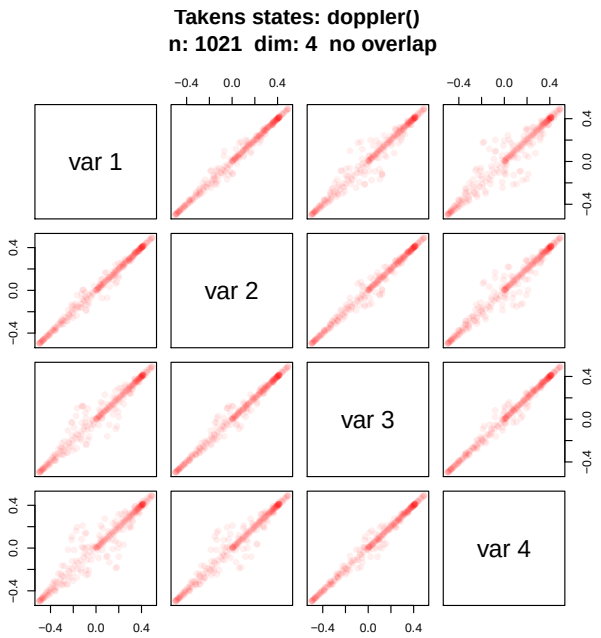


FIGURE 18. Takens states. Test case: doppler, no overlap. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 1.446 sec.

Input

```
statecoplot(dopplertakens) #4
```

See Figure 19 on the following page.

Input

```
statecoplot(dopplertakenslag16) #4
```

See Figure 20 on the next page.

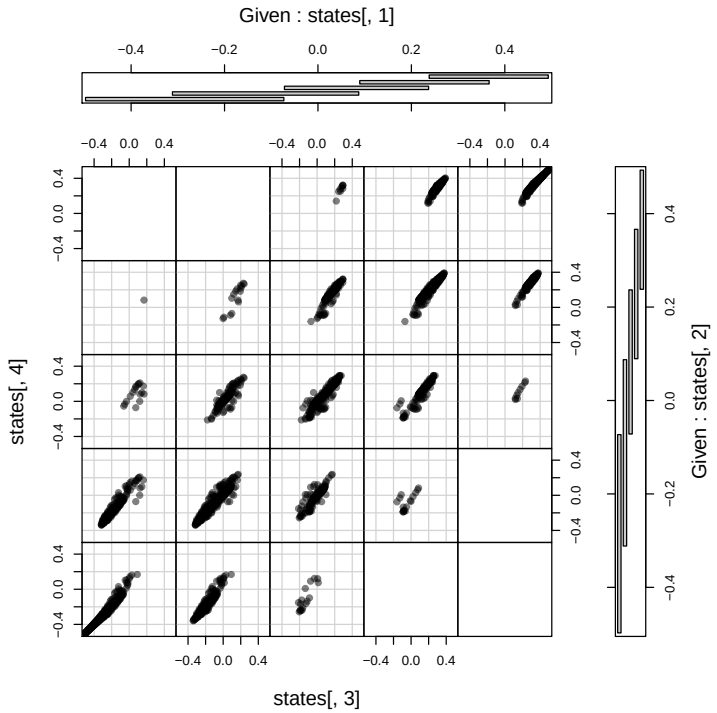


FIGURE 19. State coplot. Test case: Doppler. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.242 sec.

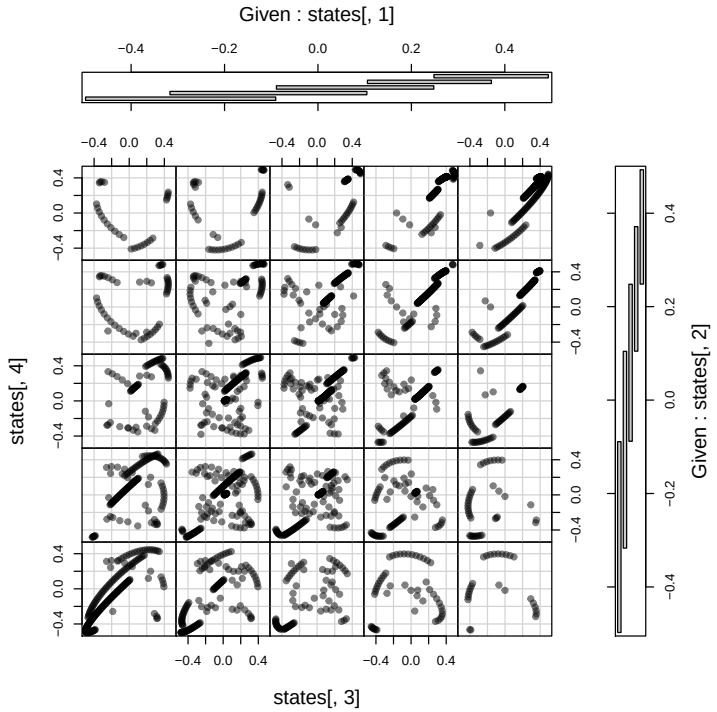


FIGURE 20. State coplot. Test case: Doppler, time lag 16. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.217 sec.

6. Recurrence Plots for Test Signals

6.0.1. *Sinus Recurrence Plots.*

Input

```
sin10neighs <- local.findAllNeighbours(sintakens, radius=0.8)
save(sin10neighs, file="sin10neighs.Rdata")
# load(file="sin10neighs.RData")
local.recurrencePlotAux(sin10neighs, dim=dim(sintakens)[2], radius=0.8)
```

See Figure 21.

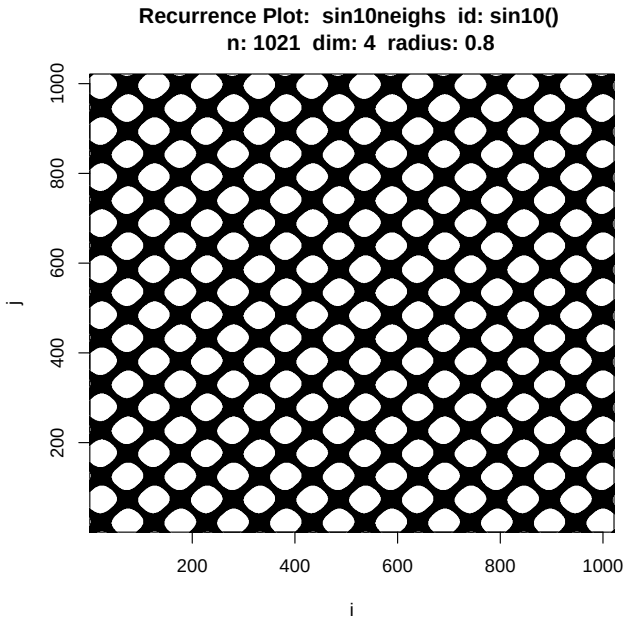


Figure 21. Recurrence plot. Test case: sinus. Time used: 2.344 sec.

Input

```
sin10rqa <- showrqa(sintakens, radius=0.8, log=TRUE)
```

Output

```
sin10()  n: 1021  Dim: 4
Radius: 0.8  Recurrence coverage REC: 0.496  log(REC)/log(R): 3.143
Determinism: 1  Laminarity: 1
DIV: 0.001
Trend: 0  Entropy: 3.213
Diagonal lines max: 1020  Mean: 32.712  Mean off main: 32.65
Vertical lines max: 66  Mean: 41.479
```

RQA Test case: sinus. Radius=0.8.Time used: 4.404 sec.. For graphical ouput, see Figure 22 on the next page.

Input

```
sin10lag16neighs <- local.findAllNeighbours(sintakenslag16, radius=0.2)
save(sin10lag16neighs, file="sin10lag16neighs.Rdata")
```

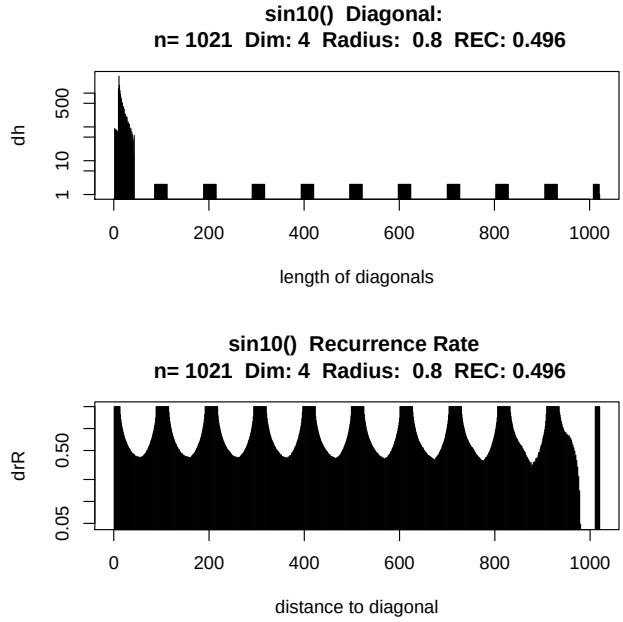


FIGURE 22. RQA. Test case: sinus. Radius=0.8.Time used: 4.404 sec., radius=0.64

```
# load(file="sin10lag16neighs.RData")
local.recurrencePlotAux(sin10lag16neighs, dim=4, radius=0.2)
```

See Figure 23.

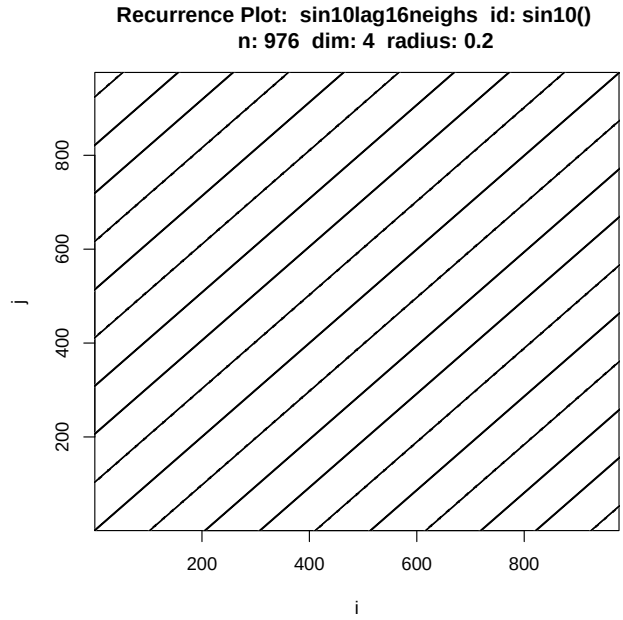


FIGURE 23. Recurrence plot. Test case: sinus curve, time lag 16. Time used: 4.933 sec.

Input

```
sin10lag16rqa <- showrqa(sintakenslag16, radius=0.2)
```

Output

```
sin10()  n: 976  Dim: 4
Radius: 0.2  Recurrence coverage REC: 0.067  log(REC)/log(R): 1.683
Determinism: 0.999  Laminarity: 1
DIV: 0.001
Trend: 0  Entropy: 2.316
Diagonal lines max: 975  Mean: 124.503  Mean off main: 122.827
Vertical lines max: 8  Mean: 6.774
```

RQA Test case: sinus curve, time lag 16. Time used: 5.744 sec. . For graphical ouput, see Figure 24.

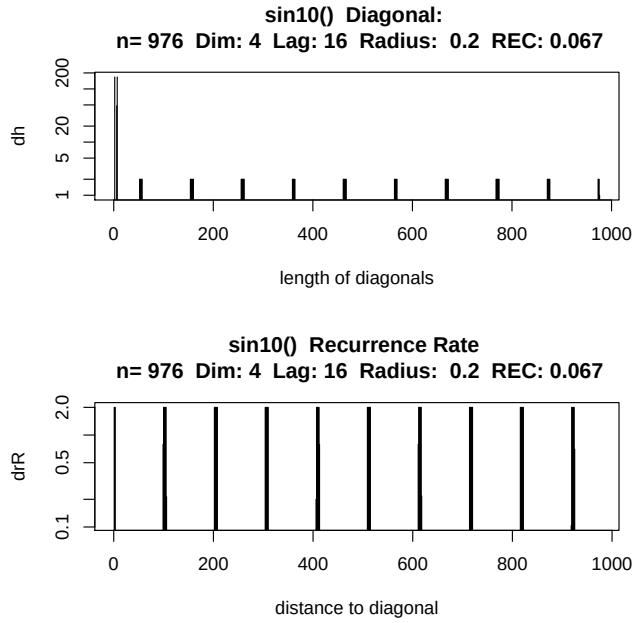


FIGURE 24. RQA. Test case: sinus curve, time lag 16. Time used: 5.744 sec. , radius=0.64

6.0.2. *Uniform Random Numbers Recurrence Plots.*

Input

```
unifneighs <- local.findAllNeighbours(uniftakens, radius=0.6)#0.4
save(unifneighs, file="unifneighs.RData")
# load(file="unifneighs.RData")
local.recurrencePlotAux(unifneighs, radius=0.6)
```

See Figure 25.

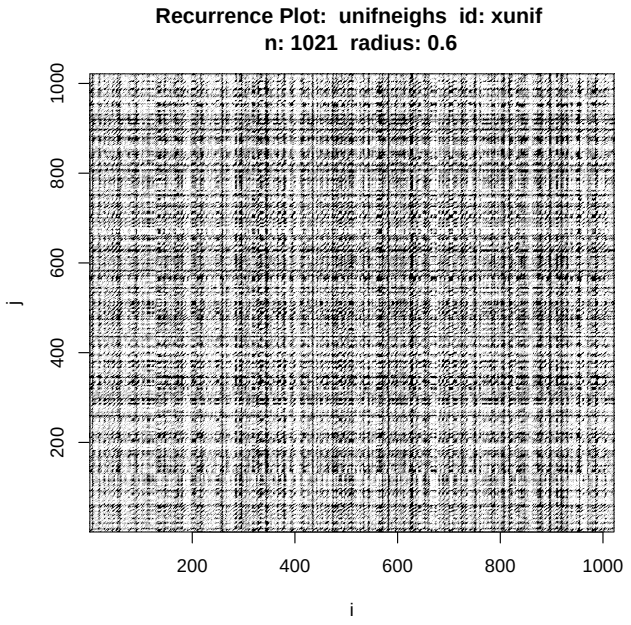


FIGURE 25. Recurrence plot. Test case: uniform random numbers. Time used: 2.804 sec.

Input

```
showrqa(uniftakens, radius=0.6, log=TRUE)
```

Output

```
xunif  n: 1021  Dim: 4
Radius: 0.6  Recurrence coverage REC: 0.491  log(REC)/log(R): 1.392
Determinism: 0.973  Laminarity: 0.785
DIV: 0.017
Trend: 0  Entropy: 2.716
Diagonal lines max: 60  Mean: 7.092  Mean off main: 7.078
Vertical lines max: 1021  Mean: 3.867
```

RQA Test case: uniform random numbers, radius=0.6. Time used: 4.067 sec. . For graphical ouput, see Figure 26 on the next page.

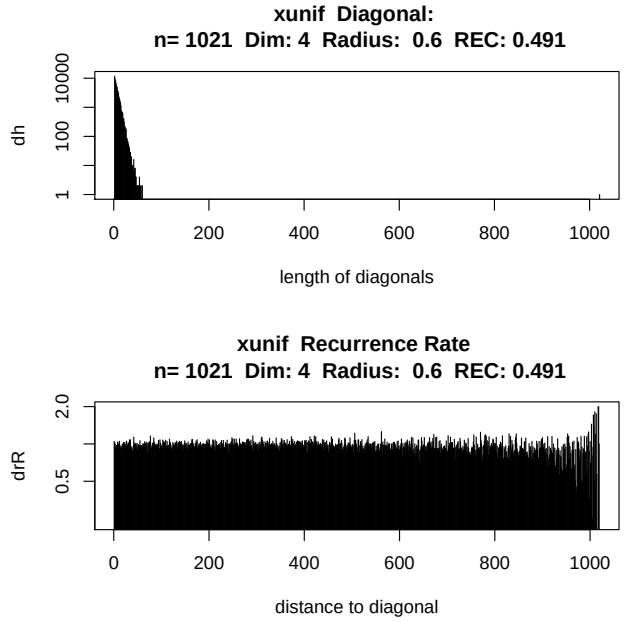


FIGURE 26. RQA. Test case: uniform random numbers, radius=0.6. Time used: 4.067 sec. , radius=0.64

6.0.3. *Chirp Signal Recurrence Plots.*

Input

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

See Figure 27.

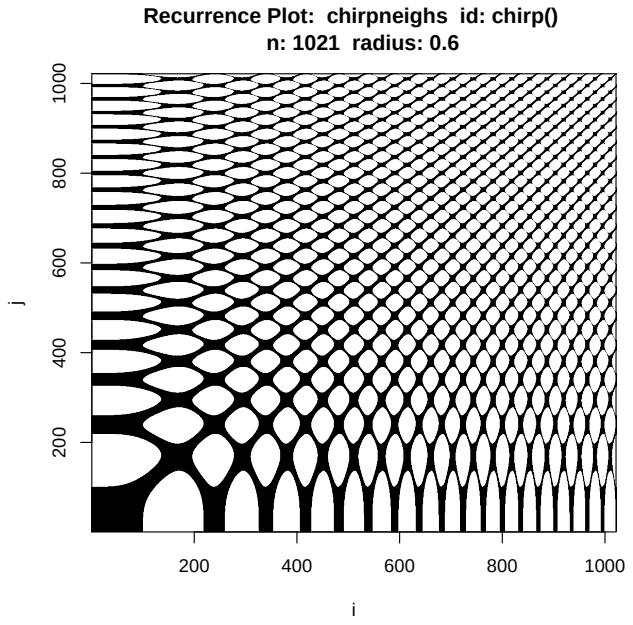


FIGURE 27. Recurrence plot. Test case: chirp signal. Time used: 1.805 sec.

Input

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

Output

```
chirp()  n: 1021  Dim: 4
Radius: 0.6  Recurrence coverage REC: 0.341  log(REC)/log(R): 2.108
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
```

RQA Test case: chirp signal. Time used: 2.889 sec.. For graphical ouput, see Figure 28 on the next page.

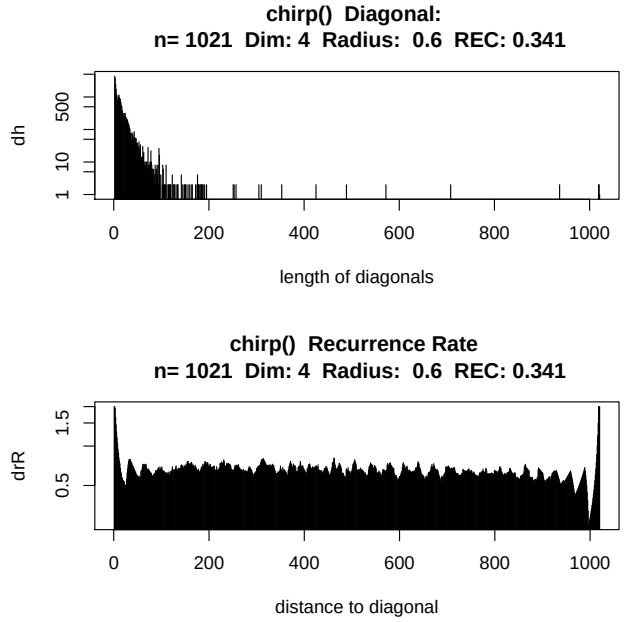


FIGURE 28. RQA. Test case: chirp signal. Time used: 2.889 sec., radius=0.64

6.0.4. *Doppler Recurrence Plots.*

Input

```
dopplerneighs <- local.findAllNeighbours(dopplertakens, radius=0.2)
save(dopplerneighs, file="dopplerneighs.Rdata")
```

Input

```
load(file="dopplerneighs.RData")
local.recurrencePlotAux(dopplerneighs, dim=4, radius=0.2)
```

See Figure 29.

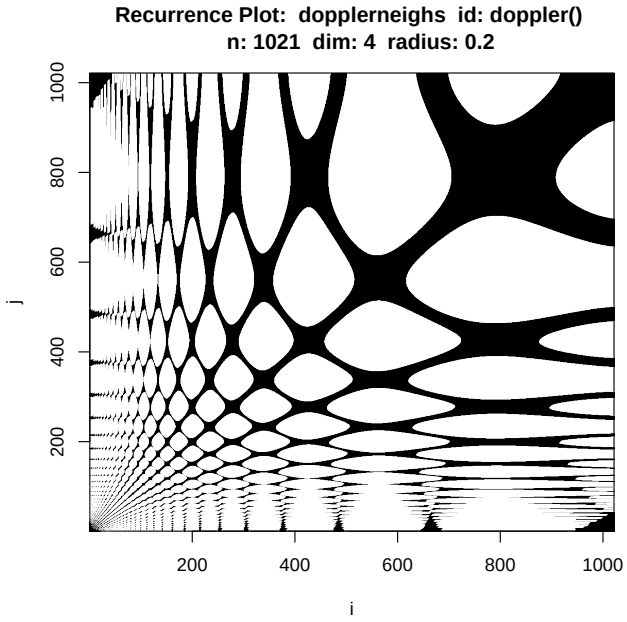


FIGURE 29. Recurrence plot. Test case: Doppler. Time used: 1.367 sec.

Input

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

Output

```
doppler()  n: 1021  Dim: 4
Radius: 0.2  Recurrence coverage REC: 0.299  log(REC)/log(R): 0.75
Determinism: 0.991  Laminarity: 0.995
DIV: 0.001
Trend: 0  Entropy: 3.445
Diagonal lines max: 1020  Mean: 17.496  Mean off main: 17.439
Vertical lines max: 329  Mean: 24.835
```

RQA Test case: Doppler. Time used: 0.679 sec.. For graphical ouput, see Figure 30 on the next page.

Input

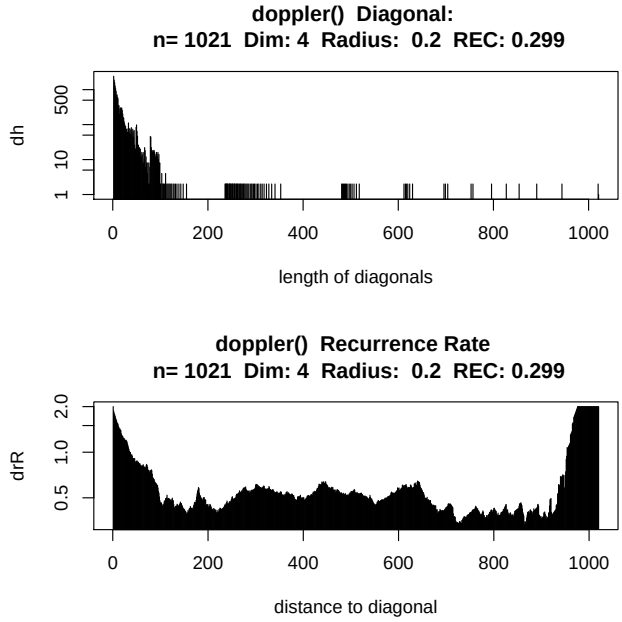


FIGURE 30. RQA. Test case: Doppler. Time used: 0.679 sec., radius=0.64

```
dopplerlag16neighs <- local.findAllNeighbours(dopplertakenslag16, radius=0.2)
save(dopplerlag16neighs, file="dopplerlag16neighs.Rdata")
# load(file="dopplerlag16neighs.RData")
local.recurrencePlotAux(dopplerlag16neighs, dim=4, radius=0.2)
```

See Figure 31.

