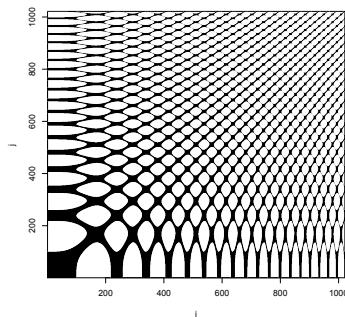


# STATISTICAL DATA ANALYSIS: RECURRENCE PLOT

GÜNTHER SAWITZKI



## CONTENTS

1. Background	1
1.1. Takens' Recurrence States	2
1.2. Recurrence Plots	3
1.3. Recurrence Quantification Analysis	3
2. R Setup	3
2.1. Local Bottleneck	6
2.2. Recurrence Plots	8
3. Test Signals	10
3.1. Sinus	10
3.2. Uniform Random Numbers	10
3.3. Chirp Signal	11
3.4. Doppler signal	12
4. Takens' States for Test Signals	14
5. Recurrence Plots for Test Signals	26
6. Case Study: Geyser data	36
6.1. Geyser Eruption Durations	36
6.2. Geyser Waiting	48
6.3. Geyser - linearized	50
7. Case Study: HRV data example.beats	58
7.1. RHRV: example.beats - Hart Rate Variation	66
8. Case Study: HRV data example2.beats	74
8.1. RHRV: example2.beats - Hart Rate Variation	82
References	91
Index	92

---

Date: 2013-11.

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

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

*Typeset, with minor revisions: August 23, 2014 from svn/cvs Revision : 163*

*gs@statlab.uni-heidelberg.de .*

## 1. BACKGROUND

**1.1. Takens' Recurrence States.** Recurrence plots have been introduced in an attempt to understand near periodic behaviour in hydrodynamics, in particular the transition to turbulence. On the one hand, and extended theory on dynamical systems was available, covering deterministic models. A fundamental concept is that at a certain time a system is in some state, and developing from this. Defining the proper state space is a critical step in modelling.

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

Recurrence, coming back to some state, is often a key to understand a near periodic system. A classical field is the movement of celestial bodies.

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

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

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

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

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

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

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

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

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

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

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

Takens states are vectors in  $M$  dimensions. There are standard statistical techniques to visualize 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

**ToDo:** add support  
for higher dimensional signals

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 use a variant of pairs().

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.

**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.

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

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

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

See Table 1 on the following page.

## 2. R SETUP

---

*Input*

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

---

*Output*

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

**ToDo:** consider dimension-adjusted radius

**ToDo:** support distance instead of 0/1 indicators

TABLE 1. Recurrence Quantification Analysis (RQA)

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

[1] "Mersenne-Twister" "Inversion"

---

*Input*

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

---



---

*Input*

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

---

2.0.1. *Takens States*. Takens states are represented as a matrix, one state per row. The number of columns 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

---

<code>statepairs</code>	<i>Show marginal scatterplots of Takens states</i>
-------------------------	--

---

*Usage.*

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

*Arguments.*

---

<code>states</code>	A matrix: Takens states by dimension, one state per row.
<code>main</code>	Optional: the main header.
<code>range</code>	Optional: an interval selecting values to be displayed.
<code>rank</code>	An experimental variant. If <code>rank</code> , the values are rank transformed.
<code>nooverlap</code>	An experimental variant. If <code>nooverlap</code> , the cases are subsampled by dimension.
<code>alpha</code>	The alpha value used for plotting. The current choice is $\sqrt{\frac{1}{nr\ states}}$ as a default.

