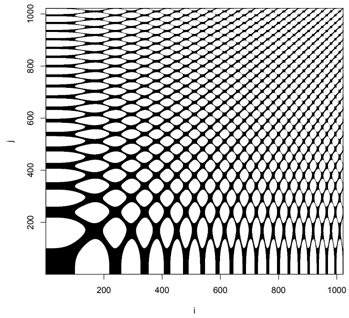
STATISTICAL DATA ANALYSIS: RECURRENCE PLOT

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 29, 2017 from svn/cvs *Revision* : 259
`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 phase space of a system, for a choice of "dimension" d, take the sequence of d tuples taken from your data to define the states.

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

Under mild conditions, this Takens' delay embedding allows to reconstruct the topogy of the phase space from this embedded sequence [Takens, 1981].

The delay imbedding implicitly depends on the sampling rate with which the signal has been sampled. For example, if the rate is sufficiently high, all differentiable dynamics appear to be linear sine linear approximation holds. The imbedding construction allows a sub-sampling using a lag parameter m. So you take

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

ToDo: add support for higher dimensional signals

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>`, for example Figure 1

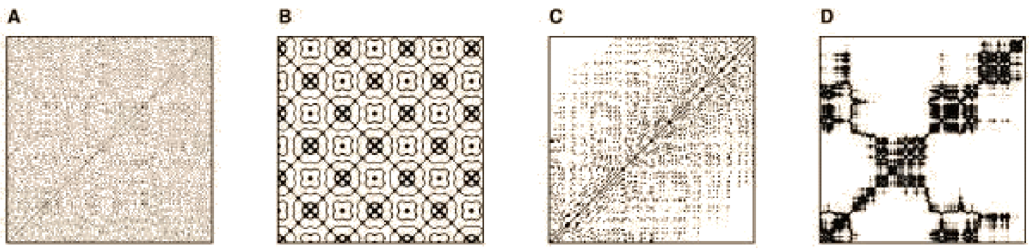


FIGURE 1. Characteristic typology of recurrence plots: (A) homogeneous (uniformly distributed noise), (B) periodic (super-positioned harmonic oscillations), (C) drift (logistic map corrupted with a linearly increasing term) and (D) disrupted (Brownian motion). These examples illustrate how different RPs can be. The used data have the length 400 (A, B, D) and 150 (C), respectively; no embeddings are used; the thresholds are $\epsilon = 0.2$ (A, C, D) and $\epsilon = 0.4$ (B). From `<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). References: [Zbilut and Webber, 2006], [Webber Jr and Zbilut, 2005]. See Table 1 on the next page.

ToDo: check RATIO - remove $RR =^? Radius^2$

ToDo: make scale indpendent

2. R Setup

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

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.

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

For signal representation, we use a common layout.

Input

```
plotsignal <- function(signal, main, ylab, alpha=0.4) {
        if (missing(ylab)) { ylab <- deparse(substitute(signal)) }

        par(mfrow = c(1, 2))
        plot(signal,
                main = "", xlab = "index",  ylab = ylab,
                col = rgb(0, 0, 1, 0.75*alpha), pch = 20)

        plot(signal, type = "l",
                main = "",  xlab = "index", ylab = ylab,
                col = rgb(0, 0, 0, alpha))
        points(signal,
                col = rgb(0, 0, 1, 0.75*alpha), 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)
}
```

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, rgb(1,0,0, 1/sqrt(dim(states)[1])),
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 $rgb(1, 0, 0, \alpha = \sqrt{\frac{1}{nr\ states}})$, 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.

This assumes that correlated neighbours are collected in lists. Just for reference. This function is inlined.

Input

```
    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)
    }
```

Input

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


  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
                )
 #        )

}
```

This assumes that correlated neighbours are collected in lists. Just for reference. This function is inlined.

Input

```
        # 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)
        }
```

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 ,...)
{        #takens <- takens[!is.na(takens)]
        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$time.lag <- attr(takens,"time.lag")
        cat(id, " n:", dim(takens)[1], " Dim:", dim(takens)[2], " Lag:",xxrqa$time.lag, " Radius:", radiu
        xxrqa$n <- dim(takens)[1];         xxrqa$dim <- dim(takens)[2]

        cat(paste(" 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("Ratio", xxrqa$RATIO)
        cat(paste(" Determinism:", round(xxrqa$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"))

        if (do.hist){
        if (log==TRUE) log<-"y"
        oldpar <- par(mfrow=c(1,2))
        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)
        try(plot(dh, type="h", main=paste( id, " Diagonal:",
                pars," Radius: ",radius, " REC:", round(xxrqa$REC, digits)),
                xlab="length of diagonals",
                log = log, ...))        #needs handling of nan, Inf.

        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",
                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 50) and two examples of heart rate data sets (Section 9 on page 72 and Section 10 on page 97) from `library(rhrv)`. The Geyser data set gives an example of a bi-variate point process. The HRV data sets give real live point process 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, if restricting is appropriate.

Input
```
nsignal <- 1024 #nsignal <- 4096 #nsignal <- 256
system.time.start <- proc.time()
```

3.1. **Sinus.** 10 sinus waves in 1024 steps.

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

See Figure 2 on page 15.

3.2. **Uniform Random Numbers.**

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

See Figure 3 on page 15,

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 4 on the next page,

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 5 on the facing page,

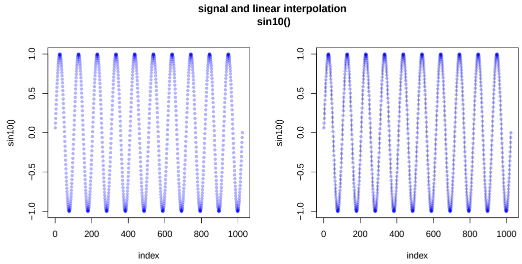


FIGURE 2. Test case: sin10. Signal and linear interpolation.

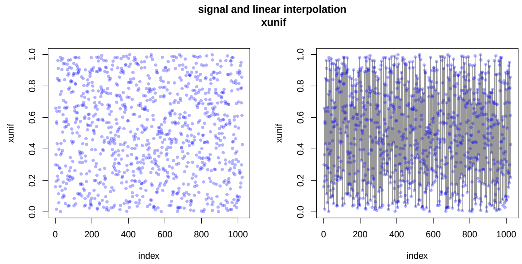


FIGURE 3. Test case: unif - uniform random numbers. Signal and linear interpolation.

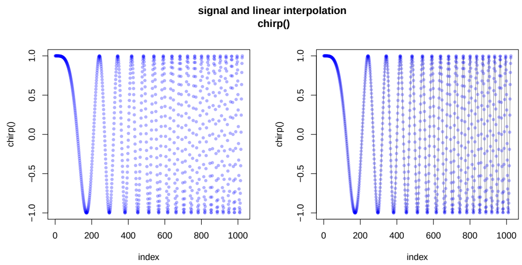


FIGURE 4. Test case: chirp signal. Signal and linear interpolation.

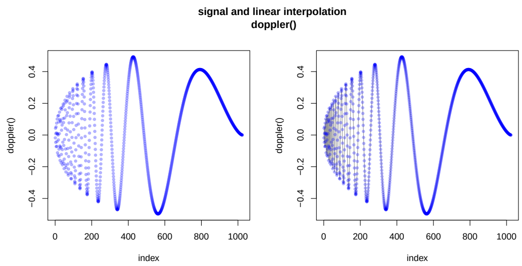


FIGURE 5. Test case: Doppler signal. Signal and linear interpolation.

4. nonlinearTseries Quick-Start

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("$\tau_{delay}$ estimation based on acf\n","sin()"))
# taudelay estimation based on the mutual information function
tau.ami=NA
try(tau.ami <- timeLag(sinN, technique = "ami",
        do.plot = T, selection.method="first.minimum",
        main=paste("$\tau_{delay}$ estimation based on ami\n","sin()")) )
par(oldpar)
cat("Sinus tau.ami:",tau.ami," tau.acf:",tau.acf)
```

Output

```
Sinus tau.ami: 6  tau.acf: 20
```

See Figure 6.

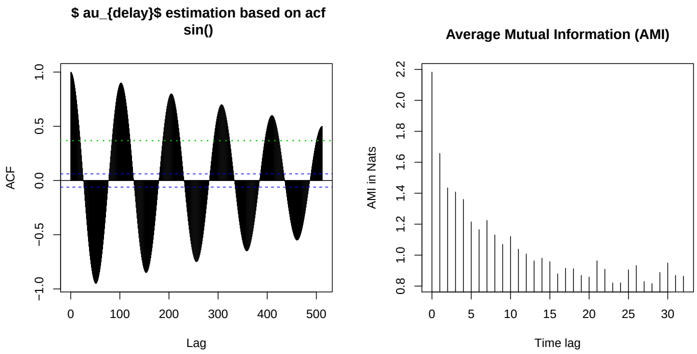


FIGURE 6. τ_{delay} estimation. "Sinus" Left: based on the autocorrelation function ACF. Right: AMI

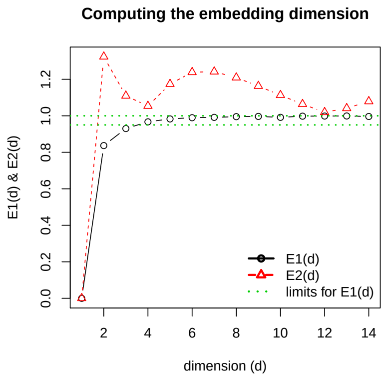
Input

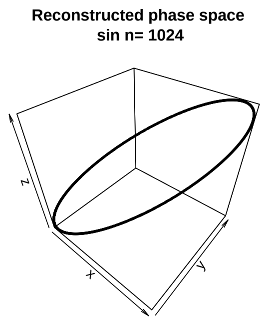
```
oldpar <-  par(mfrow=c(1,2))
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(sinN, time.lag = tau.ami,
max.embedding.dim = 15)
cat("sin() Lag tau.ami", tau.ami,"Estimated embedding dim:", emb.dim)
}  else  {
emb.dim = estimateEmbeddingDim(sinN, time.lag = tau.acf,
max.embedding.dim = 15)
cat("sin() Lag tau.acf", tau.acf,"Estimated embedding dim:", emb.dim)
}
```

Output

```
sin() Lag tau.ami 6 Estimated embedding dim: 4
```

Input
```
#taksinN = buildTakens(sinN, embedding.dim = 2, time.lag = 1)
taksinN = buildTakens(sinN, embedding.dim = emb.dim, time.lag = tau.ami)
scatter3D(taksinN[,1], taksinN[,2], taksinN[,3],
        main = paste("Reconstructed phase space\n sin n=",length(sinN)),
        col = 1, type="o",cex = 0.3)
par(oldpar)
```

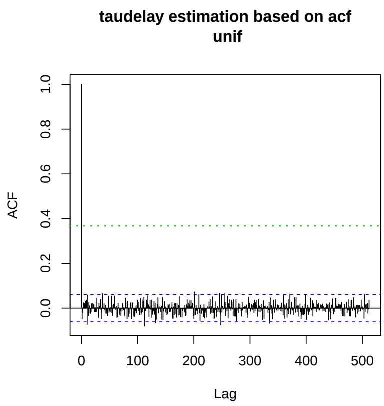


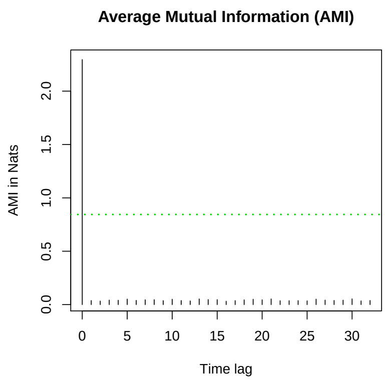


4.2. **Uniform Random Numbers.**

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("taudelay estimation based on acf\n", "unif"))
# taudelay estimation based on the mutual information function
tau.ami = timeLag(unifN, technique = "ami", do.plot = T,
        main=paste("taudelay estimation based on ami\n", "unif"))
par(oldpar)
cat("Uniform ", "tau.acf:", tau.acf ," tau.ami:",tau.ami)
```

Output
```
Uniform  tau.acf: 1  tau.ami: 1
```



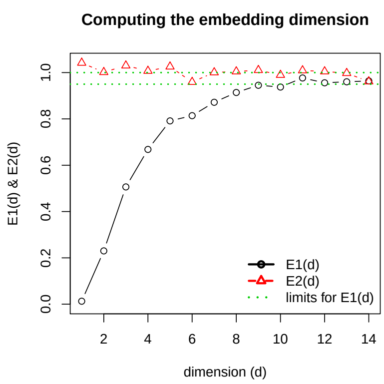


Input

```
oldpar <-  par(mfrow=c(1,2))
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(unifN, time.lag = tau.ami,
        max.embedding.dim = 15)
par(oldpar)
cat("unif Lag tau.ami", tau.ami,"Estimated embedding dim:", emb.dim)
}  else  {
emb.dim = estimateEmbeddingDim(unifN, time.lag = tau.acf,
        max.embedding.dim = 15)
par(oldpar)
cat("unif Lag tau.acf", tau.acf,"Estimated embedding dim:", emb.dim)
}
```

Output
```
unif Lag tau.ami 1 Estimated embedding dim: 11
```

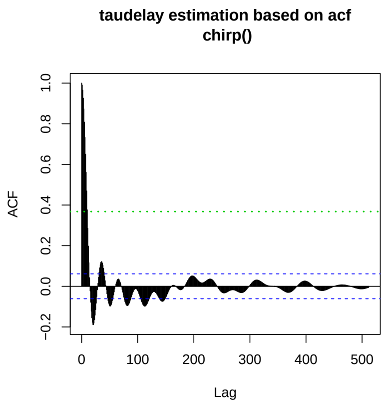
Input
```
#<<takfunifN, fig=TRUE>>=
takUnif = buildTakens(unifN, embedding.dim = emb.dim, time.lag = tau.ami)
#range(takUnif[4,])
col4 <- gray((1-takUnif[,4]),0.1)
scatter3D(takUnif[,1], takUnif[,2], takUnif[,3],
        main = paste("Reconstructed phase space","unif n=",length(unifN)),
        col = col4, type="o",cex = 0.3)
par(oldpar)
```

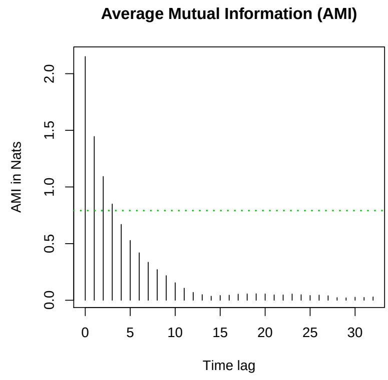


4.3. Chirp.

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("taudelay estimation based on acf\n","chirp()"))
# taudelay estimation based on the mutual information function
tau.ami = timeLag(chirpN, technique = "ami", do.plot = T,
main=paste("taudelay estimation based on ami\n n=",length(chirpN),"chirp()"))
par(oldpar)
cat("Chirp ", "tau.acf:", tau.acf ," tau.ami:",tau.ami)
```

Output
```
Chirp  tau.acf: 11  tau.ami: 4
```





Input

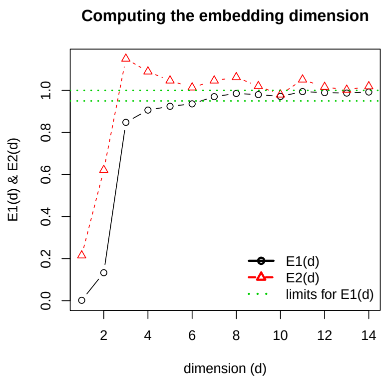
```
oldpar <-  par(mfrow=c(1,2))
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(chirpN, time.lag = tau.ami,
        max.embedding.dim = 15)
cat("chirp Lag tau.ami", tau.ami,"Estimated embedding dim:", emb.dim)
}  else  {
emb.dim = estimateEmbeddingDim(chirpN, time.lag = tau.acf,
        max.embedding.dim = 15)
cat("chirp Lag tau.acf", tau.acf,"Estimated embedding dim:", emb.dim)
}
```

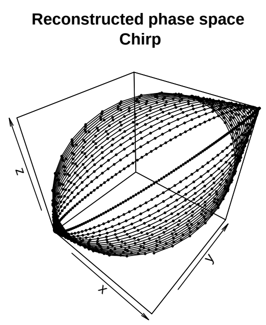
Output

```
chirp Lag tau.ami 4 Estimated embedding dim: 7
```

Input

```
#<<takfchirpN, fig=TRUE>>=
takChirp = buildTakens(chirpN, embedding.dim = emb.dim, time.lag = tau.ami)
scatter3D(takChirp[,1], takChirp[,2], takChirp[,3],
        main = paste("Reconstructed phase space\n","Chirp"),
        col = 1, type="o",cex = 0.3)
par(oldpar)
```

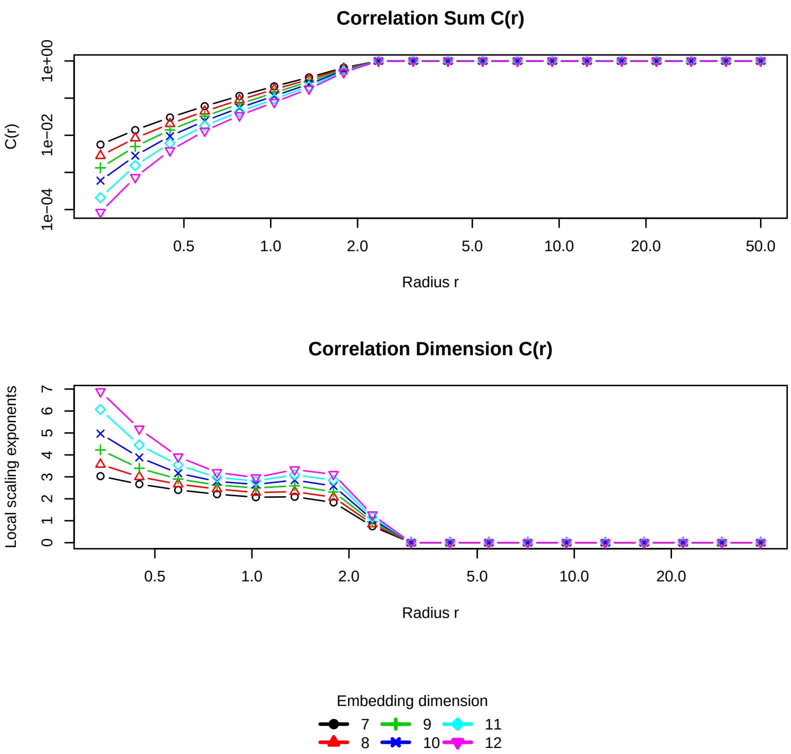




Input

```
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)
par(oldpar)
```



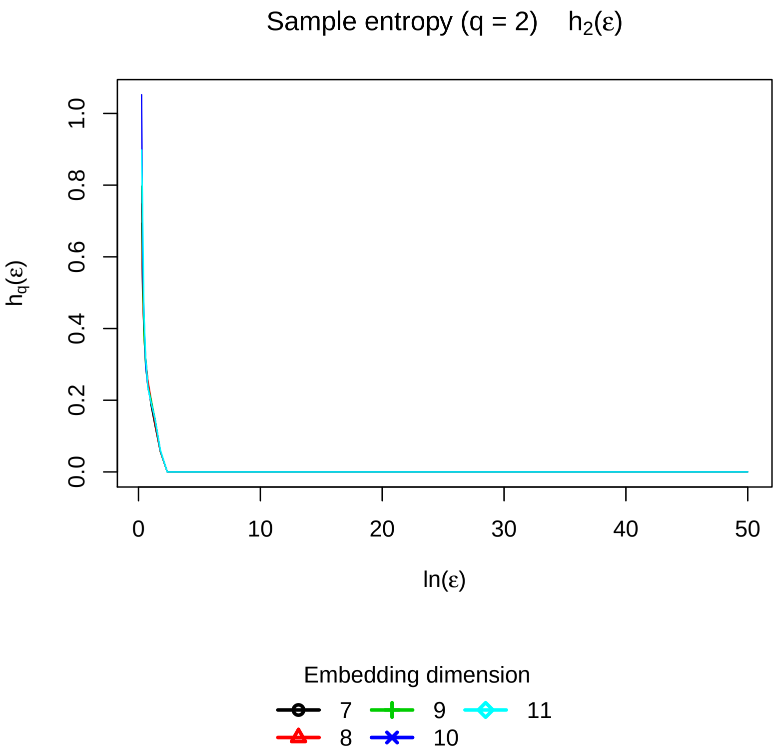
Input

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

Output

```
Sample entropy estimate:  0
```

Input



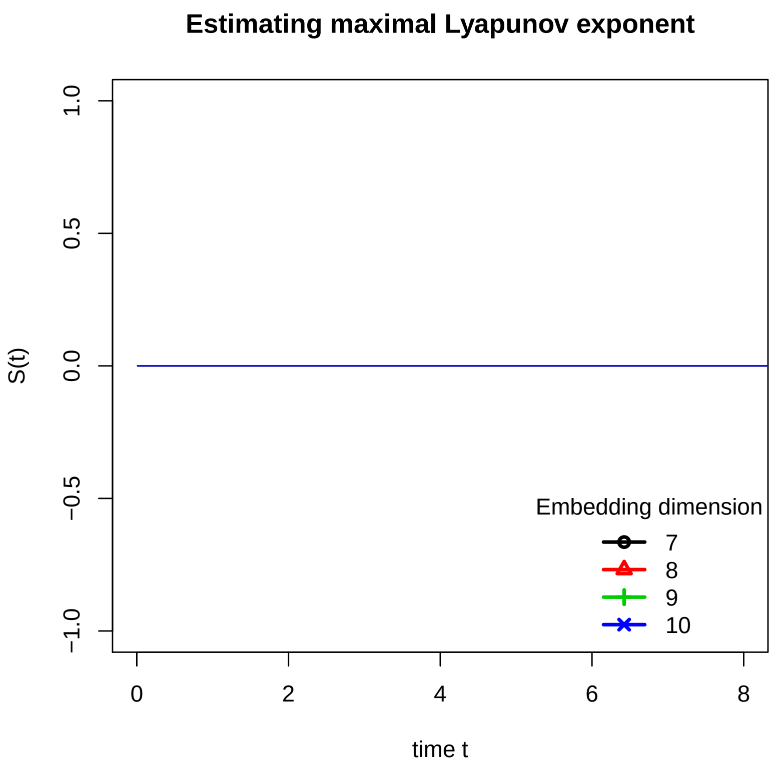
Input

```
#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

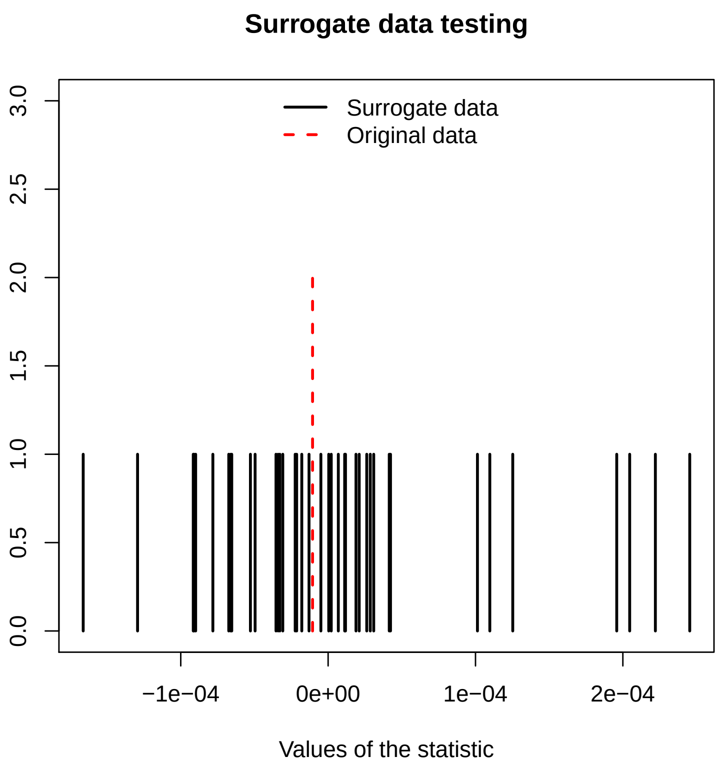
```
estimate: 0
```

Input



Input

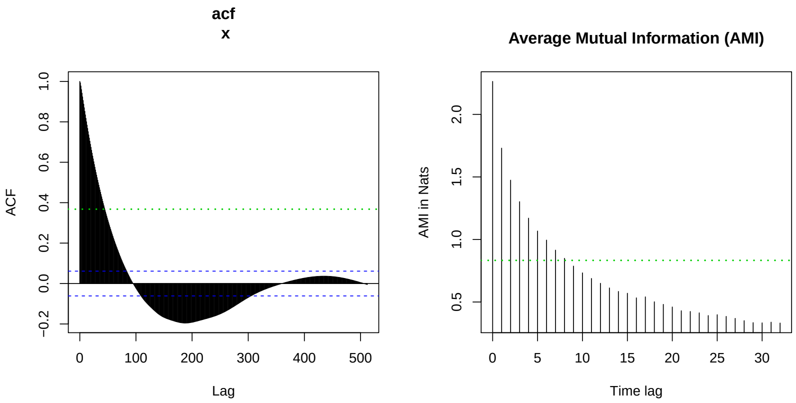
```
st = surrogateTest(chirpN,significance = 0.05,one.sided = F,
FUN = timeAsymmetry, do.plot=F)
## Computing statistics
##
## Null Hypothesis: Data comes from a linear stochastic process
## Reject Null hypothesis:
## Original data's stat is significant larger than surrogates' stats
plot(st)
```


Surrogate data testing
Surrogate data
Original data
0.0
0.5
1.0
1.5
2.0
2.5
3.0
−1e−04
0e+00
1e−04
2e−04
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))))
# taudelay estimation based on the mutual information function
tau.ami = timeLag(dopplerN, technique = "ami", do.plot = T,
main=paste("ami\n",deparse(substitute(x))))
par(oldpar)
cat("Doppler ", "tau.acf:", tau.acf ," tau.ami:",tau.ami)
```

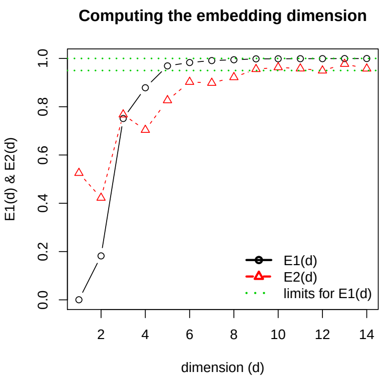
Output
```
Doppler  tau.acf: 45  tau.ami: 9
```