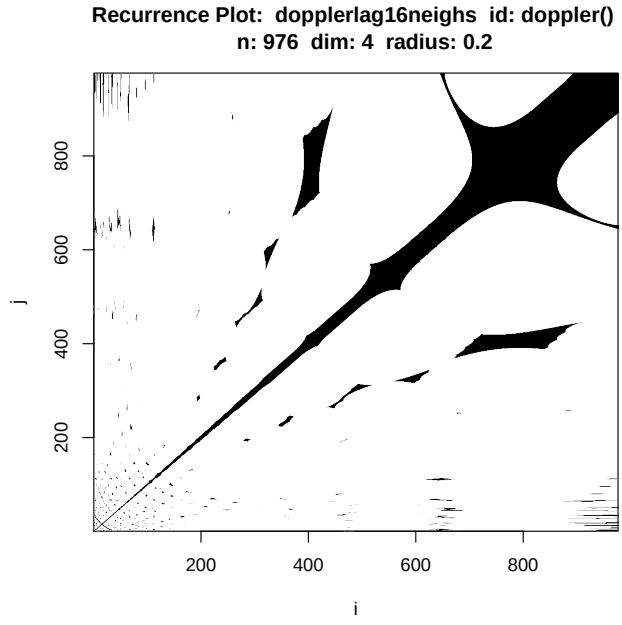


FIGURE 31. Recurrence plot. Test case: Doppler. Time used: 1.277 sec.

Input

```
showrqa(dopplertakenslag16, radius=0.2)
```

Output

```
doppler()  n: 976  Dim: 4
Radius: 0.2  Recurrence coverage REC: 0.098  log(REC)/log(R): 1.443
Determinism: 0.98  Laminarity: 0.989
DIV: 0.001
Trend: 0  Entropy: 2.815
Diagonal lines max: 975  Mean: 28.978  Mean off main: 28.678
Vertical lines max: 256  Mean: 31.539
```

RQA RQA. Test case: Doppler. Time used: 1.471 sec. . For graphical ouput, see Figure 32.

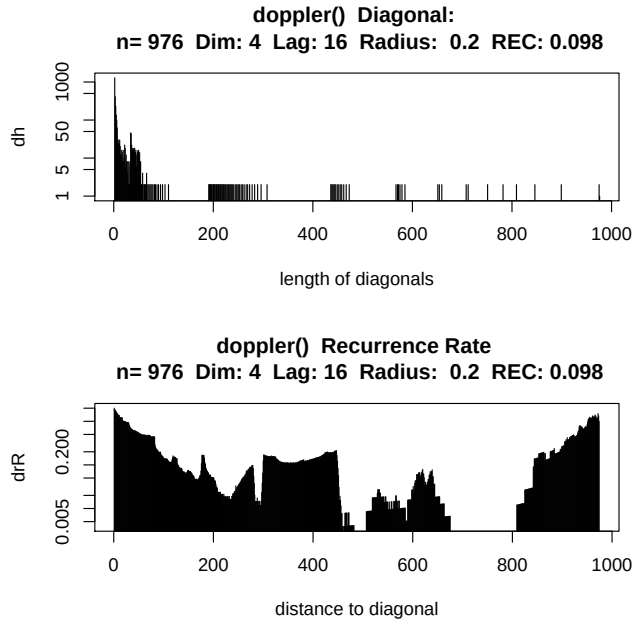


FIGURE 32. RQA. RQA. Test case: Doppler. Time used: 1.471 sec. , radius=0.64

7. Geyser example: Bivariate recurrence plots

ToDo: double check: `MASS:::geyser` should be used, not `faithful`

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

More specific the Geyser is a marked point process, with points marked as duration and waiting.

Decomposing it, we get two sequences, duration and waiting. Both sequences have (at least) bi-modal structures - a peculiarity of this data set. The main application problem is to predict waiting, based on the available information from the past.

Note for data analysis: The time to predict is the waiting time for the next eruption; so for prediction use a shifted sequence $waiting[-1]$.

Note for data analysis: Some data are not recorded, but replaced by codes such as short, long. Handling of partially encoded data is required.

After decomposition, both sequences can be handled separately.

Input
```
try(detach("package:MASS" ), silent=TRUE)
try(detach(faithful), silent=TRUE)
try(detach(geyser), silent=TRUE)
library(MASS)
data(geyser)
attach(geyser)
```

Input
```
plotsignal(duration)
```

Input
```
plotsignal(waiting)
```

Time used: 0.799 sec. Signal components and linear interpolation. See Figure 33 on the following page.

Input
```
attach(geyser)
takens.duration4 <- buildTakens( time.series=duration, embedding.dim=4, time.lag=1)
takens.waiting4 <- buildTakens( time.series=waiting, embedding.dim=4,  time.lag=1)
```

Time used: 0.831 sec.

Input
```
statepairs(takens.duration4) #dim=4
```

Input
```
statepairs(takens.waiting4) #dim=4
```

Example case: Old Faithful Geyser eruption data. Time used: 1.918 sec. See Figure 34 on the next page.

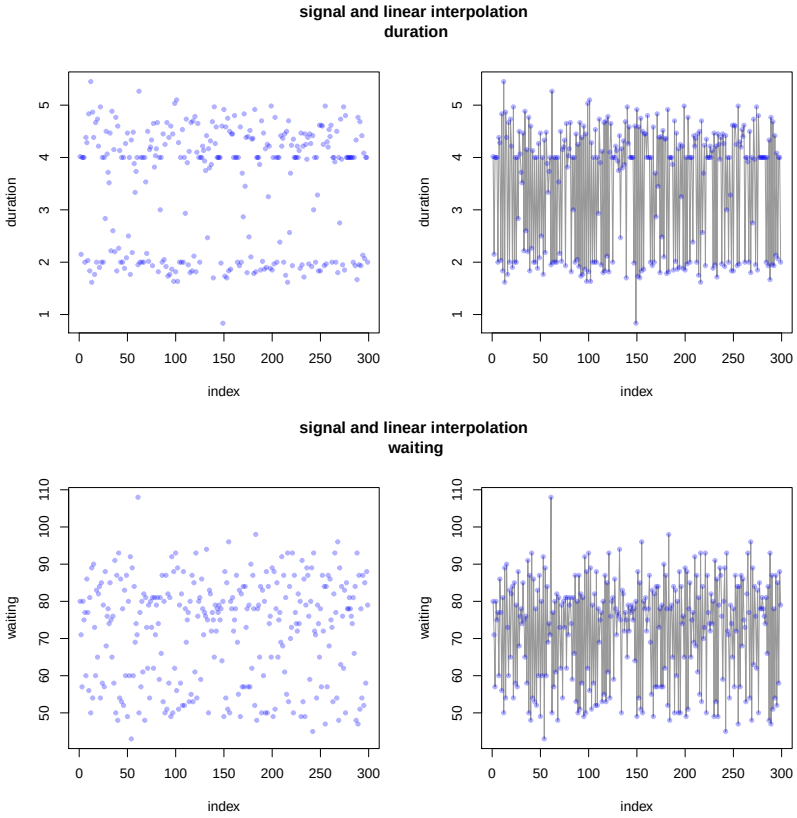


FIGURE 33. Example case: Old Faithful Geyser eruption durations and waiting time. Signals and linear interpolation. Time used: 0.8 sec.

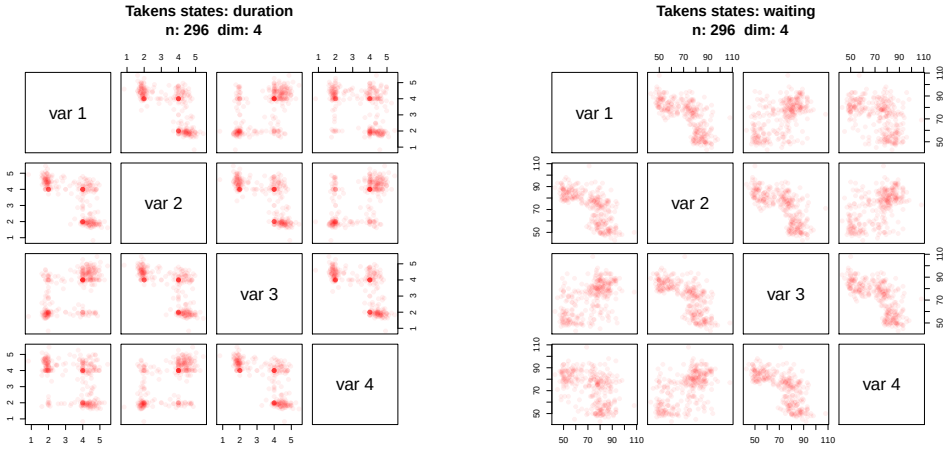


FIGURE 34. Old Faithful Geyser eruption durations and waiting time. Takens states, dim=4. Time used: 1.919 sec.

Input

```
durationneighs4<- findAllNeighbours(takens.duration4, radius=2.0)
waitingneighs4<-findAllNeighbours(takens.waiting4, radius=20.0)
```

Input

```
local.recurrencePlotAux(durationneighs4)
```

Input

```
local.recurrencePlotAux(waitingneighs4)
```

See Figure 35.

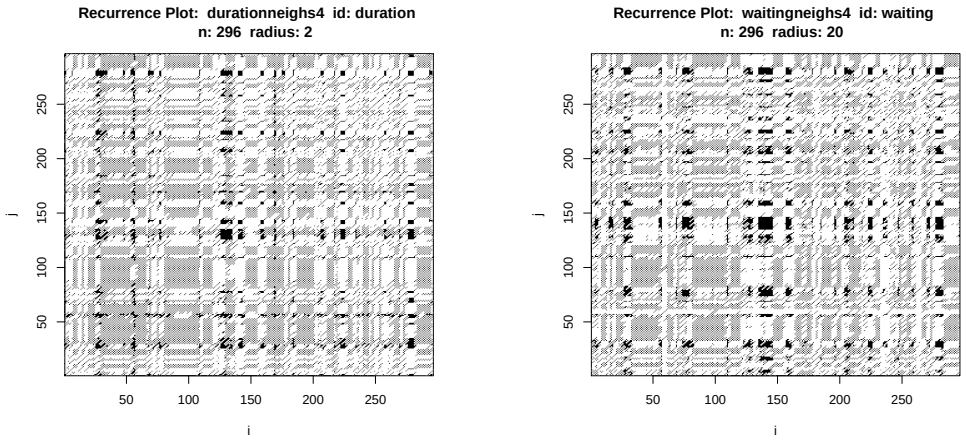


FIGURE 35. Old Faithful Geyser eruption durations and waiting time. Time used: 2.167 sec.

Assuming that labels etc. are propragated as attributes, we can use a modified version of *recurencePlot()*

Input

```
# univariate variant, assuming attributes
local.recurrencePlot1=function(neighs){
    dim <- attr(neighs,"embedding.dim")
    lag <- attr(neighs,"time.lag")
    radius <- attr(neighs,"radius")
        # just for reference. This function is inlined
        neighbourListNeighbourMatrix  = function(){
          #neighs.matrix = Diagonal(ntakens)
          for (i in 1:ntakens){
            if (length(neighs[[i]])>0){
              for (j in neighs[[i]]){
                neighs.matrix[i,j] = 1
              }
            }
          }
          return (neighs.matrix)
        }

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

}
```

Input

```
oldpar <- par(mfrow=c(1,2))
local.recurrencePlot1(durationneighs4)
local.recurrencePlot1(waitingneighs4)
par(oldpar)
```

Each plot shows a symmetric matrix. Since dimensions match, we can use the upper and ower triangle to allow comparison of both series in one plot, comparing the upper and lower triangle.

Input

```
# bivariate variant, assuming attributes
# same dimension and size required
local.recurrencePlot2=function(neighs1, neighs2){
    dim1 <- attr(neighs1,"embedding.dim")
    lag1 <- attr(neighs1,"time.lag")
    radius1 <- attr(neighs1,"radius")

    dim2 <- attr(neighs2,"embedding.dim")
    lag2 <- attr(neighs2,"time.lag")
    radius2 <- attr(neighs2,"radius")

        # just for reference. This function is inlined
        neighbourListNeighbourMatrix  = function(){
          #neighs.matrix = Diagonal(ntakens)
          for (i in 1:ntakens){
            if (length(neighs[[i]])>0){
```

```
            for (j in neighs[[i]]){
              neighs.matrix[i,j] = 1
            }
          }
        }
        return (neighs.matrix)
      }

#1
  ntakens1=length(neighs1)
  neighs1.matrix <- matrix(nrow=ntakens1,ncol=ntakens1)
  #neighbourListNeighbourMatrix()
    #neighs.matrix = Diagonal(ntakens)
          for (i in 1:ntakens1){
                  neighs1.matrix[i,i] = 1 # do we want the diagonal fixed to 1
            if (length(neighs1[[i]])>0){
              for (j in neighs1[[i]]){
                neighs1.matrix[i,j] = 1
              }
            }
          }
#2
  ntakens2=length(neighs2)
  neighs2.matrix <- matrix(nrow=ntakens2,ncol=ntakens2)
  #neighbourListNeighbourMatrix()
    #neighs.matrix = Diagonal(ntakens)
          for (i in 1:ntakens2){
                  neighs2.matrix[i,i] = 1 # do we want the diagonal fixed to 1
            if (length(neighs2[[i]])>0){
              for (j in neighs2[[i]]){
                neighs2.matrix[i,j] = 1
              }
            }
          }
# merge
neighs.matrix <- neighs1.matrix
# replace upper triangle by neighs2.matrix
for (i in 1:ntakens2){
        for (j in i:ntakens2)
        neighs.matrix[i,j] <- - neighs2.matrix[i,j]  #for colour
}


        main <- paste("Recurrence Plot: ",
                      deparse(substitute(neighs1)),
                      deparse(substitute(neighs2))
                      )
        more <- NULL

        #use compones of neights if available
        if (!is.null(dim1)) more <- paste(more," dim:",dim1, dim2)
        if (!is.null(lag1)) more <- paste(more," lag:",lag1, lag2)
        if (!is.null(radius1)) more <- paste(more," radius:",radius1, radius2)

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

ntakens <- ntakens1

  # 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=c("red","blue"),
                #xlim=c(1,ntakens), ylim=c(1,ntakens),
                useRaster=TRUE,  #? is this safe??
                main=main
                )
 #        )

}
```

Input

```
oldpar <- par(mfrow=c(1,1))
local.recurrencePlot2(durationneighs4,waitingneighs4)
par(oldpar)
```

See Figure 36.

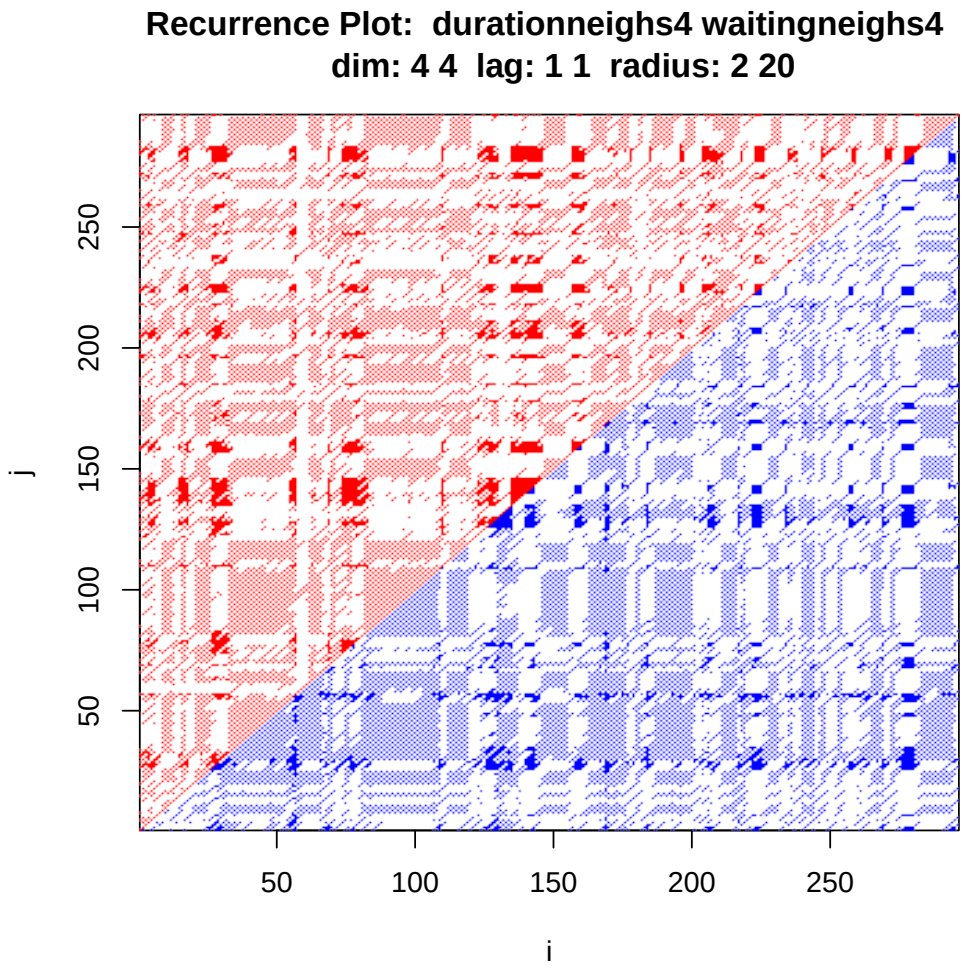


FIGURE 36. Recurrene plot: Lower triangle: duration. Upper triangle: waiting.

So far, the data have been handled as a pair of one-dimensional problems. Looking at the data as a bivariate problem requires to use a bivariate distance to define neighbours.

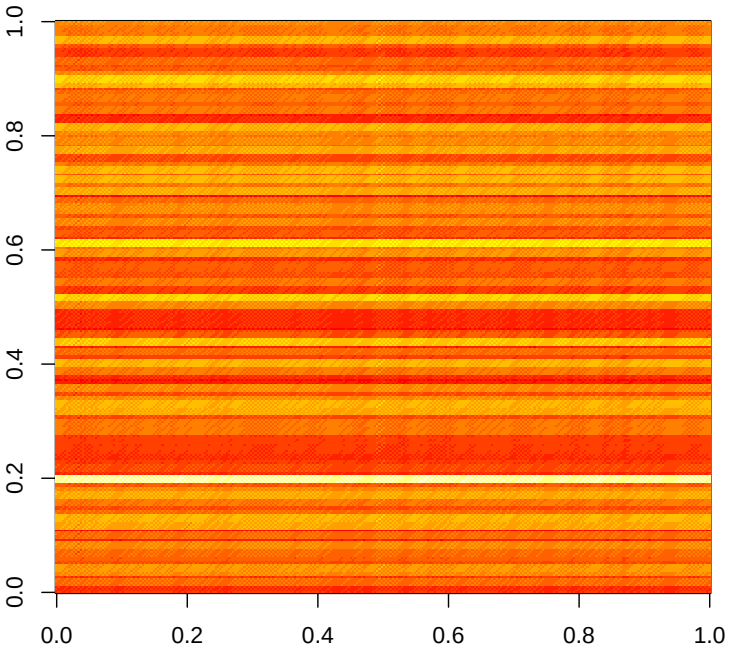
Input

```
#
# covariance
# see \cite{marwan2002nonlinear}
# CR[i,j] = \Theta( \epsilon - \norm( xv[i] - yv[j] ))

# Brute force

# signed distances may be used for experiments.
maxdist <- function (x){ max(abs(x))} # works on delta
cordist <- function (x, y){ suppressWarnings(cor(x,y))} # using signed. Warnings for zero #
# variances are suppressed

# epsilon/radius and heaviside not use here - may be added in image rendering.

# propagate names from takens

CR0 <- function (xtakens, ytakens, sdist= maxdist) {
        xl <- nrow(xtakens); yl <- nrow(ytakens)
        xname <- deparse(substitute(xtakens))
        yname <- deparse(substitute(ytakens))
        cr <- matrix(nrow=xl, ncol=yl)
        for (i in 1:xl)
         for (j in 1:yl)
        {
        cr[i,j] <- sdist(xtakens[i,]- ytakens[j,])
        }
        return(cr)
}
CR2 <- function (xtakens, ytakens, cdist= cordist) {
        xl <- nrow(xtakens); yl <- nrow(ytakens)
        xname <- deparse(substitute(xtakens))
        yname <- deparse(substitute(ytakens))
        cr <- matrix(nrow=xl, ncol=yl)
        for (i in 1:xl)
         for (j in 1:yl)
        {
        cr[i,j] <- cdist(xtakens[i,], ytakens[j,])
        }
        return(cr)
}
## for experiments only. do not copy large matrices
crossrecurrencePlotFromMatrix <- function(neighs.matrix,
                zlim= range(neighs.matrix, na.rm=TRUE),
                main="Cross Recurrence plot",
                xlab="x Takens vector's index",
                ylab="y Takens vector's index",...){
# need a print because it is (possibly) a trellis object!!
rec.plot = image(neighs.matrix,
                zlim= zlim,
                x = 1: ncol(neighs.matrix),
                y = 1: nrow(neighs.matrix),
                   main = main, xlab = xlab, ylab = ylab,
                   ...)
print(rec.plot)
rec.plot
```

```
}
# raw data may give a poor impression. Adjust e.g. for scale and location
cr4 <- CR0(takens.duration4,takens.waiting4)
cr4C <- CR2(takens.duration4,takens.waiting4)
```

Input

```
image(cr4)
# neighs.matrix <- cr4;range(cr4) #  70.5500 107.1667
```

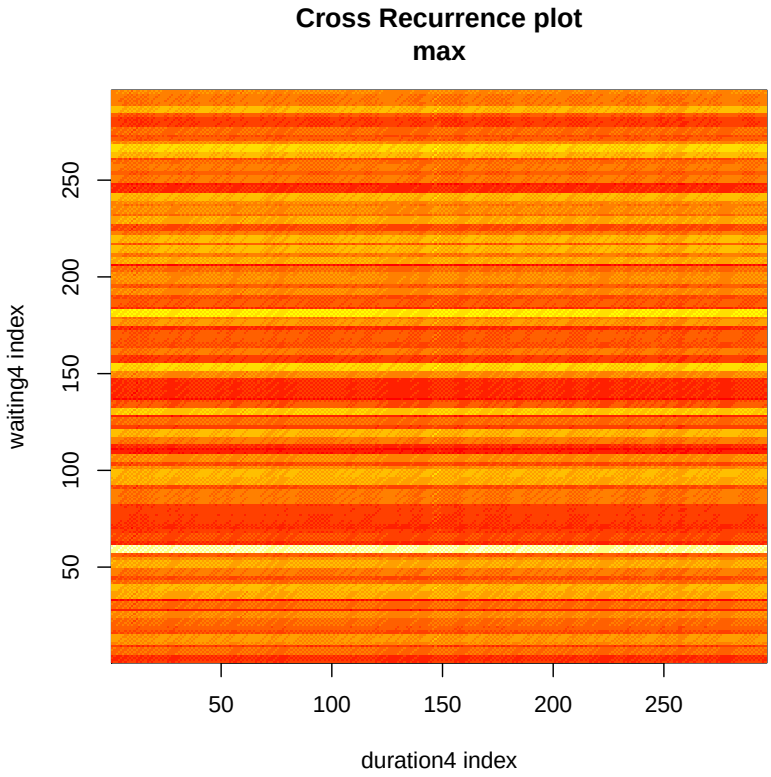


Input

```
# a max distance
crossrecurrencePlotFromMatrix(cr4,
main="Cross Recurrence plot\nmax",
xlab="duration4 index", ylab="waiting4 index")  # ugly heat matrix
```