---

**ToDo:** colour by time

---

*Input*

```

statepairs <- function(states, main,
                       rank=FALSE, nooverlap= FALSE, range=NULL,
                       alpha, ...){
  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 (missing(alpha)) {alpha <- 1/sqrt(dim(states)[1])}
  #cat("alpha=",alpha)

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

  pairs(states, main=main,
        col=rgb(0,0,0, alpha), pch=19, ...)
  #title(main=main, outer=TRUE, line=-2, cex.main=0.8)
}

```

---

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

---

<code>statecoplot</code>	<i>Show conditioning marginal scatterplots of Takens states</i>
--------------------------	---

---

*Usage.*

```

statecoplot(states, main,
            range = NULL,
            rank = FALSE, nooverlap = FALSE,
            alpha, number = c(5,5))

```

*Arguments.*

---

<code>states</code>	A matrix: Takens states by dimension, one state per row.
<code>main</code>	Optional: the main header.
<code>range</code>	Optional: an interval selecting values to be displayed.
<code>rank</code>	An experimental variant. If <code>rank</code> , the values are rank transformed.
<code>nooverlap</code>	An experimental variant. If <code>nooverlap</code> , the cases are subsampled by dimension.
<code>alpha</code>	The alpha value used for plotting. The current choice is $\sqrt{\frac{1}{nr\ states}}$ as a default.
<code>number</code>	integer; the number of conditioning intervals, for a and b, possibly of length 2.

---



---

```
statecoplot <- function(states, main,
                           rank = FALSE, nooverlap = FALSE, range = NULL,
                           alpha, 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 (missing(alpha)) {alpha <- 1/sqrt(dim(states)[1]/32)}
  #cat("alpha=",alpha)

  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)
}
```

---

**2.1. Local Bottleneck.** To allow experimental implementations, functions from `non-linearTseries` are aliased here.

---

```
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)
}



---


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



---


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

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

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

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

more <- NULL

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

#use components of neights if available
embedding.dim <- attr(neighs, "embedding.dim")

```

minor cosmetics  
added to recurrence-  
PlotAux

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

```

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:",
  " does not match radius argument=", radius))
}
# if (!is.null(attrradius)) more <- paste(more," radius attr:",attrradius)
if (!is.null(more)) main <- paste(main,"\n",more)

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

```

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

**2.2. Recurrence Plots.** This is a hack to report RQA information.  $dim = \text{NULL}$  is added to align calling with other functions.

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

Needs improvement.

```
showrqa <- function(takens, dim=NULL, radius,
                      digits=3,
                      do.hist = TRUE, rm.hist = TRUE, log = TRUE ,...)
{
  xxrqa <- rqa(takens=takens, radius=radius)
  xxrqa$radius <- radius
  id <- attr(takens, "id")
  if (is.null(id)) id <- deparse(substitute(takens))
  xxrqa$id<- id
  xxrqa$time.lag <- attr(takens, "time.lag")
  cat(id, " n:", dim(takens)[1], " Dim:", dim(takens)[2], "\n")
  xxrqa$n <- dim(takens)[1]
  xxrqa$dim <- dim(takens)[2]
  cat(paste("Radius:", radius,
            " Recurrence coverage REC:", round(xxrqa[1]$REC, digits),
            " log(REC)/log(R):", round(log(xxrqa[1]$REC)/log(radius), digits), "\n"))
  xxrqa$logratio <- log(xxrqa[1]$REC)/log(radius)
  cat(paste("Determinism:", round(xxrqa$DET, digits),
            " Laminarity:", round(xxrqa$LAM, digits), "\n"))
  cat(paste("DIV:", round(xxrqa$DIV, digits), "\n"))
}
```

```

cat(paste("Trend:", round(xxrqa$TREND, digits),
          " Entropy:", round(xxrqa$ENTR, digits), "\n"))
cat(paste("Diagonal lines max:", round(xxrqa$Lmax, digits),
          " Mean:", round(xxrqa$Lmean, digits),
          " Mean off main:", round(xxrqa$LmeanWithoutMain, digits), "\n"))
cat(paste("Vertical lines max:", round(xxrqa$Vmax, digits),
          " Mean:", round(xxrqa$Vmean, digits), "\n"))
# str(xxrqa[4:12])

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

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

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

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

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

  par(oldpar)
}

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

```

### 3. TEST SIGNALS

We set up a small series of test signals. Some synthetic test signals are introduced here. More test cases used later are the Geyser data set ( Section 6 on page 36) and two examples of heart rate data sets ( Section 7 on page 58 and Section 8 on page 74) from *library(rhrv)*.