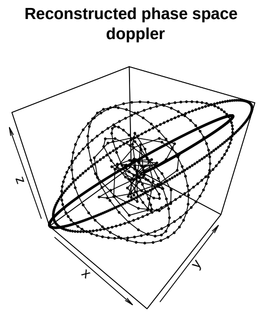


Input
```
oldpar <-  par(mfrow=c(1,2))
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(dopplerN, time.lag = tau.ami,
        max.embedding.dim = 15)
cat("Doppler lag tau.ami", tau.ami,"Estimated embedding dim:", emb.dim)
}  else  {
emb.dim = estimateEmbeddingDim(dopplerN, time.lag = tau.acf,
        max.embedding.dim = 15)
cat("Doppler lag tau.acf", tau.acf,"Estimated embedding dim:", emb.dim)
}
```

Output
```
Doppler lag tau.ami 9 Estimated embedding dim: 5
```

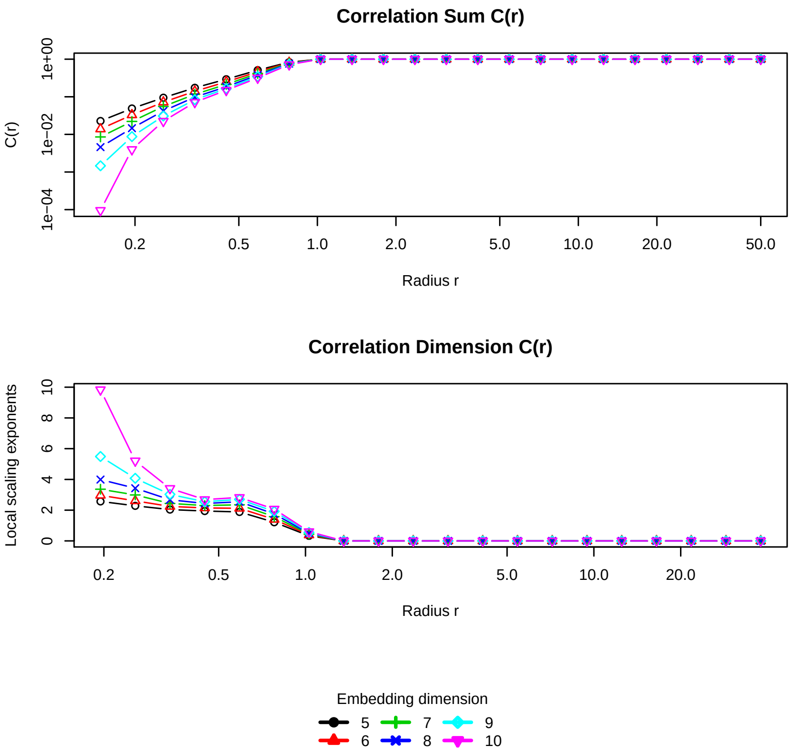
Input
```
#<<takfdopplerN, fig=TRUE>>=
tak = buildTakens(dopplerN, embedding.dim = emb.dim, time.lag = tau.ami)
scatter3D(tak[,1], tak[,2], tak[,3],
        main = "Reconstructed phase space \n doppler" ,
        col = 1, type="o",cex = 0.3)
par(oldpar)
```





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

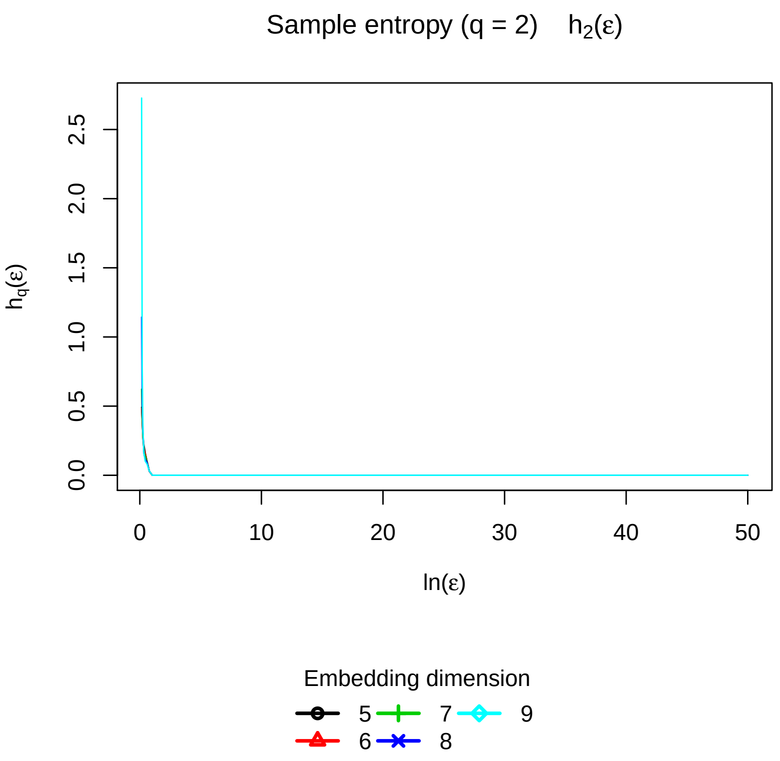
```
oldpar <- par(mfrow=c(1,2))
se <- sampleEntropy(cd, do.plot = T)
se.est <- estimate(se, do.plot = T,
```

```
        regression.range = c(8,15))
par(oldpar)
        cat("Sample entropy estimate: ", mean(se.est), "\n")
```

Output

```
Sample entropy estimate:  0
```

Input



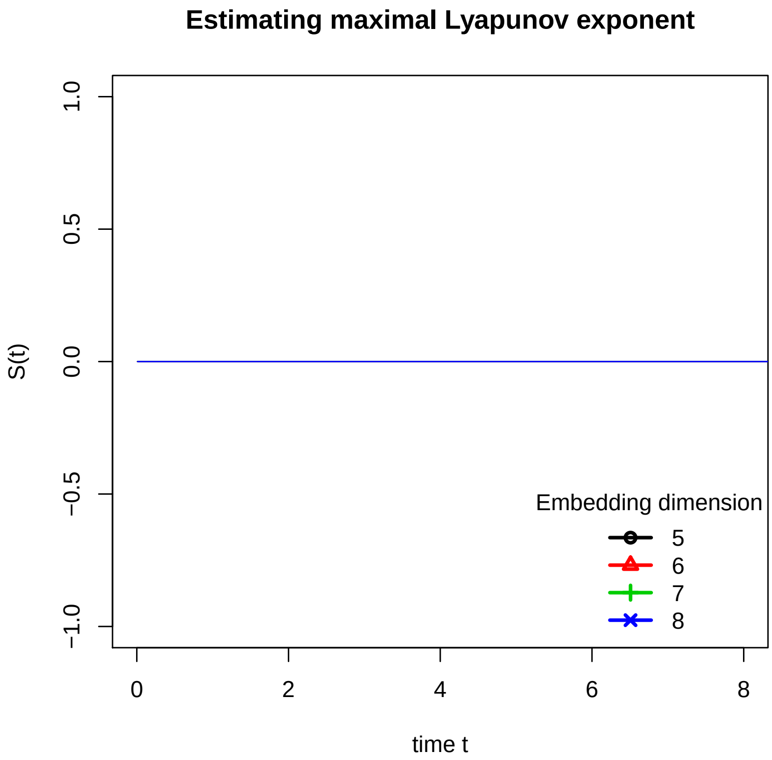
Input

```
#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

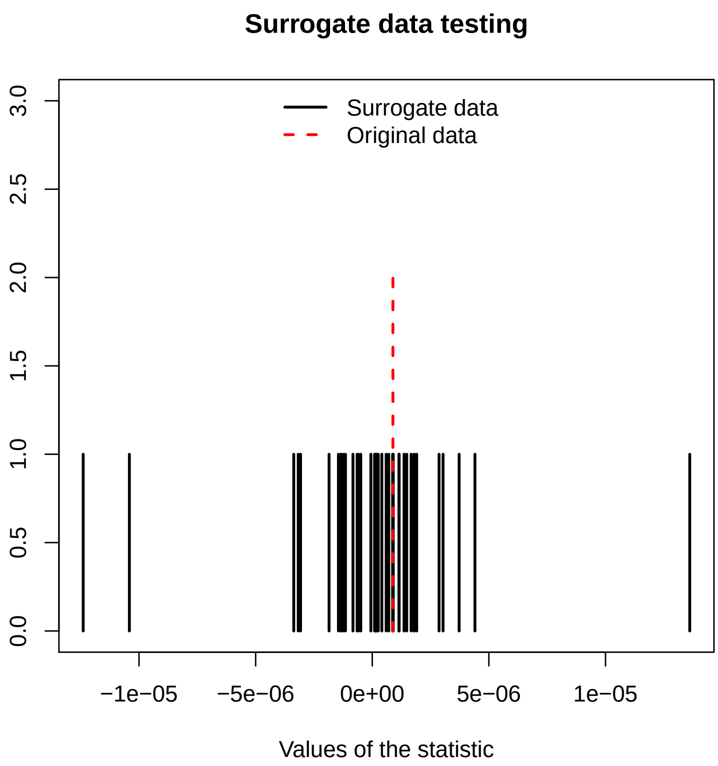
```
estimate: 0
```

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 7.

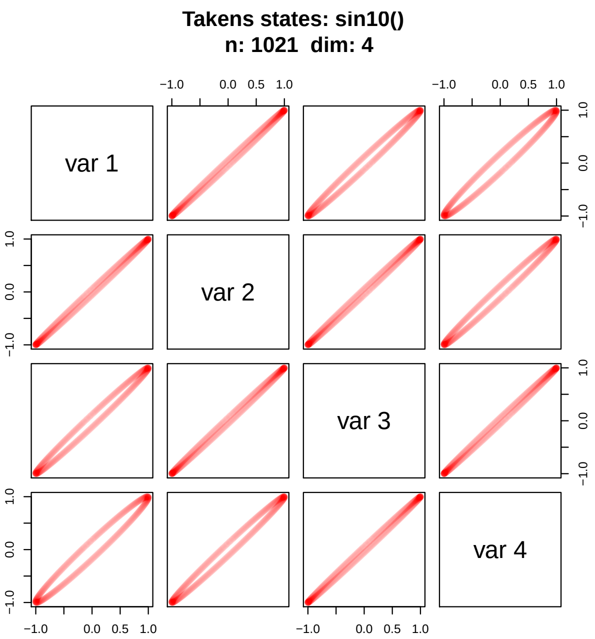


Figure 7. Takens states. Test case: sinus. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 1.122 sec.

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 8 on the next page.

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

See Figure 9 on the following page.

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

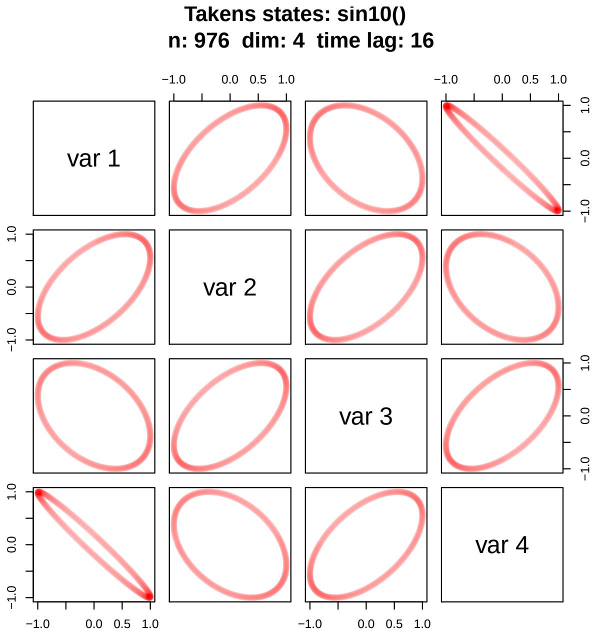


FIGURE 8. Takens states. Test case: sinus at time lag 16. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.592 sec.

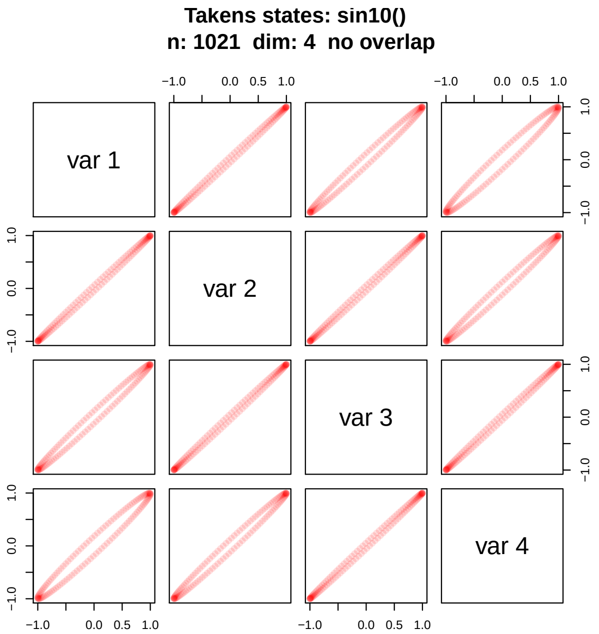


FIGURE 9. Takens states. Test case: sinus, no overlap. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.774 sec.

See Figure 10.

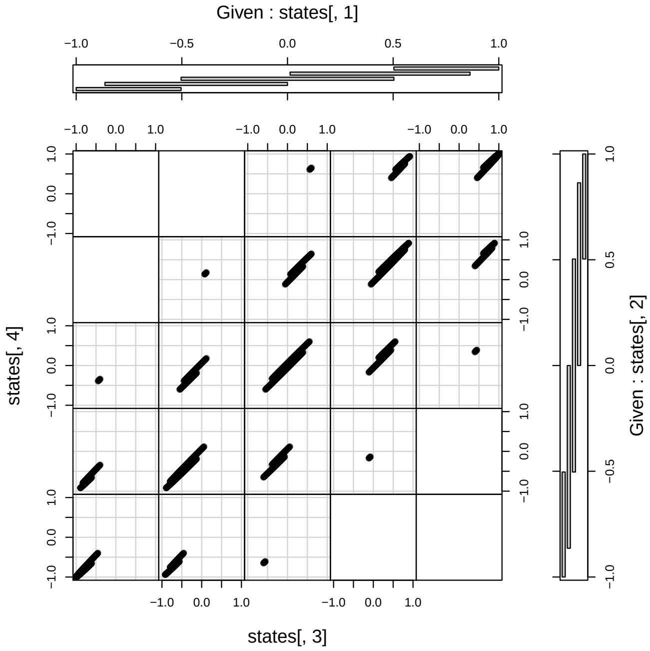


FIGURE 10. State coplot. Test case: sinus. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.601 sec.

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

See Figure 11 on the following page.

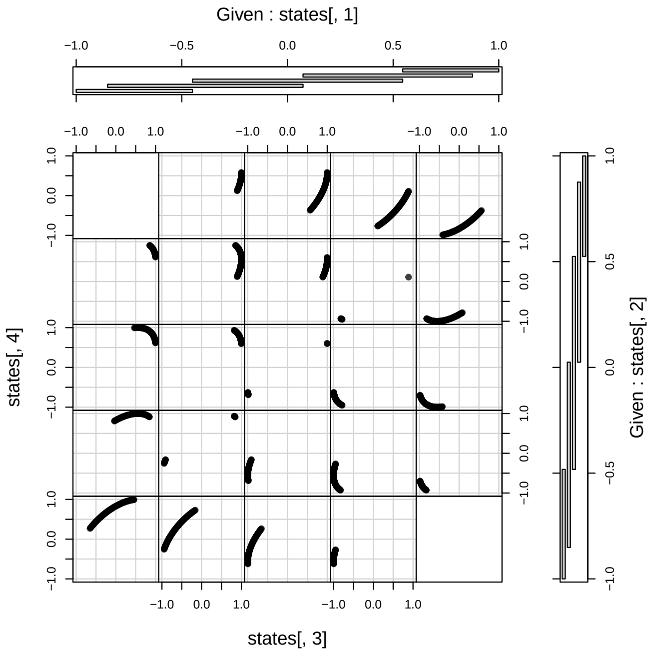


FIGURE 11. State coplot. Test case: sinus, time lag 16. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.198 sec.

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

See Figure 12.

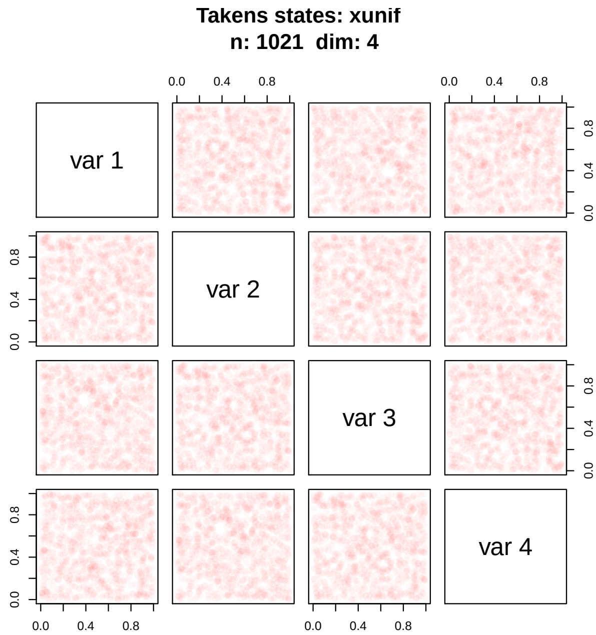


FIGURE 12. Takens states. Test case: uniform random numbers. Time used: 1.016 sec.

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

See Figure 13 on the following page.

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

See Figure 14 on the next page.

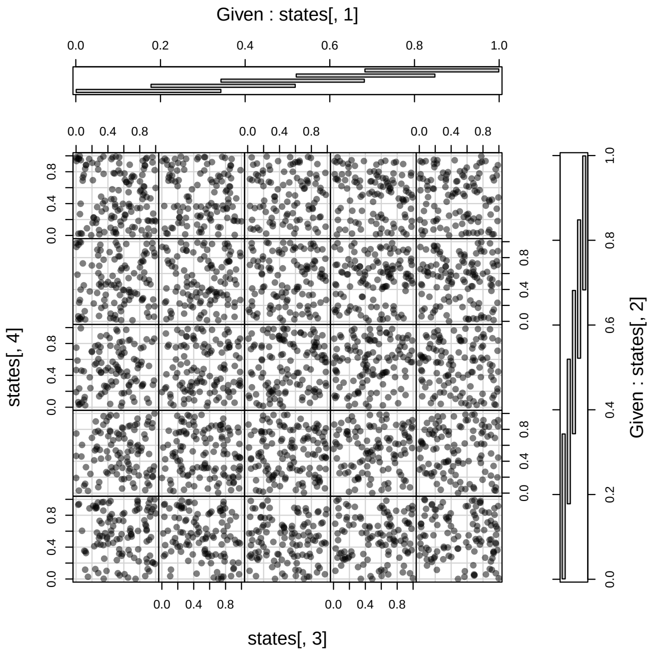


FIGURE 13. State coplot. Test case: uniform random numbers. Time used: 1.246 sec.

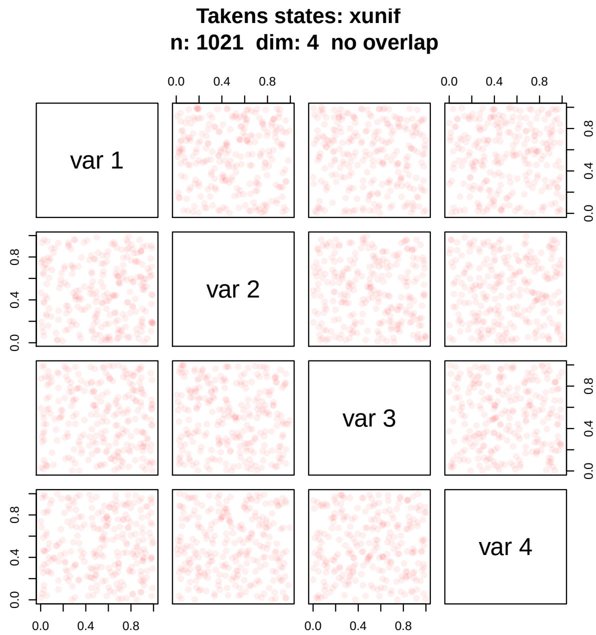


FIGURE 14. Takens states. Test case: uniform random numbers. Time used: 1.462 sec.

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

See Figure 15.

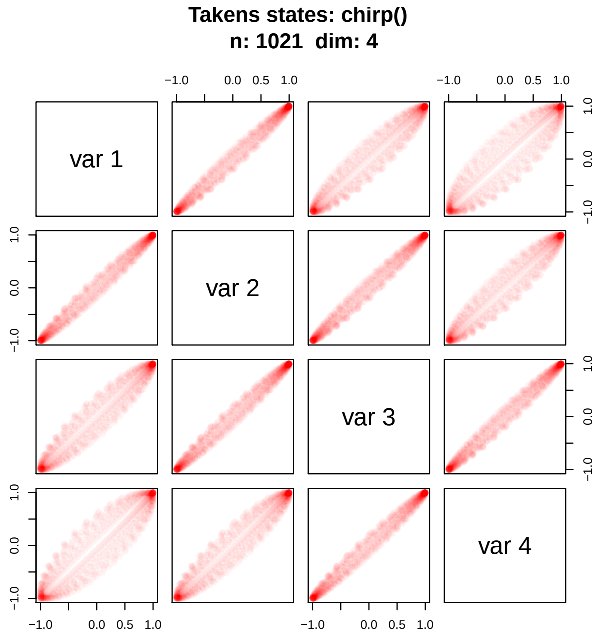


FIGURE 15. Takens states. Test case: chirp signal. Time used: 1.014 sec.

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

See Figure 16 on the next page.

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

See Figure 17 on the following page.

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

See Figure 18 on page 37.

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.

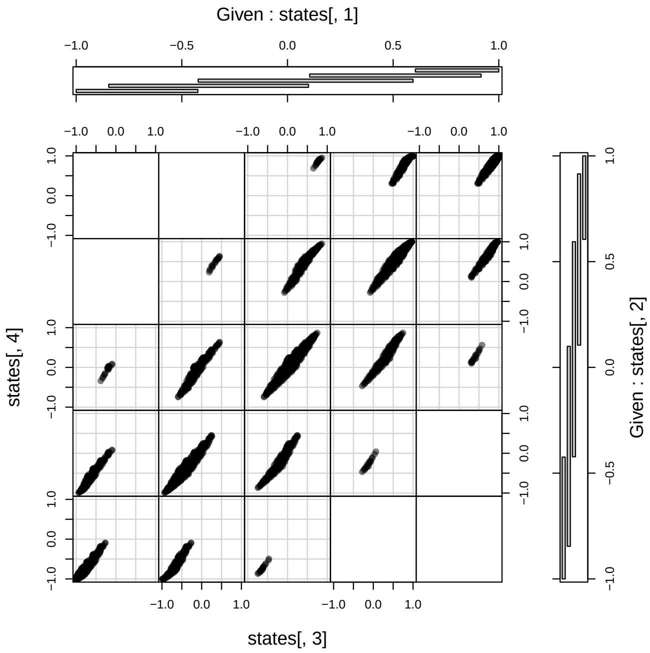


FIGURE 16. State coplot. Test case: chirp signal. Time used: 1.232 sec.

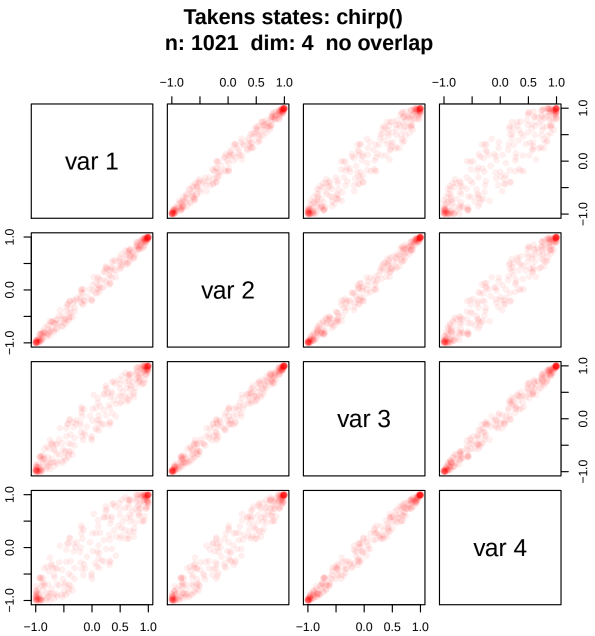


FIGURE 17. Takens states. Test case: chirp signal. Time used: 1.648 sec-

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

See Figure 19 on the facing page.

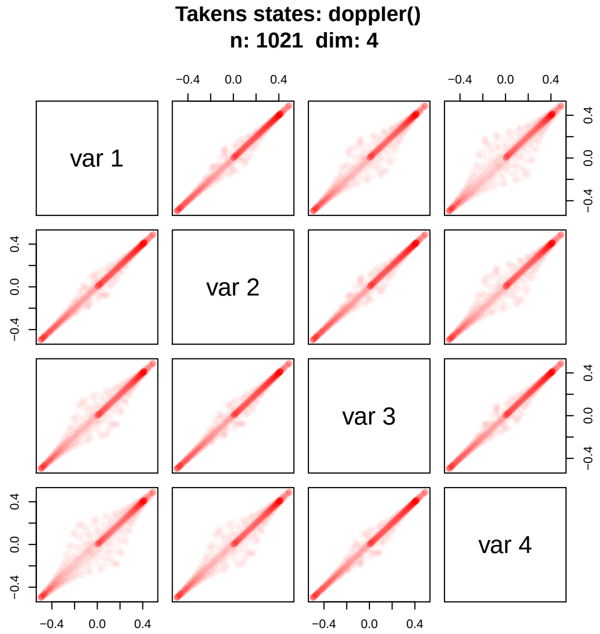


FIGURE 18. Takens states. Test case: Doppler. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 1.002 sec.

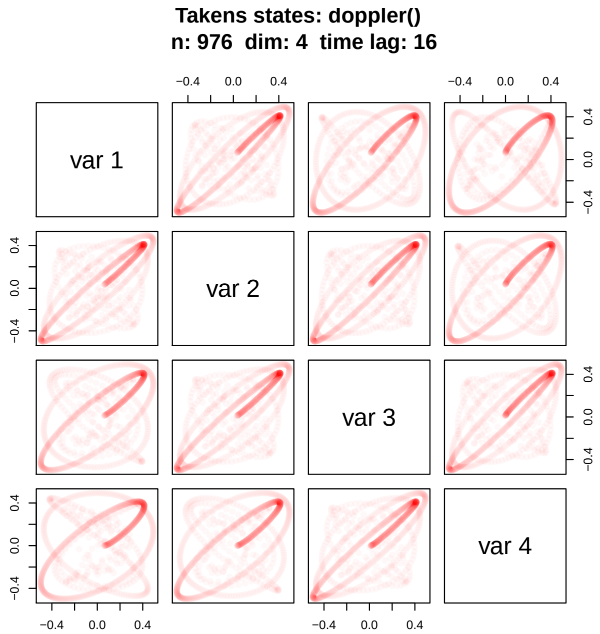


FIGURE 19. Takens states. Test case: doppler at time lag 16. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.976 sec.

Input

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

See Figure 20.

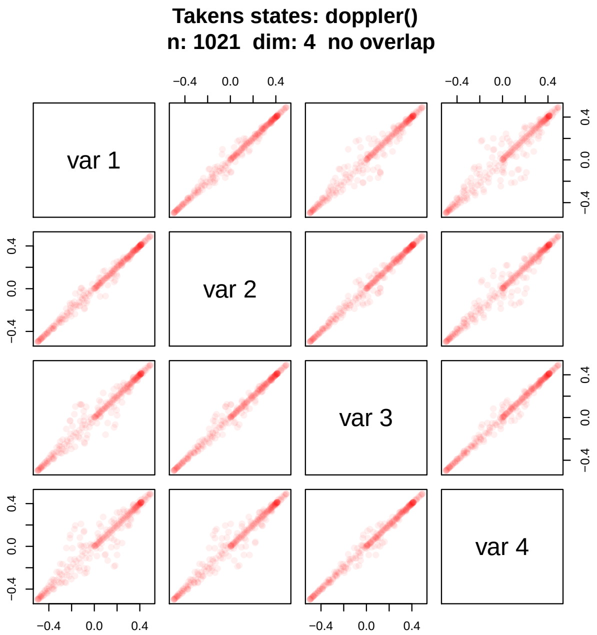


FIGURE 20. Takens states. Test case: doppler, no overlap. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 1.2 sec.

Input

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

See Figure 21 on the facing page.

Input

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

See Figure 22 on the next page.

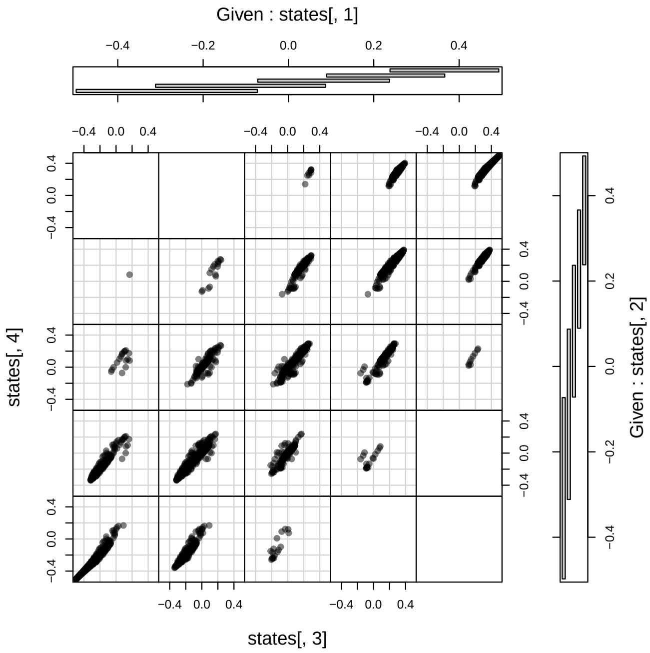


FIGURE 21. State coplot. Test case: Doppler. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.228 sec.

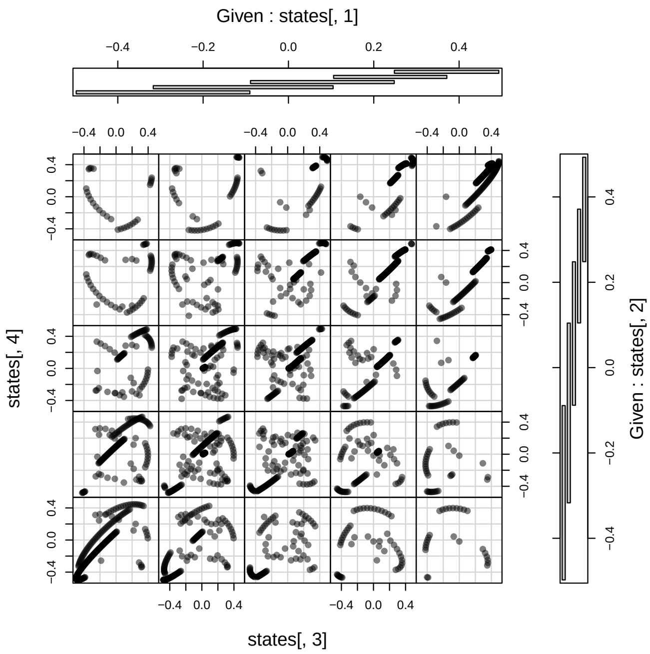


FIGURE 22. State coplot. Test case: Doppler, time lag 16. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.585 sec.

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 23.

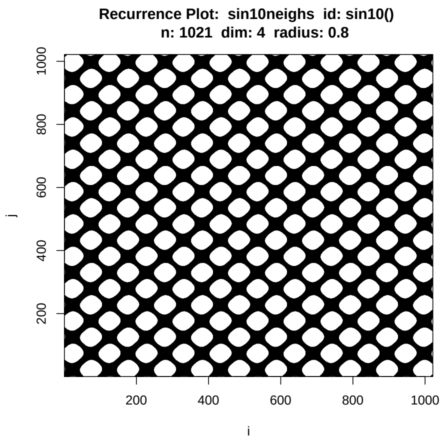


FIGURE 23. Recurrence plot. Test case: sinus. Time used: 1.799 sec.

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

Output
```
sin10()  n: 1021  Dim: 4  Lag: 1  Radius: 0.8
 Recurrence coverage REC: 0.496  log(REC)/log(R): 3.143
Ratio 2.016253 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. log scale. Time used: 2.746 sec. For graphical ouput, see Figure 24 on the next page.