Output

```
NULL
NULL
```



Input

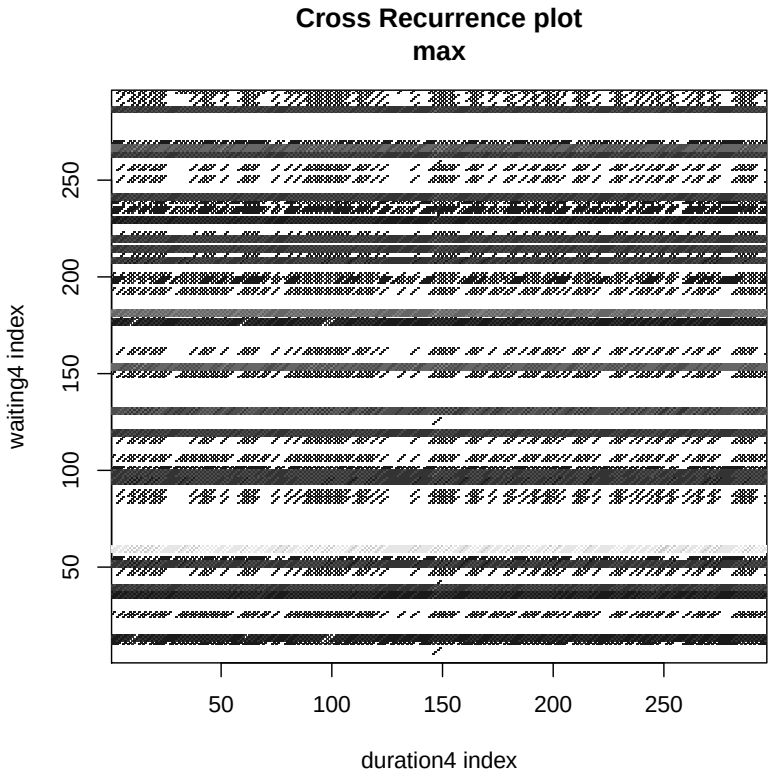
```
crossrecurrencePlotFromMatrix(cr4, zlim=c(85,108), col=grey((1:10)/10),
main="Cross Recurrence plot\nmax",
xlab="duration4 index", ylab="waiting4 index")
```

Output

```
NULL
NULL
```

Input

```
# near conventional bw,
# introducing radius/cut by zlim
```

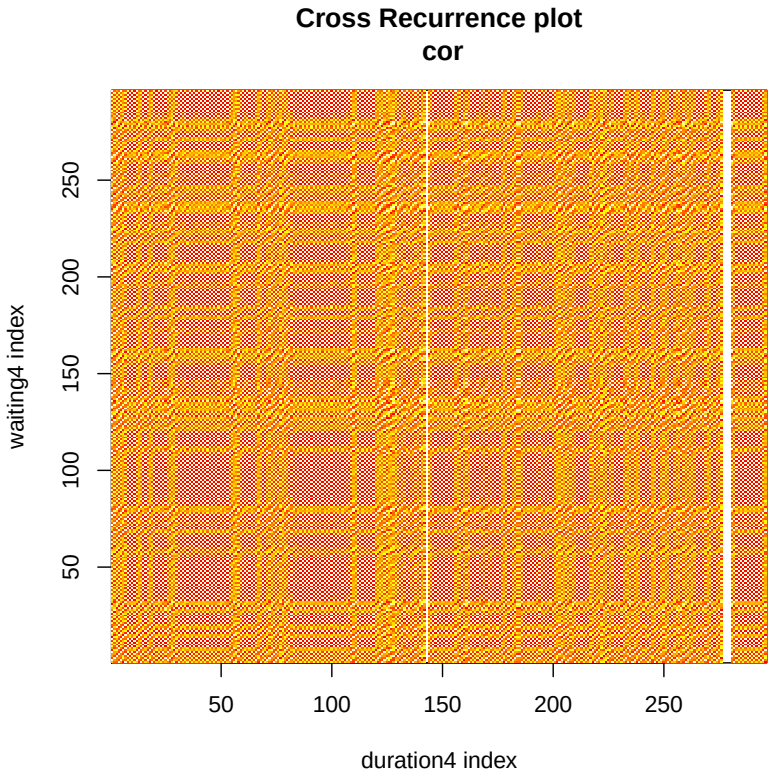


Input

```
# a correlation distance
crossrecurrencePlotFromMatrix(cr4C,
main="Cross Recurrence plot\n cor",
xlab="duration4 index", ylab="waiting4 index")  # ugly heat matrix
```

Output

```
NULL
NULL
```



Input

```
quantile(cr4C, na.rm=TRUE)
```

Output

```
         0%         25%         50%         75%        100%
-1.00000000 -0.68688423 -0.03466478  0.72172812  1.00000000
```

Input

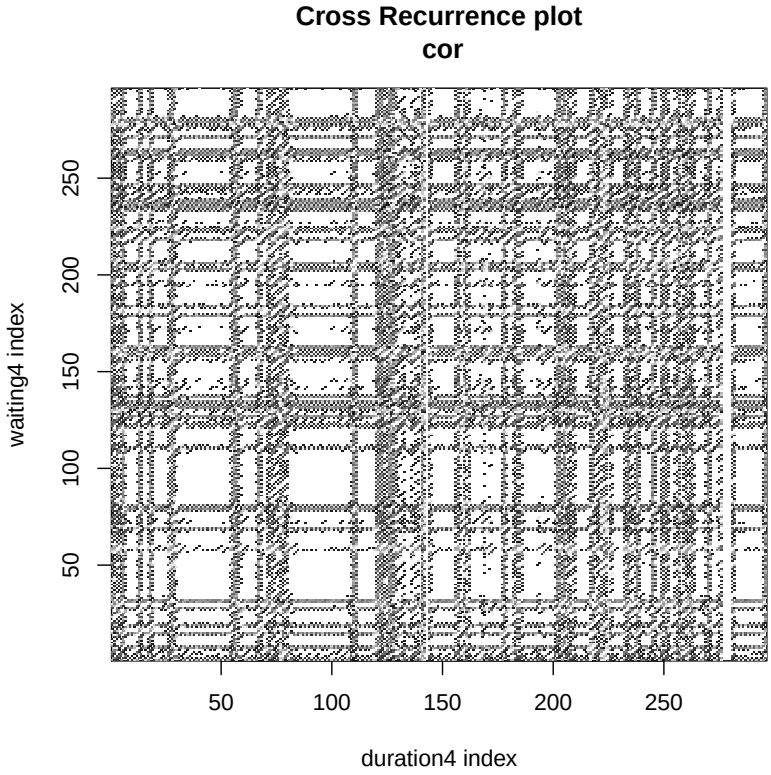
```
crossrecurrencePlotFromMatrix(cr4C, zlim=c(-0.7,0.7), col=grey((1:10)/10),
main="Cross Recurrence plot\n cor",
xlab="duration4 index", ylab="waiting4 index")
```

Output

```
NULL
NULL
```

Input

```
# near conventional
# introducing radius/cut by zlim
```


Cross Recurrence plot
cor
waiting4 index
duration4 index
50
100
150
200
250

8. Case Study: Geyser data –defunct

Defunct. hopefully completely replaced by bivariate analysis in previous chapter.

8.1. **Geyser Eruption Durations.**

Input

```
takens_duration4_x1 <-
        local.buildTakens( time.series=geyser$duration,
                embedding.dim=4, time.lag=1)
        statepairs(takens_duration4_x1) #dim=4
```

See Figure 37.

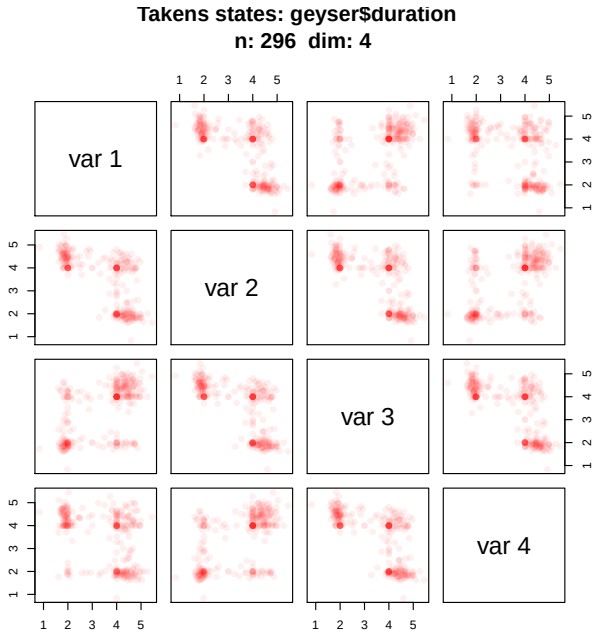


Figure 37. Recurrence plot. Old Faithful Geyser eruption durations. Time used: 0.887 sec.

Input

```
        statecoplot(takens_duration4_x1) #dim=4
```

See Figure 38 on the next page.

%

Input

```
eruptionsneighs4 <- local.findAllNeighbours(takens_duration4_x1, radius=3*0.8)
save(eruptionsneighs4, file="eruptionsneighs4.RData")
```

Input

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

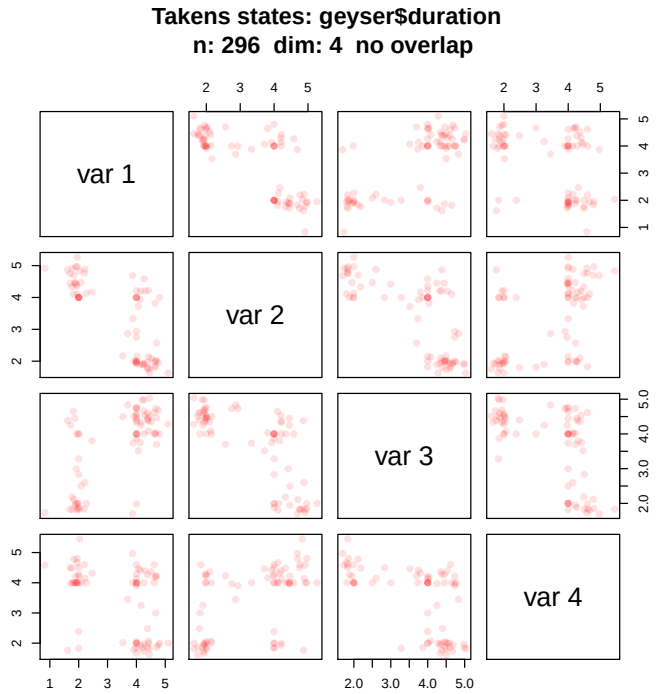


FIGURE 38. State coplot. Example case: Old Faithful Geyser eruption durations. Time used: 1.031 sec.

See Figure 39.

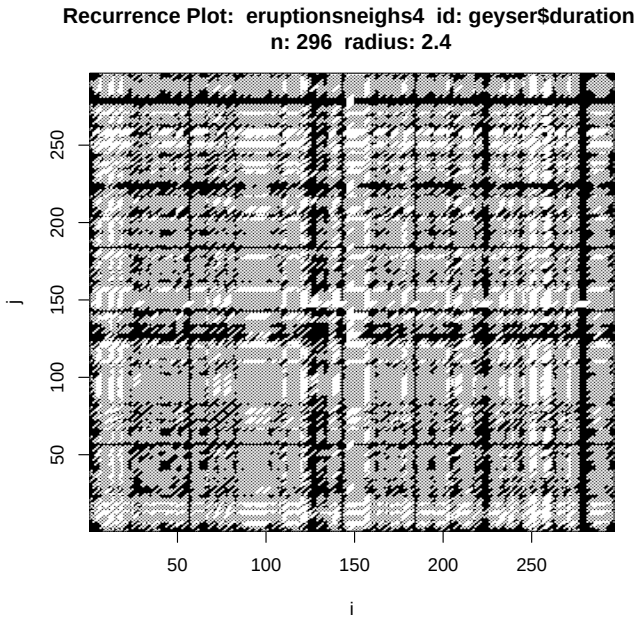


FIGURE 39. Recurrence plot. Old Faithful Geyser eruption durations. Dim=4.. Time used: 0.149 sec.

```
showrqa(takens_duration4_x1, radius=3*0.8)
```

Output

```
geyser$duration  n: 296  Dim: 4
Radius: 2.4  Recurrence coverage REC: 0.554  log(REC)/log(R): -0.675
Determinism: 0.986  Laminarity: 0.562
DIV: 0.011
Trend: 0  Entropy: 2.994
Diagonal lines max: 88  Mean: 9.147  Mean off main: 9.092
Vertical lines max: 147  Mean: 4.264
```

RQA Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.584 sec.. For graphical ouput, see Figure 40.

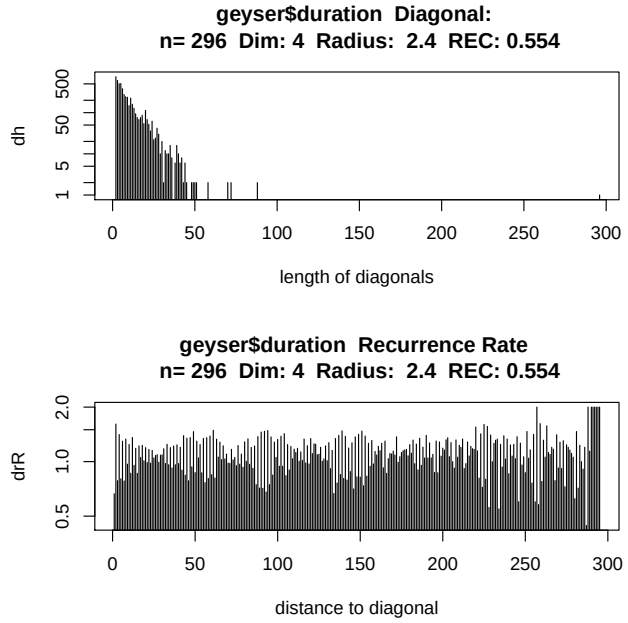


FIGURE 40. RQA. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.584 sec., radius=0.64

ToDo: remove this section

8.1.1. *Geyser eruption durations. Dim=2.*

Input

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

See Figure 41 on the following page.

Input

```
statepairs(eruptionstakens2, nooverlap=TRUE) #dim=2
```

See Figure 42 on the next page

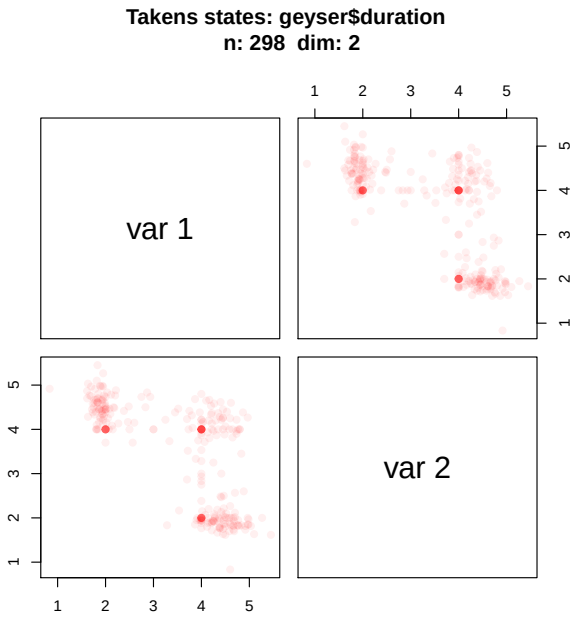


FIGURE 41. Recurrence plot. Old Faithful Geyser eruption durations. Dim=2. Time used: 0.076 sec.

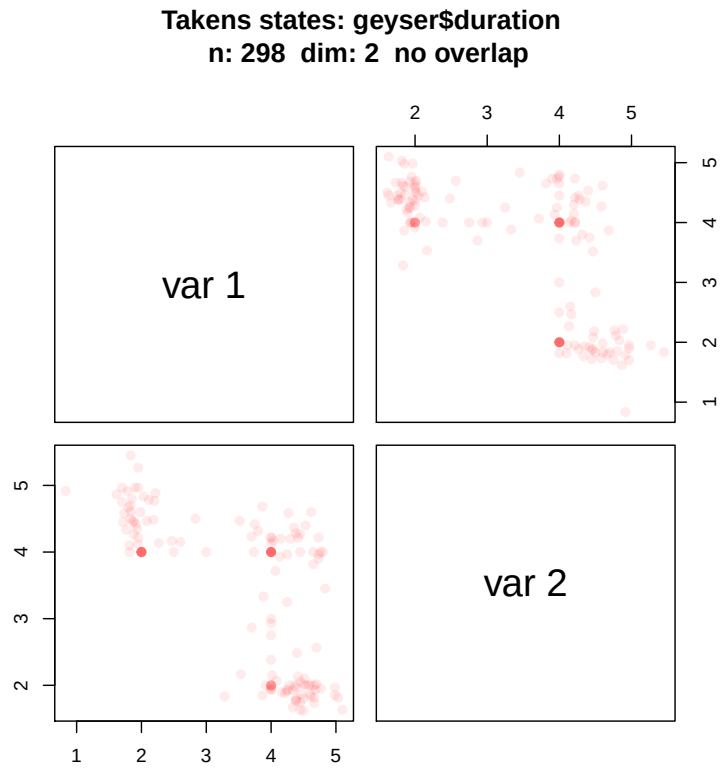


FIGURE 42. Example case: Old Faithful Geyser eruption durations. Dim=2. Time used: 0.132 sec.

Input

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

See Figure 43.

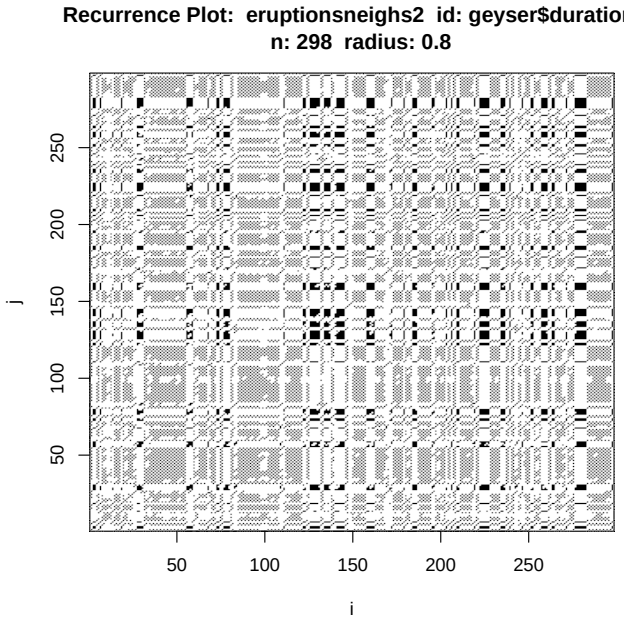


FIGURE 43. Recurrence plot. Old Faithful Geyser eruption durations. Dim=2. Time used: 0.753 sec.

Input

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

Output

```
geyser$duration  n: 298  Dim: 2
Radius: 0.8  Recurrence coverage REC: 0.274  log(REC)/log(R): 5.806
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
```

RQA Example case: Old Faithful Geyser eruption durations. Dim=2. Time used: 0.857 sec.. For graphical ouput, see Figure 44 on the next page.

8.1.2. *Geyser eruptions. Dim=4.*

Input

```
takens_duration4_x1 <- local.buildTakens( time.series=geyser$duration,
        embedding.dim=4, time.lag=1)
statepairs(takens_duration4_x1) #dim=4
```

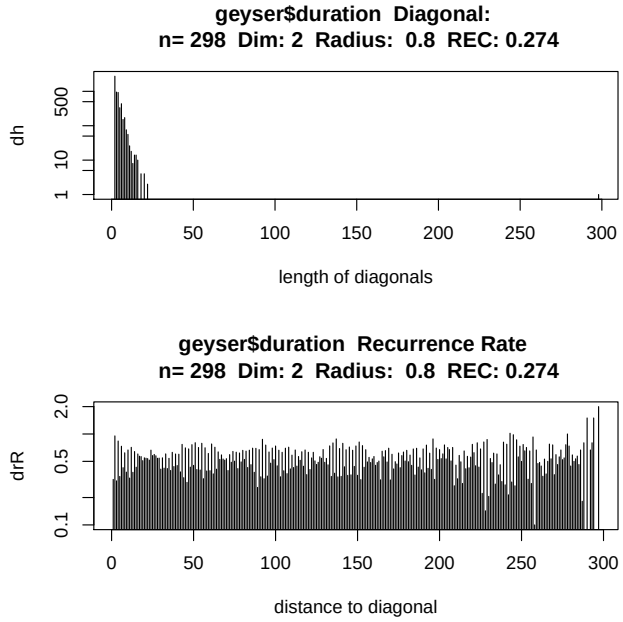


FIGURE 44. RQA. Example case: Old Faithful Geyser eruption durations. Dim=2. Time used: 0.857 sec., radius=0.64

See Figure 45.

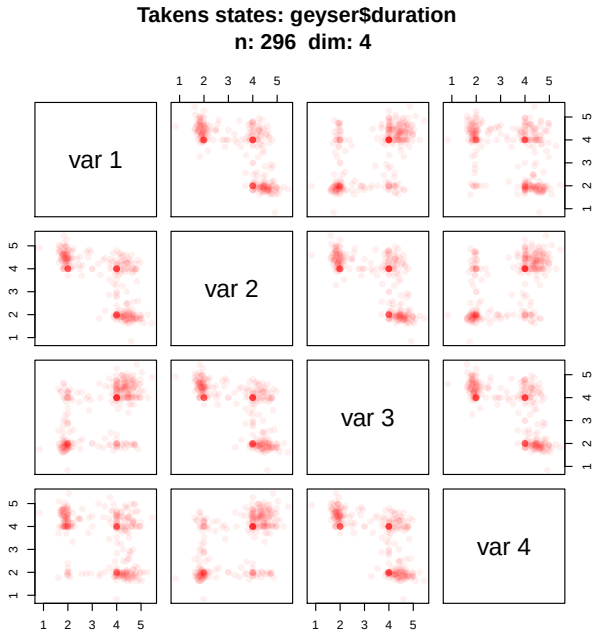


FIGURE 45. Recurrence plot. Old Faithful Geyser eruption durations. Dim=4. Time used: 0.234 sec.

Input

```
statepairs(takens_duration4_x1, nooverlap=TRUE) #dim=4
```

See Figure 46.