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.

---

	<i>Input</i>
#nsignal <- 256	
nsignal <- 1024	
#nsignal <- 4096	
system.time.start <- proc.time()	

---

For signal representation, we use a common layout.

---

	<i>Input</i>
plotsignal <- function(signal, main, ylab) {	
#! alpha level should depend on expected number of overlaps	
if (missing(ylab)) { ylab <- deparse(substitute(signal)) }	
par(mfrow = c(1,	
2))	
plot(signal,	
main = "", xlab = "index", ylab = ylab,	
col = rgb(0, 0, 1, 0.3), pch = 20)	
plot(signal, type = "l",	
main = "", xlab = "index", ylab = ylab,	
col = rgb(0, 0, 0, 0.4))	
points(signal,	
col = rgb(0, 0, 1, 0.3), pch = 20)	
if (missing(main)) { main = deparse(substitute(signal)) }	
title(main = main,	
outer = TRUE, line = -2, cex.main = 1.2)	
}	

---

#### 3.1. Sinus.

---

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

---

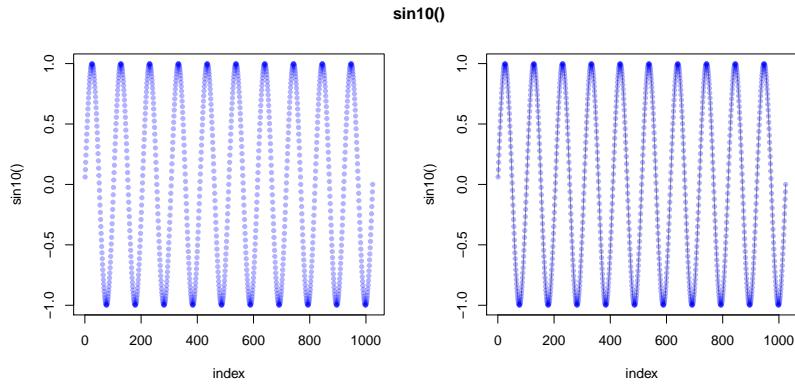
See Figure 1 on the facing page.

#### 3.2. Uniform Random Numbers.

---

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

---

FIGURE 1. Test case:  $\sin 10t$ . Signal and linear interpolation.

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

See Figure 2,

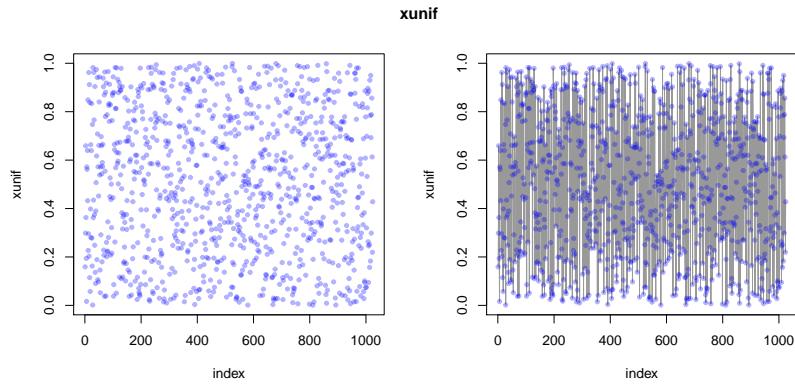


FIGURE 2. Test case: unif - uniform random numbers. Signal and linear interpolation.

### 3.3. Chirp Signal.

---

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

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

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

```

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

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

```

See Figure 3,

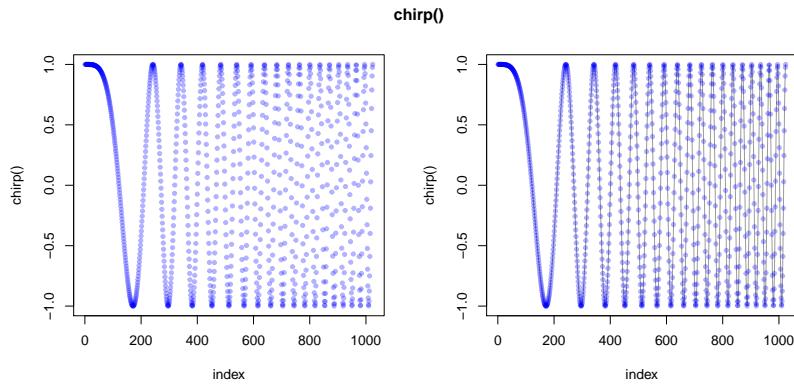


FIGURE 3. Test case: chirp signal. Signal and linear interpolation.