Input
```
sin10lag16neighs <- local.findAllNeighbours(sintakenslag16, radius=0.2)
save(sin10lag16neighs, file="sin10lag16neighs.Rdata")
# load(file="sin10lag16neighs.RData")
local.recurrencePlotAux(sin10lag16neighs, dim=4, radius=0.2)
```

See Figure 25 on the facing page.

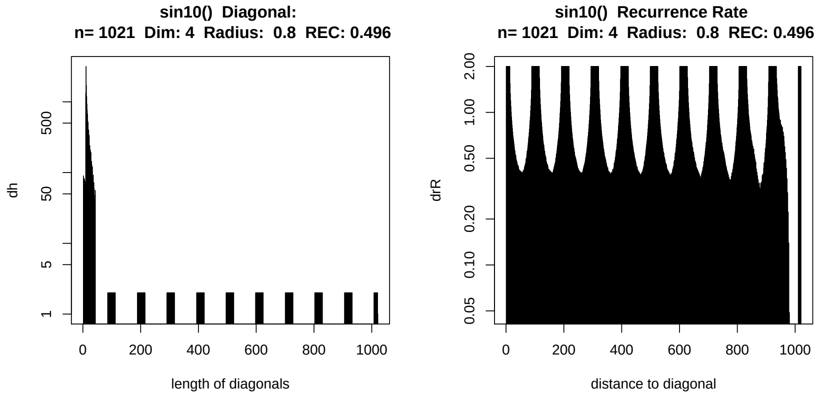


FIGURE 24. RQA. Test case: sinus. Radius=0.8. log scale

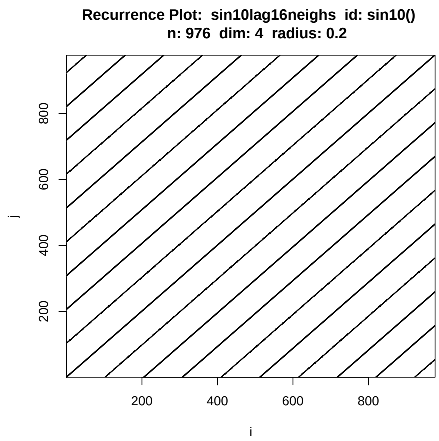


FIGURE 25. Recurrence plot. Test case: sinus curve, time lag 16. Time used: 3.667 sec.

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

Output
```
sin10()  n: 976  Dim: 4  Lag: 16  Radius: 0.2
 Recurrence coverage REC: 0.067  log(REC)/log(R): 1.683
Ratio 15.00875 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: 4.273 sec. For graphical ouput, see Figure 26 on the next page.

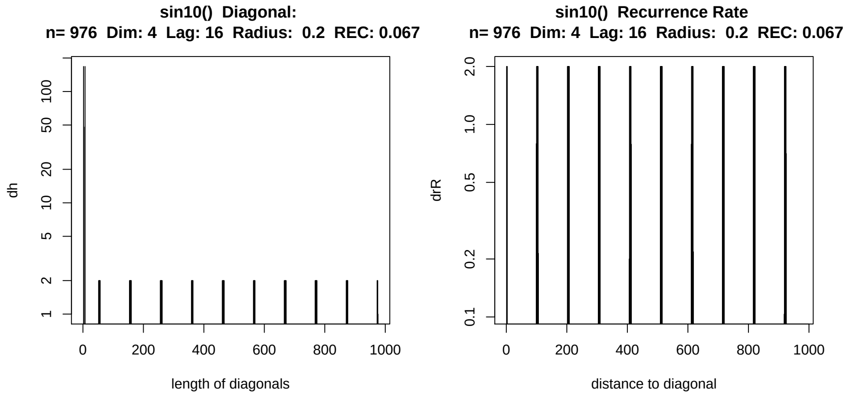


FIGURE 26. RQA. Test case: sinus curve, time lag 16

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 27.

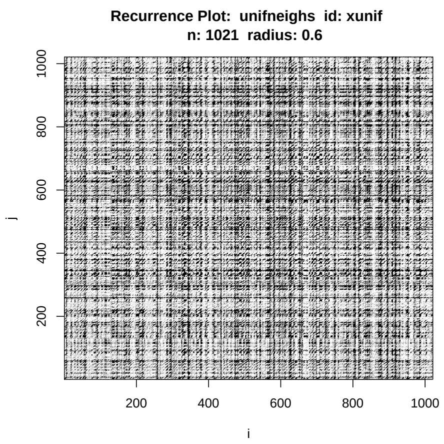


FIGURE 27. Recurrence plot. Test case: uniform random numbers. Time used: 2.014 sec.

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

Output
```
xunif  n: 1021  Dim: 4  Lag: 1  Radius: 0.6
 Recurrence coverage REC: 0.491  log(REC)/log(R): 1.392
Ratio 1.981289 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, log scale. Time used: 3.211 sec. For graphical ouput, see Figure 28 on the next page.

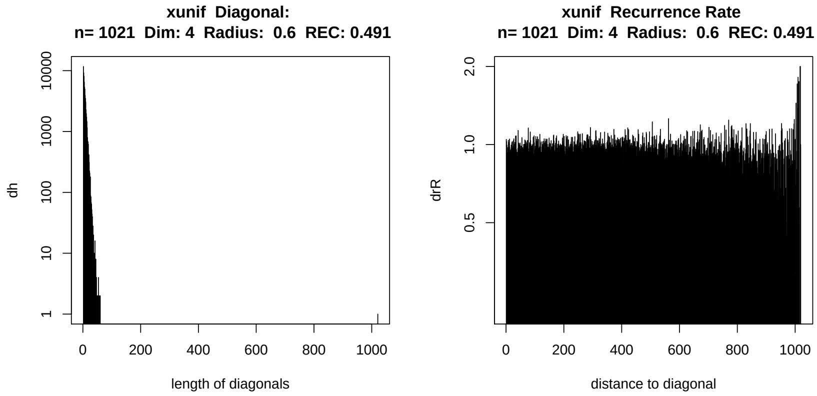


Figure 28. RQA. Test case: uniform random numbers, radius=0.6, log scale

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 29.

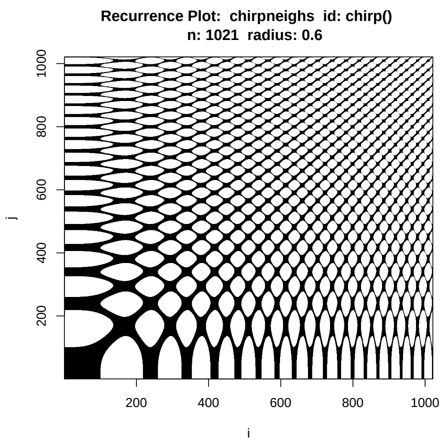


FIGURE 29. Recurrence plot. Test case: chirp signal. Time used: 1.457 sec.

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

Output
```
chirp()  n: 1021  Dim: 4  Lag: 1  Radius: 0.6
 Recurrence coverage REC: 0.341  log(REC)/log(R): 2.108
Ratio 2.899648 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: 1.913 sec. For graphical ouput, see Figure 30 on the next page.

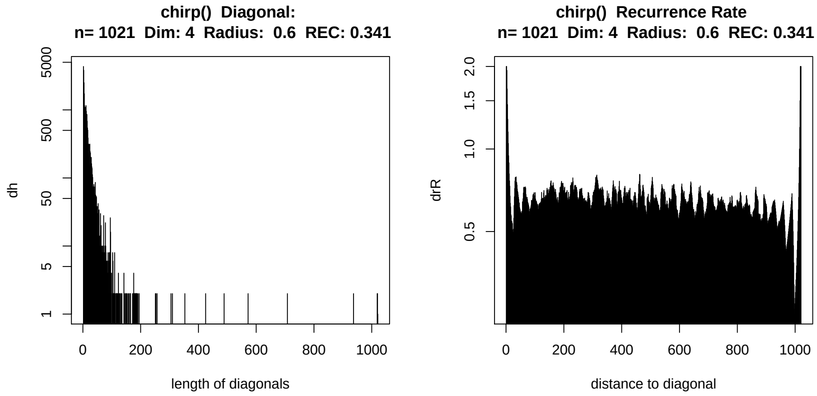


FIGURE 30. RQA. Test case: chirp signal

6.0.4. *Doppler Recurrence Plots.*

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

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

See Figure 31.

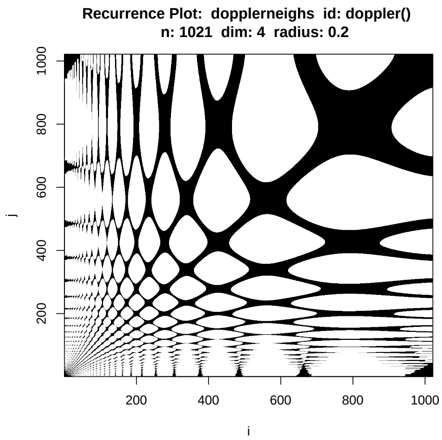


FIGURE 31. Recurrence plot. Test case: Doppler. Time used: 1.062 sec.

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

Output
```
doppler()  n: 1021  Dim: 4  Lag: 1  Radius: 0.2
 Recurrence coverage REC: 0.299  log(REC)/log(R): 0.75
Ratio 3.314572 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: 0.395 sec. For graphical ouput, see Figure 32 on the next page.

Input
```
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 33 on the following page.

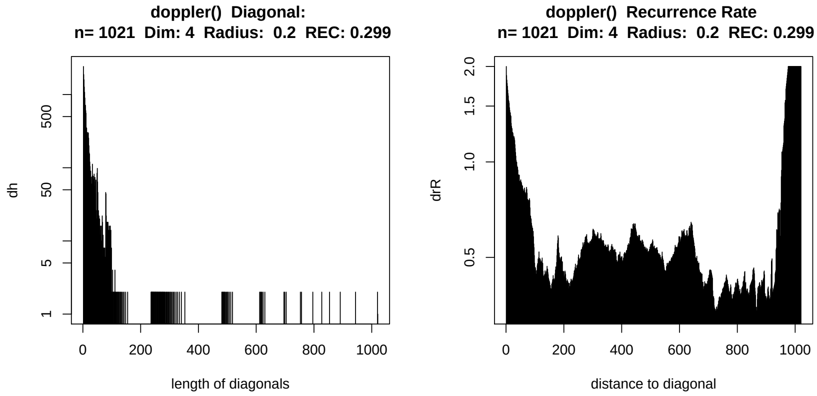


FIGURE 32. RQA. Test case: Doppler

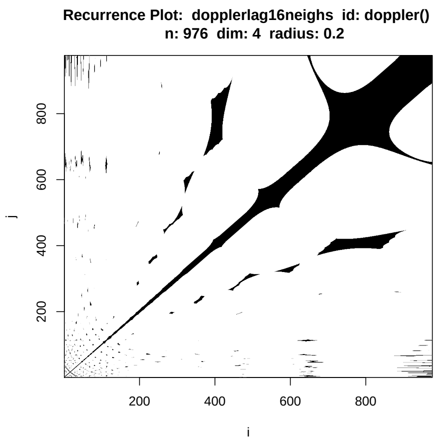


FIGURE 33. Recurrence plot. Test case: Doppler. Time used: 0.927 sec.

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

Output
```
doppler() n: 976 Dim: 4 Lag: 16 Radius: 0.2
 Recurrence coverage REC: 0.098 log(REC)/log(R): 1.443
Ratio 9.984234 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 Test case: Doppler. Time used: 1.181 sec. For graphical ouput, see Figure 34 on the next page.

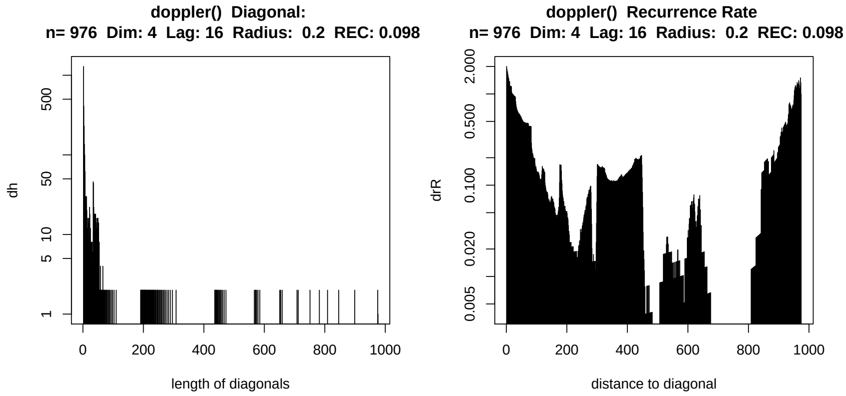


FIGURE 34. RQA. Test case: Doppler

7. Case Study: Geyser example and Bivariate recurrence plots

7.1. **Paired sequences.** 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(geyser), silent=TRUE)
library(MASS)
data(geyser)
attach(geyser)
```

To remove scale effects when comparing two sequences, we standardize both.

Input
```
duration <- duration - median(duration, na.rm=TRUE)
iqr_duration <- quantile(duration, probs = c(0.25, 0.75), na.rm = TRUE)
duration <- duration / (iqr_duration[2]-iqr_duration[1])
waiting <- waiting - median(waiting, na.rm=TRUE)
iqr_waiting <- quantile(waiting, probs = c(0.25, 0.75), na.rm = TRUE)
waiting <- waiting / (iqr_waiting[2]-iqr_waiting[1])
```

Input
```
plotsignal(duration)
```

Input
```
plotsignal(waiting)
```

Time used: 0.548 sec. (Standardized) signal components and linear interpolation. See Figure 35 on the facing page.

Input
```
oldpar <-  par(mfrow=c(1,2))
# taudelay estimation based on the autocorrelation function
tau.acf = timeLag(duration, technique = "acf", do.plot = T,
main=paste("taudelay estimation based on acf\n","duration()"))
# taudelay estimation based on the mutual information function
```

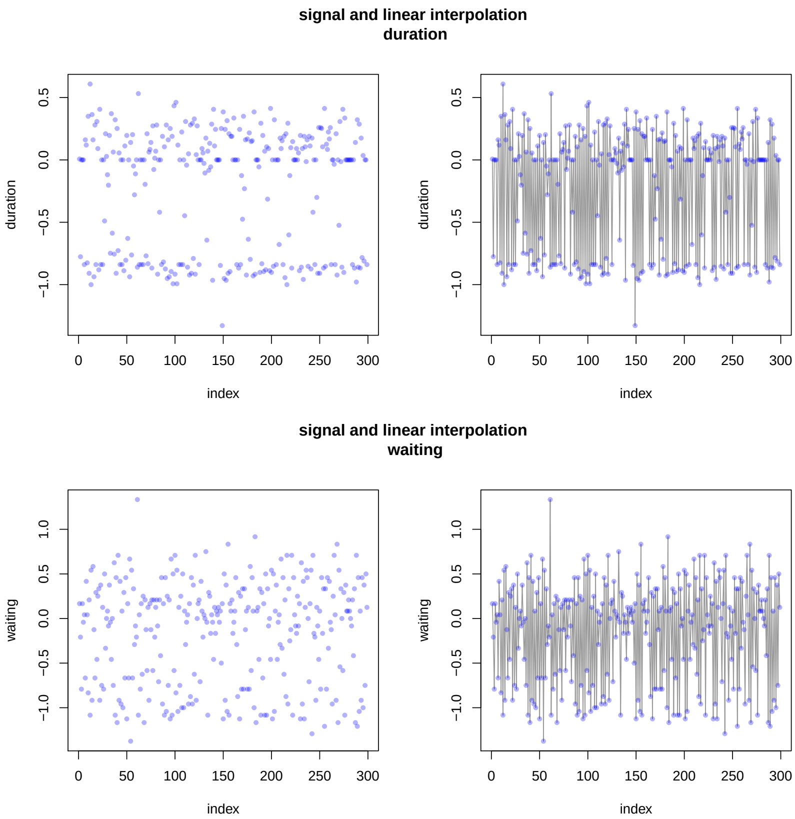
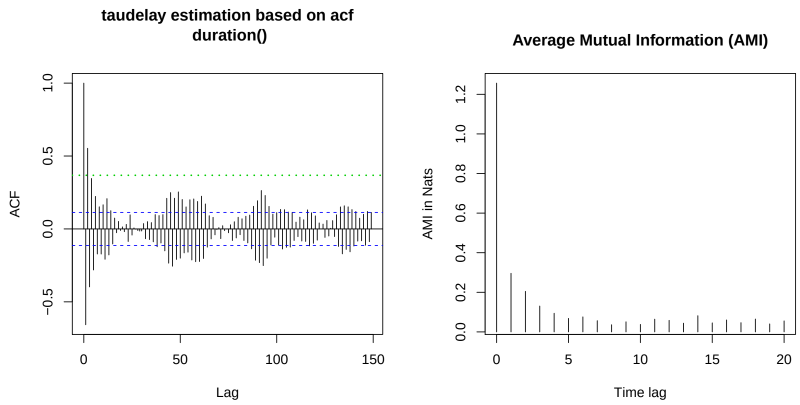


FIGURE 35. Example case: Old Faithful Geyser eruption standardized durations and waiting time. Signals and linear interpolation. Time used: 0.548 sec.

```
tau.ami=NA
#try(tau.ami <- timeLag(duration, technique = "ami", do.plot = T,
#main=paste("taudelay estimation based on ami\n","duration")) )
try(tau.ami <- timeLag(duration, technique = "ami",
        do.plot = T, selection.method="first.minimum",
main=paste("taudelay estimation based on ami\n","sin()")) )
par(oldpar)
cat("duration tau.acf:",tau.acf," tau.ami:",tau.ami)
```

Output

```
duration tau.acf: 1  tau.ami: 5
```



Input

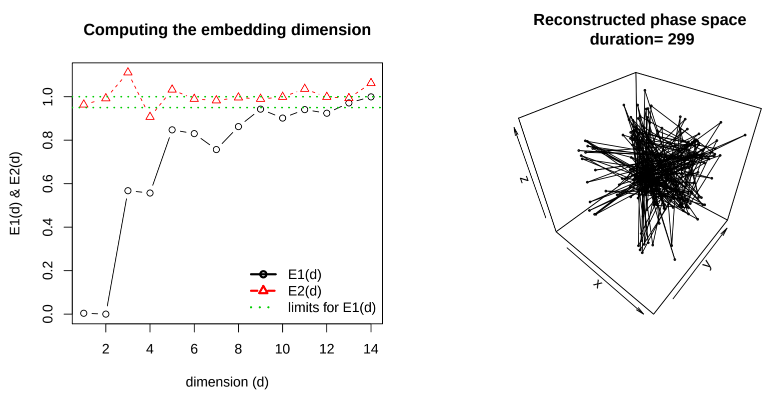
```
oldpar <-  par(mfrow=c(1,2))
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(duration, time.lag = tau.ami,
max.embedding.dim = 15)
cat("duration Lag tau.ami", tau.ami,"Estimated embedding dim:", emb.dim)
}  else  {
emb.dim = estimateEmbeddingDim(duration, time.lag = tau.acf,
max.embedding.dim = 15)
cat("duration Lag tau.acf", tau.acf,"Estimated embedding dim:", emb.dim)
}
```

Output

```
duration Lag tau.ami 5 Estimated embedding dim: 13
```

Input

```
#%<<takfsinN, fig=TRUE>>=
takduration = buildTakens(duration, embedding.dim = emb.dim, time.lag = tau.ami)
scatter3D(takduration[,1], takduration[,2], takduration[,3],
        main = paste("Reconstructed phase space\n duration=",length(duration)),
        col = 1, type="o",cex = 0.3)
par(oldpar)
```

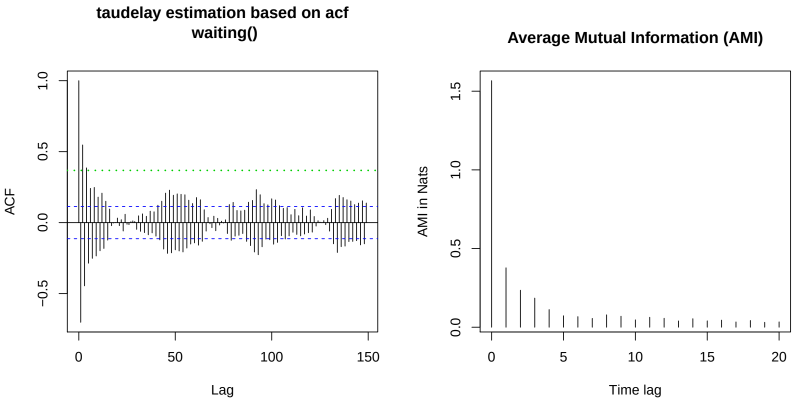


Input

```
oldpar <-  par(mfrow=c(1,2))
# taudelay estimation based on the autocorrelation function
```

```
tau.acf = timeLag(waiting, technique = "acf", do.plot = T,
main=paste("taudelay estimation based on acf\n","waiting()"))
# taudelay estimation based on the mutual information function
tau.ami=NA
#try(tau.ami <- timeLag(waiting, technique = "ami", do.plot = T,
#main=paste("taudelay estimation based on ami\n","waiting()")) )
try(tau.ami <- timeLag(waiting, technique = "ami",
        do.plot = T, selection.method="first.minimum",
main=paste("taudelay estimation based on ami\n","waiting()")) )
par(oldpar)
cat("waiting tau.acf:",tau.acf," tau.ami:",tau.ami)
```

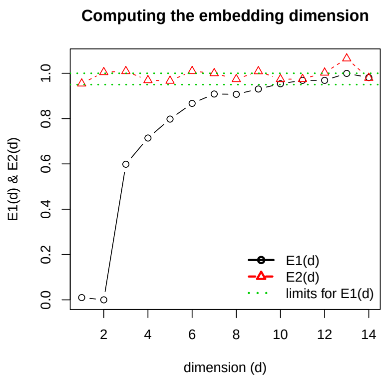
Output
```
waiting tau.acf: 1  tau.ami: 7
```

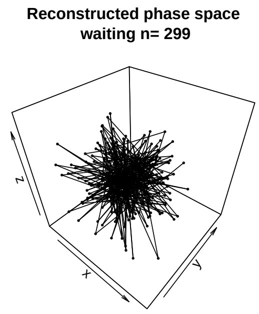


Input
```
oldpar <-  par(mfrow=c(1,2))
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(waiting, time.lag = tau.ami,
max.embedding.dim = 15)
cat("waiting Lag tau.ami", tau.ami,"Estimated embedding dim:", emb.dim)
}  else  {
emb.dim = estimateEmbeddingDim(waiting, time.lag = tau.acf,
max.embedding.dim = 15)
cat("waiting Lag tau.acf", tau.acf,"Estimated embedding dim:", emb.dim)
}
```

Output
```
waiting Lag tau.ami 7 Estimated embedding dim: 10
```

Input
```
#@
#%%
#%<<takfwaiting, fig=TRUE>>=
takwaiting = buildTakens(waiting, embedding.dim = emb.dim, time.lag = tau.ami)
scatter3D(takwaiting[,1], takwaiting[,2], takwaiting[,3],
#%main = paste("Reconstructed phase space\n",deparse(substitute(time.series))),
main = paste("Reconstructed phase space\n waiting n=",length(waiting)),
col = 1, type="o",cex = 0.3)
par(oldpar)
```





Input
```
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: 7.12 sec.