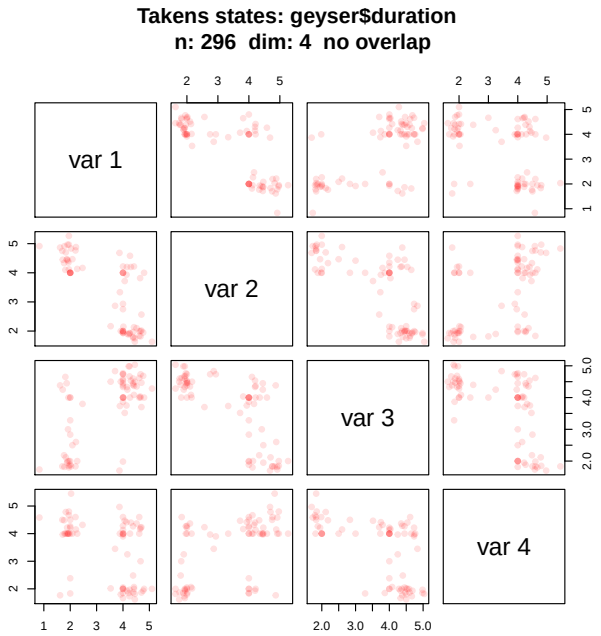


FIGURE 46. Recurrence plot. Old Faithful Geyser eruption durations. Dim=4. Time used: 0.675 sec.

Input

```
eruptionsneighs4 <- local.findAllNeighbours(takens_duration4_x1,
        radius=0.8)
save(eruptionsneighs4, file="eruptionsneighs4.RData")
```

Input

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

See Figure 47 on the following page.

Input

```
showrqa(takens_duration4_x1, radius=0.8)
```

Output

```
geyser$duration  n: 296  Dim: 4
Radius: 0.8  Recurrence coverage REC: 0.112  log(REC)/log(R): 9.822
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 48 on the next page.

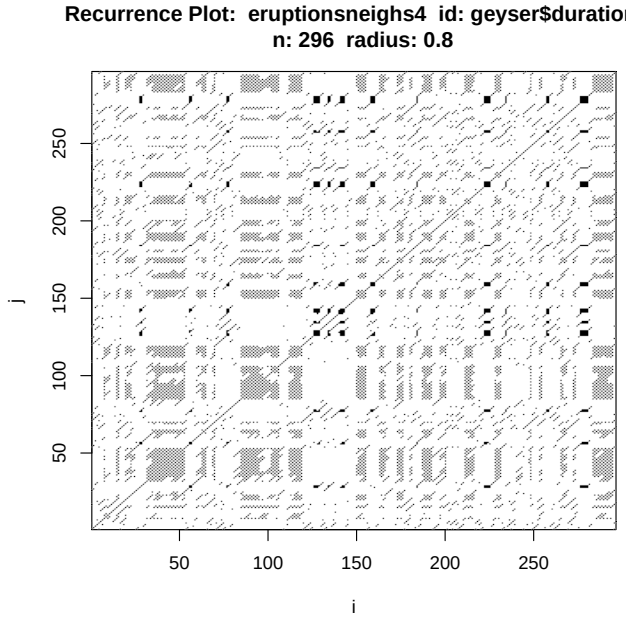


FIGURE 47. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.08 sec.

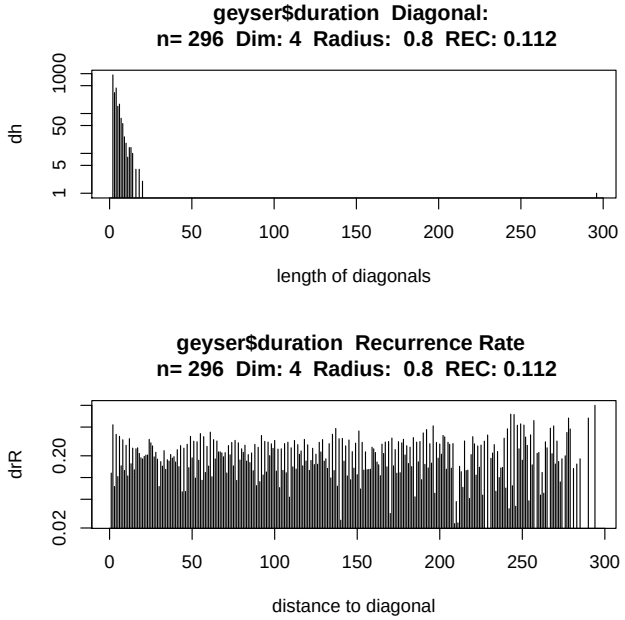


FIGURE 48. Recurrence plot. Old Faithful Geyser eruption durations. Dim=4. Time used: 0.16 sec.

8.1.3. *Geyser eruption durations. Dim=8.*

Input

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

See Figure 49.

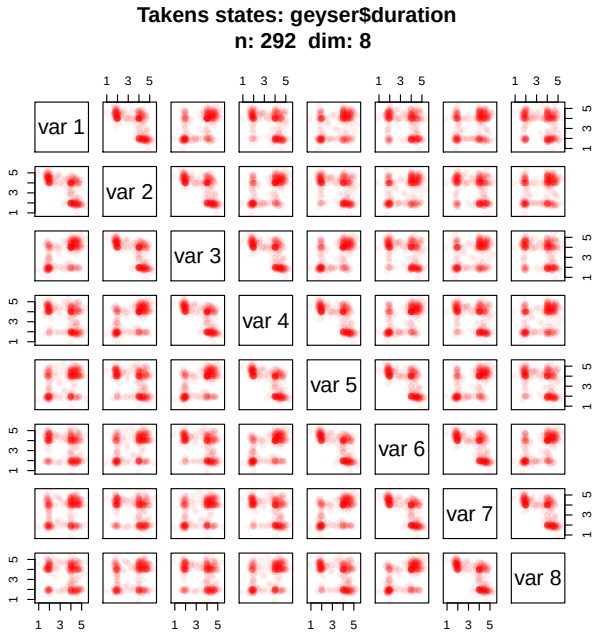


FIGURE 49. Recurrence plot. Old Faithful Geyser eruption durations. Dim=8. Time used: 1.755 sec.

Input

```
statepairs(eruptionstakens8, nooverlap=TRUE) #dim=8
```

See Figure 50 on the next page.

Input

```
eruptionsneighs8 <- local.findAllNeighbours(eruptionstakens8,
        radius=2.6)
save(eruptionsneighs8, file="eruptionsneighs8.RData")
#load(file="eruptionsneighs8.RData")
local.recurrencePlotAux(eruptionsneighs8)
```

See Figure 51 on page 77.

Input

```
showrqa(eruptionstakens8, radius=2.6)
```

Output

```
geyser$duration  n: 292  Dim: 8
Radius: 2.6  Recurrence coverage REC: 0.494  log(REC)/log(R): -0.738
Determinism: 0.991  Laminarity: 0.5
DIV: 0.011
Trend: 0  Entropy: 3.338
```

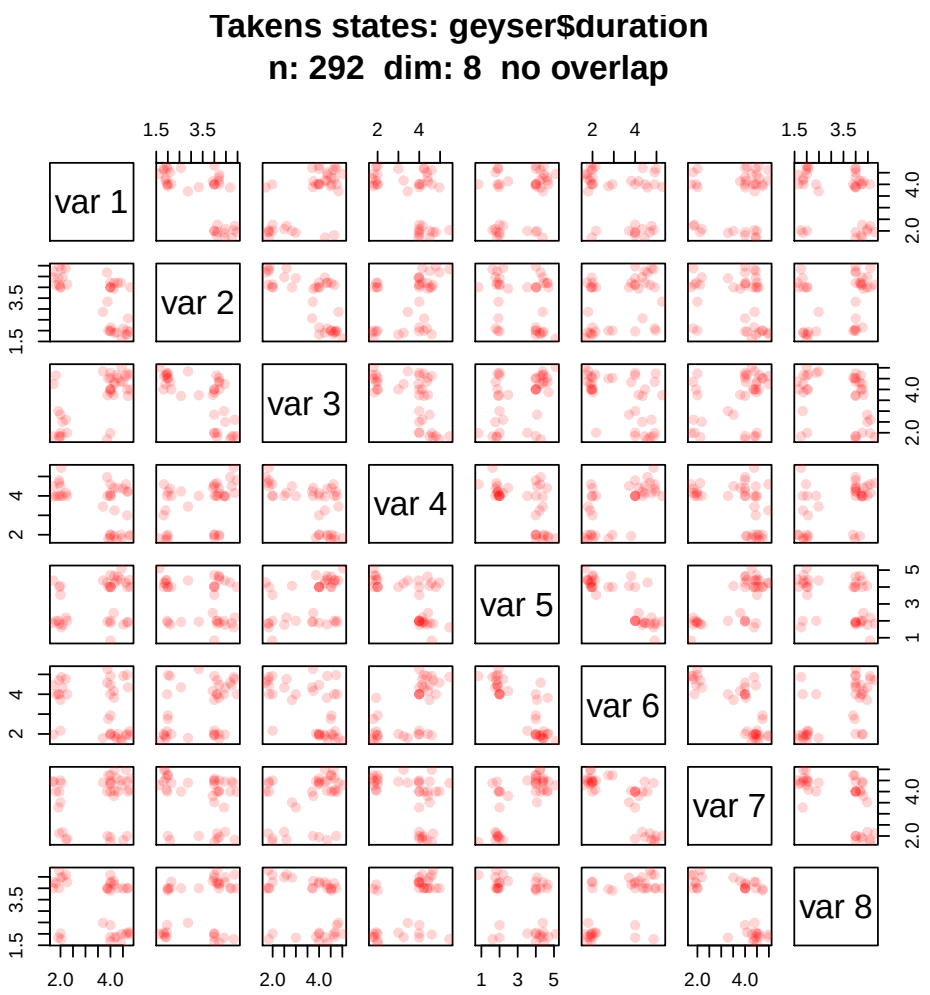


FIGURE 50. Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 2.008 sec.

```
Diagonal lines max: 93  Mean: 12.635  Mean off main: 12.55
Vertical lines max: 36  Mean: 4.073
```

RQA Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 0.608 sec. . For graphical ouput, see Figure 52 on the next page.

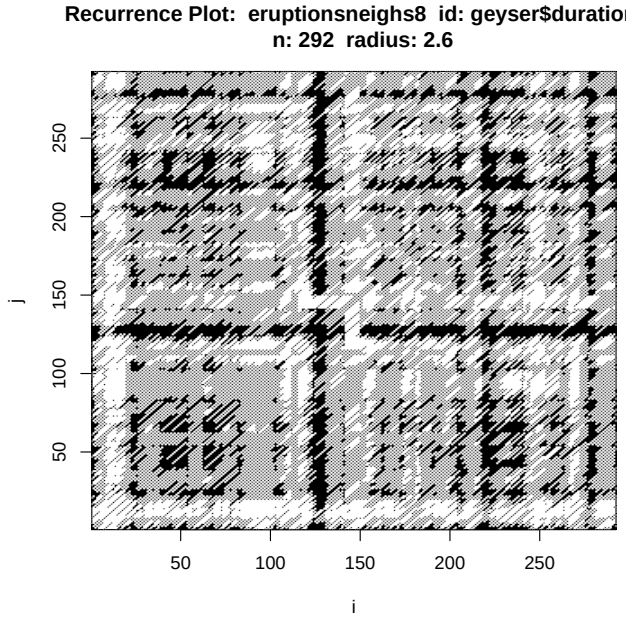


FIGURE 51. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 0.505 sec.

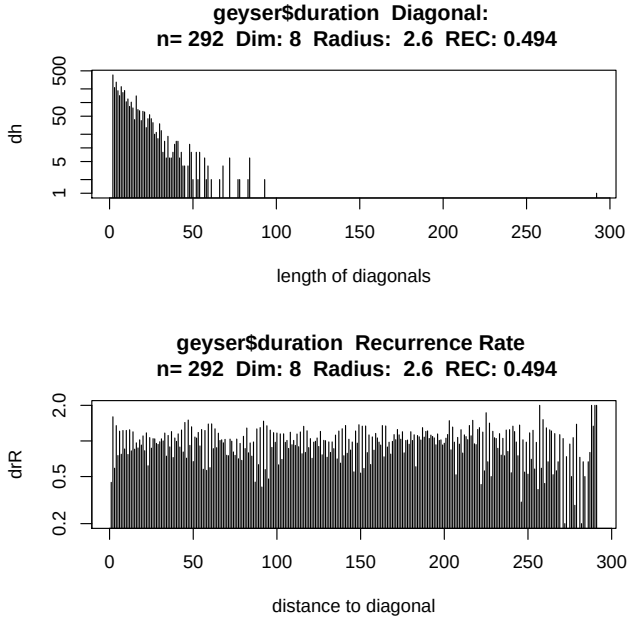


FIGURE 52. RQA. Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 0.608 sec. , radius=0.64

8.1.4. *Geyser eruption durations: Comparison by Dimension.* For comparison, recurrence plots for the Geyser data with varying dimension are in Figure 53

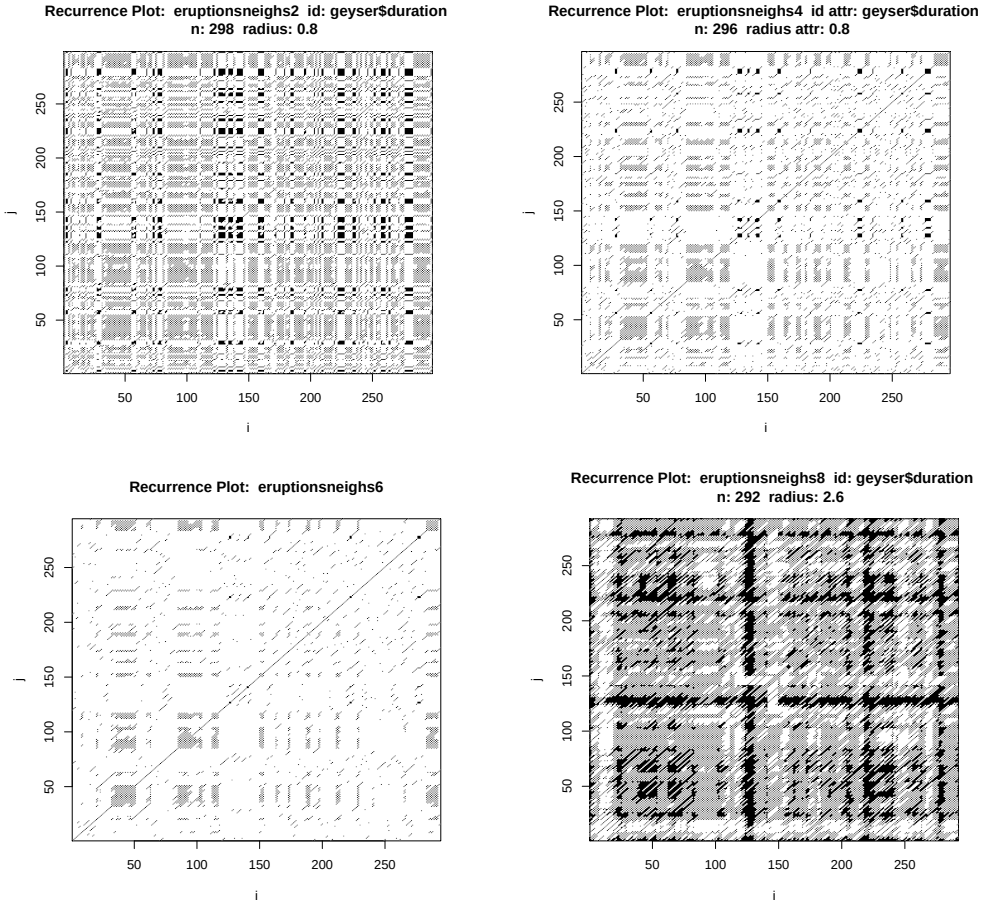


FIGURE 53. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=2, 4, 6, 8.

8.2. **Geyser Waiting, lag=4.** Note: is this of any use? Any good example for use of `lag`?

Input

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

See Figure 54 on the next page.

Input

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

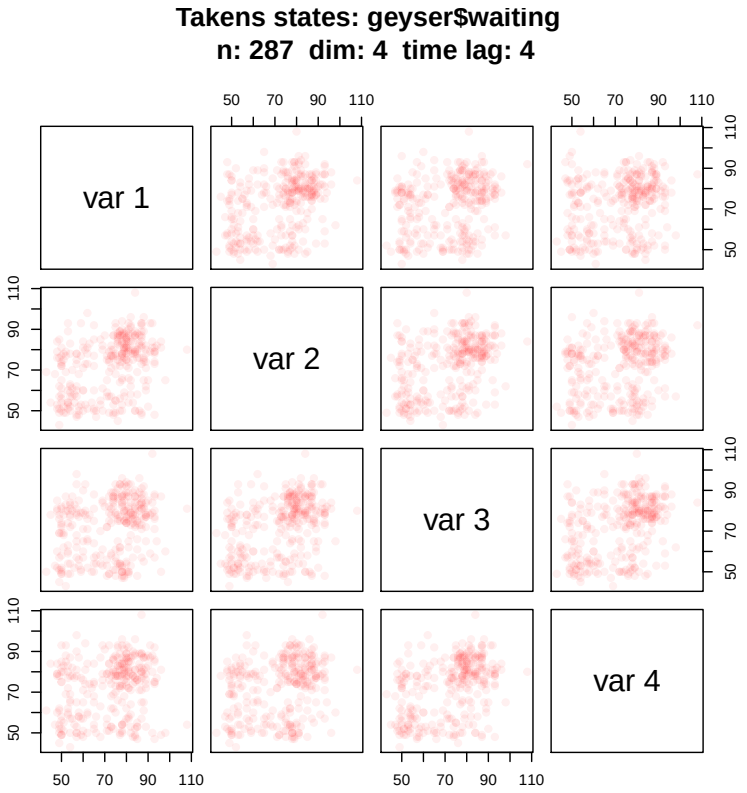


FIGURE 54. Example case: Old Faithful Geyser waiting. lag 4. Time used: 1.174 sec.

Input

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

Output

```
geyser$waiting  n: 287  Dim: 4
Radius: 16  Recurrence coverage REC: 0.137  log(REC)/log(R): -0.718
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
```

RQA Example case: Old Faithful Geyser waiting. Time used: 0.082 sec. . For graphical ouput, see Figure 55 on the following page.

Input

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

See Figure 56 on the next page.

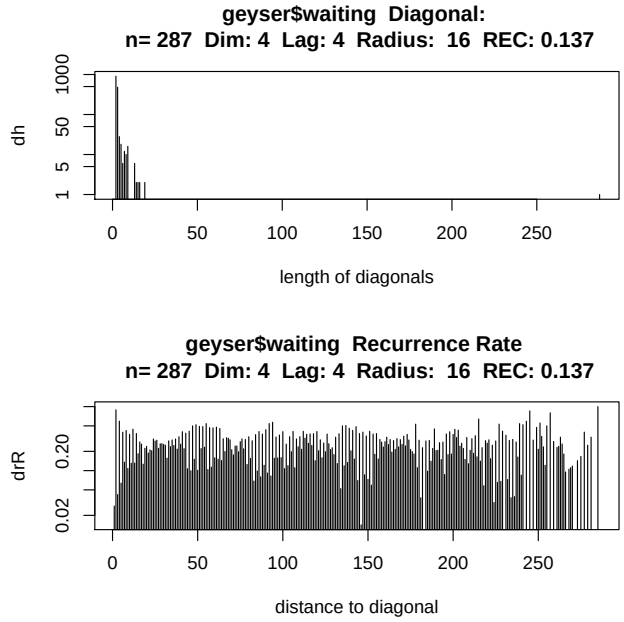


FIGURE 55. RQA. Example case: Old Faithful Geyser waiting. Time used: 0.082 sec. , radius=0.64

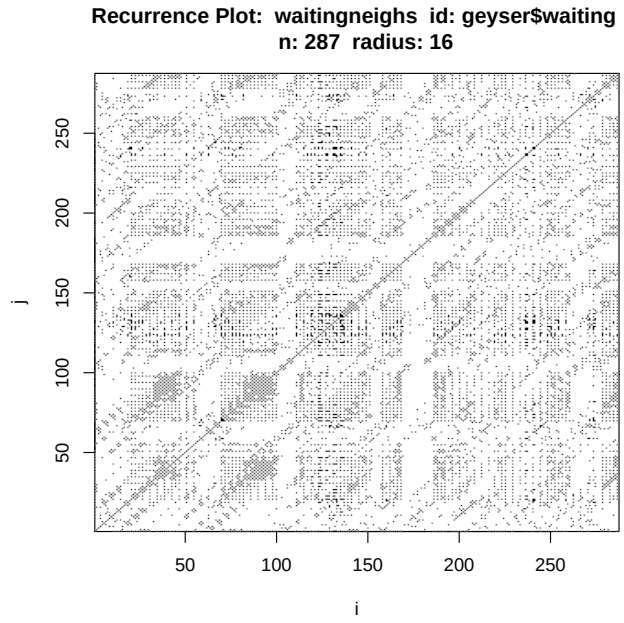


FIGURE 56. Recurrence plot. Example case: Old Faithful Geyser waiting.. Time used: 0.913 sec.

9. Case Study: HRV data example.beats

Only 1024 data points used in recurrence plots in this section

Note for data analysis: The data include missing beats, or spurious artefacts. These need to be handled.

Input

```
#stop("Stopping before RHRV")

#install.packages("RHRV",repos="http://r-forge.r-project.org",type="source")
if (!require("RHRV")) {
install.packages("RHRV")
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")
#       RecordPath = "beatsFolder")


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

Input

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

See Figure 57.

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

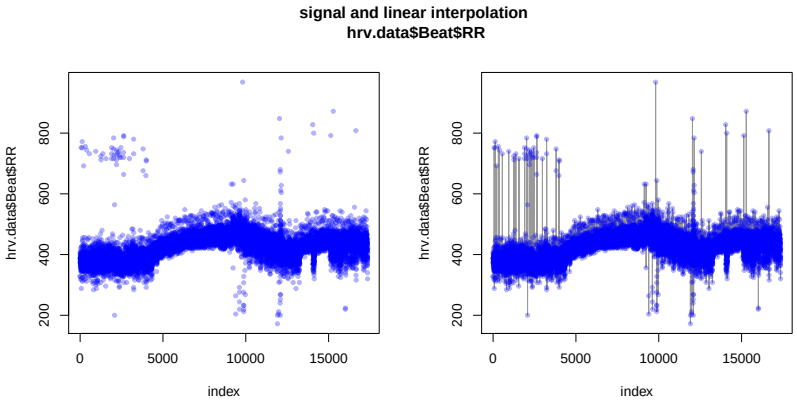


Figure 57. RHRV tutorial example.beats. Signal and linear interpolation.

Input

```
hrvRRtakens4 <- local.buildTakens( time.series=hrv.data$Beat$RR[1:nsignal],
        embedding.dim=4,time.lag=1)
statepairs(hrvRRtakens4) #dim=4
```

See Figure 58.