### 3.4. Doppler signal.

---

```

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

```

See Figure 4 on the next page,

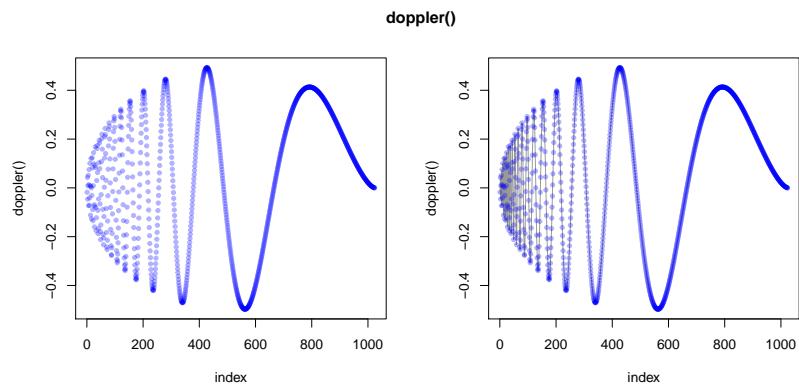


FIGURE 4. Test case: Doppler signal. Signal and linear interpolation.

#### 4. TAKENS' STATES FOR TEST SIGNALS

---

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

---

See Figure 5.

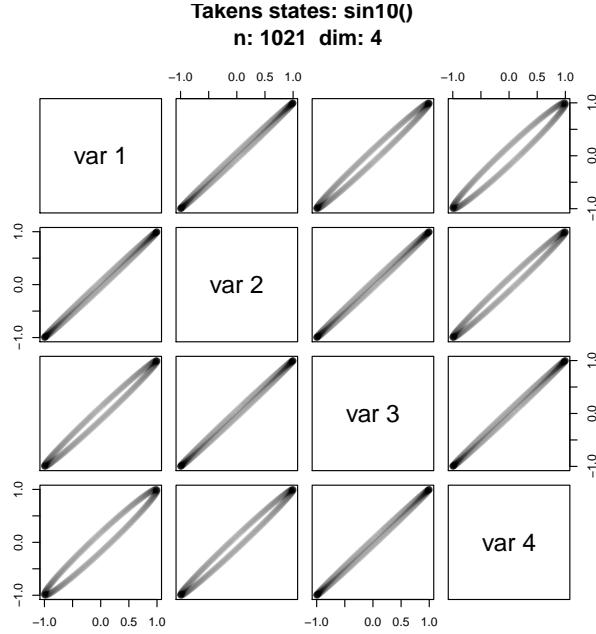


FIGURE 5. Takens states. Test case: sinus. Note that  $2 - dim$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.614 sec.

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

---

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

---

See Figure 6 on the next page.

---

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

---

See Figure 7 on the facing page.