Input
```
duration4rqa <- showrqa(takens.duration4, radius=0.5, log=TRUE)
```

Output
```
duration  n: 296  Dim: 4  Lag: 1  Radius: 0.5
 Recurrence coverage REC: 0.168  log(REC)/log(R): 2.577
Ratio 5.628595 Determinism: 0.943  Laminarity: 0.07
DIV: 0.036
Trend: 0  Entropy: 1.937
Diagonal lines max: 28  Mean: 4.359  Mean off main: 4.267
Vertical lines max: 5  Mean: 3.065
```

Input
```
waiting4rqa <- showrqa(takens.waiting4, radius=0.5, log=TRUE)
```

Output
```
waiting  n: 296  Dim: 4  Lag: 1  Radius: 0.5
 Recurrence coverage REC: 0.112  log(REC)/log(R): 3.154
Ratio 8.004975 Determinism: 0.899  Laminarity: 0.088
DIV: 0.05
Trend: 0  Entropy: 1.8
Diagonal lines max: 20  Mean: 3.904  Mean off main: 3.775
Vertical lines max: 7  Mean: 3.024
```

Time used: 7.326 sec. For graphical ouput, see Figure 36 on the next page.

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

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

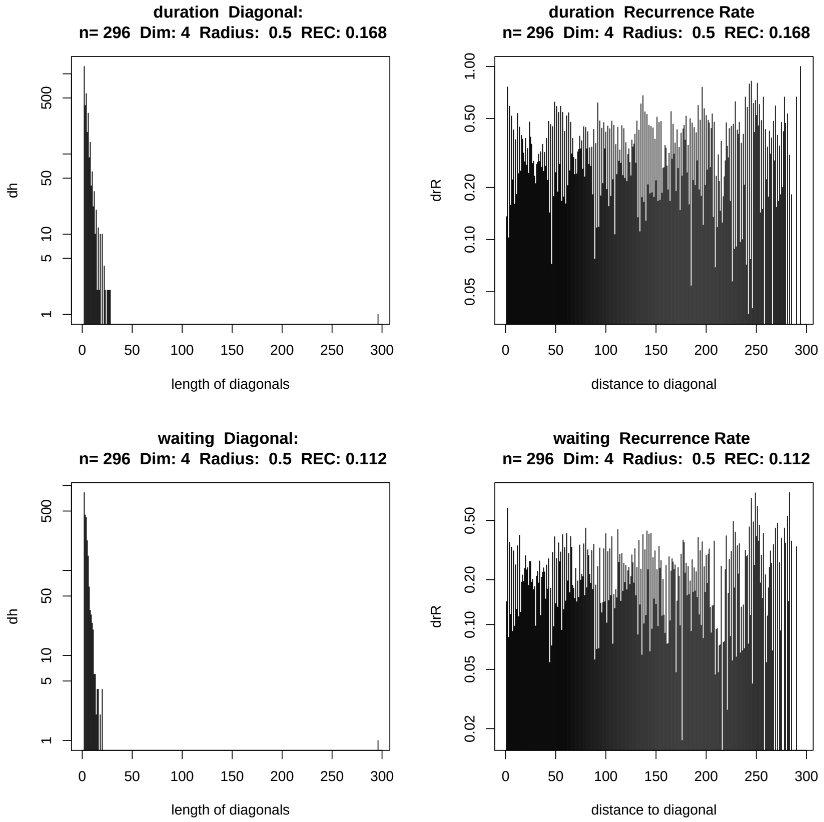


FIGURE 36. RQA. Left:Geyser duration. Radius=0.5, log scale. Right:Geyser waiting. Radius=0.5 log scale.

Example case: Old Faithful Geyser eruption data. Time used: 8.315 sec. See Figure 38 on the following page.

ToDo: use nooverlap ?

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

See Figure 37 on the next page.

Input
```
durationneighs4<- findAllNeighbours(takens.duration4, radius=0.5)
waitingneighs4<-findAllNeighbours(takens.waiting4, radius=0.5)
```

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

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

See Figure 39 on page 57.

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

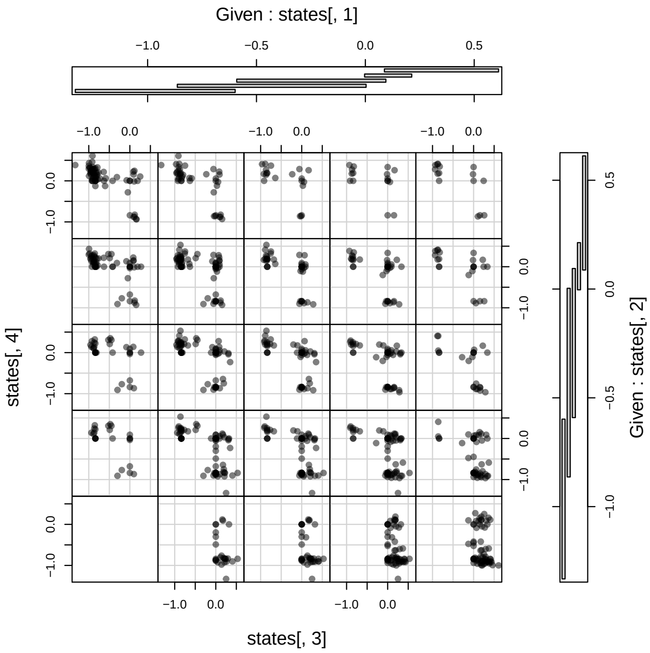


FIGURE 37. State coplot. Example case: Old Faithful Geyser eruption durations. Time used: 8.457 sec.

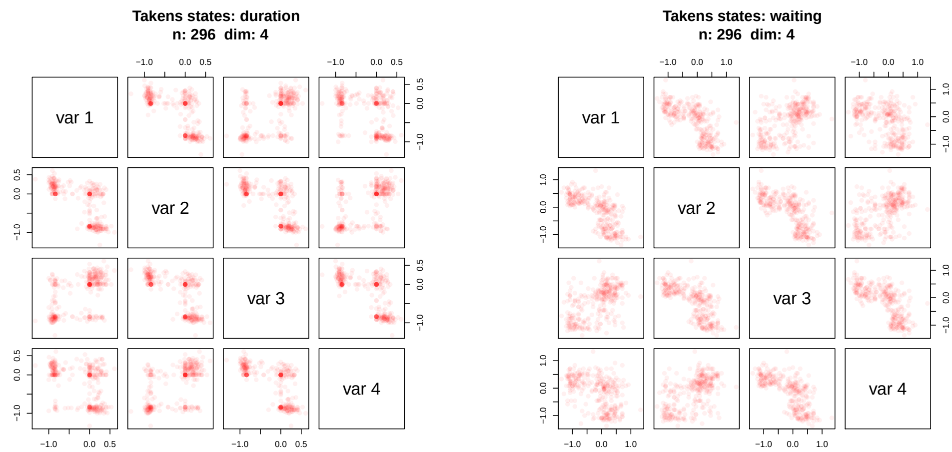


FIGURE 38. Old Faithful Geyser eruption durations and waiting time. Takens states, dim=4. Time used: 8.457 sec.

This assumes that correlated neighbours are collected in lists. Just for reference. This function is inlined.

Input
```
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
      }
```

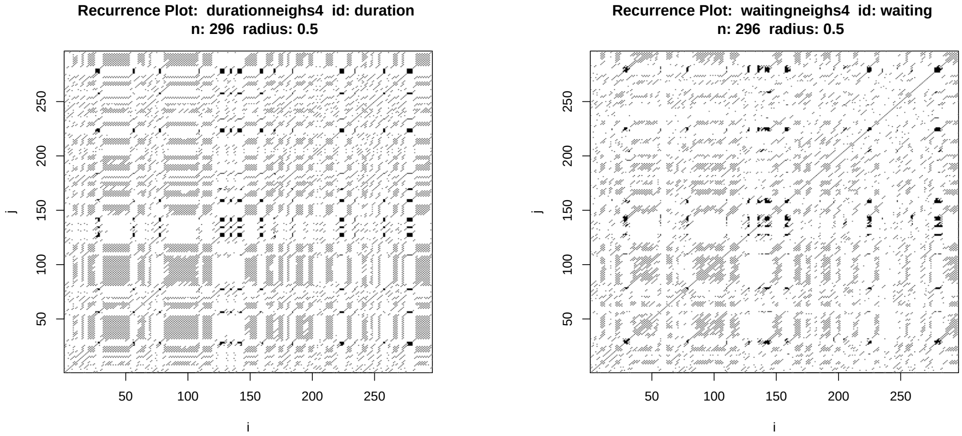


FIGURE 39. Old Faithful Geyser eruption durations and waiting time. Time used: 9.031 sec.

```
      }
    }
    return (neighs.matrix)
  }
```

Input

```
# univariate variant, assuming attributes
local.recurrencePlot1=function(neighs){
    dim <- attr(neighs,"embedding.dim")
    lag <- attr(neighs,"time.lag")
    radius <- attr(neighs,"radius")

  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 components 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)
```

Input

```
showrqa(takens.duration4, radius=0.5)
```

Output

```
duration  n: 296  Dim: 4  Lag: 1  Radius: 0.5
 Recurrence coverage REC: 0.168  log(REC)/log(R): 2.577
Ratio 5.628595 Determinism: 0.943  Laminarity: 0.07
DIV: 0.036
Trend: 0  Entropy: 1.937
Diagonal lines max: 28  Mean: 4.359  Mean off main: 4.267
Vertical lines max: 5  Mean: 3.065
```

RQA Example case: Old Faithful Geyser eruption durations. Radius=2.4 Dim=4. Time used: 9.538 sec. For graphical ouput, see Figure 40.

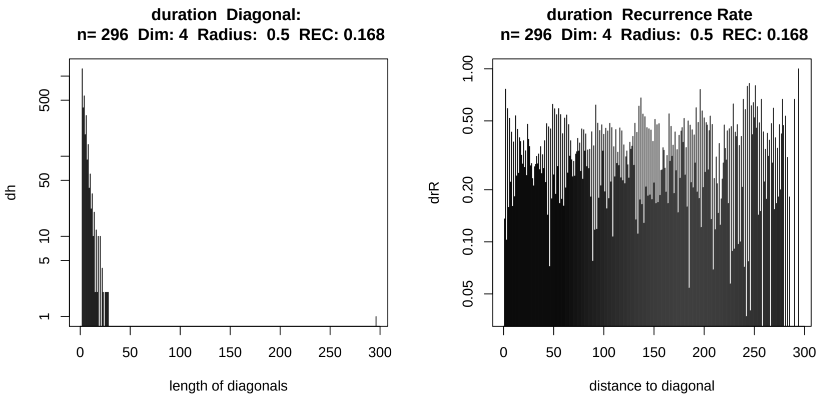


FIGURE 40. RQA. Example case: Old Faithful Geyser eruption durations. Radius=2.4 Dim=4

Input

```
showrqa(takens.waiting4, radius=0.5)
```

Output
```
waiting  n: 296  Dim: 4  Lag: 1  Radius: 0.5
 Recurrence coverage REC: 0.112  log(REC)/log(R): 3.154
Ratio 8.004975 Determinism: 0.899  Laminarity: 0.088
DIV: 0.05
Trend: 0  Entropy: 1.8
Diagonal lines max: 20  Mean: 3.904  Mean off main: 3.775
Vertical lines max: 7  Mean: 3.024
```

RQA Example case: Old Faithful Geyser eruption waiting. Radius=0.5 Dim=4. Time used: 9.611 sec. For graphical ouput, see Figure 41.

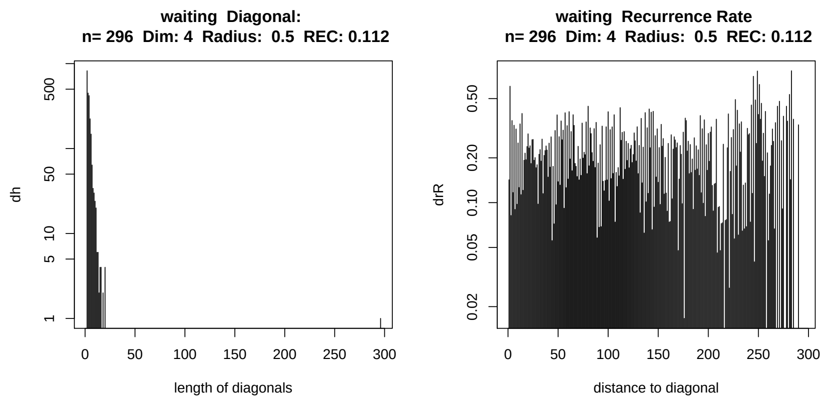


FIGURE 41. RQA. Example case: Old Faithful Geyser eruption waiting. Radius=0.5 Dim=4

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.

This assumes that correlated neighbours are collected in lists. Just for reference. This function is inlined.

Input
```
         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)
         }
```

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")


#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 42.

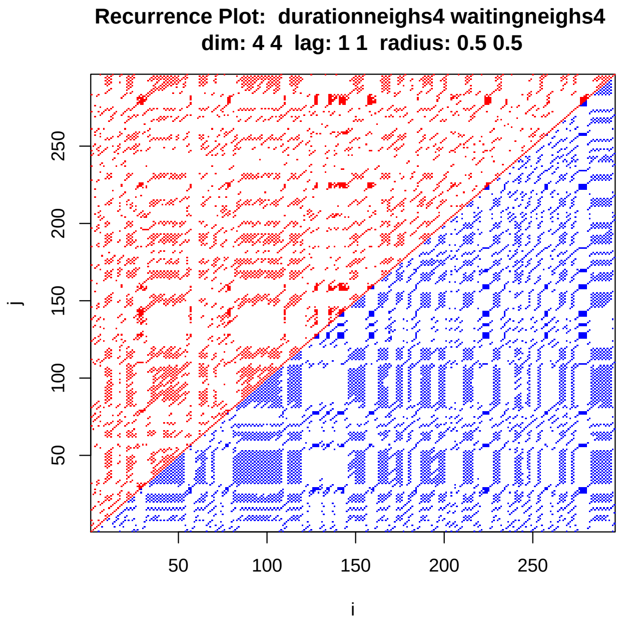


FIGURE 42. Recurrene plot: Lower triangle: duration. Upper triangle: waiting.

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. See Marwan and Kurths [2002], Marwan *et al.* [2002].

Marwan uses a "covariance"

$$CR[i,j] = \Theta(\epsilon - |\vec{x}_i - \vec{y}_j|)$$

with a maximum distance as $|\cdot|$, where $\vec{x}, \vec{y}$ are imbeddings. Other distances may be used. Signed distances for example may be used for experiments.

We use two examples:

Input

```
maxdist <- function (x){ max(abs(x), na.rm=TRUE)} # works on delta
cordist <- function (x, y){ suppressWarnings(cor(x,y))}
```

ToDo: Epsilon/radius and Heaviside not use here - thresholding is handled in image rendering.

ToDo: propagate names from Takens

Here is a brute force implementation.

Input
```
CR0 <- function (xtakens, ytakens, sdist= maxdist) {
# returns a cross distance matrix usin sdist
# using signed. Warnings for zero #
# variances are suppressed

        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)
}
```

ToDo: CR0 this needs optimisation on triangle

Variant $CR2$ allows a distance that is not factored via $x - y$.

Input
```
CR2 <- function (xtakens, ytakens, cdist= cordist) {
# returns a cross distance matrix usin cdist
        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)
}
```

Input
```
## 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)
        invisible(rec.plot)
```

```
}
```

Raw data may give a poor impression. Adjust e.g. for scale and location. This is done in this section for the Geyser data (see above).

Input
```
cr4 <- CR0(takens.duration4,takens.waiting4) # maxdist by default
cr4C <- CR2(takens.duration4,takens.waiting4) # cordist by default
```

Input
```
oldpar <- par(mfrow=c(1,2))
image(cr4, x=1:ncol(cr4), y=1:nrow(cr4),
        xlab="duration index", ylab="waiting index",main="CR0 maxdist")
image(cr4C, x=1:ncol(cr4C), y=1:nrow(cr4C),
        xlab="duration index", ylab="waiting index", main="CR2 cor distance")
# neighs.matrix <- cr4;range(cr4) #  70.5500 107.1667
par(oldpar)
cat("Range cr4:", range(cr4),"Range cr4C:", range(cr4C, na.rm=TRUE, finite=TRUE))
```

Output
```
Range cr4: 0.03350814 2.662005 Range cr4C: -1 1
```

See Figure 43.

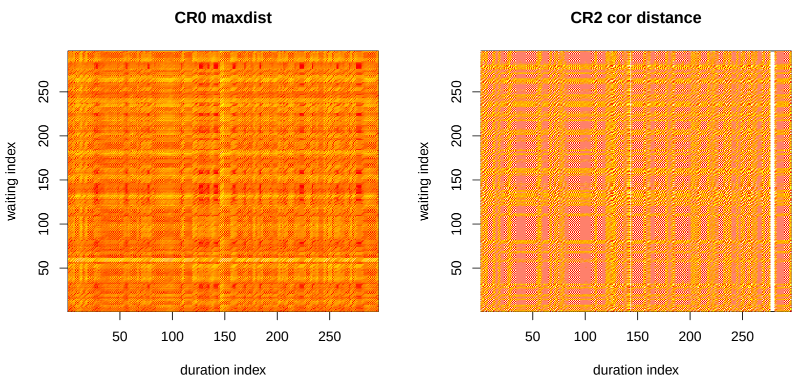


FIGURE 43. Recurrence plot. Cross recurrence. Left:using maxdist distance. Right:cor. Using default colour. Time used: 19.598 sec.

A max distance.

Input
```
oldpar <- par(mfrow=c(1,2))
crossrecurrencePlotFromMatrix(cr4,
        main="Cross Recurrence plot\n max",
        xlab="duration4 index", ylab="waiting4 index")  # ugly heat matrix
```

Output
```
NULL
```

Input

```
#@
#\insertrecurrence{CRPFromMatrix}{Cross recurrence ??using max distance}{\Sexpr{laptime()}}
#
#<<crossrecurrencePlot, fig=TRUE, include=FALSE>>=
crossrecurrencePlotFromMatrix(cr4,  col=grey((1:10)/10),
main="Cross Recurrence plot\nmax",
xlab="duration4 index", ylab="waiting4 index")
```

Output

```
NULL
```

Input

```
# near conventional bw,
# introducing radius/cut by zlim
par(oldpar)
```

See Figure 44.

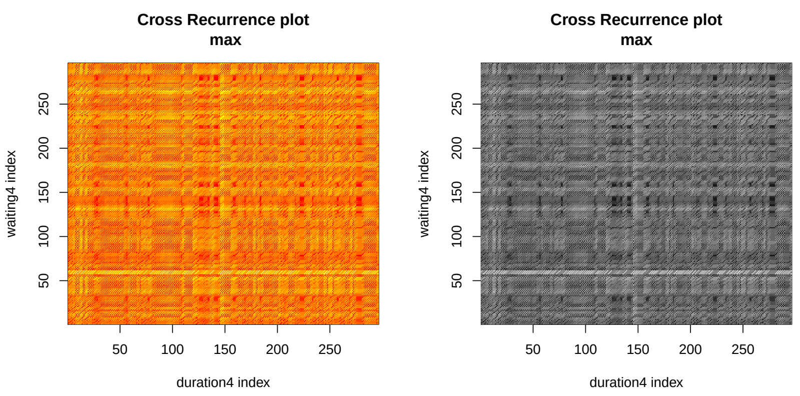


FIGURE 44. Recurrence plot. Cross recurrence using max distance - default colours and grey scale. Time used: 22.256 sec.

Input

```
oldpar <- par(mfrow=c(1,2))
# a correlation distance
crossrecurrencePlotFromMatrix(cr4C,
main="Cross Recurrence plot\n cor",
xlab="duration4 index", ylab="waiting4 index")  # ugly heat matrix
```

Output

```
NULL
```

Input

```
#<<CRPFromMatrixC, fig=TRUE, include=FALSE>>=
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
```

Input
```
# near conventional
# introducing radius/cut by zlim
par(oldpar)
```

See Figure 45.

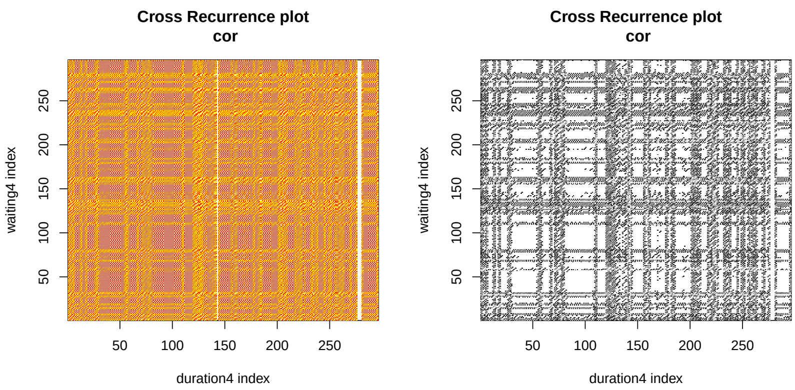


FIGURE 45. Recurrence plot. Cross recurrence using cor distance - default colours and grey scale. Time used: 24.456 sec.

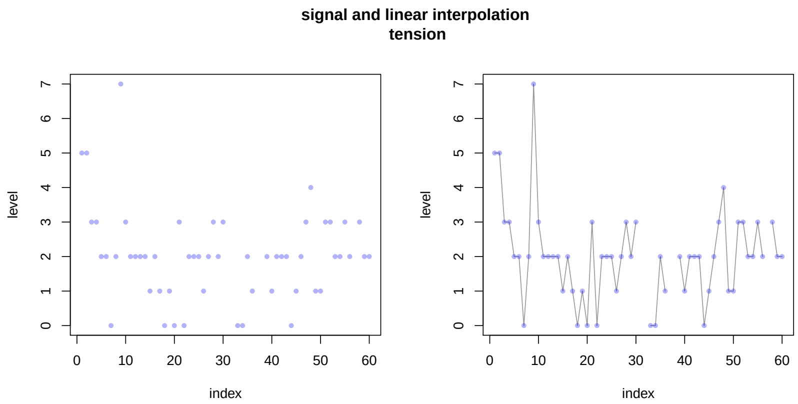
8. Case Study: Tension data

8.1. Raw data.

Note for data analysis: a logistic transform is used in 8.2.

The tension data set (from [Ebner-Priemer *et al.*, 2007], [Ebner-Priemerx and Sawitzki, 2007]) is one sample case of recordings of psychological tension data, recorded in regular interval on a subjective tension level ranked $0 \ldots 10$. The data are ranked data, and may contain missing data. Missing data are just removed here.

Input
```
tension <- scan("../media/tension.dat")
plotsignal(tension,"tension","level")
tension<-tension[!is.na(tension)]
```



ToDo: fix segfault in `mutualInformation()` if NA is included

Input
```
oldpar <-  par(mfrow=c(1,2))
# taudelay estimation based on the autocorrelation function
        tau.acf = timeLag(tension, technique = "acf", do.plot = T,
        main=paste("$\tau_{delay}$ estimation based on acf\n","tension()"))
# taudelay estimation based on the mutual information function
tau.ami=NA
try(tau.ami <- timeLag(tension, technique = "ami",
        do.plot = T, selection.method="first.minimum",
        main=paste("$\tau_{delay}$ estimation based on ami\n","tension()")) )
par(oldpar)
cat("tension tau.acf:",tau.acf," tau.ami:",tau.ami)
```

Output
```
tension tau.acf: 1  tau.ami: 1
```

See Figure 46 on the next page.