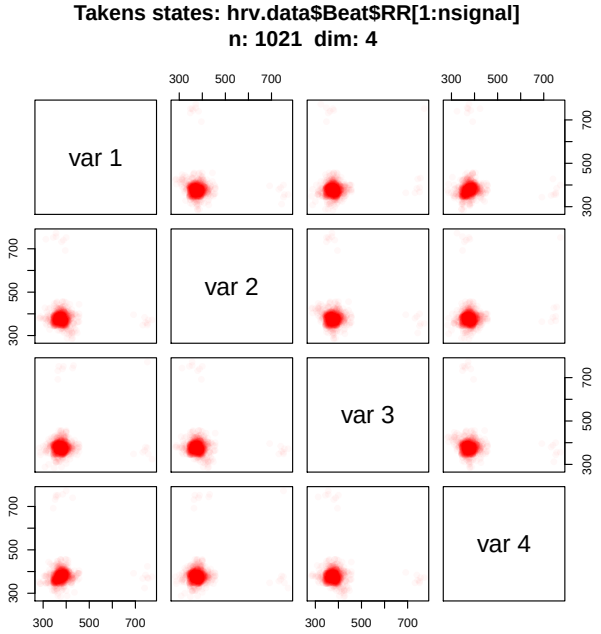


FIGURE 58. Recurrence plot. RHRV tutorial example.beats.. Time used: 0.86 sec.

Input

```
statepairs(hrvRRtakens4, rank=TRUE) #dim=4
```

See Figure 59 on the facing page.

Input

```
statecoplot(hrvRRtakens4) #dim=4
```

See Figure 60 on page 84.

Input

```
hrvRRtakens4 <- local.buildTakens( time.series=hrv.data$Beat$RR[1:nsignal],
        embedding.dim=4,time.lag=1)
statepairs(hrvRRtakens4)
```

See Figure 61 on page 84.

Input

```
hrvRRneighs4  <- local.findAllNeighbours(hrvRRtakens4, radius=16)
save(hrvRRneighs4, file="hrvRRneighs4.Rdata")
```

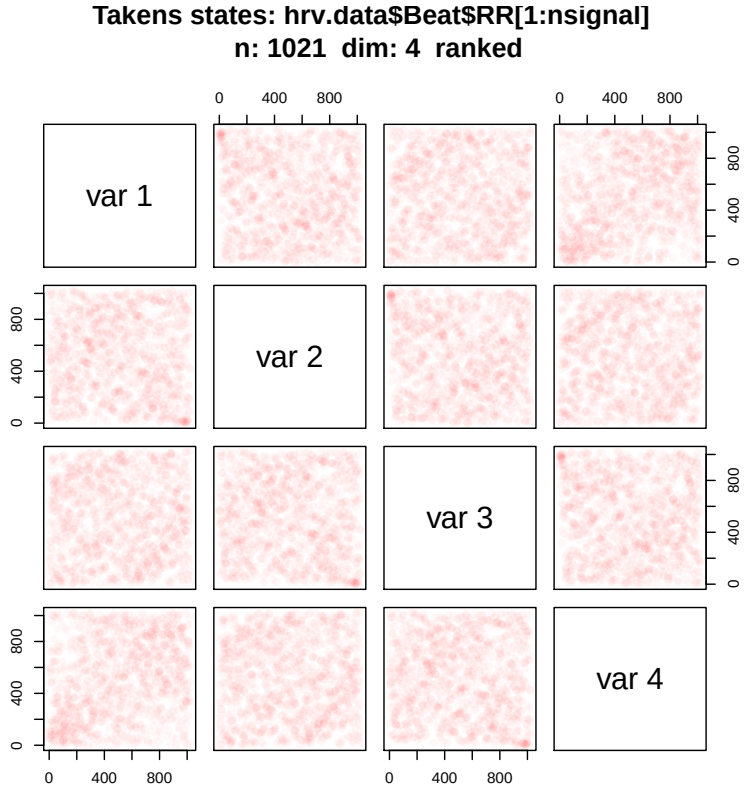


FIGURE 59. RHRV tutorial example.beats. Ranked data. Time used: 1.818 sec.

Time used: 0.91 sec.

Input

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

See Figure 62 on page 85.

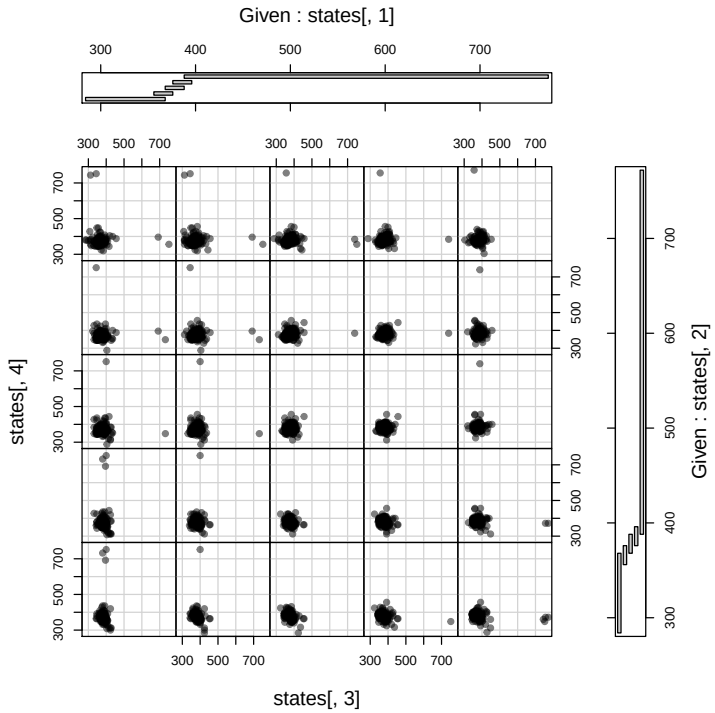


FIGURE 60. State coplot. RHRV tutorial example.beats. Time used: 0.218 sec.

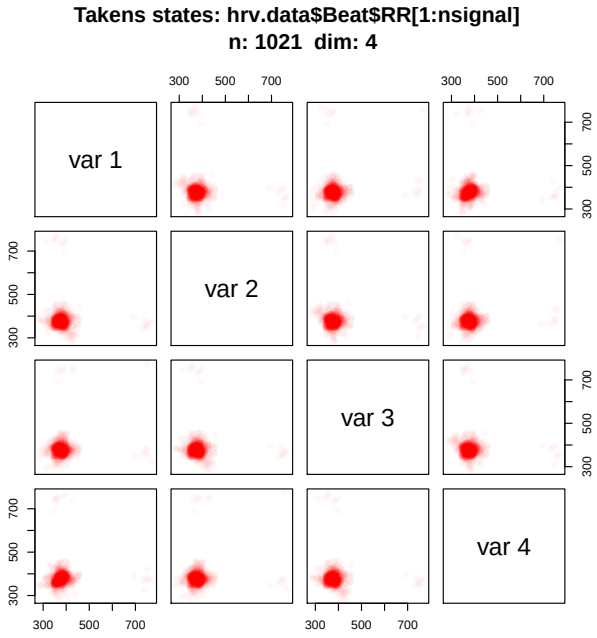


FIGURE 61. Recurrence plot. RHRV tutorial example.beats. Time used: 0.858 sec.

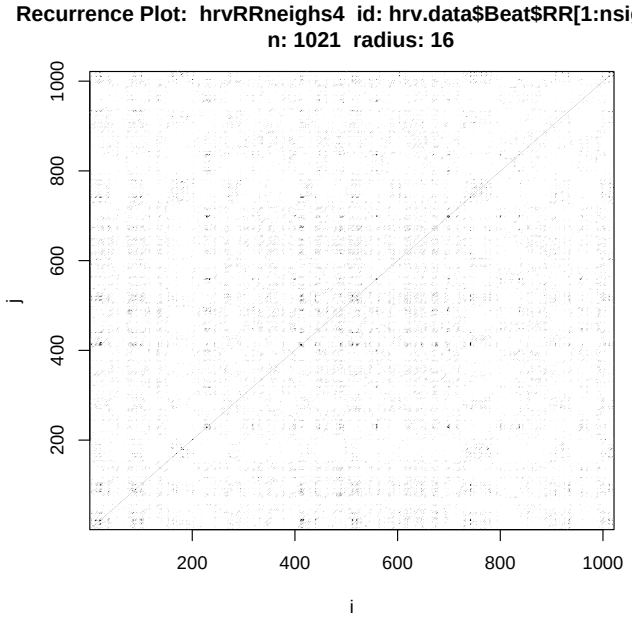


FIGURE 62. Recurrence plot. Example: RHRV tutorial example.beats. Dim=4.. Time used: 1.199 sec.

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

9.0.1. *RHRV: example.beats - 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")
# load(file="hrvRRneighs2.RData")
local.recurrencePlotAux(hrvRRneighs2)
showrqa(hrvRRtakens2, do.hist=FALSE, radius=16)
```

Output

```
hrv.data$Beat$RR[1:nsignal]  n: 1023  Dim: 2
Radius: 16  Recurrence coverage REC: 0.165  log(REC)/log(R): -0.65
Determinism: 0.651  Laminarity: 0.376
DIV: 0.056
Trend: 0  Entropy: 1.248
Diagonal lines max: 18  Mean: 2.841  Mean off main: 2.816
Vertical lines max: 14  Mean: 2.463
```

See Figure 63.

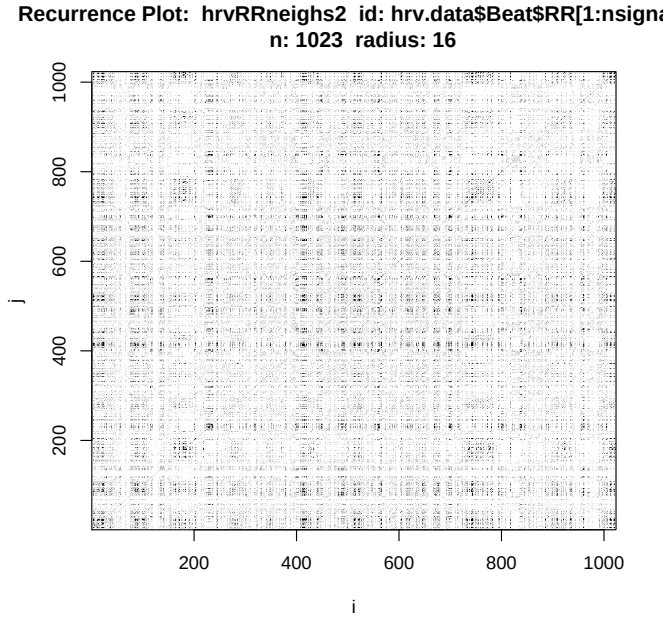


FIGURE 63. Recurrence plot. Dim=2.. Time used: 2.458 sec.

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

Dim=6. Time used: 0.949 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")
# load(file="hrvRRneighs8.RData")
local.recurrencePlotAux(hrvRRneighs8)
```

Dim=8. Time used: 1.44 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")
# load(file="hrvRRneighs12.RData")
local.recurrencePlotAux(hrvRRneighs12)
```

Dim=12. Time used: 3.078 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")
# load(file="hrvRRneighs16.RData")
local.recurrencePlotAux(hrvRRneighs16)
```

Dim=16. Time used: 1.505 sec.

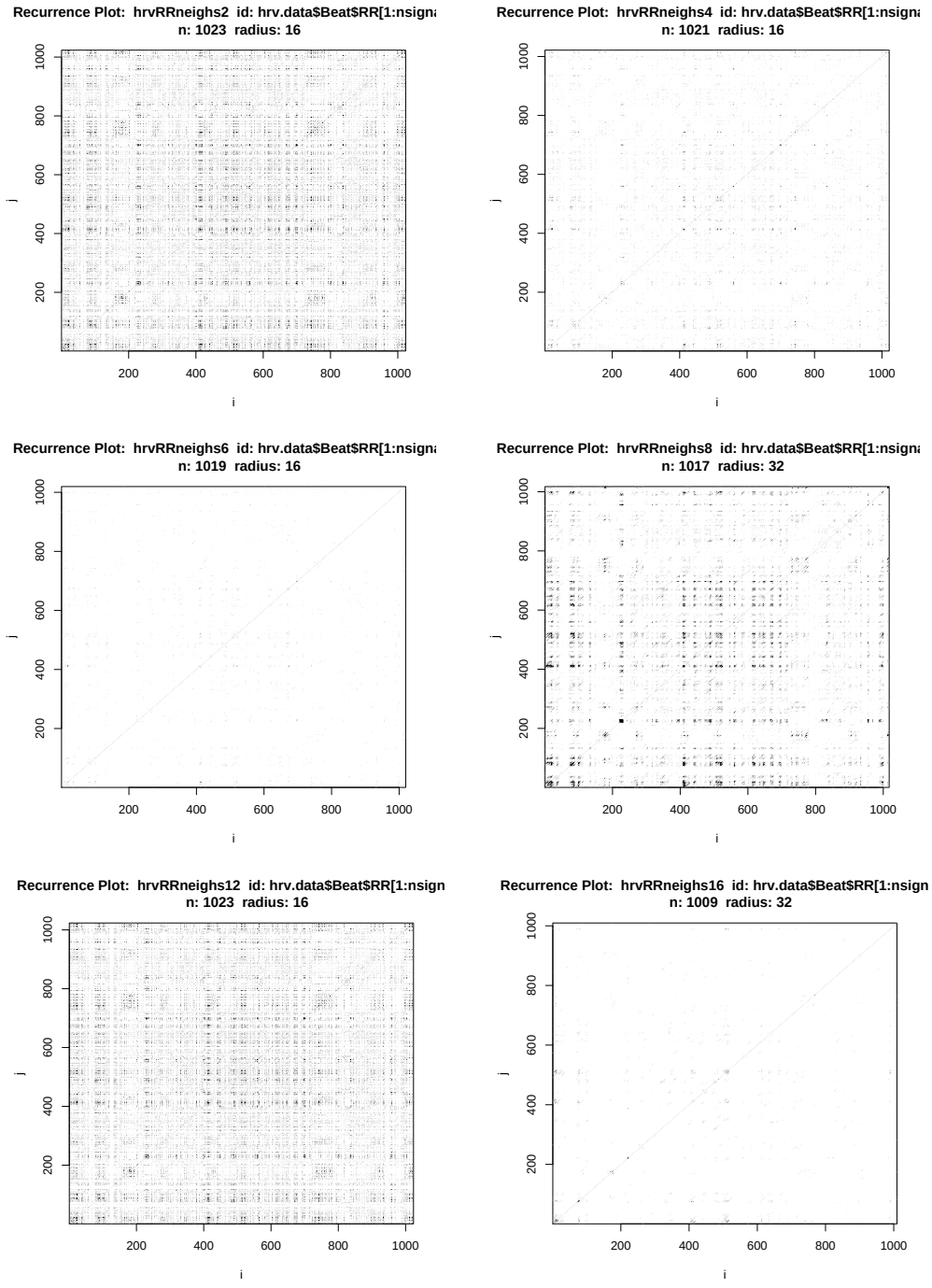


FIGURE 64. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 1.505 sec.

ToDo: This is an experimental proposal

9.1. **RHRV: example.beats - 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. The scaled difference is recorded as variation HRRV
#------------------------------------------------------
        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
        # end addition gs
        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:

Input
```
plotsignal(HRRV)
```

See Figure 65,

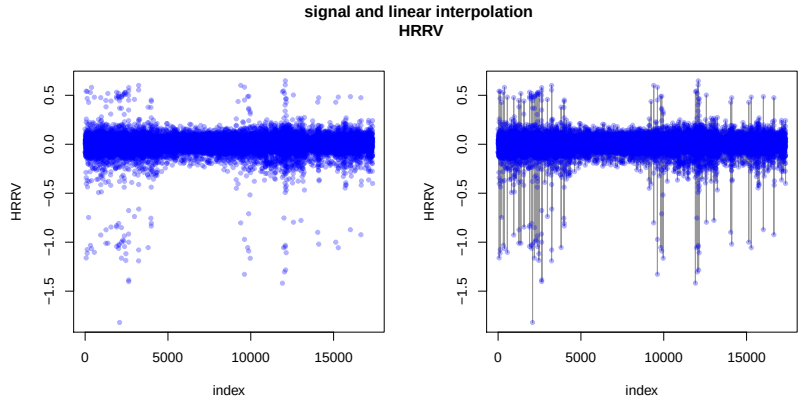


FIGURE 65. RHRV tutorial example.beats. HRRV Signal and linear interpolation.

Only 1024 data points used in this section

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

See Figure 66 on the facing page

Input
```
statepairs(hrvRRVtakens4, rank=TRUE) #dim=4
```

ToDo: findAllNeighbours does not handle NAs

See Figure 67 on page 92

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

Time used: 0.185 sec.

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

ToDo: check. There seem to be strange artefacts.

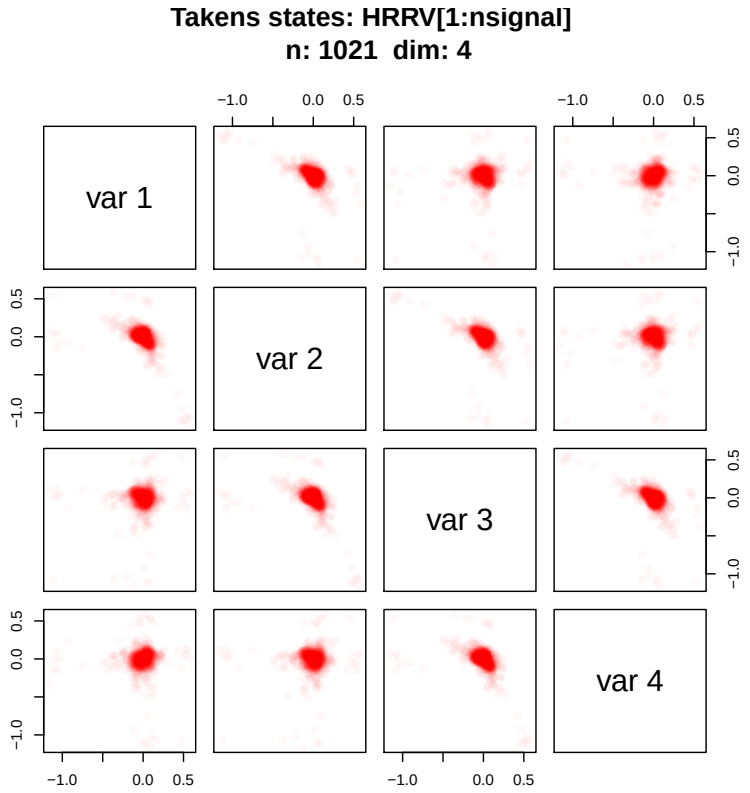


FIGURE 66. RHRV tutorial example.beats. HRRV Time used: 1.307 sec.

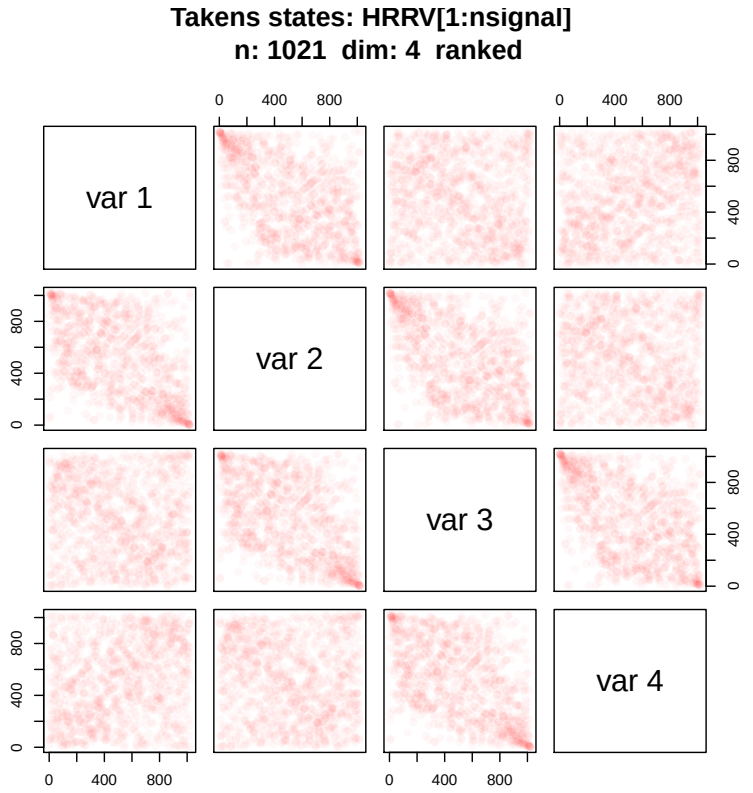


FIGURE 67. RHRV tutorial example.beats. Ranked HRRV data. Time used: 2.633 sec.

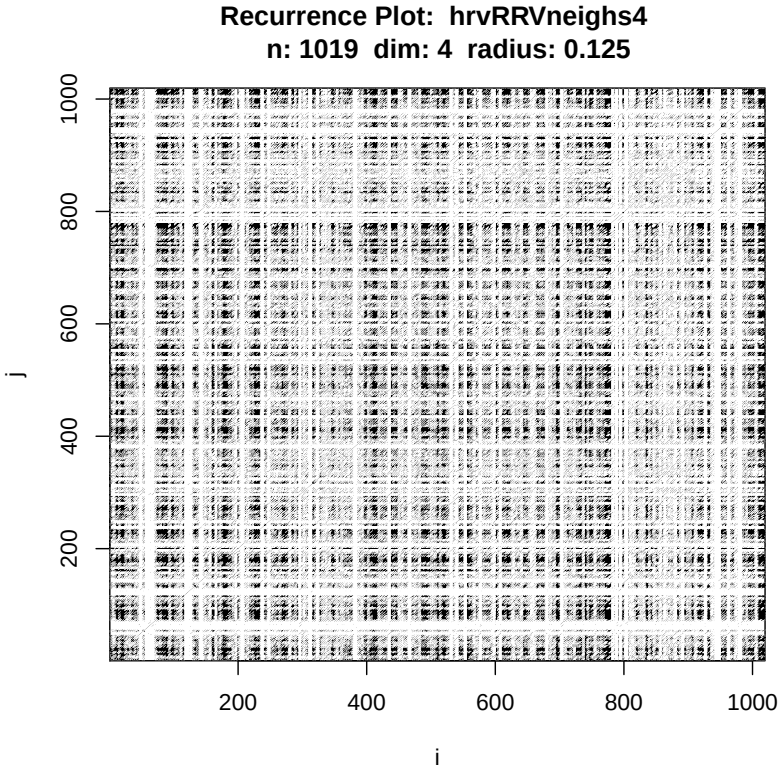


FIGURE 68. Recurrence Plot. Example case: RHRV tutorial example.beats. HRRV Dim=4. Time used: 1.102 sec.

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. Eckmann uses nearest neighbours with NN=10

9.1.1. *RHRV: example.beats - RR Variation: Comparison by Dimension.*

Input

```
hrvRRVtakens2 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=2, time.lag=1)
hrvRRVneighs2  <- local.findAllNeighbours(hrvRRVtakens2[-(1:2),],
        radius=0.125)
save(hrvRRVneighs2, file="hrvRRVneighs2.Rdata")
# load(file="hrvRRVneighs2.RData")
local.recurrencePlotAux(hrvRRVneighs2, dim=2,  radius=0.125)
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  log(REC)/log(R): 0.319
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
```

Dim=2. Time used: 4.523 sec.

Input

```
hrvRRVtakens6 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=6,time.lag=1)
hrvRRVneighs6  <- local.findAllNeighbours(hrvRRVtakens6[-(1:2),], radius=0.125)
save(hrvRRVneighs6, file="hrvRRVneighs6.Rdata")
# load(file="hrvRRVneighs6.RData")
local.recurrencePlotAux(hrvRRVneighs6, dim=6, radius=0.125)
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  log(REC)/log(R): 0.827
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
```

Dim=6. Time used: 3.081 sec.