---

```
Input
stateecplot(sintakens) #4
```

---

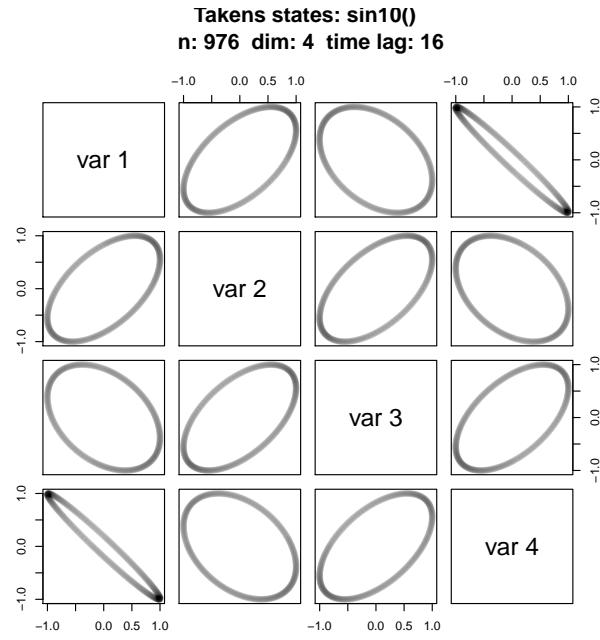


FIGURE 6. Takens states. Test case: sinus at time lag 16. Note that 2-dim marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.642 sec.

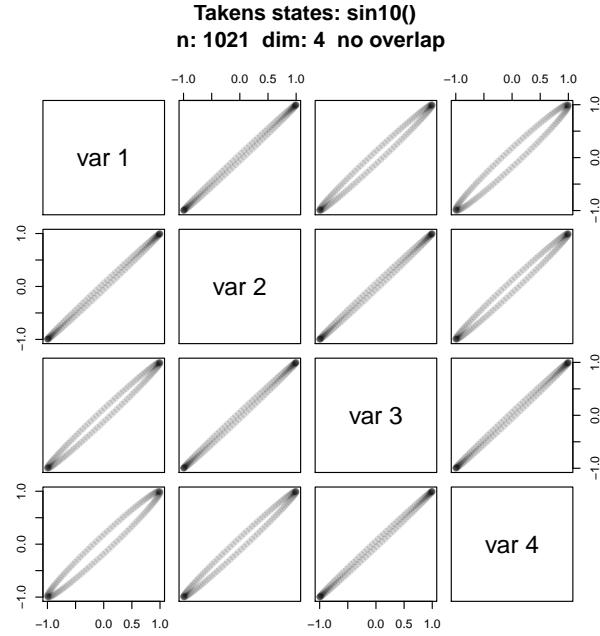


FIGURE 7. Takens states. Test case: sinus, no overlap. Note that 2-dim marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.894 sec.

See Figure 8.

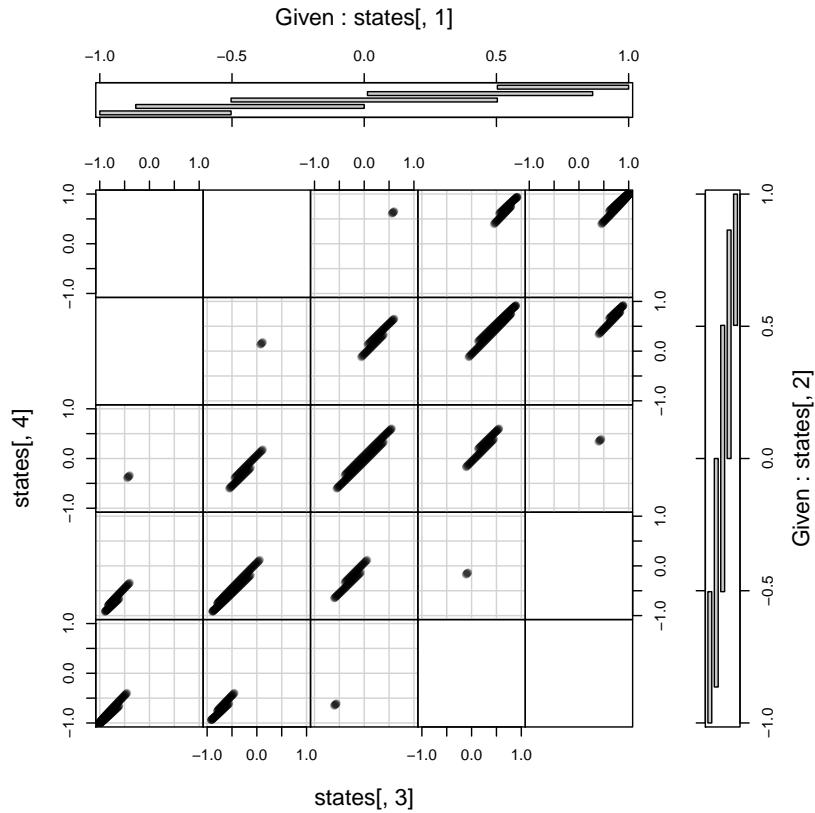


FIGURE 8. State coplot. Test case: sinus. Note that  $2 - \dim$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.241 sec.