Input
```
oldpar <-  par(mfrow=c(1,2))
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(tension, time.lag = tau.ami,
        max.embedding.dim = 15)
cat("tension() Lag tau.ami", tau.ami,"Estimated embedding dim:", emb.dim)
```

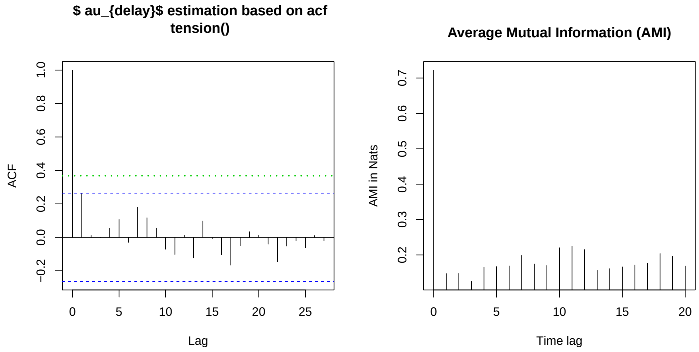


FIGURE 46. τ_{delay} estimation. "Tension" Left: based on the autocorrelation function ACF. Right: AMI

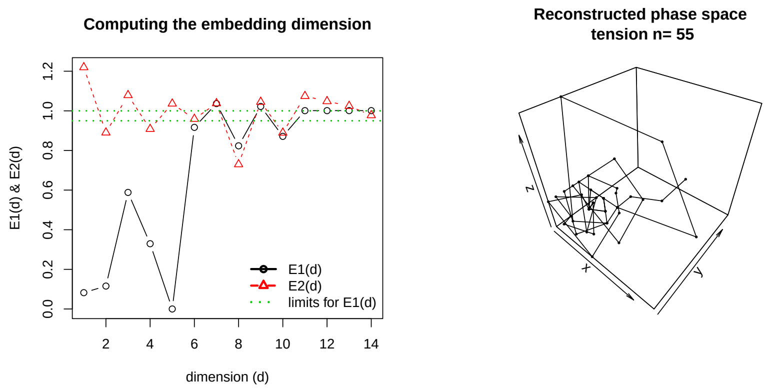
```
} else {
        emb.dim = estimateEmbeddingDim(tension, time.lag = tau.acf,
        max.embedding.dim = 15)
cat("tension() Lag tau.acf", tau.acf,"Estimated embedding dim:", emb.dim)
}
```

Output

```
tension() Lag tau.ami 1 Estimated embedding dim: 9
```

Input

```
#%<<takfsinN, fig=TRUE>>=
# taktension = buildTakens(tension, embedding.dim = 6, time.lag = 1)
taktension = buildTakens(tension, embedding.dim = emb.dim, time.lag = tau.ami)
scatter3D(taktension[,1], taktension[,2], taktension[,3],
#%main = paste("Reconstructed phase space\n",deparse(substitute(time.series))),
        main = paste("Reconstructed phase space\n tension n=",length(tension)),
        col = 1, type="o",cex = 0.3)
par(oldpar)
```



Input

```
tensionneighs <- local.findAllNeighbours(taktension, radius=2.0)
save(tensionneighs, file="tensionneighs.Rdata")
 # load(file="tensionneighs.Rdata")
local.recurrencePlotAux(tensionneighs, dim=dim(taktension)[2], radius=2.0)
```

See Figure 47.

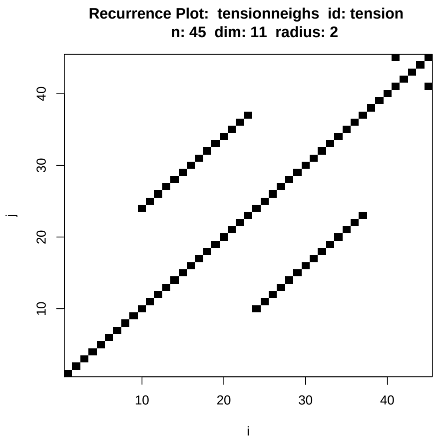


FIGURE 47. Recurrence plot. Test case: tension. Time used: 26.291 sec.

Input
```
tensionrqa <- showrqa(taktension, radius=2.0, log=TRUE)
```

Output
```
tension  n: 45  Dim: 11  Lag: 1  Radius: 2
 Recurrence coverage REC: 0.037  log(REC)/log(R): -4.755
Ratio 26.28 Determinism: 0.973  Laminarity: 0
DIV: 0.071
Trend: -0.001  Entropy: 0.637
Diagonal lines max: 14  Mean: 24.333  Mean off main: 14
Vertical lines max: 0  Mean: 0
```

RQA Test case: tension. Radius=2.0. Time used: 26.34 sec. For graphical ouput, see Figure 48.

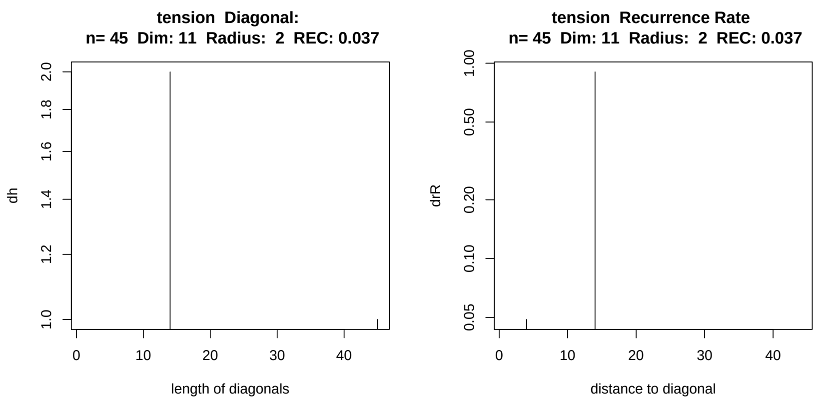


FIGURE 48. RQA. Test case: tension. Radius=2.0

8.2. **Tension: transformed scale.** Try rescaled logits:

Input
```
k=11 # levels 0..11
#
tension1 <- (tension+1)/(k+2)
tension1 <- log(tension1/(1-tension1))
```

Input
```
oldpar <-  par(mfrow=c(1,2))
# taudelay estimation based on the autocorrelation function
        tau.acf = timeLag(tension1, technique = "acf", do.plot = T,
        main=paste("$\tau_{delay}$ estimation based on acf\n","tension1()"))
# taudelay estimation based on the mutual information function
tau.ami=NA
try(tau.ami <- timeLag(tension1, technique = "ami",
        do.plot = T, selection.method="first.minimum",
        main=paste("$\tau_{delay}$ estimation based on ami\n","tension1()")) )
par(oldpar)
cat("tension rescaled. tau.acf:",tau.acf," tau.ami:",tau.ami)
```

Output
```
tension rescaled. tau.acf: 1  tau.ami: 1
```

See Figure 49.

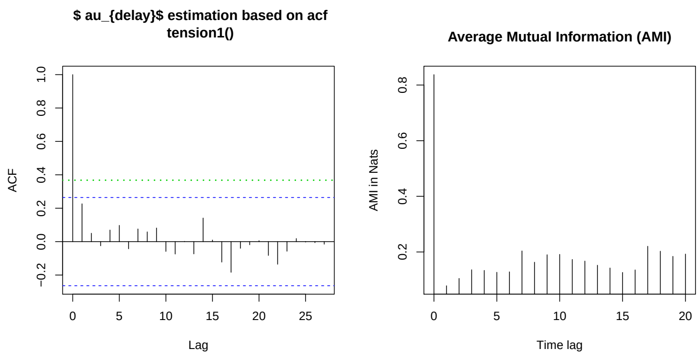
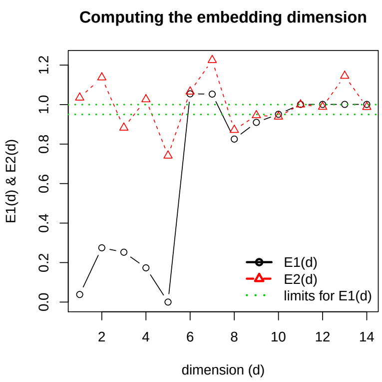


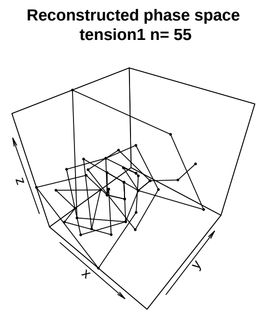
FIGURE 49. τ_{delay} estimation. "tension1" Left: based on the autocorrelation function ACF. Right: AMI

Input
```
oldpar <-  par(mfrow=c(1,2))
if (is.numeric(tau.ami)) {
emb.dim = estimateEmbeddingDim(tension1, time.lag = tau.ami,
        max.embedding.dim = 15)
cat("tension1() Lag tau.ami", tau.ami,"Estimated embedding dim:", emb.dim)
}  else  {
emb.dim = estimateEmbeddingDim(tension1, time.lag = tau.acf,
        max.embedding.dim = 15)
cat("tension1() Lag tau.acf", tau.acf,"Estimated embedding dim:", emb.dim)
}
```

Output
```
tension1() Lag tau.ami 1 Estimated embedding dim: 9
```

Input
```
#%<<takfsinN, fig=TRUE>>=
taktension1 = buildTakens(tension1, embedding.dim = emb.dim, time.lag = tau.ami)
scatter3D(taktension1[,1], taktension1[,2], taktension1[,3],
        main = paste("Reconstructed phase space\n tension1 n=",length(tension1)),
        col = 1, type="o",cex = 0.3)
par(oldpar)
```





Input
```
tension1neighs <- local.findAllNeighbours(taktension1, radius=1.4)
save(tension1neighs, file="tension1neighs.Rdata")
# load(file="tension1neighs.Rdata")
local.recurrencePlotAux(tension1neighs, dim=dim(taktension1)[2], radius=1.4)
```

See Figure 50.

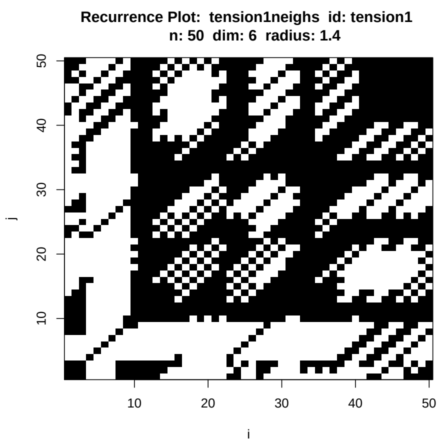


FIGURE 50. Recurrence plot. Test case: tension1. Time used: 28.287 sec.

Input
```
tension1rqa <- showrqa(taktension1, radius=2, log=TRUE)
```

Output
```
tension1  n: 50  Dim: 6  Lag: 1  Radius: 2
 Recurrence coverage REC: 0.877  log(REC)/log(R): -0.19
Ratio 1.137389 Determinism: 0.997  Laminarity: 0.971
```

```
DIV: 0.02
Trend: 0.001  Entropy: 3.137
Diagonal lines max: 49  Mean: 15.956  Mean off main: 15.706
Vertical lines max: 50  Mean: 11.628
```

RQA Test case: tension1. Radius=1.4. log scale. Time used: 28.545 sec. For graphical ouput, see Figure 51.

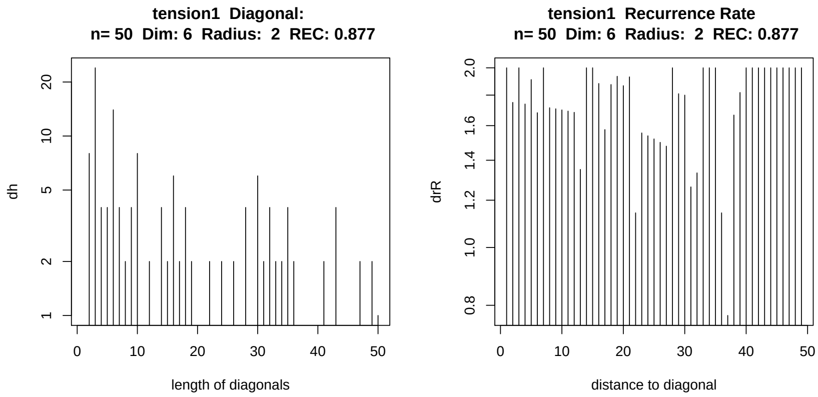


FIGURE 51. RQA. Test case: tension1. Radius=1.4. log scale

Only 1024 data points used in recurrence plots in this section

9. Case Study: HRV data example.beats

Note for data analysis: The data include missing beats, or spurious artefacts. These need to be handled. Graphical display is heavily affected by the scales implied by outliers.

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)
```

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

See Figure 52.

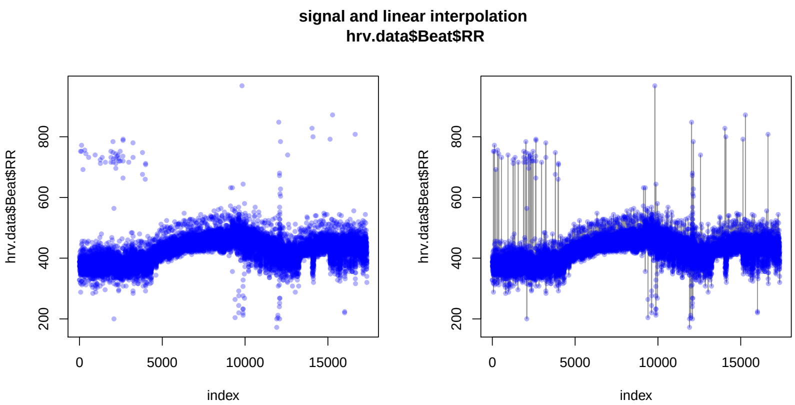


Figure 52. RHRV tutorial example.beats. Signal and linear interpolation.

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

See Figure 53.

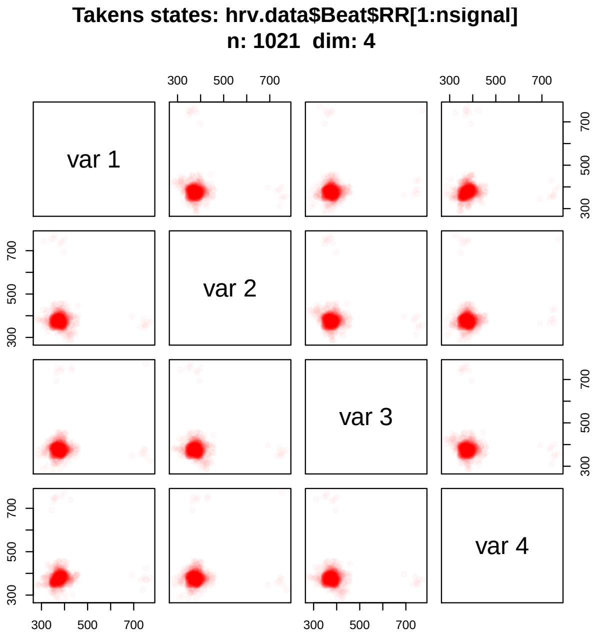


FIGURE 53. Takens states. RHRV tutorial example.beats.

0.72

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

See Figure 54 on the next page.

1.767

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

See Figure 55 on the following page.

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

See Figure 56 on page 75.

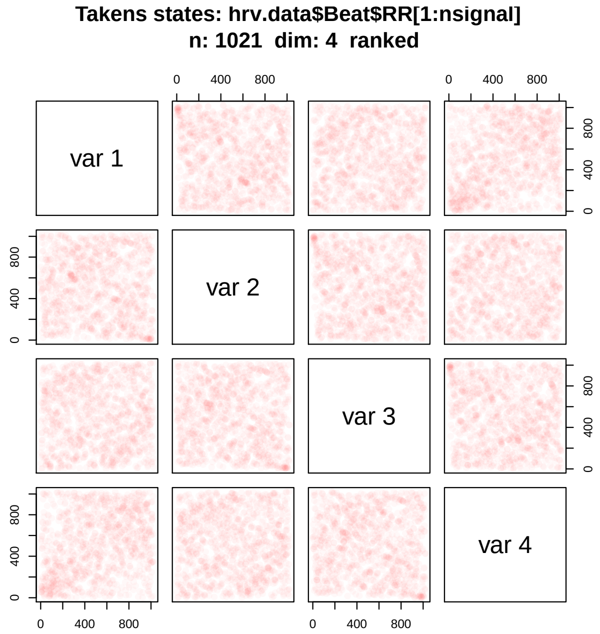


FIGURE 54. Takens states. RHRV tutorial example.beats. Ranked data

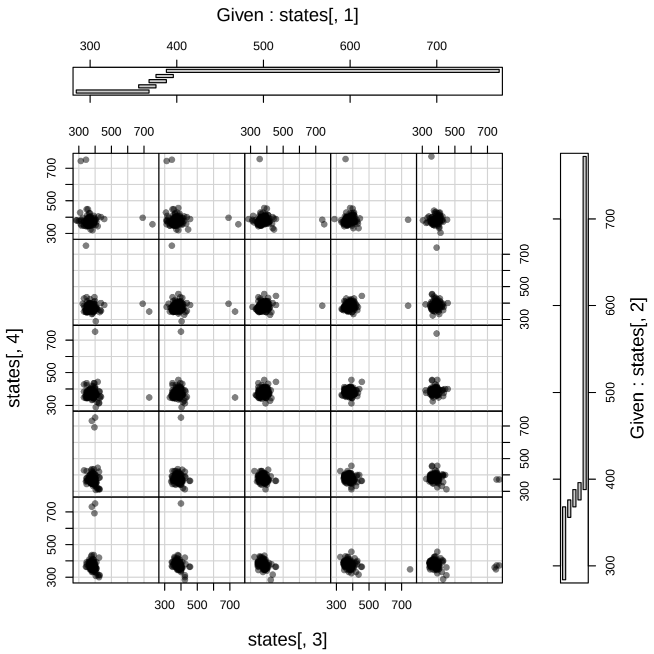


FIGURE 55. State coplot. RHRV tutorial example.beats. Time used: 0.23 sec.

Input

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

Time used: 0.574 sec.

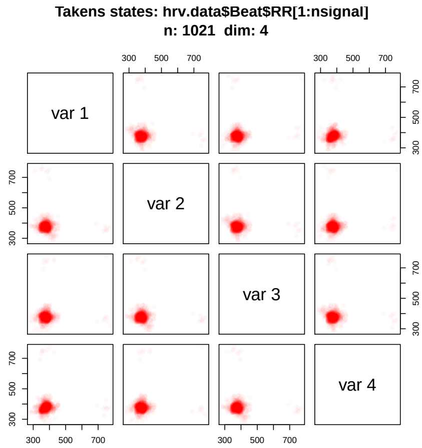


FIGURE 56. Recurrence plot. RHRV tutorial example.beats. Time used: 0.516 sec.

Input

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

See Figure 57.

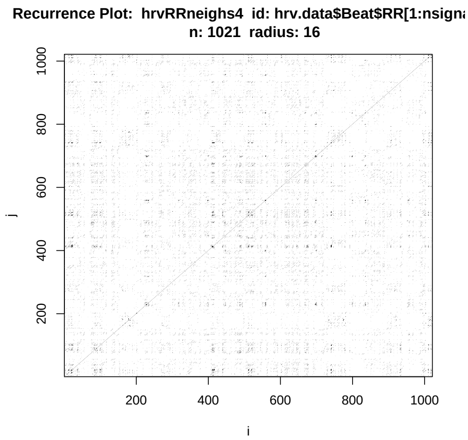


FIGURE 57. Recurrence plot. Example: RHRV tutorial example.beats. Dim=4.. Time used: 0.903 sec.

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=TRUE, radius=16)
```

Output
```
hrv.data$Beat$RR[1:nsignal]  n: 1023  Dim: 2  Lag: 1  Radius: 16
 Recurrence coverage REC: 0.165  log(REC)/log(R): -0.65
Ratio 3.945784 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 58 on page 78.

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: 0.777 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: 0.996 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: 1.93 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: 1.331 sec.

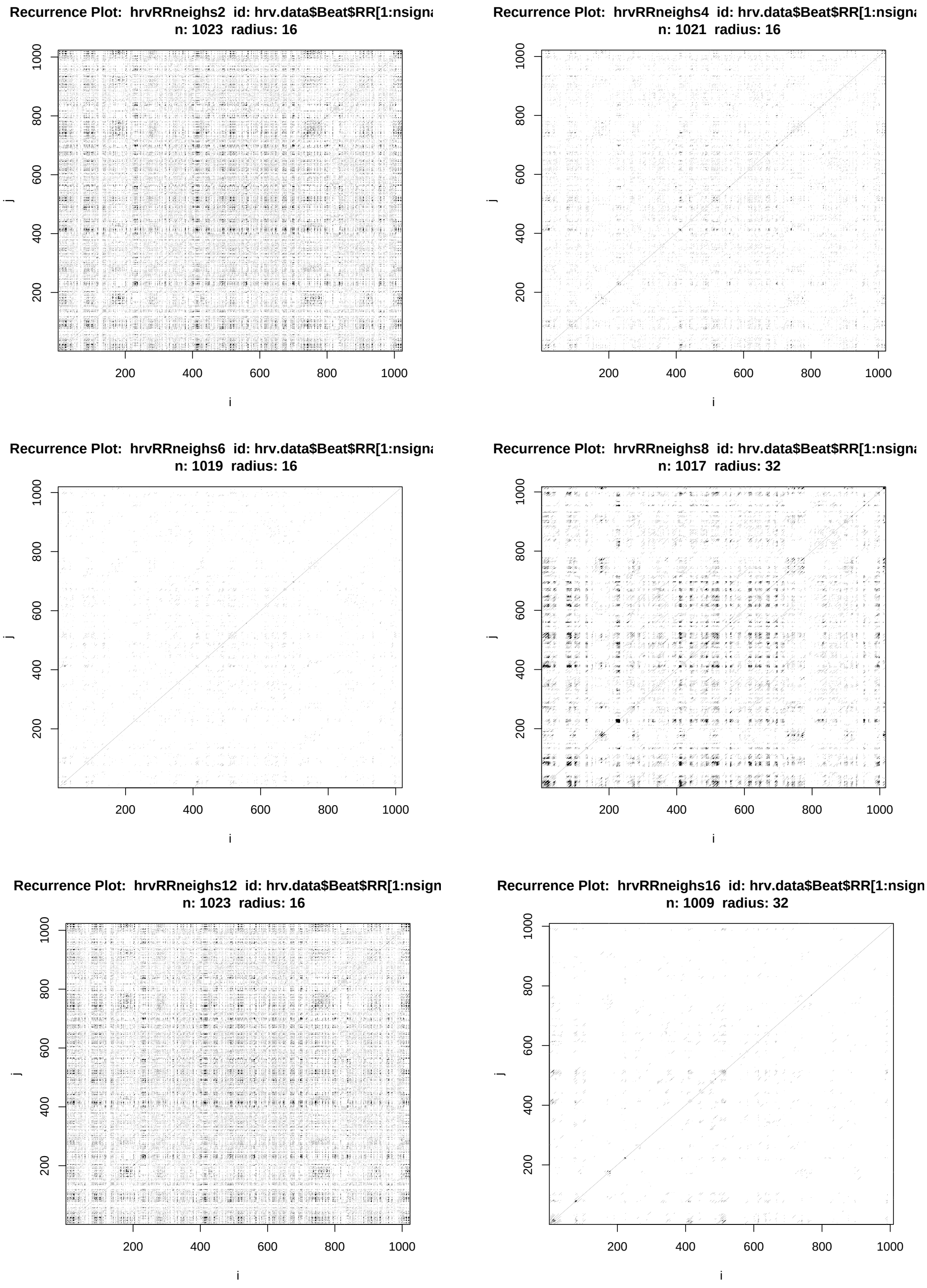


Figure 58. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 1.331 sec.

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 59,

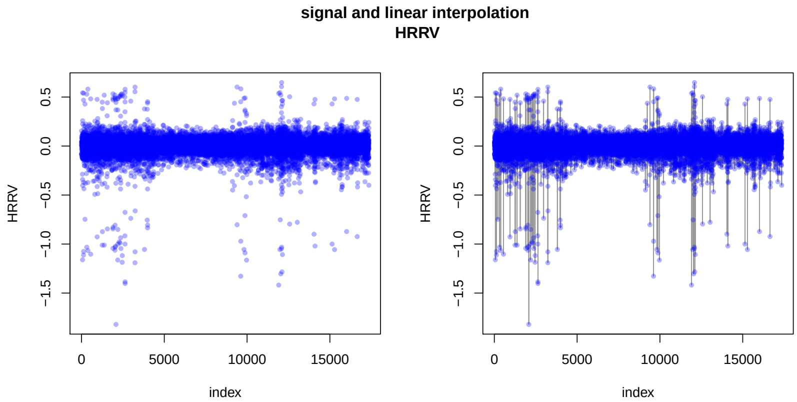


FIGURE 59. RHRV tutorial example.beats. HRRV Signal and linear interpolation.

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 60 on the next page.

1.055

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

See Figure 61 on the facing page.

ToDo: findAllNeighbours does not handle NAs

1.7

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

Time used: 0.151 sec.

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

ToDo: check. There seem to be strange artefacts.

See Figure 62 on page 82.

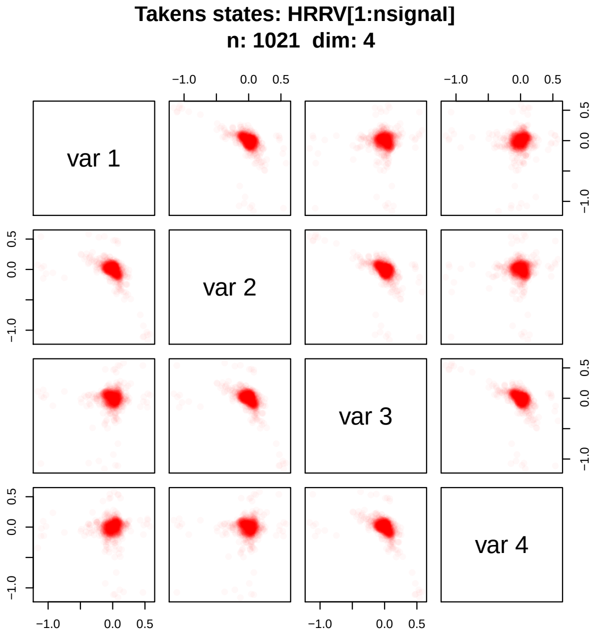


FIGURE 60. Takens states. RHRV tutorial example.beats. HRRV

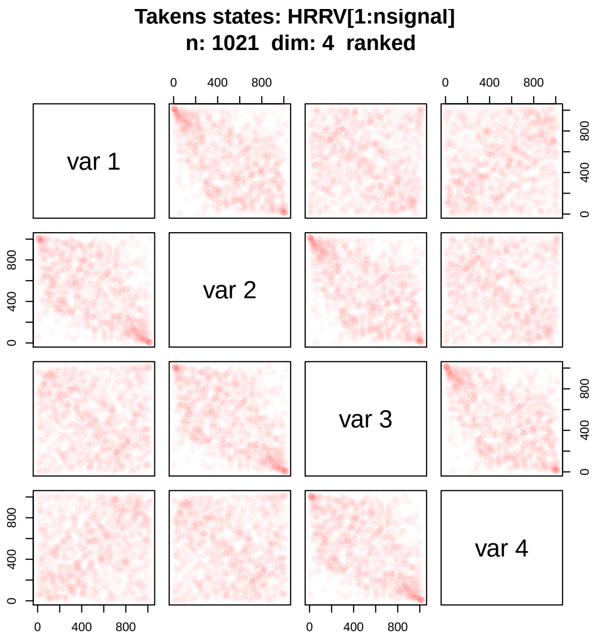


FIGURE 61. Takens states. RHRV tutorial example.beats ranked HRRV data

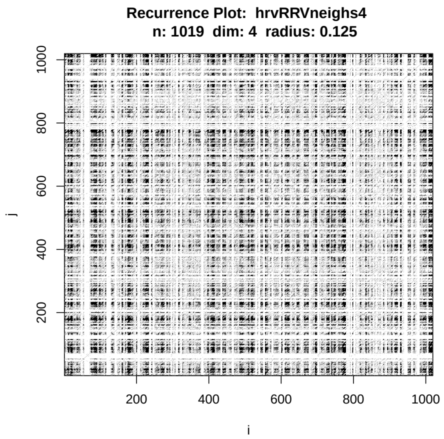


FIGURE 62. Recurrence plot. RHRV tutorial example.beats. HRRV Dim=4. Time used: 1.417 sec.

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)
```

Dim=2. Time used: 2.192 sec. See figure 68

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

Output
```
hrvRRVtakens2[-(1:2), ]  n: 1021  Dim: 2  Lag:  Radius: 0.125
 Recurrence coverage REC: 0.515  log(REC)/log(R): 0.319
Ratio 1.824474 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
```

RQA HRRV Dim=2. Time used: 3.247 sec. For graphical ouput, see Figure 63 on the following page.