Input

```
hrvRRVtakens8 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=8,time.lag=1)
hrvRRVneighs8  <- local.findAllNeighbours(hrvRRVtakens8[-(1:2),], radius=0.125)
save(hrvRRVneighs8, file="hrvRRVneighs8.Rdata")
# load(file="hrvRRVneighs8.RData")
local.recurrencePlotAux(hrvRRVneighs8, dim=8, radius=0.125)
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  log(REC)/log(R): 1.084
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
```

Dim=8. Time used: 2.163 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")
# load(file="hrvRRVneighs12.RData")
local.recurrencePlotAux(hrvRRVneighs12, dim=12,  radius=3/16)
showrqa(hrvRRVtakens12[-(1:2),], radius=3/16, do.hist=FALSE)
```

Output

```
hrvRRVtakens12[-(1:2), ]  n: 1011  Dim: 12
Radius: 0.1875  Recurrence coverage REC: 0.235  log(REC)/log(R): 0.865
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
```

Dim=12: Time used: 3.762 sec.

Input

```
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")
# load(file="hrvRRVneighs16.RData")
local.recurrencePlotAux(hrvRRVneighs16, dim=16, radius=3/16)
showrqa(hrvRRVtakens16[-(1:2),], radius=3/16, do.hist=FALSE)
```

Output

```
hrvRRVtakens16[-(1:2), ]  n: 1007  Dim: 16
Radius: 0.1875  Recurrence coverage REC: 0.147  log(REC)/log(R): 1.144
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
```

Dim=16. Time used: 2.963 sec.

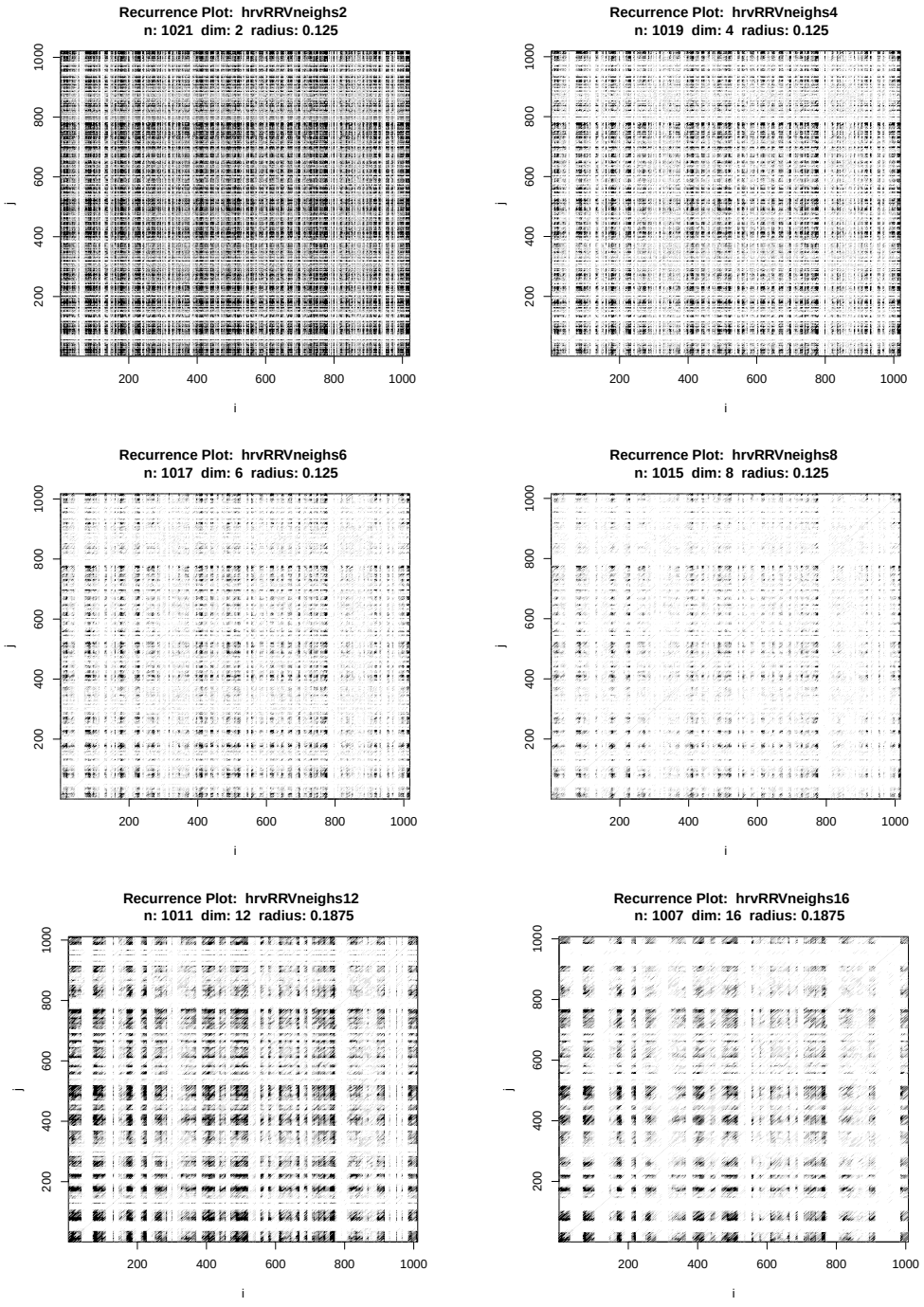


FIGURE 69. Recurrence Plot. Example case: RHRV tutorial example.beats variation. Dim=2, 4, 6, 8, 12, 16. Time used: 2.969 sec.

9.1.2. *RHRV: example.beats - RR Variation: Comparison by Dimension, Time.Lag = 8.*

Input

```
hrvRRVtakens2Lag08 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=2, time.lag=8)
hrvRRVneighs2Lag08  <- local.findAllNeighbours(hrvRRVtakens2Lag08[-(1:2),],
        radius=0.125)
save(hrvRRVneighs2Lag08, file="hrvRRVneighs2Lag08.Rdata")
# load(file="hrvRRVneighs2Lag08.RData")
local.recurrencePlotAux(hrvRRVneighs2Lag08, dim=2,  radius=0.125)
showrqa(hrvRRVtakens2Lag08[-(1:2),], radius=0.125, do.hist=FALSE)
```

Output

```
hrvRRVtakens2Lag08[-(1:2), ]  n: 1014  Dim: 2
Radius: 0.125  Recurrence coverage REC: 0.469  log(REC)/log(R): 0.364
Determinism: 0.809  Laminarity: 0.792
DIV: 0.033
Trend: 0  Entropy: 1.637
Diagonal lines max: 30  Mean: 3.451  Mean off main: 3.442
Vertical lines max: 41  Mean: 3.445
```

Dim=2. Time used: 4.104 sec.

Input

```
hrvRRVtakens4Lag08 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=4, time.lag=8)
hrvRRVneighs4Lag08  <- local.findAllNeighbours(hrvRRVtakens4Lag08[-(1:2),],
        radius=0.125)
save(hrvRRVneighs4Lag08, file="hrvRRVneighs4Lag08.Rdata")
# load(file="hrvRRVneighs4Lag08.RData")
local.recurrencePlotAux(hrvRRVneighs4Lag08, dim=2,  radius=0.125)
showrqa(hrvRRVtakens2Lag08[-(1:2),], radius=0.125, do.hist=FALSE)
```

Output

```
hrvRRVtakens2Lag08[-(1:2), ]  n: 1014  Dim: 2
Radius: 0.125  Recurrence coverage REC: 0.469  log(REC)/log(R): 0.364
Determinism: 0.809  Laminarity: 0.792
DIV: 0.033
Trend: 0  Entropy: 1.637
Diagonal lines max: 30  Mean: 3.451  Mean off main: 3.442
Vertical lines max: 41  Mean: 3.445
```

Dim=4. Time used: 2.999 sec.

Input

```
statepairs(hrvRRVtakens4Lag08)
```

Input

```
statecoplot(hrvRRVtakens4Lag08)
```

Output

```
Missing rows: 1, 2
```

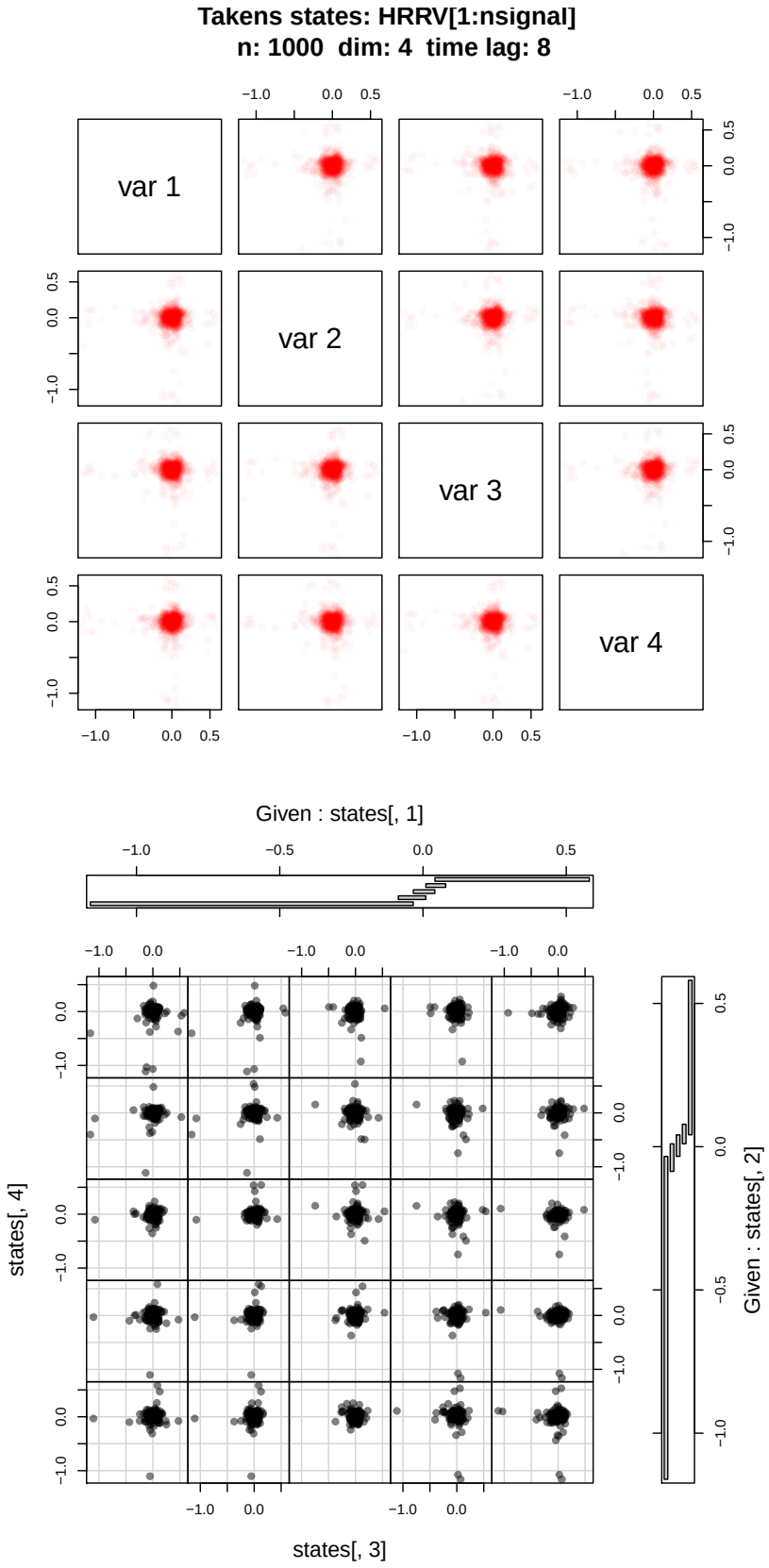


FIGURE 70. Takens states. Example case: RHRV tutorial example.beats variation. Dim=4, time.lag=8. Time used: 4.523 sec.

Input

```
statepairs(hrvRRVtakens4Lag08, range=c(-0.5, 0.5))
```

Input

```
statecoplot(hrvRRVtakens4Lag08, range=c(-0.5, 0.5))
```

Output

```
Missing rows: 1, 2, 38, 39, 46, 47, 54, 55, 62, 63, 100, 101, 102, 108, 109, 110, 116, 117, 118, 124, 125
```

Input

```
hrvRRVtakens6Lag08 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=6,time.lag=8)
hrvRRVneighs6Lag08  <- local.findAllNeighbours(hrvRRVtakens6Lag08[-(1:2),], radius=0.125)
save(hrvRRVneighs6Lag08, file="hrvRRVneighs6Lag08.Rdata")
# load(file="hrvRRVneighs6Lag08.RData")
local.recurrencePlotAux(hrvRRVneighs6Lag08, dim=6, radius=0.125)
showrqa(hrvRRVtakens6Lag08[-(1:2),], radius=0.125, do.hist=FALSE)
```

Output

```
hrvRRVtakens6Lag08[-(1:2), ]  n: 982  Dim: 6
Radius: 0.125  Recurrence coverage REC: 0.1  log(REC)/log(R): 1.107
Determinism: 0.357  Laminarity: 0.311
DIV: 0.143
Trend: 0  Entropy: 0.604
Diagonal lines max: 7  Mean: 2.302  Mean off main: 2.237
Vertical lines max: 8  Mean: 2.234
```

Dim=6. Time used: 8.232 sec.

Input

```
hrvRRVtakens8Lag08 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=8,time.lag=8)
hrvRRVneighs8Lag08  <- local.findAllNeighbours(hrvRRVtakens8Lag08[-(1:2),], radius=0.125)
save(hrvRRVneighs8Lag08, file="hrvRRVneighs8Lag08.Rdata")
# load(file="hrvRRVneighs8Lag08.RData")
local.recurrencePlotAux(hrvRRVneighs8Lag08, dim=8, radius=0.125)
showrqa(hrvRRVtakens8Lag08[-(1:2),], radius=0.125, do.hist=FALSE)
```

Output

```
hrvRRVtakens8Lag08[-(1:2), ]  n: 966  Dim: 8
Radius: 0.125  Recurrence coverage REC: 0.045  log(REC)/log(R): 1.488
Determinism: 0.223  Laminarity: 0.161
DIV: 0.2
Trend: 0  Entropy: 0.354
Diagonal lines max: 5  Mean: 2.348  Mean off main: 2.108
Vertical lines max: 5  Mean: 2.089
```

Dim=8. Time used: 1.862 sec.

Input

```
hrvRRVtakens12Lag08 <-
        local.buildTakens( time.series=HRRV[1:nsignal],
                embedding.dim=12,time.lag=8)
hrvRRVneighs12Lag08 <-
```

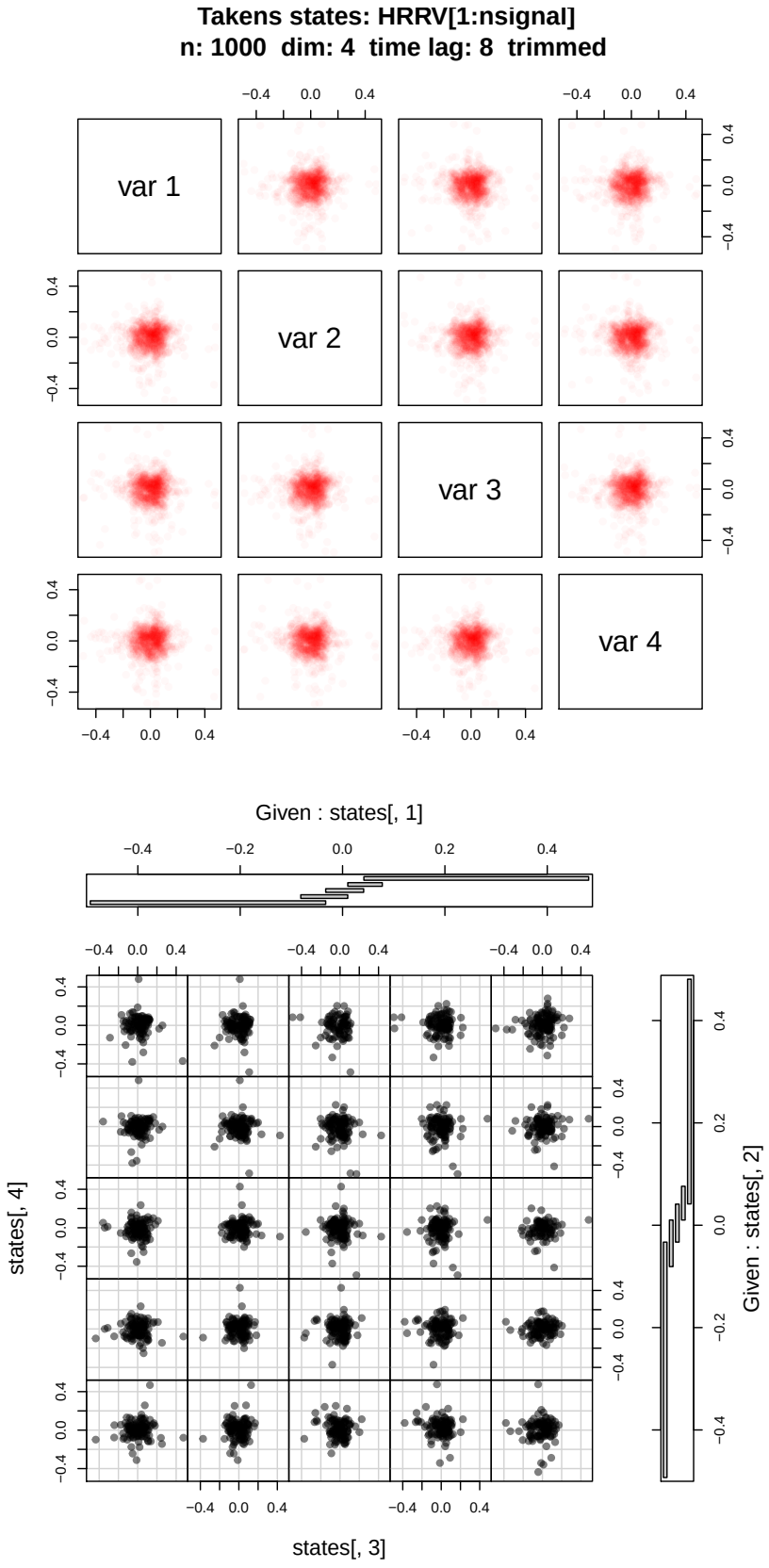


FIGURE 71. Takens states. Example case: RHRV tutorial example.beats variation, trimmed. Dim=4, time.lag=8. Time used: 6.002 sec.

```
        local.findAllNeighbours(hrvRRVtakens12Lag08[-(1:2),], radius=3/16)
save(hrvRRVneighs12Lag08, file="hrvRRVneighs12Lag08.Rdata")
# load(file="hrvRRVneighs12Lag08.RData")
local.recurrencePlotAux(hrvRRVneighs12Lag08, dim=12,  radius=3/16)
showrqa(hrvRRVtakens12Lag08[-(1:2),], radius=3/16, do.hist=FALSE)
```

Output

```
hrvRRVtakens12Lag08[-(1:2), ]  n: 934  Dim: 12
Radius: 0.1875  Recurrence coverage REC: 0.133  log(REC)/log(R): 1.207
Determinism: 0.472  Laminarity: 0.443
DIV: 0.143
Trend: 0  Entropy: 0.77
Diagonal lines max: 7  Mean: 2.39  Mean off main: 2.349
Vertical lines max: 7  Mean: 2.457
```

Dim=12: Time used: 2.599 sec.

Input

```
hrvRRVtakens16Lag08 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=16,time.lag=8)
hrvRRVneighs16Lag08  <- local.findAllNeighbours(hrvRRVtakens16Lag08[-(1:2),], radius=3/16)
save(hrvRRVneighs16Lag08, file="hrvRRVneighs16Lag08.Rdata")
# load(file="hrvRRVneighs16Lag08.RData")
local.recurrencePlotAux(hrvRRVneighs16Lag08, dim=16, radius=3/16)
showrqa(hrvRRVtakens16Lag08[-(1:2),], radius=3/16, do.hist=FALSE)
```

Output

```
hrvRRVtakens16Lag08[-(1:2), ]  n: 902  Dim: 16
Radius: 0.1875  Recurrence coverage REC: 0.071  log(REC)/log(R): 1.581
Determinism: 0.339  Laminarity: 0.31
DIV: 0.167
Trend: 0  Entropy: 0.608
Diagonal lines max: 6  Mean: 2.347  Mean off main: 2.239
Vertical lines max: 6  Mean: 2.321
```

Dim=16. Time used: 2.504 sec.

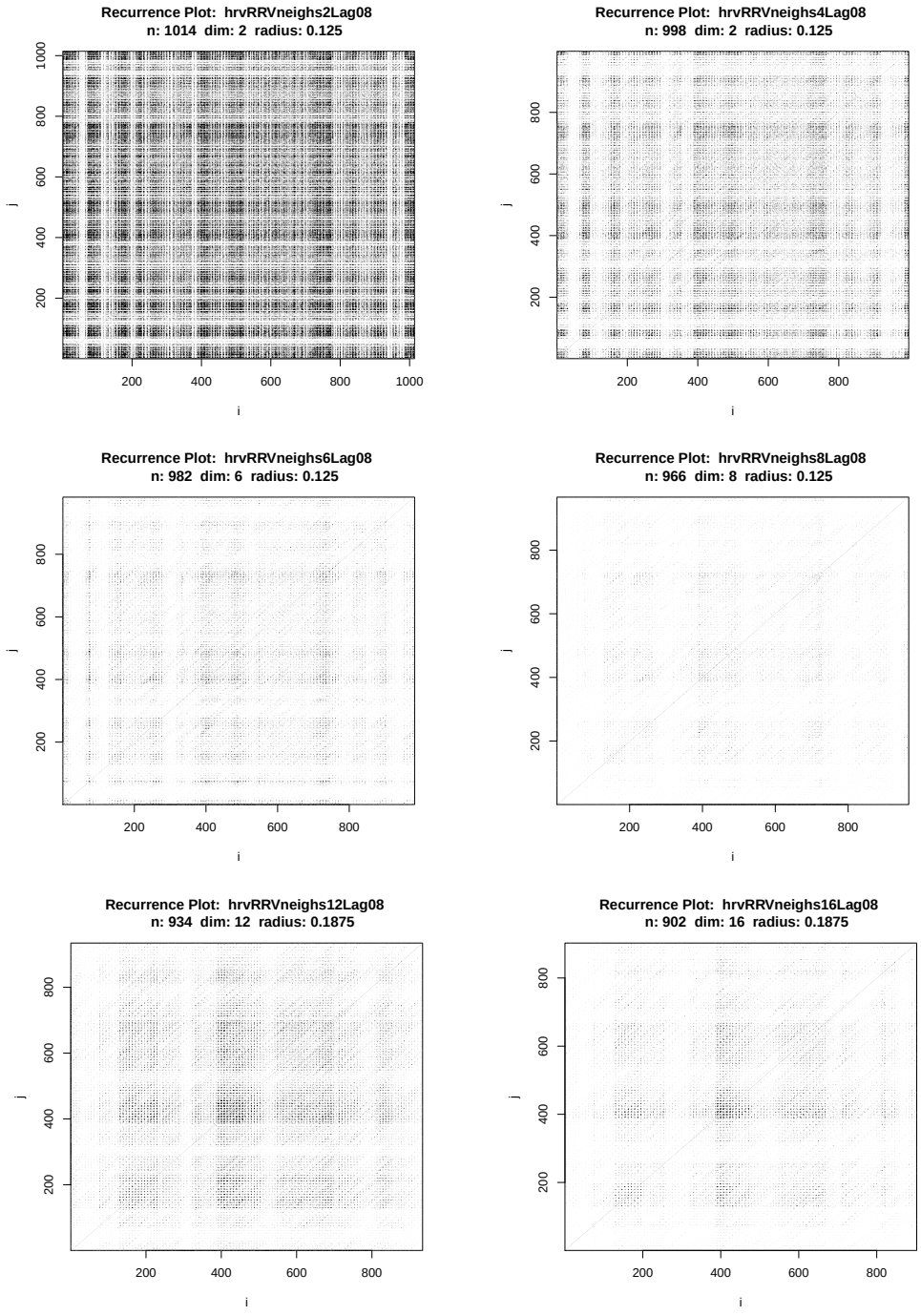


FIGURE 72. Recurrence Plot. Example case: RHRV tutorial example.beats variation. Dim=2, 4, 6, 8, 12, 16. Time.lag=8. Time used: 2.504 sec.

10. Case Study: HRV data example2.beats

This is a copy of the previous section, now applied to HRV data example2.beats.

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
#####################################################
hrv2.data  = CreateHRVData()
hrv2.data = SetVerbose(hrv2.data, TRUE )
#####################################################
### code chunk number 3: loading
#####################################################
hrv2.data = LoadBeatAscii(hrv2.data, "example2.beats",
       RecordPath = "/users/gs/projects/rforge/rhrv/tutorial/beatsFolder")
#       RecordPath = "beatsFolder")


#####################################################
### code chunk number 4: derivating
#####################################################
hrv2.data = BuildNIHR(hrv2.data)
```

Input

```
plotsignal(hrv2.data$Beat$RR)
```

See Figure 73.