---

*Input*

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

---

See Figure 9 on the facing page.

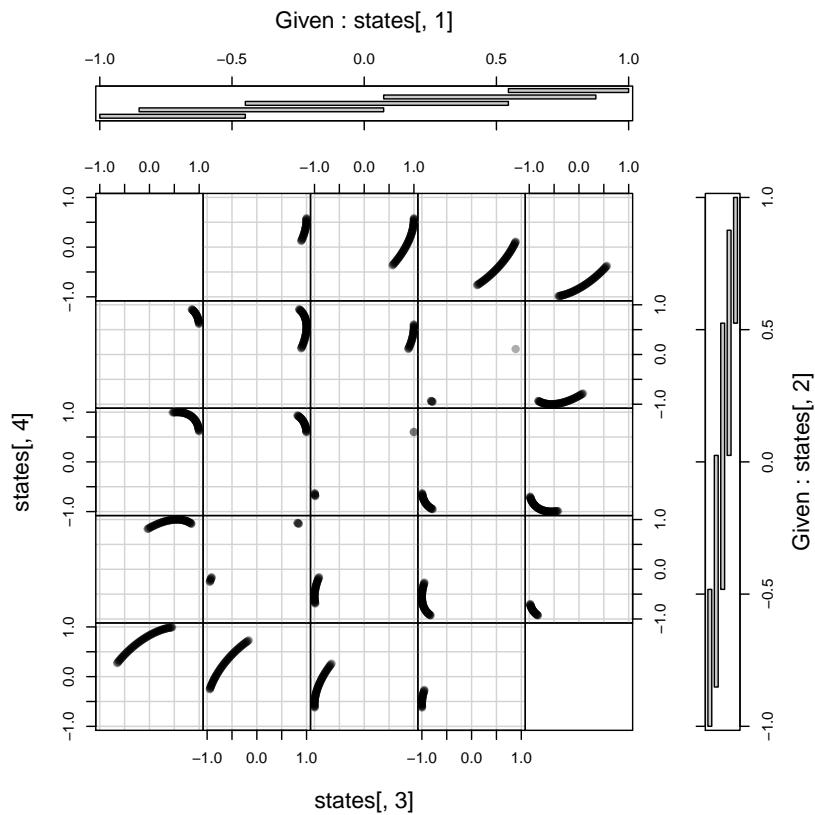


FIGURE 9. State coplot. Test case: sinus, time lag 16. Note that  $2 - \dim$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.237 sec.

---

*Input*

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

---

See Figure 10.

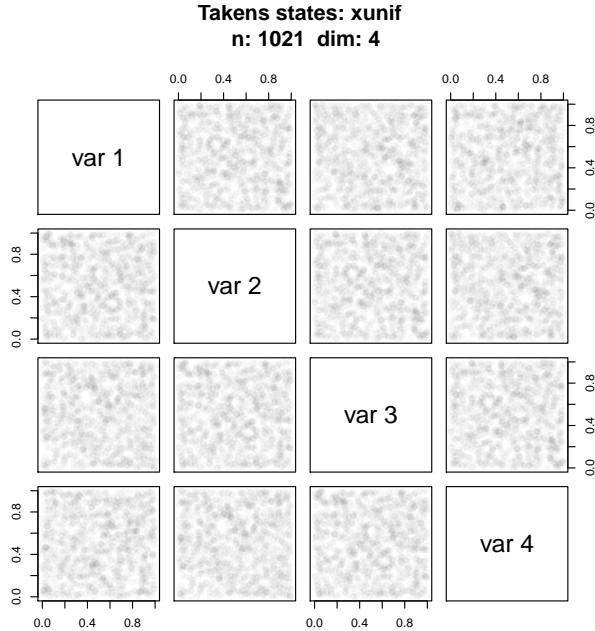


FIGURE 10. Takens states. Test case: uniform random numbers. Time used: 0.643 sec.