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)
```

Dim=8. Time used: 1.491 sec. See figure 68

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

Output
```
hrvRRVtakens6[-(1:2), ]  n: 1017  Dim: 6  Lag:  Radius: 0.125
 Recurrence coverage REC: 0.179  log(REC)/log(R): 0.827
Ratio 5.261147 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
```

RQA HRRV Dim=6. Time used: 1.85 sec. For graphical ouput, see Figure 64 on the next page.

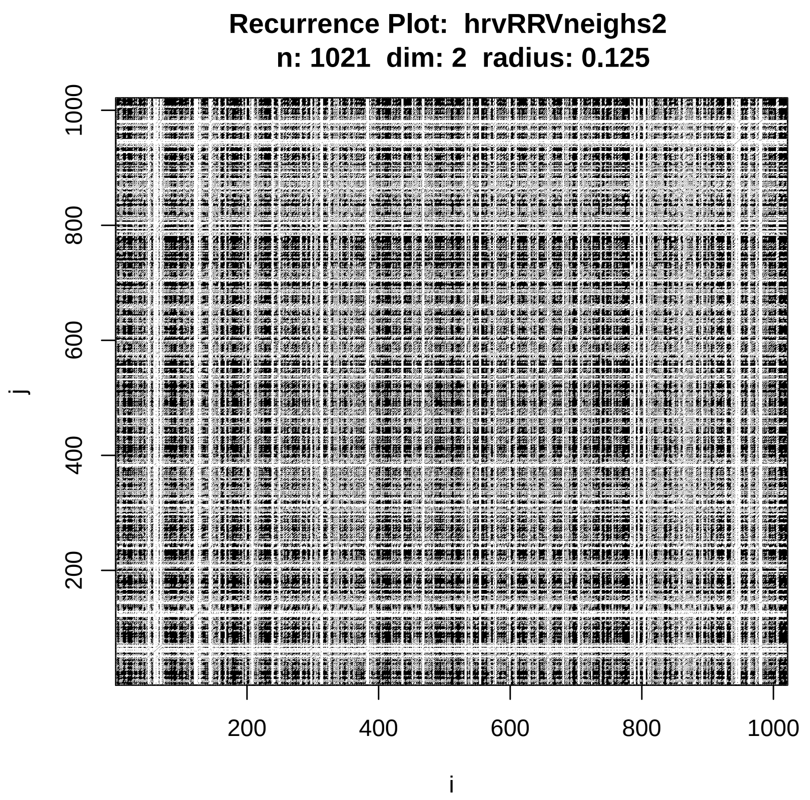


FIGURE 63. RQA. HRRV Dim=2

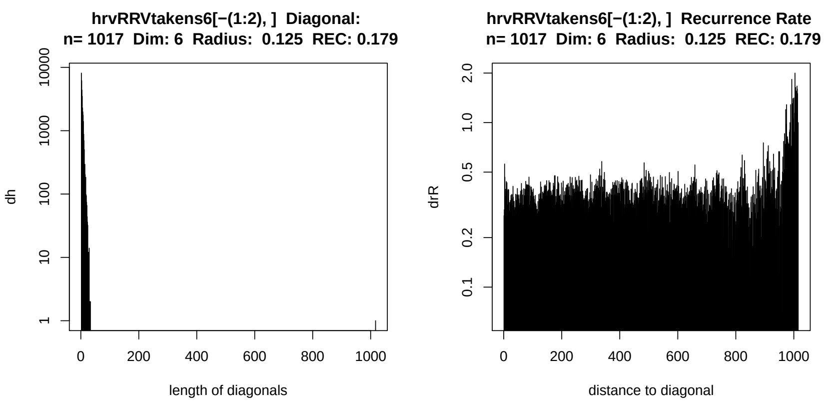


FIGURE 64. RQA. HRRV Dim=6

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)
```

Dim=8. Time used: 1.513 sec. See figure 68

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

Output
```
hrvRRVtakens8[-(1:2), ]  n: 1015  Dim: 8  Lag:  Radius: 0.125
 Recurrence coverage REC: 0.105  log(REC)/log(R): 1.084
Ratio 8.977958 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
```

RQA HRRV Dim=8. Time used: 2.212 sec. For graphical ouput, see Figure 65.

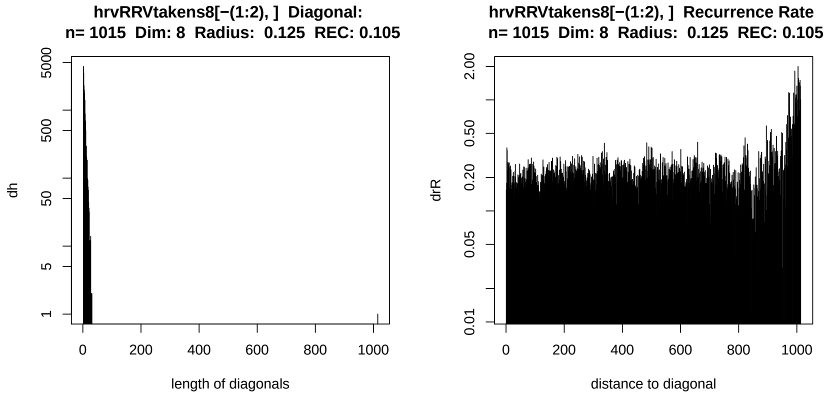


FIGURE 65. RQA. HRRV Dim=8

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)
```

Dim=12. Time used: 1.986 sec. See figure 68

Input
```
showrqa(hrvRRVtakens2[-(1:2),], radius=3/16, do.hist=TRUE)
```

Output
```
hrvRRVtakens2[-(1:2), ]  n: 1021  Dim: 2  Lag:  Radius: 0.1875
 Recurrence coverage REC: 0.75  log(REC)/log(R): 0.172
Ratio 1.312218 Determinism: 0.984  Laminarity: 0.951
DIV: 0.013
Trend: 0  Entropy: 3.136
Diagonal lines max: 78  Mean: 10.13  Mean off main: 10.117
Vertical lines max: 85  Mean: 8.721
```

RQA HRRV Dim=12. Time used: 3.272 sec. For graphical ouput, see Figure 67 on the next page.

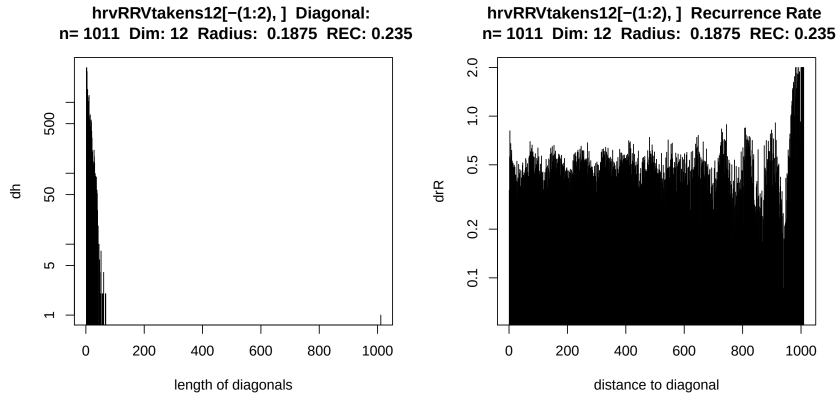


FIGURE 66. RQA. HRRV Dim=12

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)
```

Dim=12. Time used: 2.098 sec. See figure 68

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

Output
```
hrvRRVtakens12[-(1:2), ]  n: 1011  Dim: 12  Lag:  Radius: 0.1875
 Recurrence coverage REC: 0.235  log(REC)/log(R): 0.865
Ratio 4.202561 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
```

RQA HRRV Dim=12. Time used: 3.038 sec. For graphical ouput, see Figure 67 on the facing page.

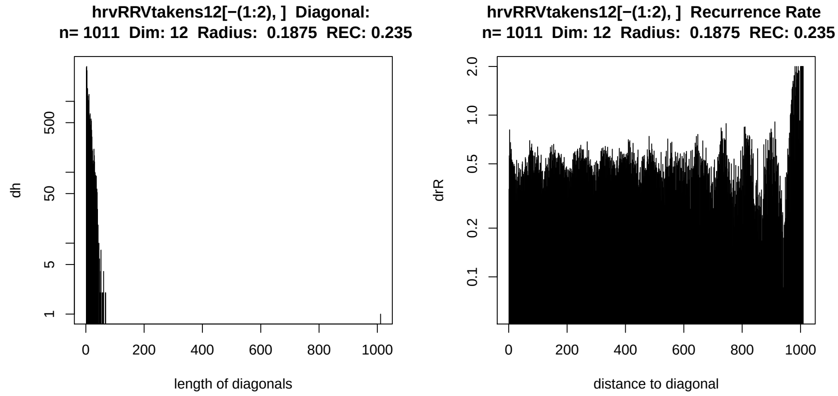


FIGURE 67. RQA. HRRV Dim=12

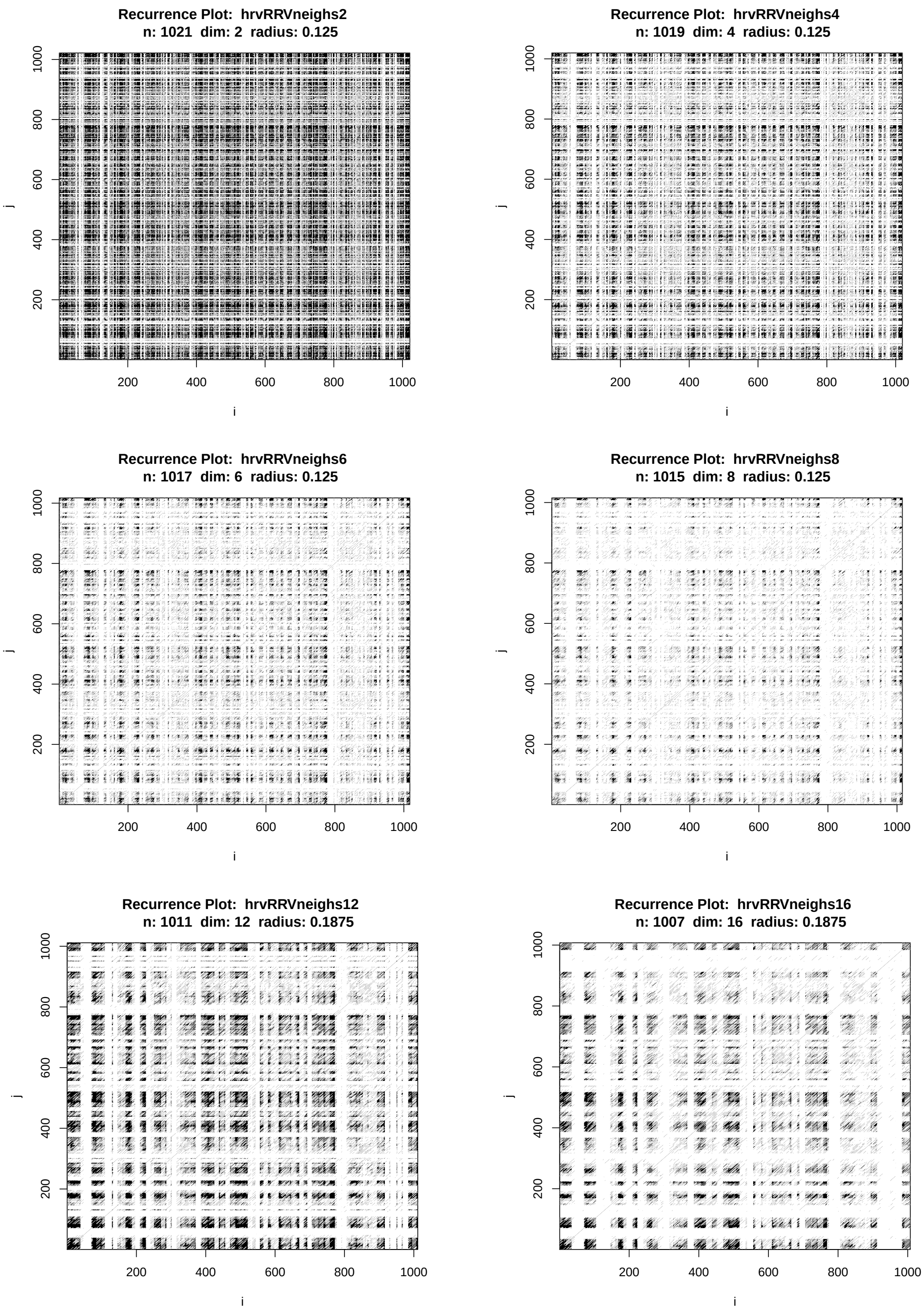


FIGURE 68. Recurrence Plot. Example case: RHRV tutorial example.beats variation. Dim=2, 4, 6, 8, 12, 16. Time used: 3.038 sec.

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)
```

Dim=2, lag=8. Time used: 1.904 sec. See figure 76.

Input
```
showrqa(hrvRRVtakens2Lag08[-(1:2),], radius=0.125, do.hist=TRUE)
```

Output
```
hrvRRVtakens2Lag08[-(1:2), ]  n: 1014  Dim: 2  Lag:  Radius: 0.125
 Recurrence coverage REC: 0.469  log(REC)/log(R): 0.364
Ratio 1.726809 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
```

RQA HRRV Dim=2, lag=8. Time used: 2.906 sec. For graphical ouput, see Figure 69.

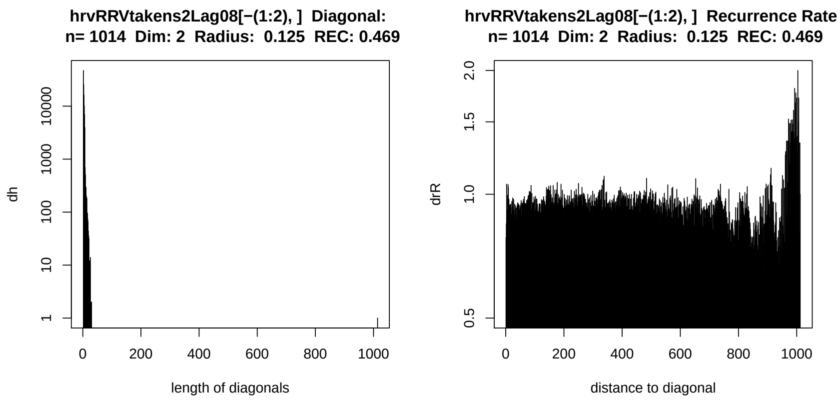


FIGURE 69. RQA. HRRV Dim=2, lag=8

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)
```

Dim=4, lag=8. Time used: 1.517 sec. See figure 76.

Input
```
showrqa(hrvRRVtakens4Lag08[-(1:2),], radius=0.125, do.hist=TRUE)
```

Output
```
hrvRRVtakens4Lag08[-(1:2), ]  n: 998  Dim: 4  Lag:  Radius: 0.125
 Recurrence coverage REC: 0.22  log(REC)/log(R): 0.729
Ratio 2.550804 Determinism: 0.56  Laminarity: 0.53
DIV: 0.071
Trend: 0  Entropy: 0.994
Diagonal lines max: 14  Mean: 2.559  Mean off main: 2.538
Vertical lines max: 25  Mean: 2.576
```

Input
```
statepairs(hrvRRVtakens4Lag08)
```

Input
```
statecoplot(hrvRRVtakens4Lag08)
```

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

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

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

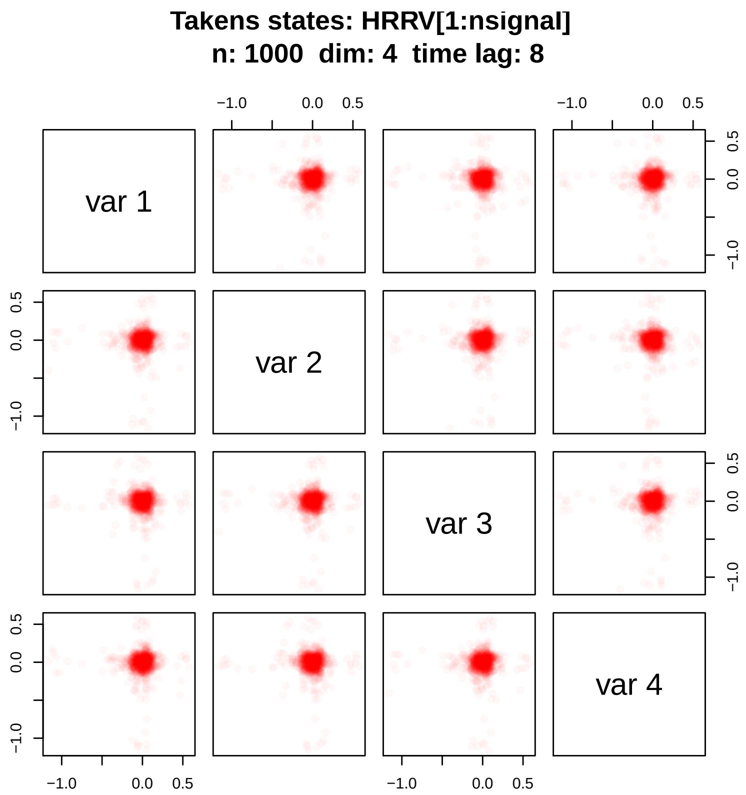
Output
```
 Missing rows: 1, 2, 38, 39, 46, 47, 54, 55, 62, 63, 100, 101, 102, 108, 109, 110, 116, 117, 118, 124, 125
```

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)
```

Dim=6, lag=8. Time used: 6.283 sec. See figure 76.

Input
```
showrqa(hrvRRVtakens6Lag08[-(1:2),], radius=0.125, do.hist=TRUE)
```

Output
```
hrvRRVtakens6Lag08[-(1:2), ]  n: 982  Dim: 6  Lag:  Radius: 0.125
 Recurrence coverage REC: 0.1  log(REC)/log(R): 1.107
Ratio 3.566204 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
```



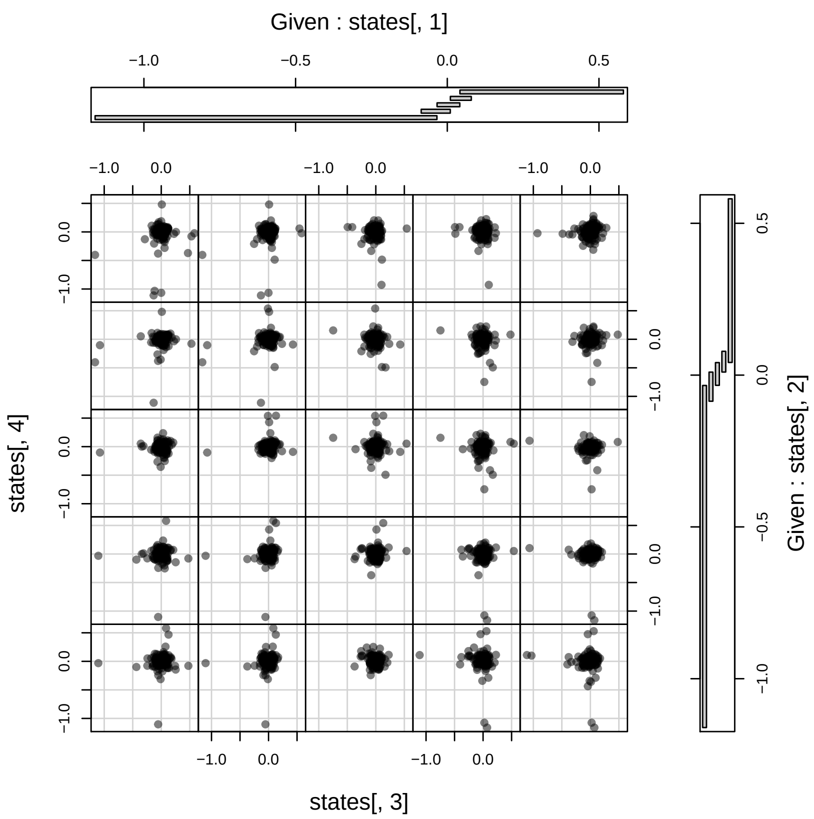
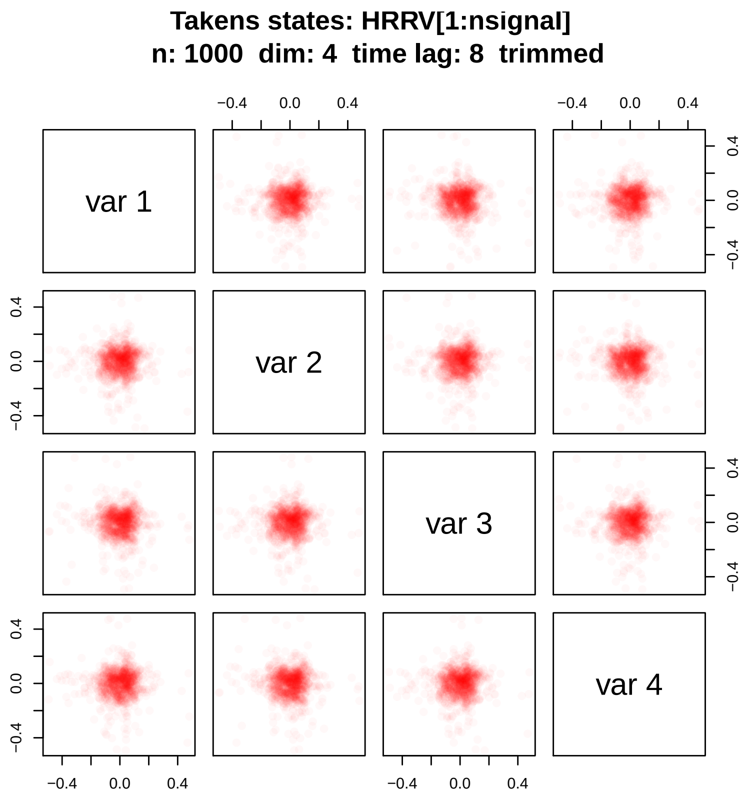


FIGURE 70. Takens states. Example case: RHRV tutorial example.beats variation. Dim=4, time.lag=8. Time used: 3.119 sec.



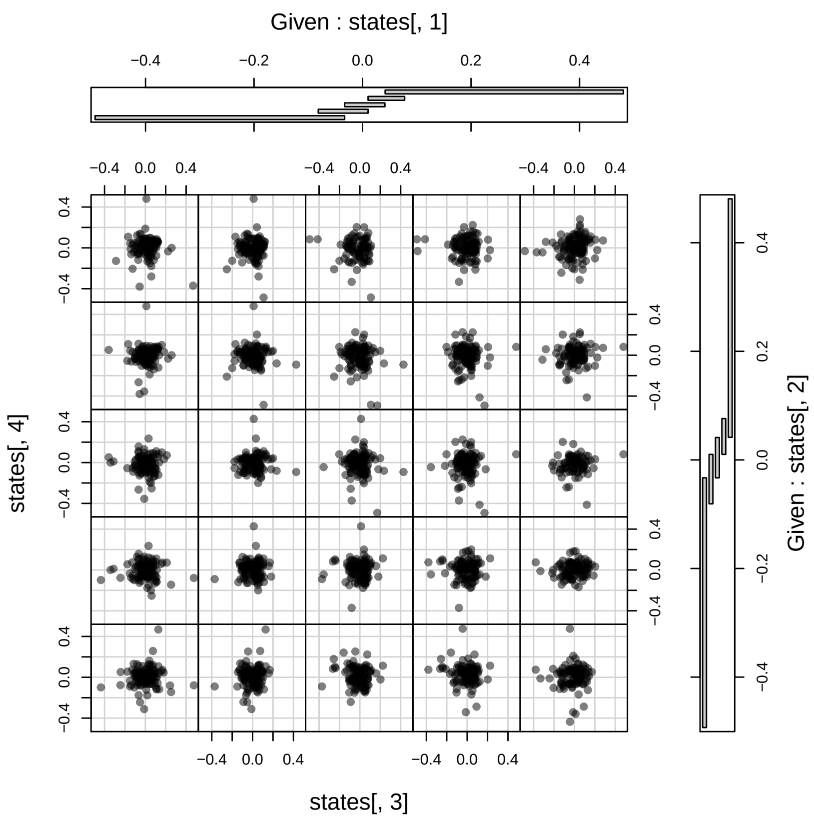


FIGURE 71. Takens states. Example case: RHRV tutorial example.beats variation, trimmed. Dim=4, time.lag=8. Time used: 4.729 sec.

RQA HRRV Dim=6, lag=8. Time used: 6.754 sec. For graphical ouput, see Figure 73 on the following page.

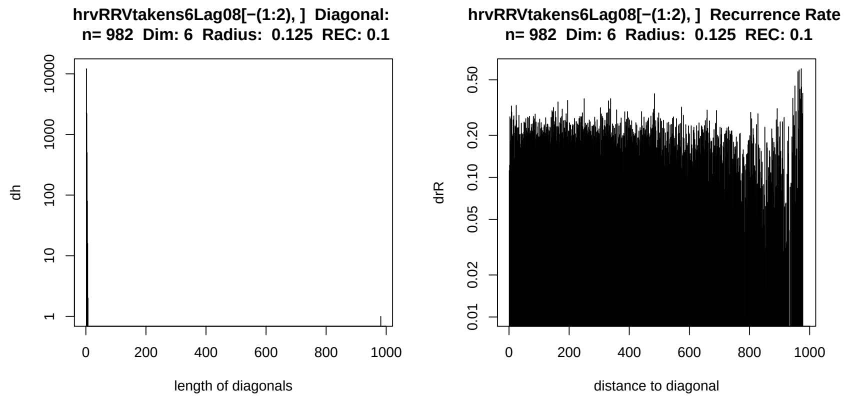


FIGURE 72. RQA. HRRV Dim=6, lag=8

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)
```

Dim=8, lag=8. Time used: 1.065 sec. See figure 76.