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

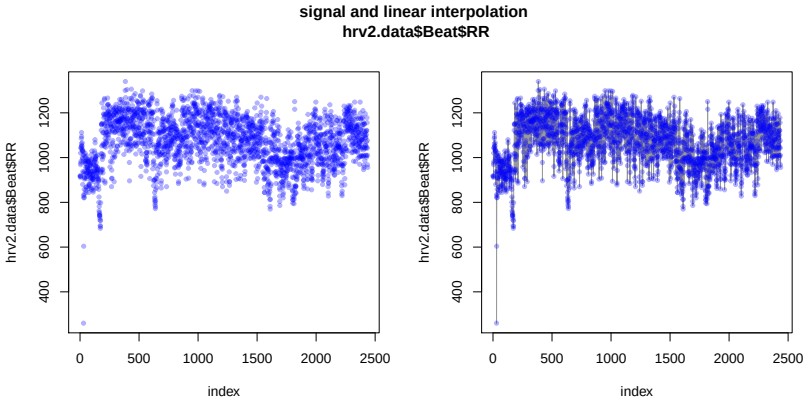


Figure 73. RHRV tutorial example2.beats. Signal and linear interpolation.

Input

```
hrv2RRtakens4 <- local.buildTakens( time.series=hrv2.data$Beat$RR[1:nsignal],
        embedding.dim=4, time.lag=1)
statepairs(hrv2RRtakens4) #dim=4
```

See Figure 74 on the next page.

Only 1024 data points used in this plot

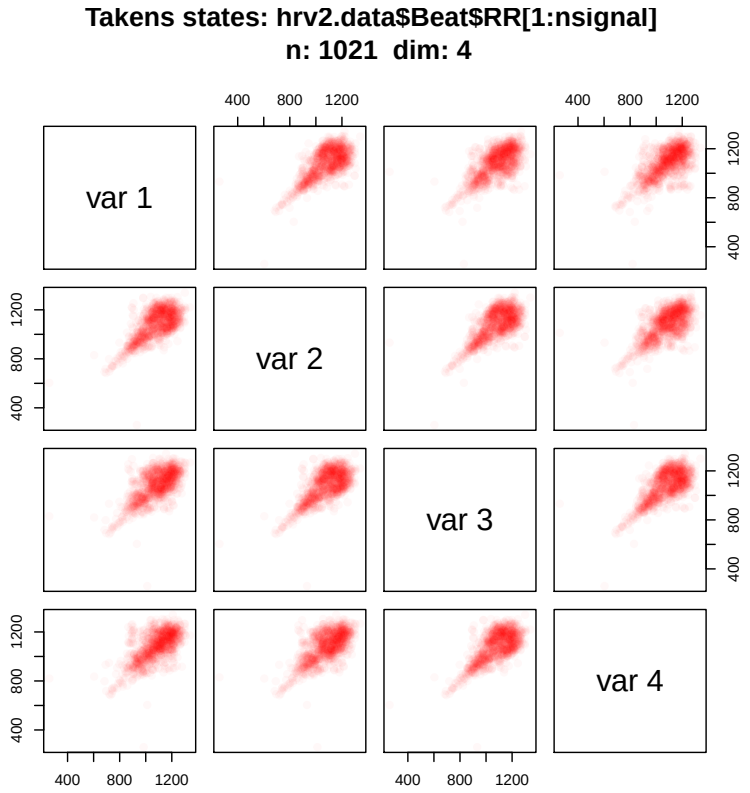


FIGURE 74. RHRV tutorial example2.beats. Time used: 1.29 sec.

Input

```
statepairs(hrv2RRtakens4, rank=TRUE) #dim=4
```

See Figure 75 on the facing page.

Input

```
statecoplot(hrv2RRtakens4) #dim=4
```

See Figure 76 on page 106.

Input

```
hrv2RRneighs4  <- local.findAllNeighbours(hrv2RRtakens4, radius=12*16)
save(hrv2RRneighs4, file="hrv2RRneighs4.Rdata")
# load(file="hrv2RRneighs4.RData")
local.recurrencePlotAux(hrv2RRneighs4, radius=12*16)
showrqa(hrv2RRtakens4[-(1:2),], radius=12*16, do.hist=FALSE)
```

Output

```
hrv2RRtakens4[-(1:2), ]  n: 1019  Dim: 4
Radius: 192  Recurrence coverage REC: 0.493  log(REC)/log(R): -0.135
Determinism: 0.982  Laminarity: 0.906
DIV: 0.005
Trend: 0  Entropy: 3.158
Diagonal lines max: 191  Mean: 10.396  Mean off main: 10.375
Vertical lines max: 136  Mean: 7.145
```

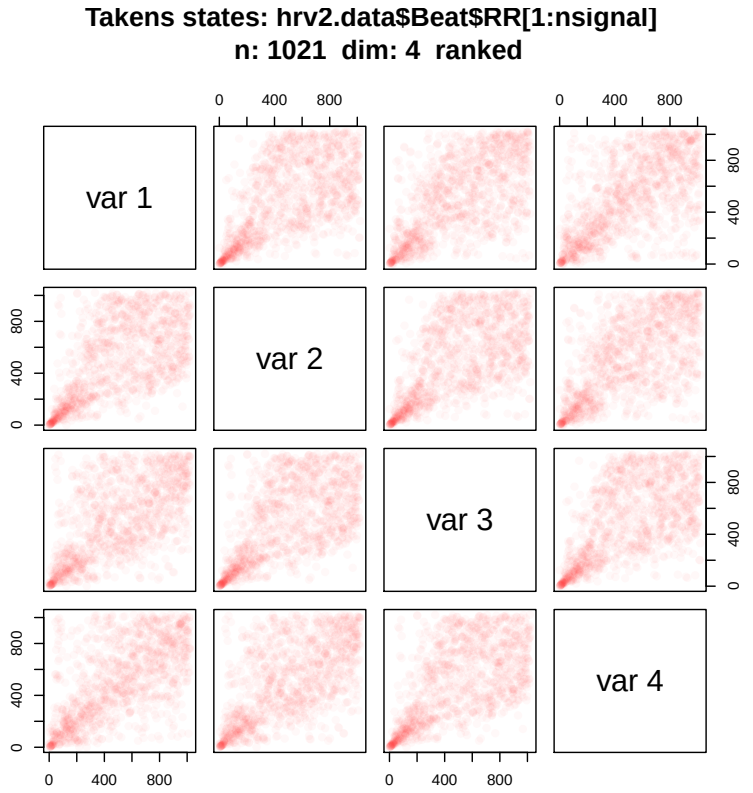


FIGURE 75. RHRV tutorial example2.beats. Ranked data. Time used: 2.662 sec.

Dim=4. Time used: 3.813 sec.

See Figure 77 on page 107.

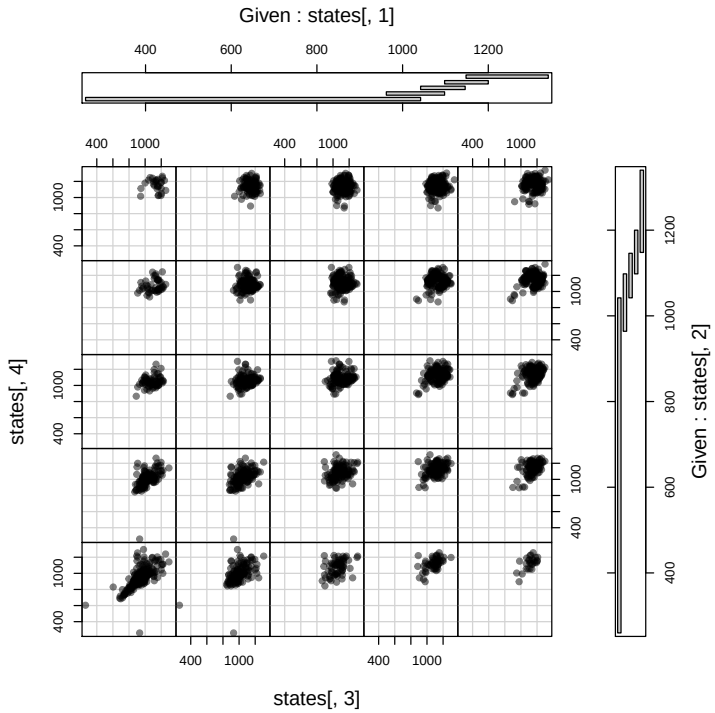


FIGURE 76. State coplot. RHRV tutorial example2.beats. Time used: 0.226 sec.

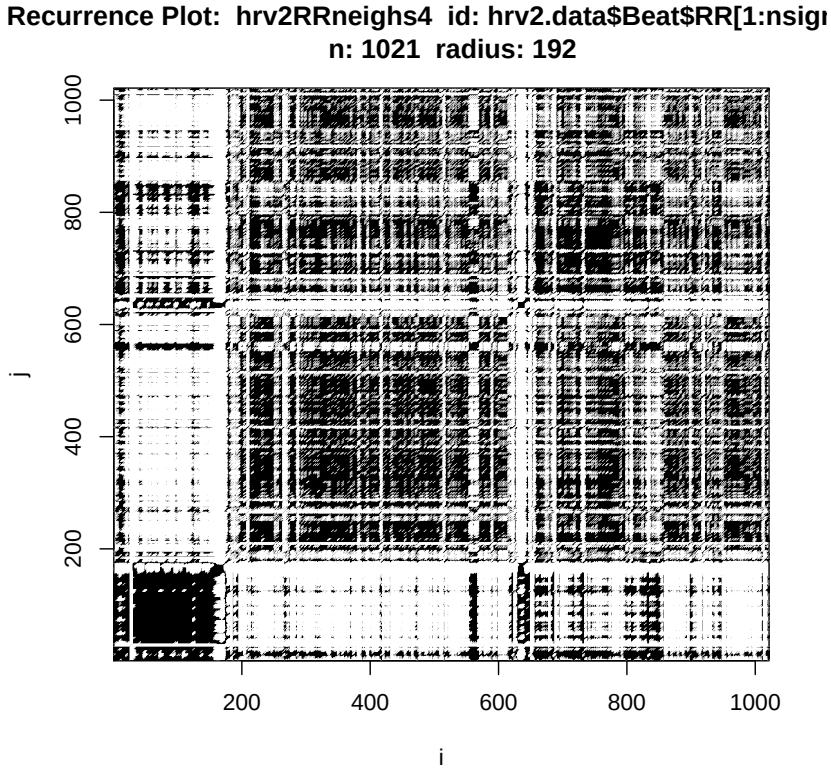


FIGURE 77. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=4. Time used: 3.813 sec.

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

10.0.1. *RHRV: example2.beats, RR-intervals. Comparison by Dimension.*

Input

```
hrv2RRtakens2 <- local.buildTakens( time.series=hrv2.data$Beat$RR[1:nsignal],
        embedding.dim=2,time.lag=1)
hrv2RRneighs2  <- local.findAllNeighbours(hrv2RRtakens2, radius=10*16)
save(hrv2RRneighs2, file="hrv2RRneighs2.Rdata")
# load(file="hrv2RRneighs2.RData")
local.recurrencePlotAux(hrv2RRneighs2, radius=10*16)
showrqa(hrv2RRtakens2[-(1:2),], radius=10*16, do.hist=FALSE)
```

Output

```
hrv2RRtakens2[-(1:2), ]  n: 1021  Dim: 2
Radius: 160  Recurrence coverage REC: 0.52  log(REC)/log(R): -0.129
Determinism: 0.948  Laminarity: 0.91
DIV: 0.007
Trend: 0  Entropy: 2.691
Diagonal lines max: 152  Mean: 7.057  Mean off main: 7.043
Vertical lines max: 130  Mean: 6.607
```

Dim=2. Time used: 3.773 sec.

See Figure 78 on page 110.

Input

```
hrv2RRtakens6 <- local.buildTakens( time.series=hrv2.data$Beat$RR[1:nsignal],
        embedding.dim=6,time.lag=1)
hrv2RRneighs6  <- local.findAllNeighbours(hrv2RRtakens6, radius=14*16)
save(hrv2RRneighs6, file="hrv2RRneighs6.Rdata")
# load(file="hrv2RRneighs6.RData")
local.recurrencePlotAux(hrv2RRneighs6, radius=14*16)
showrqa(hrv2RRtakens6[-(1:2),], radius=14*16, do.hist=FALSE)
```

Output

```
hrv2RRtakens6[-(1:2), ]  n: 1017  Dim: 6
Radius: 224  Recurrence coverage REC: 0.528  log(REC)/log(R): -0.118
Determinism: 0.993  Laminarity: 0.925
DIV: 0.004
Trend: 0  Entropy: 3.584
Diagonal lines max: 225  Mean: 15.258  Mean off main: 15.23
Vertical lines max: 167  Mean: 9.007
```

Dim=6. Time used: 4.131 sec.

See Figure 78 on page 110.

Input

```
hrv2RRtakens8 <- local.buildTakens( time.series=hrv2.data$Beat$RR[1:nsignal],
        embedding.dim=8,time.lag=1)
hrv2RRneighs8  <- local.findAllNeighbours(hrv2RRtakens8, radius=16*16)
save(hrv2RRneighs8, file="hrv2RRneighs8.Rdata")
# load(file="hrv2RRneighs8.RData")
local.recurrencePlotAux(hrv2RRneighs8, radius=16*16)
showrqa(hrv2RRtakens8[-(1:2),], radius=16*16, do.hist=FALSE)
```

Output

```
hrv2RRtakens8[-(1:2), ]  n: 1015  Dim: 8
Radius: 256  Recurrence coverage REC: 0.584  log(REC)/log(R): -0.097
Determinism: 0.997  Laminarity: 0.942
DIV: 0.004
Trend: 0  Entropy: 3.986
Diagonal lines max: 241  Mean: 22.506  Mean off main: 22.469
Vertical lines max: 310  Mean: 11.256
```

Dim=8. Time used: 3.846 sec.

See Figure 78 on the next page.

Input

```
hrv2RRtakens12 <- local.buildTakens( time.series=hrv2.data$Beat$RR[1:nsignal],
        embedding.dim=12,time.lag=1)
hrv2RRneighs12  <- local.findAllNeighbours(hrv2RRtakens12, radius=16*16)
save(hrv2RRneighs12, file="hrv2RRneighs12.Rdata")
# load(file="hrv2RRneighs12.RData")
local.recurrencePlotAux(hrv2RRneighs12, radius=16*16)
showrqa(hrv2RRtakens12[-(1:2),], radius=16*16, do.hist=FALSE)
```

Output

```
hrv2RRtakens12[-(1:2), ]  n: 1011  Dim: 12
Radius: 256  Recurrence coverage REC: 0.488  log(REC)/log(R): -0.129
Determinism: 0.997  Laminarity: 0.914
DIV: 0.004
Trend: 0  Entropy: 4.062
Diagonal lines max: 237  Mean: 24.127  Mean off main: 24.079
Vertical lines max: 299  Mean: 8.972
```

Dim=12. Time used: 5.098 sec.

Input

```
hrv2RRtakens16 <- local.buildTakens(
        time.series=hrv2.data$Beat$RR[1:nsignal],
        embedding.dim=16,time.lag=1)
hrv2RRneighs16  <- local.findAllNeighbours(hrv2RRtakens16, radius=18*16)
save(hrv2RRneighs16, file="hrv2RRneighs16.Rdata")
# load(file="hrv2RRneighs16.RData")
local.recurrencePlotAux(hrv2RRneighs16, radius=18*16)
showrqa(hrv2RRtakens16[-(1:2),], radius=18*16, do.hist=FALSE)
```

Output

```
hrv2RRtakens16[-(1:2), ]  n: 1007  Dim: 16
Radius: 288  Recurrence coverage REC: 0.544  log(REC)/log(R): -0.107
Determinism: 0.999  Laminarity: 0.932
DIV: 0.003
Trend: 0  Entropy: 4.372
Diagonal lines max: 346  Mean: 34.916  Mean off main: 34.854
Vertical lines max: 297  Mean: 9.929
```

Dim=16. Time used: 4.368 sec.

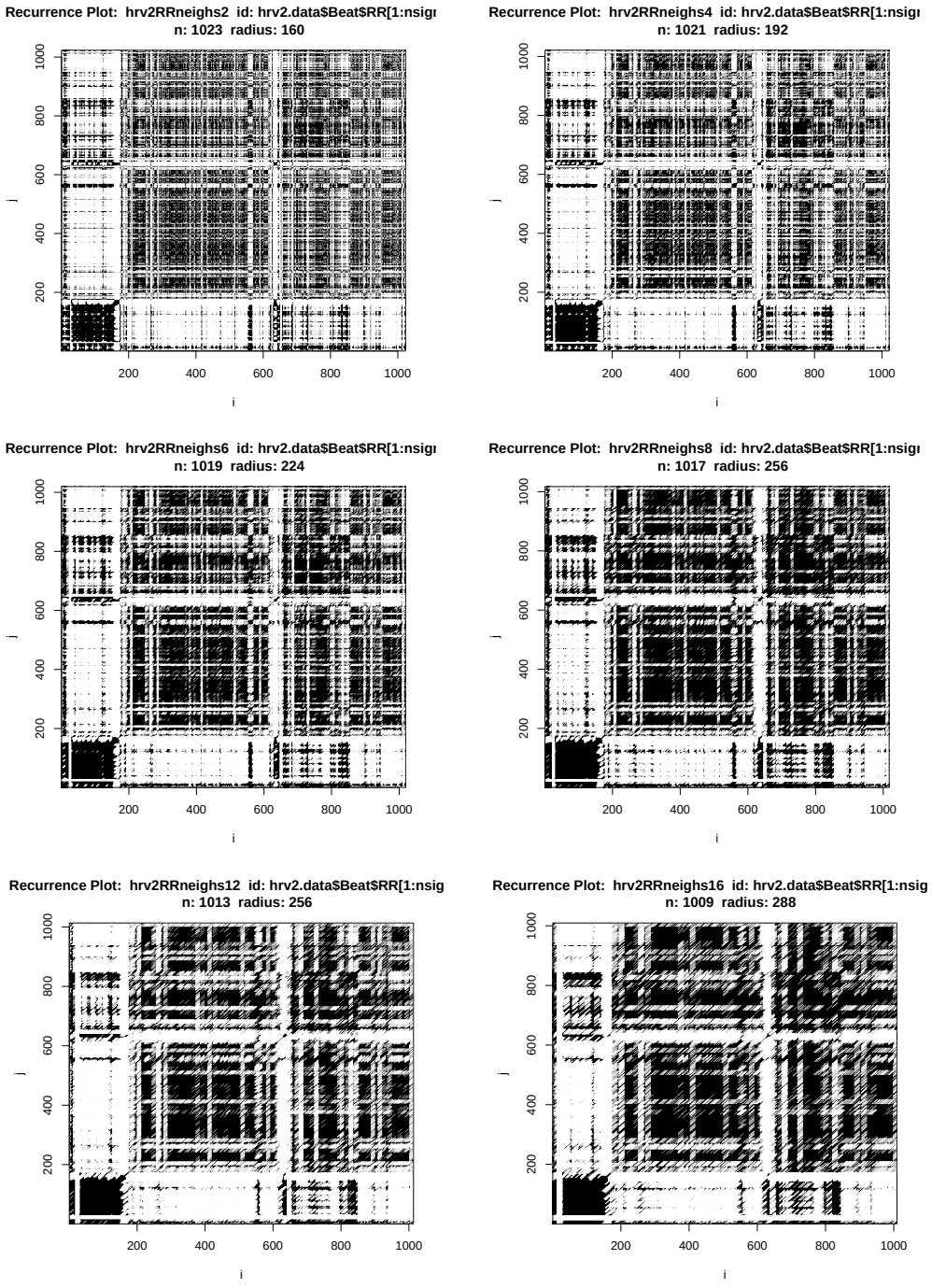


FIGURE 78. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 4.369 sec.

ToDo: Consider using differences

differences for HRV

10.1. **RHRV: example2.beats - 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

```
hrv2.data <- BuildNIDHR(hrv2.data)
```

Output

```
** Calculating non-interpolated heart rate differences **
   Number of beats: 2437
```

Input

```
HRRV <- hrv2.data$Beat$HRRV
```

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

Input

```
plotsignal(HRRV)
```

See Figure 79,

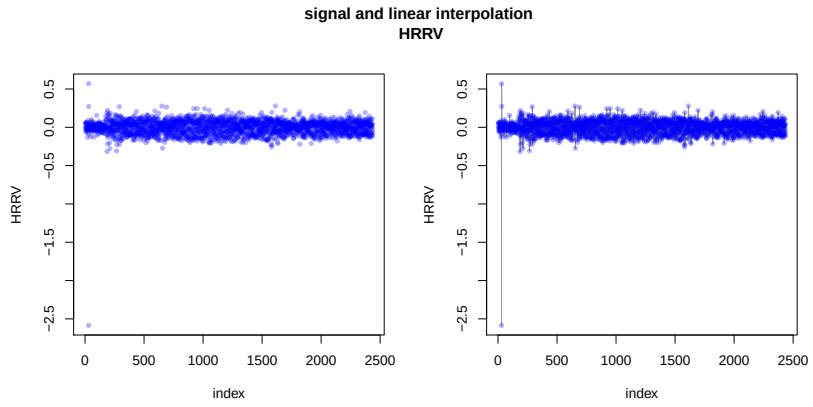


FIGURE 79. RHRV tutorial example2.beats. HRRV Signal and linear interpolation.