---

*Input*

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

---

See Figure 11 on the facing page.

---

*Input*

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

---

See Figure 12 on the next page.

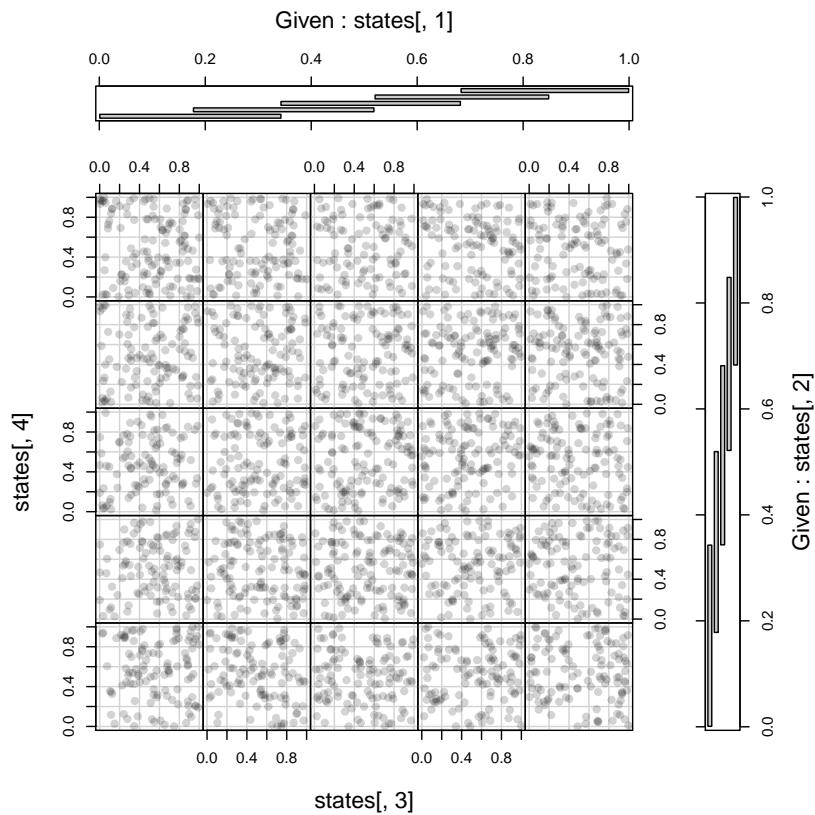


FIGURE 11. State coplot. Test case: uniform random numbers. Time used: 0.998 sec.

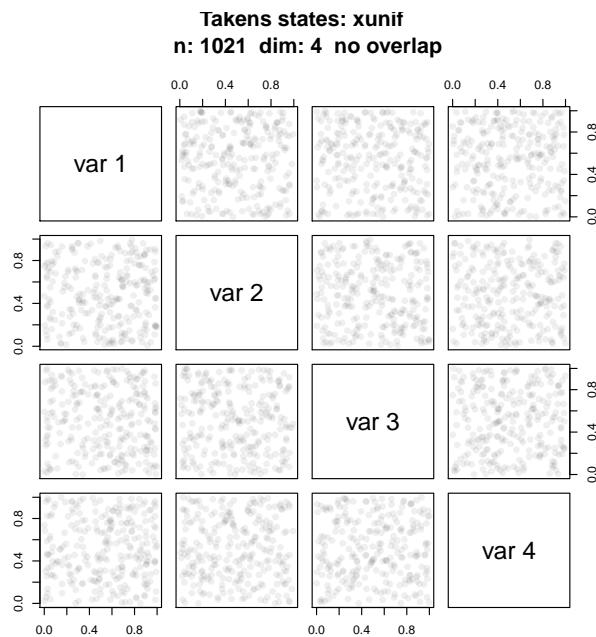


FIGURE 12. Takens states. Test case: uniform random numbers. Time used: 1.245 sec.