Input
```
showrqa(hrvRRVtakens8Lag08[-(1:2),], radius=0.125, do.hist=TRUE)
```

Output
```
hrvRRVtakens8Lag08[-(1:2), ]  n: 966  Dim: 8  Lag:  Radius: 0.125
 Recurrence coverage REC: 0.045  log(REC)/log(R): 1.488
Ratio 4.917975 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
```

RQA HRRV Dim=8, lag=8. Time used: 1.369 sec. For graphical ouput, see Figure 73 on the next page.

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

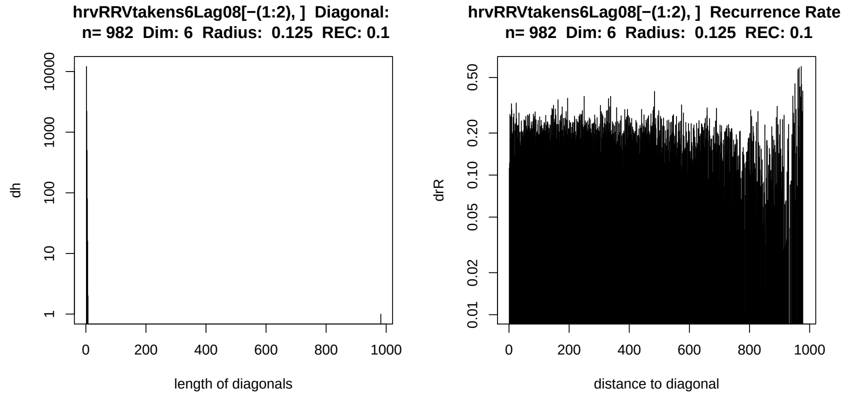


FIGURE 73. RQA. HRRV Dim=8, lag=8

Dim=12, lag=8. Time used: 1.133 sec. See figure 76.

Input
```
showrqa(hrvRRVtakens12Lag08[-(1:2),], radius=3/16, do.hist=TRUE)
```

Output
```
hrvRRVtakens12Lag08[-(1:2), ]  n: 934  Dim: 12  Lag:  Radius: 0.1875
 Recurrence coverage REC: 0.133  log(REC)/log(R): 1.207
Ratio 3.562863 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
```

RQA HRRV Dim=12, lag=8. Time used: 1.549 sec. For graphical ouput, see Figure 74.

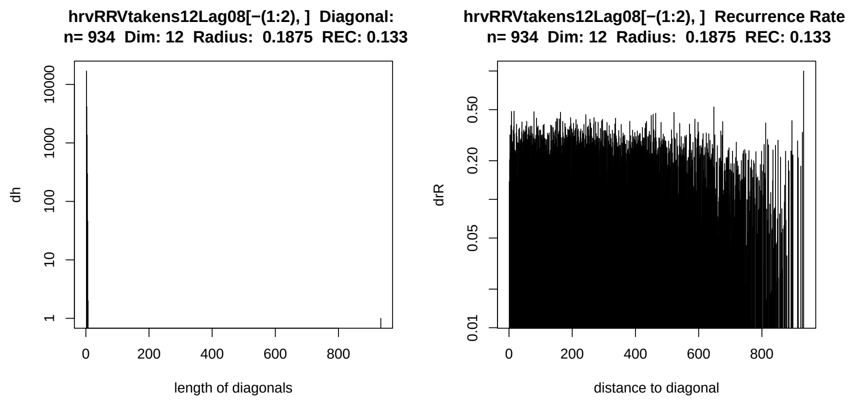


FIGURE 74. RQA. HRRV Dim=12, lag=8

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)
```

Dim=16, lag=8. Time used: 1.567 sec. See figure 76.

Input
```
showrqa(hrvRRVtakens16Lag08[-(1:2),], radius=3/16, do.hist=TRUE)
```

Output
```
hrvRRVtakens16Lag08[-(1:2), ]  n: 902  Dim: 16  Lag:  Radius: 0.1875
 Recurrence coverage REC: 0.071  log(REC)/log(R): 1.581
Ratio 4.776416 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
```

RQA HRRV Dim=16, lag=8. Time used: 2.003 sec. For graphical ouput, see Figure 75.

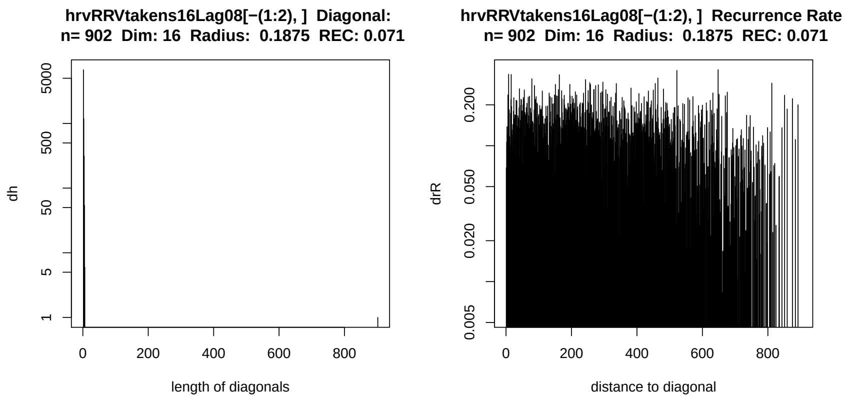


FIGURE 75. RQA. HRRV Dim=16, lag=8

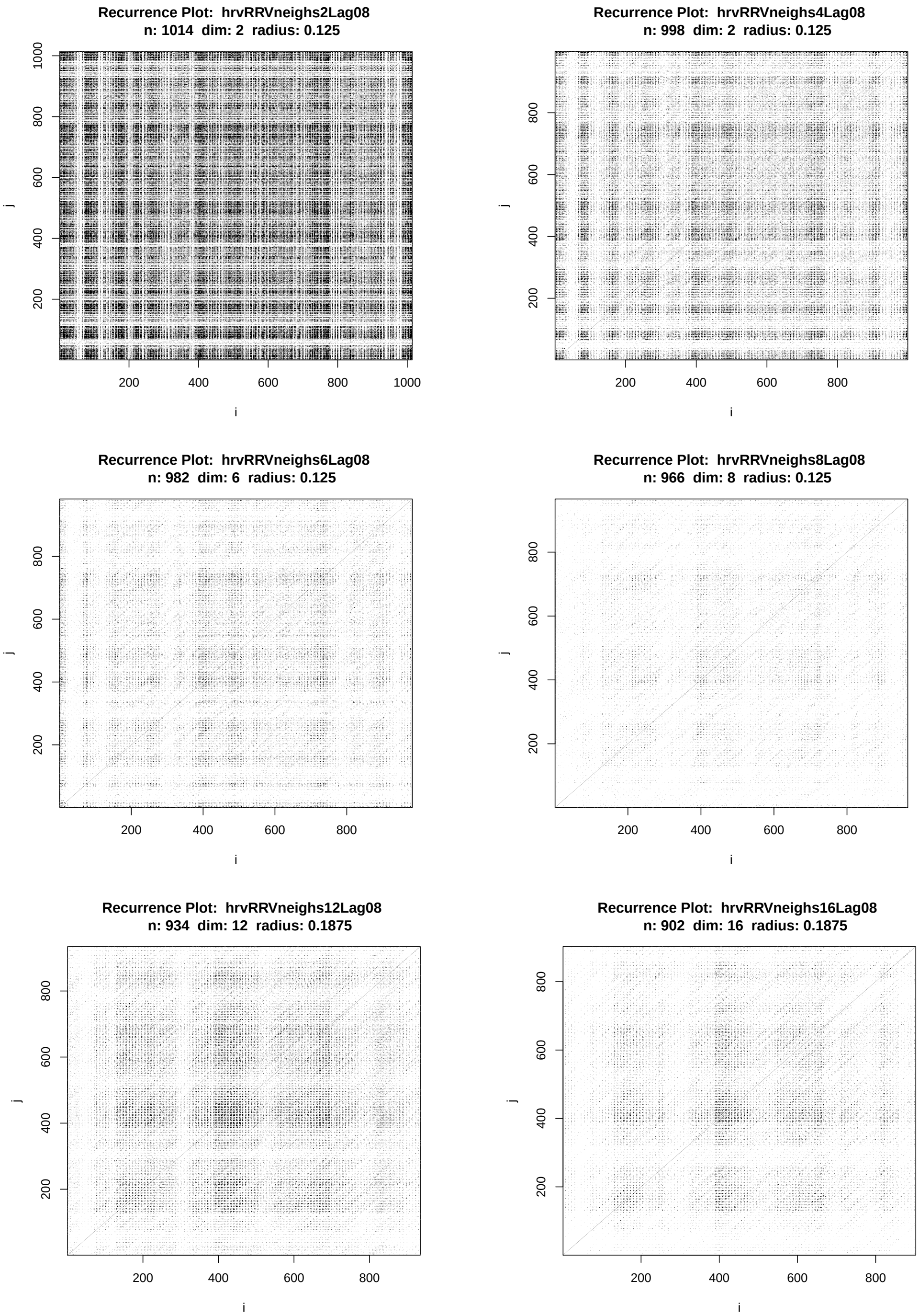


FIGURE 76. Recurrence Plot. Example case: RHRV tutorial example.beats variation. Dim=2, 4, 6, 8, 12, 16. Time.lag=8. Time used: 2.014 sec.

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 77.

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

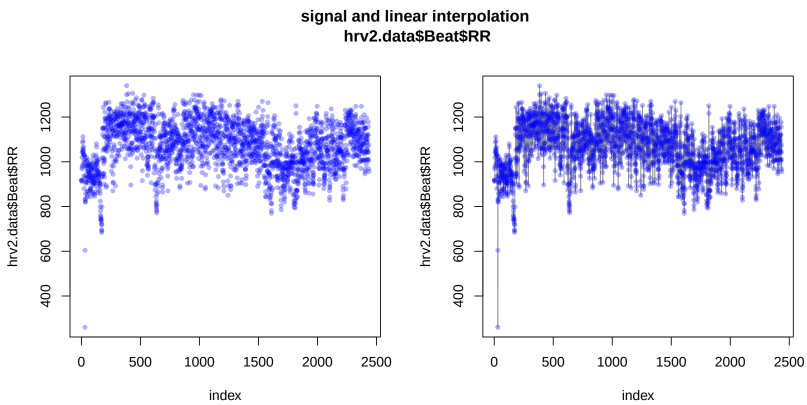


Figure 77. RHRV tutorial example2.beats. Signal and linear interpolation.

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

See Figure 78 on the next page.

Only 1024 data points used in this plot

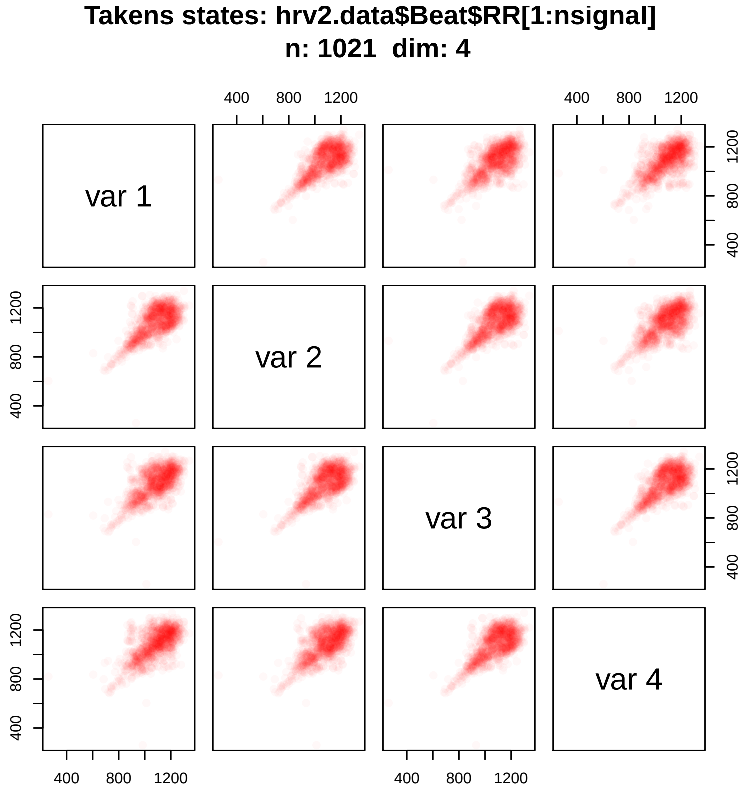


FIGURE 78. RHRV tutorial example2.beats. Time used: 0.838 sec.

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

See Figure 79 on the facing page.

1.664

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

See Figure 80 on the next page.

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=TRUE)
```

Output
```
hrv2RRtakens4[-(1:2), ]  n: 1019  Dim: 4  Lag:  Radius: 192
 Recurrence coverage REC: 0.493  log(REC)/log(R): -0.135
Ratio 1.993144 Determinism: 0.982  Laminarity: 0.906
DIV: 0.005
Trend: 0  Entropy: 3.158
```

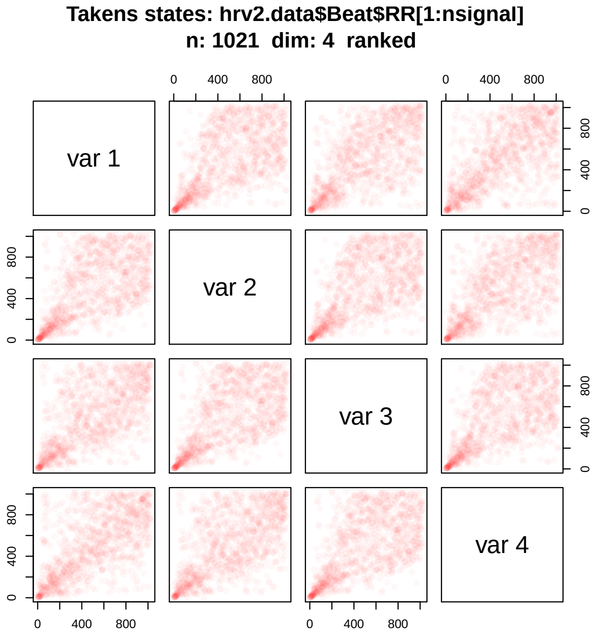


FIGURE 79. Takens states. RHRV tutorial example2.beats. Ranked data

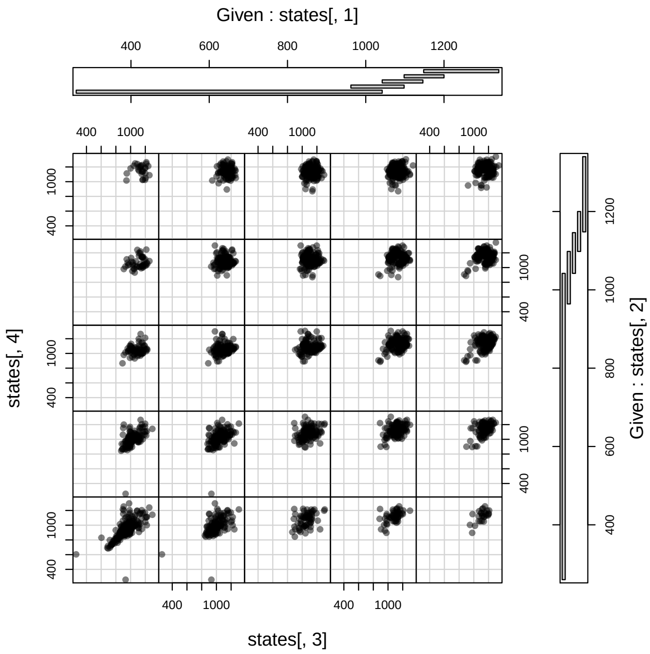


FIGURE 80. State coplot. RHRV tutorial example2.beats. Time used: 0.418 sec.

```
Diagonal lines max: 191  Mean: 10.396  Mean off main: 10.375
Vertical lines max: 136  Mean: 7.145
```

Dim=4. Time used: 3.264 sec.

See Figure 81.

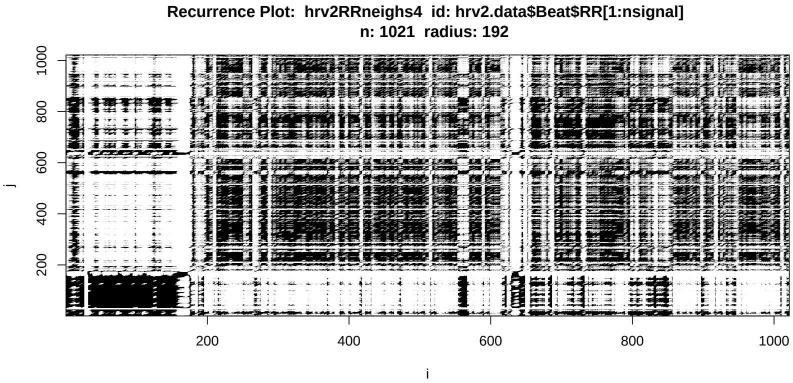


FIGURE 81. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=4. Time used: 3.264 sec.

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)
```

Dim=2. Time used: 1.732 sec.

See Figure 83 on page 104.

Input
```
hrv2RRrqa2 <- showrqa(hrv2RRtakens2[-(1:2),], radius=10*16, do.hist=TRUE)
```

Output
```
hrv2RRtakens2[-(1:2), ]  n: 1021  Dim: 2  Lag:  Radius: 160
 Recurrence coverage REC: 0.52  log(REC)/log(R): -0.129
Ratio 1.824356 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
```

RQA hrv2RR dim=2. Time used: 2.751 sec. For graphical ouput, see Figure 82 on the following page.

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=TRUE)
```

Output
```
hrv2RRtakens6[-(1:2), ]  n: 1017  Dim: 6  Lag:  Radius: 224
 Recurrence coverage REC: 0.528  log(REC)/log(R): -0.118
Ratio 1.882666 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: 3.565 sec.

See Figure 83 on page 104.

Input

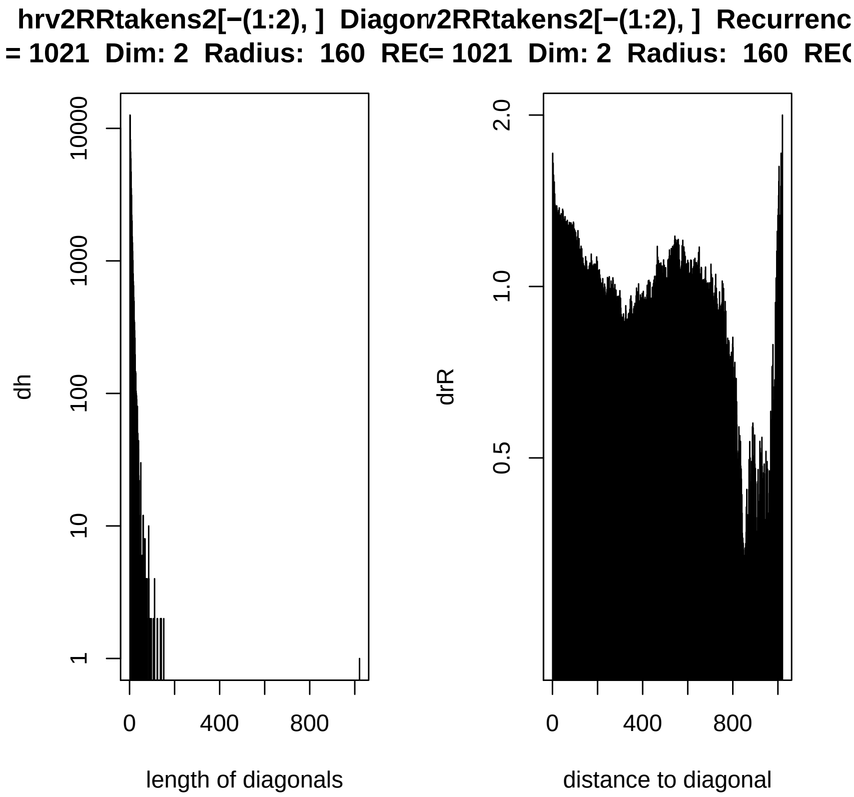


FIGURE 82. RQA. hrv2RR dim=2

```
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=TRUE)
```

Output

```
hrv2RRtakens8[-(1:2), ]  n: 1015  Dim: 8  Lag:  Radius: 256
 Recurrence coverage REC: 0.584  log(REC)/log(R): -0.097
Ratio 1.706571 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: 3.042 sec.

See Figure 83 on page 104.

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=TRUE)
```

Output

```
hrv2RRtakens12[-(1:2), ]  n: 1011  Dim: 12  Lag:  Radius: 256
 Recurrence coverage REC: 0.488  log(REC)/log(R): -0.129
Ratio 2.043437 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: 4.107 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=TRUE)
```

Output

```
hrv2RRtakens16[-(1:2), ]  n: 1007  Dim: 16  Lag:  Radius: 288
 Recurrence coverage REC: 0.544  log(REC)/log(R): -0.107
Ratio 1.83565 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: 3.659 sec.

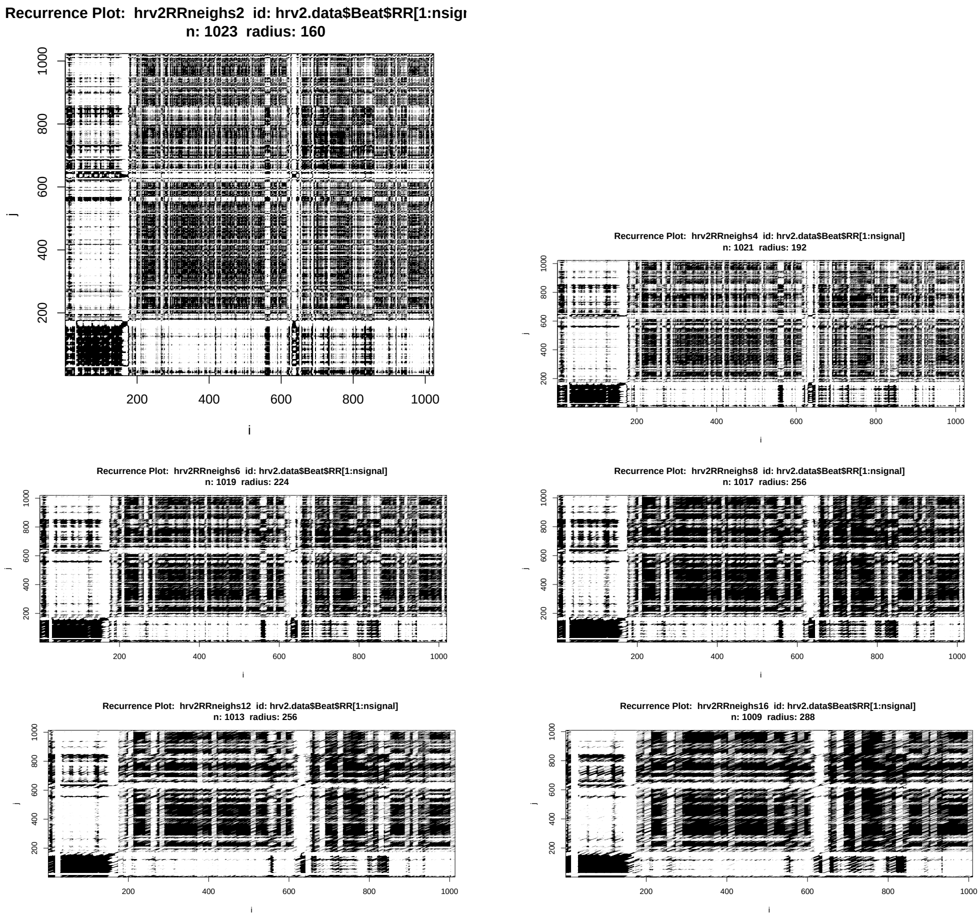


FIGURE 83. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 3.659 sec.

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 84,

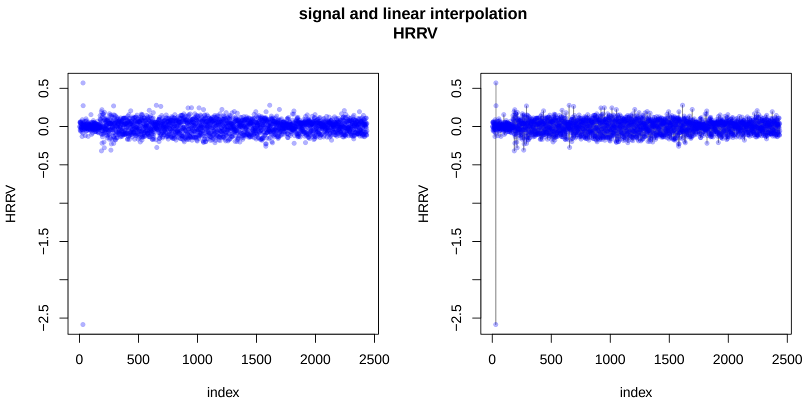


FIGURE 84. RHRV tutorial example2.beats. HRRV Signal and linear interpolation.

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 85 on the following page.