Only 1024 data points used in these plots

Input

```
hrv2RRVtakens4 <-
        local.buildTakens( time.series=HRRV[1:nsignal],
                embedding.dim=4,time.lag=1)
statepairs(hrv2RRVtakens4) #dim=4
```

See Figure 80 on the next page.

Input

```
        statecoplot(hrv2RRVtakens4) #dim=4
```

Output

```
Missing rows: 1, 2
```

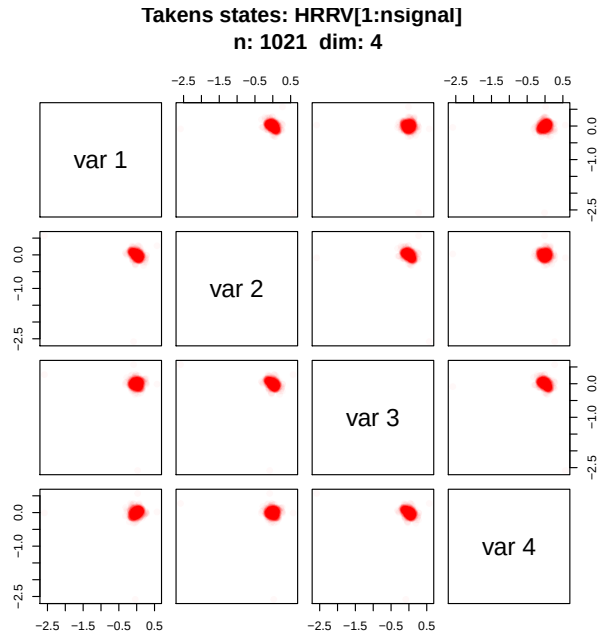


FIGURE 80. Recurrence plot. RHRV tutorial example2.beats. HRRV. Time used: 1.331 sec.

See Figure 81.

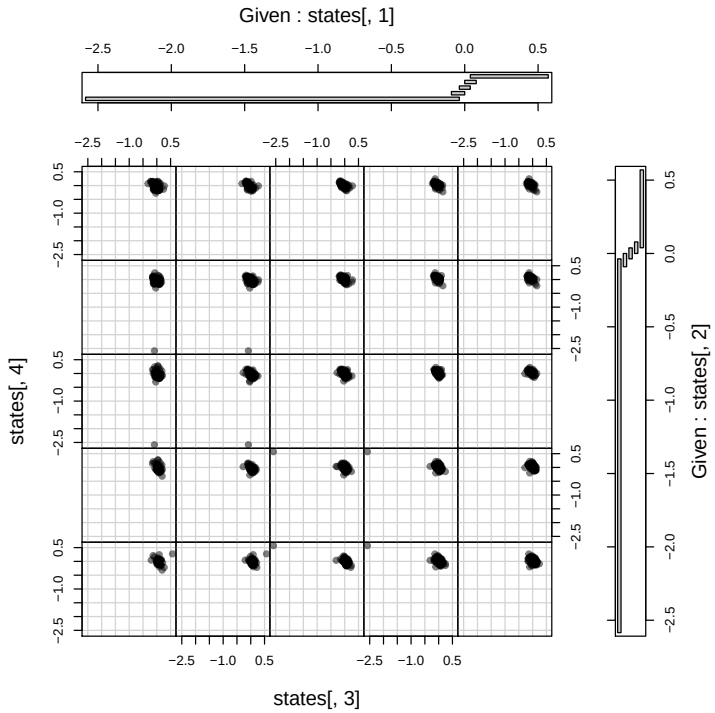


FIGURE 81. State coplot. RHRV tutorial example2.beats. HRRV. Time used: 2.224 sec.

```
statepairs(hrv2RRVtakens4, rank=TRUE) #dim=4
```

See Figure 82

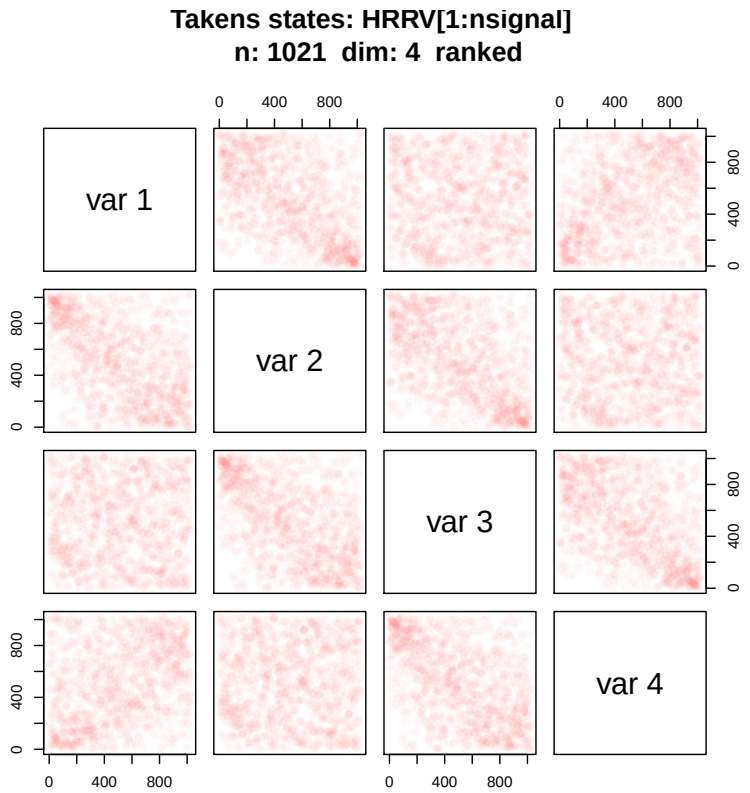


FIGURE 82. RHRV tutorial example2.beats. Ranked HRRV data. Time used: 3.524 sec.

Input

```
hrv2RRVtakens41 <- hrv2RRVtakens4
hrv2RRVtakens41[hrv2RRVtakens41 < -1.5] <- NA
hrv2RRVtakens41[hrv2RRVtakens41 > 0.45] <- NA
statepairs(hrv2RRVtakens41) #dim=4
```

See Figure 83 on the following page.

Input

```
statecoplot(hrv2RRVtakens41) #dim=4
```

Output

```
Missing rows: 1, 2, 27, 28, 29, 30, 31
```

See Figure 84 on the next page.

ToDo: findAllNeighbours does not handle NAs

Input

```
#use hack: findAllNeighbours does not handle NAs
hrv2RRVneighs4  <- local.findAllNeighbours(hrv2RRVtakens4[-(1:2),], radius=0.125)
save(hrv2RRVneighs4, file="hrv2RRVneighs4.Rdata")
```

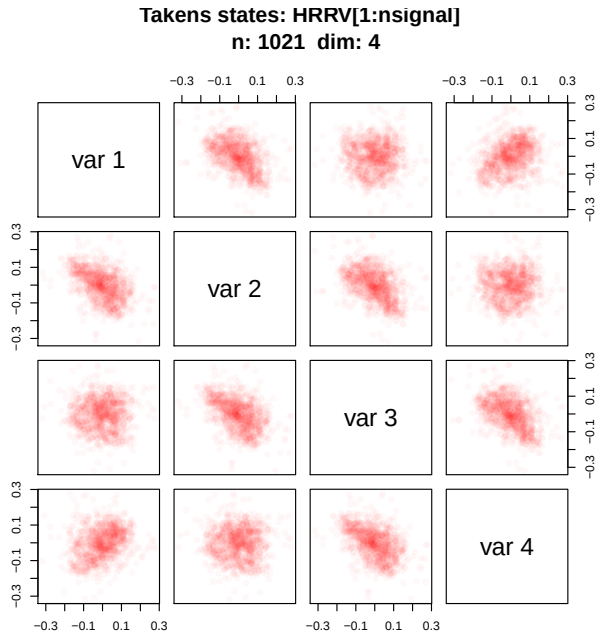


FIGURE 83. Recurrence plot. RHRV tutorial example2.beats. HRRV. Time used: 4.503 sec.

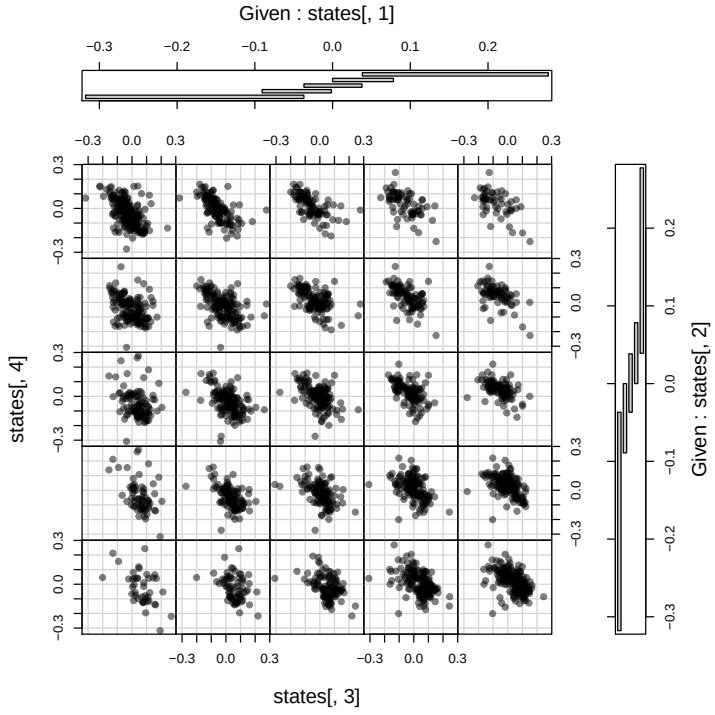


FIGURE 84. State coplot. RHRV tutorial example2.beats. HRRV. Time used: 5.044 sec.

Time used: 0.159 sec.

Input

```
load(file="hrv2RRVneighs4.RData")
local.recurrencePlotAux(hrv2RRVneighs4,dim=4, radius=0.125)
```

ToDo: check. There seem to be strange artefacts.

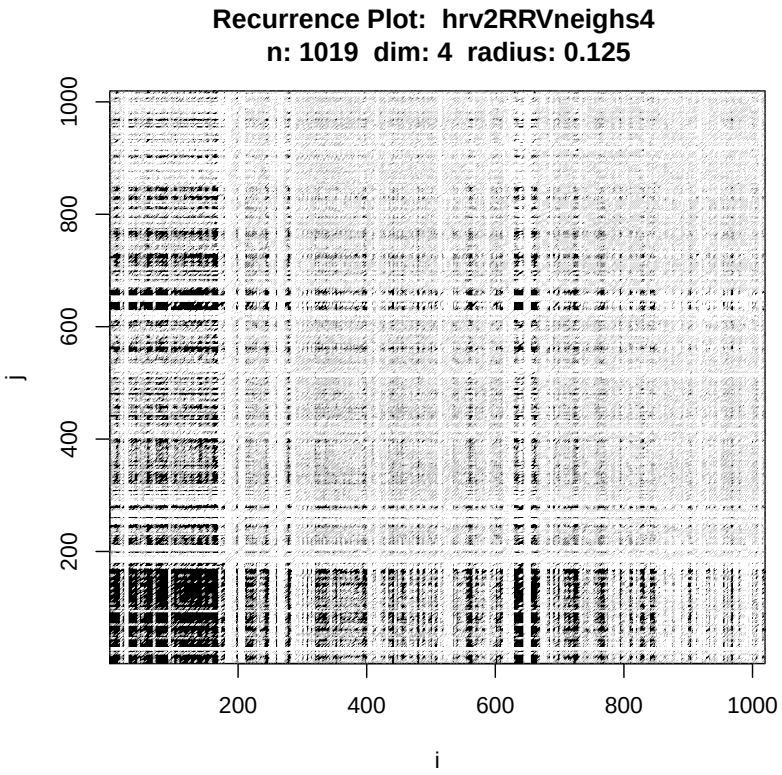


FIGURE 85. Recurrence Plot. Example case: RHRV tutorial example2.beats. HRRV Dim=4. Time used: 2.36 sec.

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

ToDo: fix default setting for radius. Eckmann uses nearest neighbours with NN=10

10.1.1. *RHRV: example2.beats - RR Variation: Comparison by Dimension.*

Input

```
hrv2RRVtakens2 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=2, time.lag=1)
hrv2RRVneighs2  <- local.findAllNeighbours(hrv2RRVtakens2[-(1:2),],
        radius=0.125)
save(hrv2RRVneighs2, file="hrv2RRVneighs2.Rdata")
# load(file="hrv2RRVneighs2.RData")
local.recurrencePlotAux(hrv2RRVneighs2, dim=2,  radius=0.125)
hrv2RRVrqa2 <- showrqa(hrv2RRVtakens2[-(1:2),], radius=0.125, do.hist=FALSE)
```

Output

```
hrv2RRVtakens2[-(1:2), ]  n: 1021  Dim: 2
Radius: 0.125  Recurrence coverage REC: 0.508  log(REC)/log(R): 0.326
Determinism: 0.913  Laminarity: 0.712
DIV: 0.009
Trend: 0  Entropy: 2.289
Diagonal lines max: 117  Mean: 5.305  Mean off main: 5.294
Vertical lines max: 86  Mean: 3.923
```

Time used: 3.636 sec.

Input

```
hrv2RRVtakens6 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=6,time.lag=1)
hrv2RRVneighs6  <- local.findAllNeighbours(hrv2RRVtakens6[-(1:2),], radius=0.125*1.5)
save(hrv2RRVneighs6, file="hrv2RRVneighs6.Rdata")
# load(file="hrv2RRVneighs6.RData")
local.recurrencePlotAux(hrv2RRVneighs6, dim=6, radius=0.125*1.5)
hrv2RRVrqa6 <- showrqa(hrv2RRVtakens6[-(1:2),], radius=0.125*1.5, do.hist=FALSE)
```

Output

```
hrv2RRVtakens6[-(1:2), ]  n: 1017  Dim: 6
Radius: 0.1875  Recurrence coverage REC: 0.516  log(REC)/log(R): 0.395
Determinism: 0.992  Laminarity: 0.736
DIV: 0.007
Trend: 0  Entropy: 3.355
Diagonal lines max: 147  Mean: 12.475  Mean off main: 12.452
Vertical lines max: 193  Mean: 4.902
```

Dim=6. Time used: 3.94 sec.

Input

```
hrv2RRVtakens8 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=8,time.lag=1)
hrv2RRVneighs8  <- local.findAllNeighbours(hrv2RRVtakens8[-(1:2),], radius=3/16)
save(hrv2RRVneighs8, file="hrv2RRVneighs8.Rdata")
# load(file="hrv2RRVneighs8.RData")
local.recurrencePlotAux(hrv2RRVneighs8, dim=8, radius=3/16)
hrv2RRVrqa8 <- showrqa(hrv2RRVtakens8[-(1:2),], radius=3/16, do.hist=FALSE)
```

Output

```
hrv2RRVtakens8[-(1:2), ]  n: 1015  Dim: 8
Radius: 0.1875  Recurrence coverage REC: 0.432  log(REC)/log(R): 0.501
Determinism: 0.992  Laminarity: 0.679
DIV: 0.007
Trend: 0  Entropy: 3.369
Diagonal lines max: 145  Mean: 12.714  Mean off main: 12.685
Vertical lines max: 147  Mean: 4.628
```

Dim=8. Time used: 4.206 sec.

Input

```
hrv2RRVtakens12 <-
        local.buildTakens( time.series=HRRV[1:nsignal],
                embedding.dim=12,time.lag=1)
hrv2RRVneighs12 <-
        local.findAllNeighbours(hrv2RRVtakens12[-(1:2),], radius=3.5/16)
save(hrv2RRVneighs12, file="hrv2RRVneighs12.Rdata")
# load(file="hrv2RRVneighs12.RData")
local.recurrencePlotAux(hrv2RRVneighs12, dim=12,  radius=3.5/16)
hrv2RRVrqa12 <- showrqa(hrv2RRVtakens12[-(1:2),], radius=3.5/16, do.hist=FALSE)
```

Output

```
hrv2RRVtakens12[-(1:2), ]  n: 1011  Dim: 12
Radius: 0.21875  Recurrence coverage REC: 0.479  log(REC)/log(R): 0.484
Determinism: 0.997  Laminarity: 0.746
DIV: 0.006
Trend: 0  Entropy: 3.803
Diagonal lines max: 156  Mean: 19.62  Mean off main: 19.58
Vertical lines max: 192  Mean: 5.298
```

Dim=12. Time used: 3.71 sec.

Input

```
hrv2RRVtakens16 <- local.buildTakens( time.series=HRRV[1:nsignal],
        embedding.dim=16,time.lag=1)
hrv2RRVneighs16  <- local.findAllNeighbours(hrv2RRVtakens16[-(1:2),], radius=4/16)
save(hrv2RRVneighs16, file="hrv2RRVneighs16.Rdata")
# load(file="hrv2RRVneighs16.RData")
local.recurrencePlotAux(hrv2RRVneighs16, dim=16, radius=4/16)
hrv2RRVrqa16 <- showrqa(hrv2RRVtakens16[-(1:2),], radius=4/16, do.hist=FALSE)
```

Output

```
hrv2RRVtakens16[-(1:2), ]  n: 1007  Dim: 16
Radius: 0.25  Recurrence coverage REC: 0.568  log(REC)/log(R): 0.408
Determinism: 0.999  Laminarity: 0.816
DIV: 0.005
Trend: 0  Entropy: 4.186
Diagonal lines max: 221  Mean: 30.105  Mean off main: 30.054
Vertical lines max: 220  Mean: 6.145
```

Dim=16. Time used: 4.917 sec.

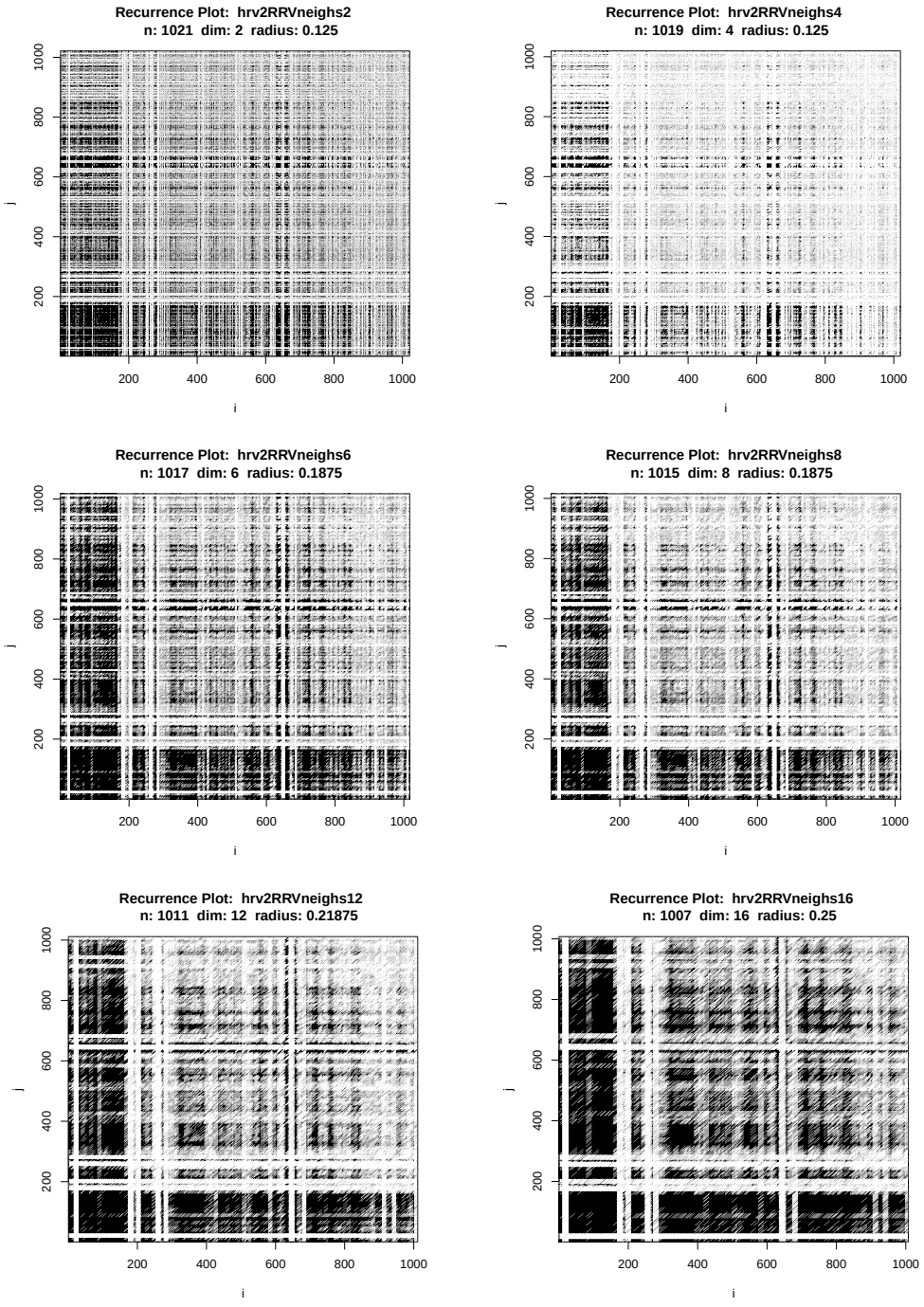


FIGURE 86. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 4.918 sec.

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`.

Index

R session info:

Total Sweave time used: 214.413 sec. at Thu Jun 1 19:49:58 2017.

- R version 3.4.0 (2017-04-21), x86_64-apple-darwin15.6.0
- Locale: en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8
- Running under: macOS Sierra 10.12.5
- Matrix products: default
- BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
- LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
- Base packages: base, datasets, graphics, grDevices, methods, stats, tcltk, utils
- Other packages: leaps 3.0, locfit 1.5-9.1, lomb 1.0, MASS 7.3-47, Matrix 1.2-9, mgcv 1.8-17, nlme 3.1-131, nonlinearTseries 0.2.3, plot3D 1.1, Rcpp 0.12.10, rgl 0.98.1, RHRV 4.2.3, sintro 0.1-6, tkrplot 0.0-23, TSA 1.01, tseries 0.10-40, waveslim 1.7.5
- Loaded via a namespace (and not attached): compiler 3.4.0, digest 0.6.12, grid 3.4.0, htmltools 0.3.6, htmlwidgets 0.8, httpuv 1.3.3, jsonlite 1.4, knitr 1.15.1, lattice 0.20-35, magrittr 1.5, mime 0.5, misc3d 0.8-4, quadprog 1.5-5, R6 2.2.0, shiny 1.0.3, tools 3.4.0, xtable 1.8-2, zoo 1.8-0

LaTeX 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/Rnw/recurrence.Rnw $

$Revision: 250 $

$Date: 2017-06-01 15:06:51 +0200 (Thu, 01 Jun 2017) $

$Name: $

$Author: gsawitzki $

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

Günther Sawitzki
StatLab Heidelberg
Im Neuenheimer Feld 294
D 69120 Heidelberg