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

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

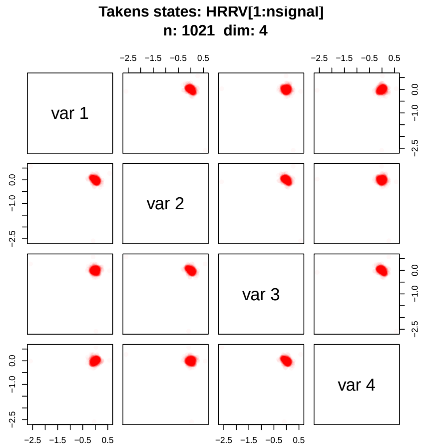


FIGURE 85. Recurrence plot. RHRV tutorial example2.beats. HRRV. Time used: 0.872 sec.

See Figure 86.

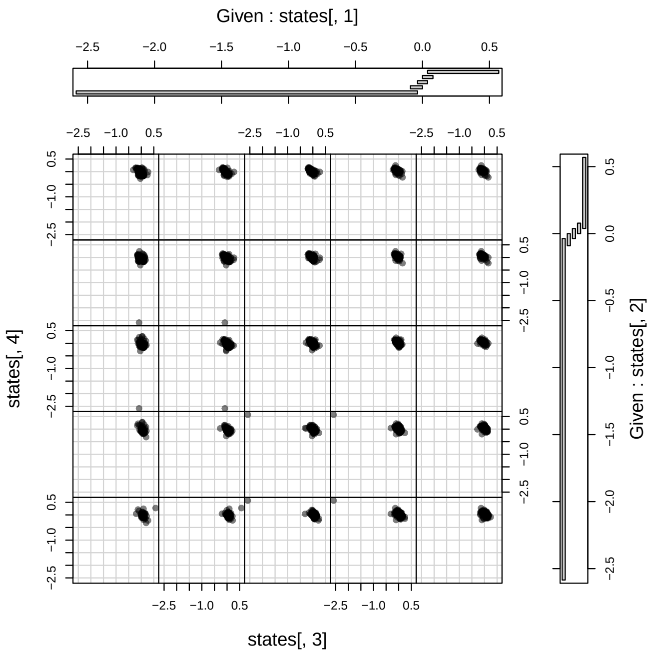


FIGURE 86. State coplot. RHRV tutorial example2.beats. HRRV. Time used: 1.124 sec.

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

See Figure 87 on the facing page

Input
```
hrv2RRVtakens41 <- hrv2RRVtakens4
hrv2RRVtakens41[hrv2RRVtakens41 < -1.5] <- NA
```

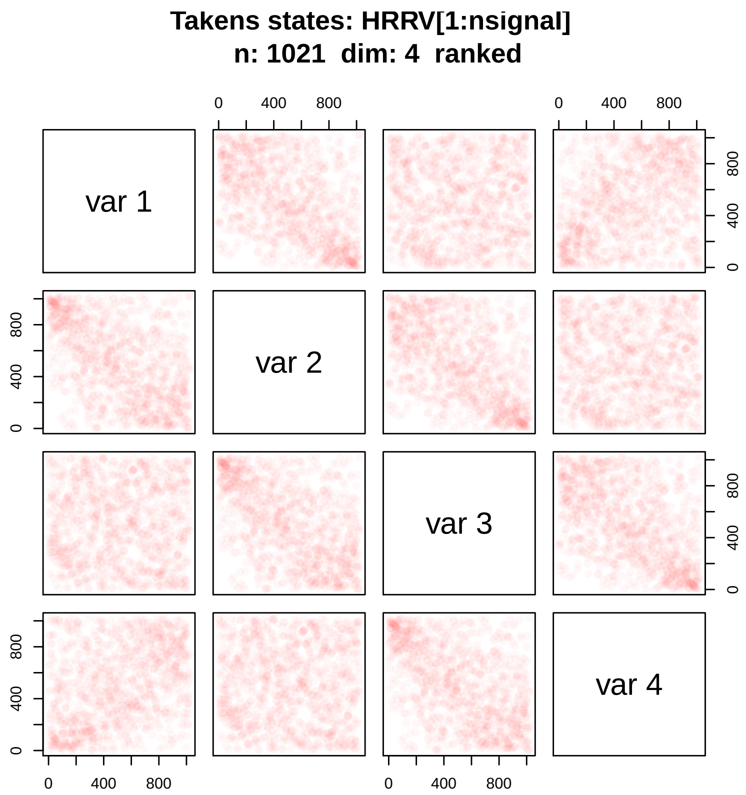


FIGURE 87. RHRV tutorial example2.beats. Ranked HRRV data. Time used: 1.977 sec.

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

See Figure 88.

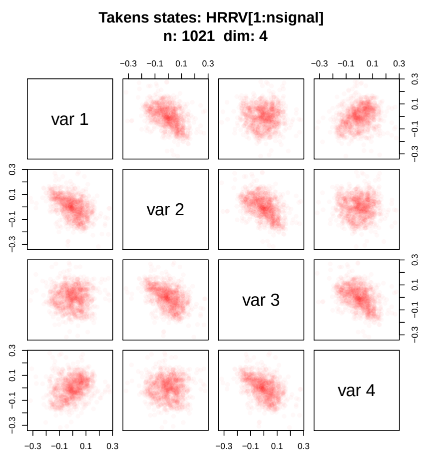


FIGURE 88. Recurrence plot. RHRV tutorial example2.beats. HRRV. Time used: 2.823 sec.

Input

```
statecoplot(hrv2RRVtakens4l) #dim=4
```

Output

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

See Figure 89.

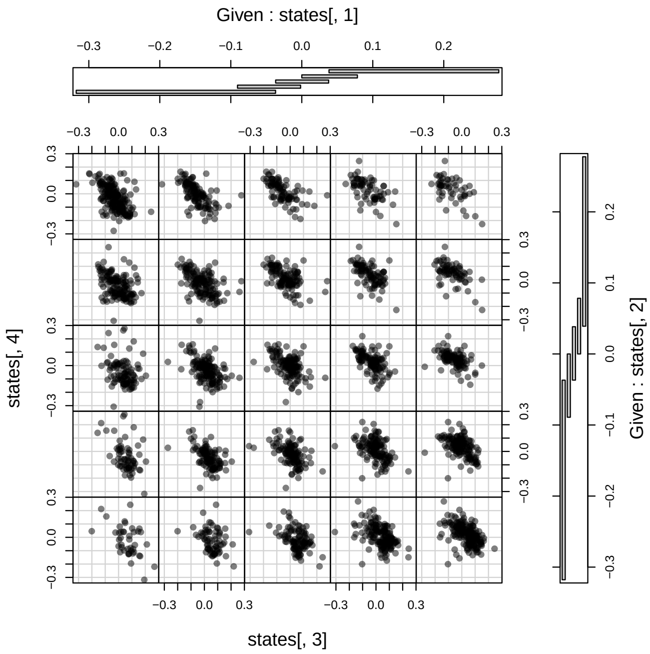


FIGURE 89. State coplot. RHRV tutorial example2.beats. HRRV. Time used: 3.05 sec.

ToDo: findAll-Neighbours 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")
```

Time used: 0.144 sec.

Input

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

ToDo: check. There seem to be strange artefacts.

See Figure 90 on the next page.

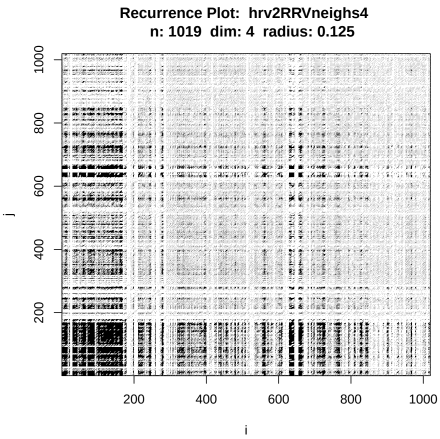


FIGURE 90. Recurrence plot. RHRV tutorial example2.beats hrv2RRV. Dim=4. Time used: 1.055 sec.

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)
```

See figure 96.

Input
```
hrv2RRVrqa2 <- showrqa(hrv2RRVtakens2[-(1:2),], radius=0.125, do.hist=TRUE)
```

Output
```
hrv2RRVtakens2[-(1:2), ]  n: 1021  Dim: 2  Lag:  Radius: 0.125
 Recurrence coverage REC: 0.508  log(REC)/log(R): 0.326
Ratio 1.797625 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
```

RQA hrv2RR dim=2. Time used: 3.088 sec. For graphical ouput, see Figure 91 on the facing page.

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)
```

See figure 96.

Input
```
hrv2RRVrqa6 <- showrqa(hrv2RRVtakens6[-(1:2),], radius=0.125*1.5, do.hist=TRUE)
```

Output
```
hrv2RRVtakens6[-(1:2), ]  n: 1017  Dim: 6  Lag:  Radius: 0.1875
 Recurrence coverage REC: 0.516  log(REC)/log(R): 0.395
Ratio 1.921114 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
```

RQA hrv2RR dim=6. Time used: 3.251 sec. For graphical ouput, see Figure 92 on page 112.

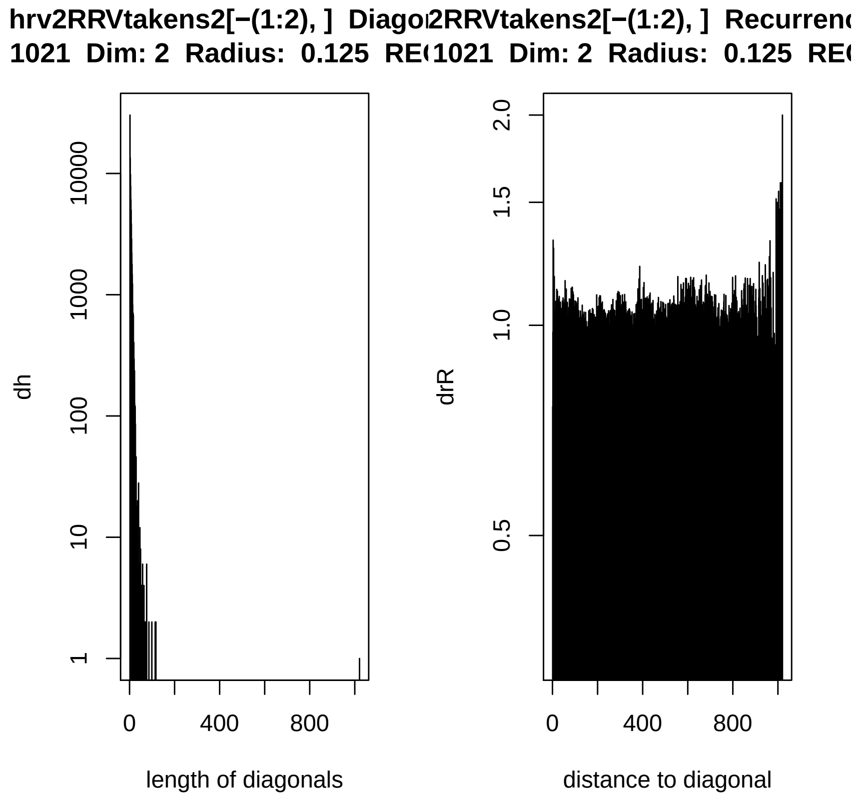


FIGURE 91. RQA. hrv2RR dim=2

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)
```

See figure 96.

Input
```
hrv2RRVrqa8 <- showrqa(hrv2RRVtakens8[-(1:2),], radius=3/16, do.hist=TRUE)
```

Output
```
hrv2RRVtakens8[-(1:2), ]  n: 1015  Dim: 8  Lag:  Radius: 0.1875
 Recurrence coverage REC: 0.432  log(REC)/log(R): 0.501
Ratio 2.296476 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
```

RQA hrv2RR dim=8. Time used: 3.339 sec. For graphical ouput, see Figure 93 on page 113.

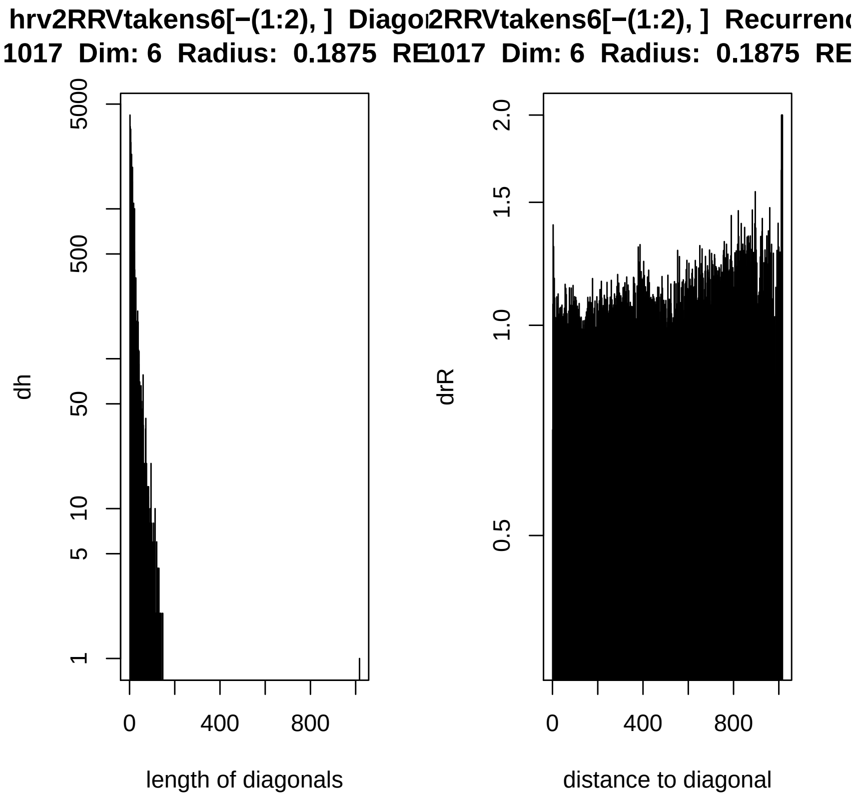


FIGURE 92. RQA. hrv2RR dim=6

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)
```

See figure 96.

Input
```
hrv2RRVrqa12 <- showrqa(hrv2RRVtakens12[-(1:2),], radius=3.5/16, do.hist=TRUE)
```

Output
```
hrv2RRVtakens12[-(1:2), ]  n: 1011  Dim: 12  Lag:  Radius: 0.21875
 Recurrence coverage REC: 0.479  log(REC)/log(R): 0.484
Ratio 2.081523 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
```

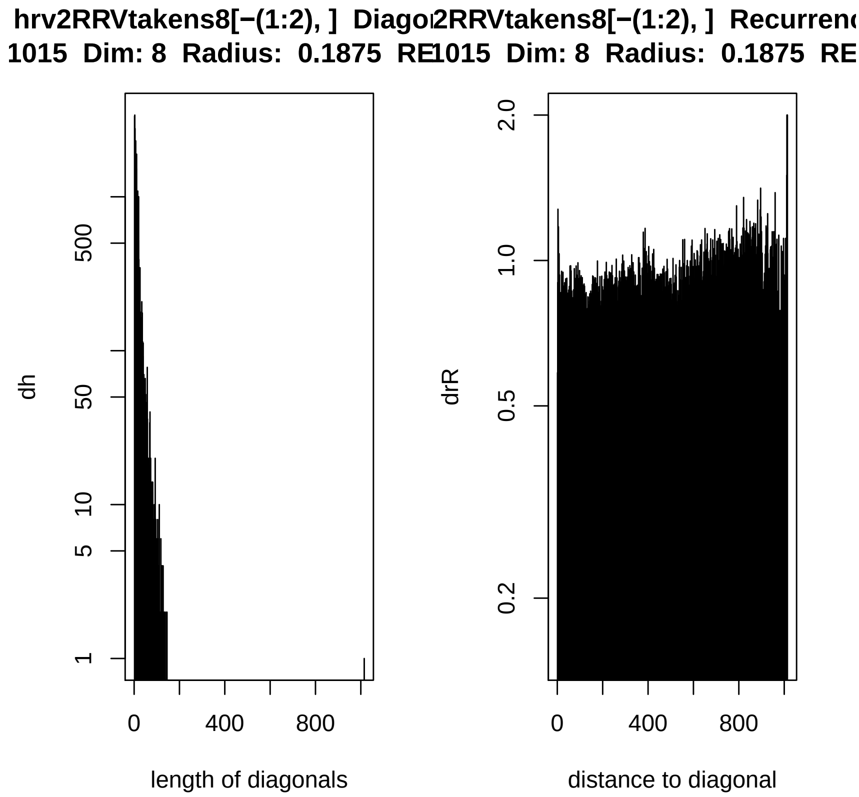


FIGURE 93. RQA. hrv2RR dim=8

RQA hrv2RR dim=12. Time used: 3.136 sec. For graphical ouput, see Figure 94 on the next page.

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)
```

See figure 96.

Input
```
hrv2RRVrqa16 <- showrqa(hrv2RRVtakens16[-(1:2),], radius=4/16, do.hist=TRUE)
```

Output
```
hrv2RRVtakens16[-(1:2), ]  n: 1007  Dim: 16  Lag:  Radius: 0.25
 Recurrence coverage REC: 0.568  log(REC)/log(R): 0.408
Ratio 1.758873 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
```

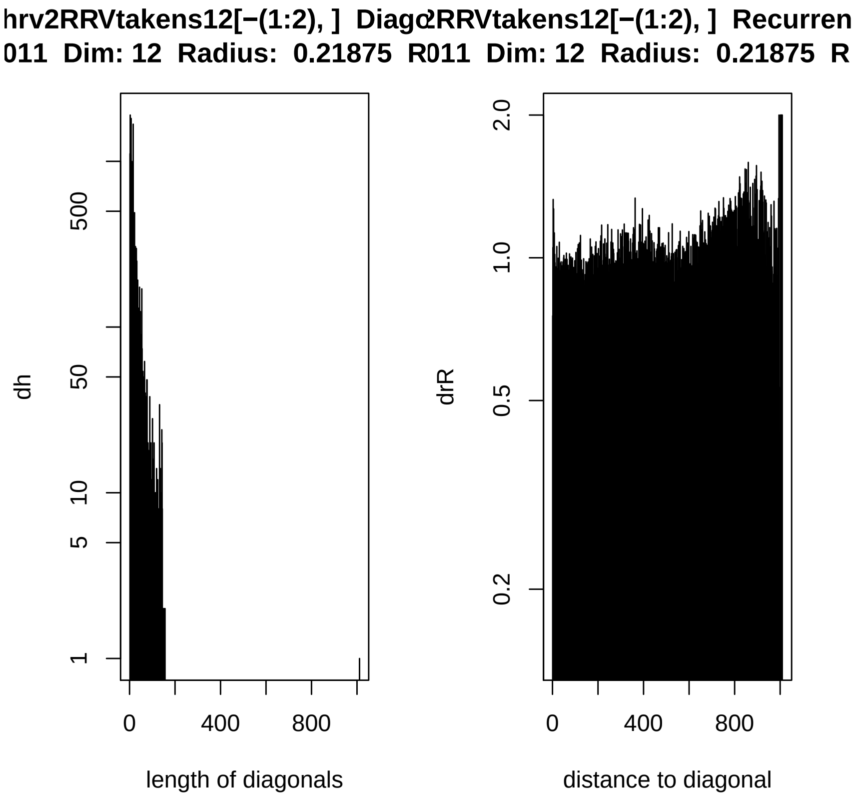


FIGURE 94. RQA. hrv2RR dim=12

RQA hrv2RR dim=16. Time used: 4.366 sec. For graphical ouput, see Figure 95 on the facing page.

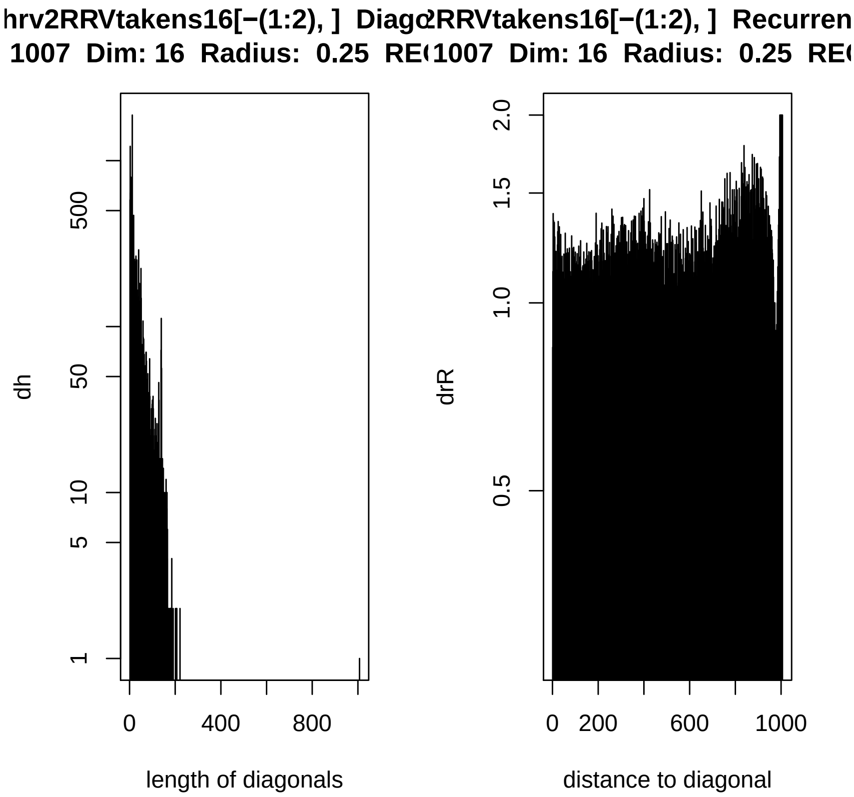


FIGURE 95. RQA. hrv2RR dim=16

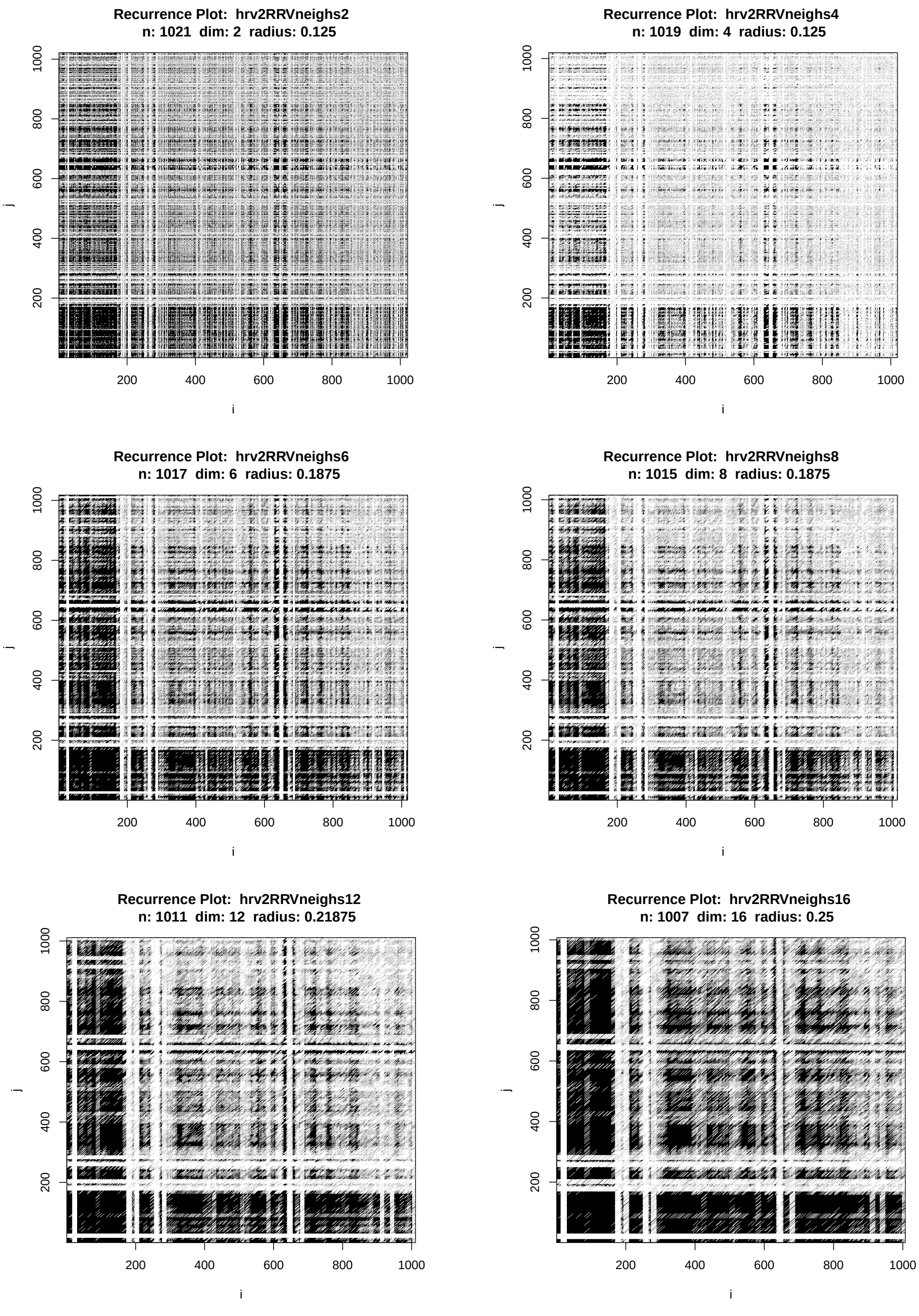


FIGURE 96. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 4.366 sec.

References

Ebner-Priemer UW, Kuo J, Kleindienst N, Welch SS, Reisch T, Reinhard I, Lieb K, Linehan MM, Bohus M (2007). "State affective instability in borderline personality disorder assessed by ambulatory monitoring." *Psychological medicine*, **37**(07), 961–970.

Ebner-Priemer UW, Sawitzki G (2007). "Ambulatory assessment of affective instability in borderline personality disorder: The effect of the sampling frequency." *European Journal of Psychological Assessment*, **23**(6), 238–247. doi:10.1027/1015-5759.23.4.238.

Eckmann JP, Kamphorst SO, Ruelle D (1987). "Recurrence plots of dynamical systems." *Europhys. Lett*, **4**(9), 973–977.

Marwan N, Kurths J (2002). "Nonlinear analysis of bivariate data with cross recurrence plots." *Physics Letters A*, **302**(5), 299–307.

Marwan N, Thiel M, Nowaczyk NR (2002). "Cross recurrence plot based synchronization of time series." *arXiv preprint physics/0201062*.

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: 195.9 sec. at Thu Jun 29 19:17:29 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: 259 $

$Date: 2017-06-23 21:25:00 +0200 (Fri, 23 Jun 2017) $

$Name: $

$Author: gsawitzki $

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

Günther Sawitzki
StatLab Heidelberg
Im Neuenheimer Feld 294
D 69120 Heidelberg