---

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

---

See Figure 13.

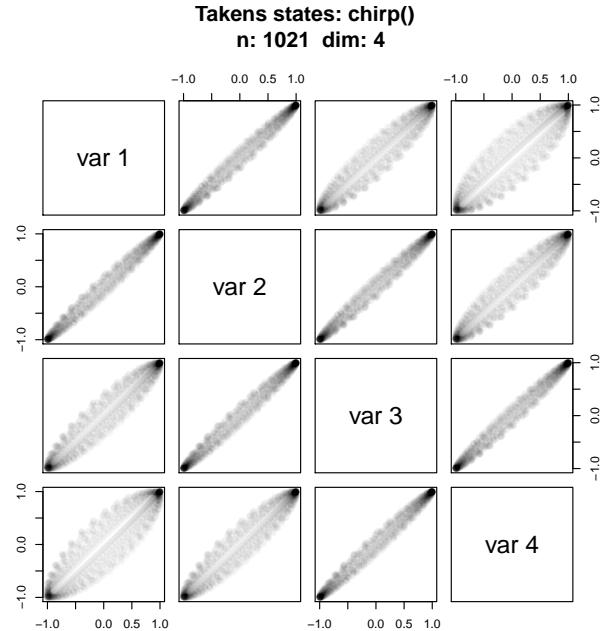


FIGURE 13. Takens states. Test case: chirp signal. Time used: 0.636 sec.

---

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

---

See Figure 14 on the next page.

---

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

---

See Figure 15 on the facing page.

---

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

---

See Figure 16 on page 22.

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

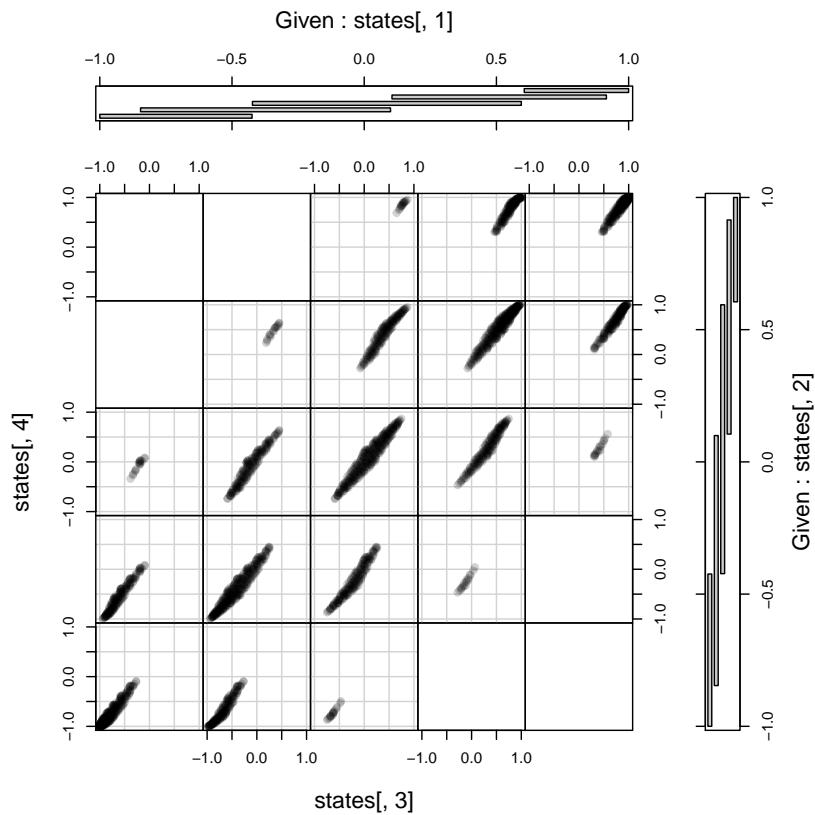


FIGURE 14. State coplot. Test case: chirp signal. Time used: 0.911 sec.

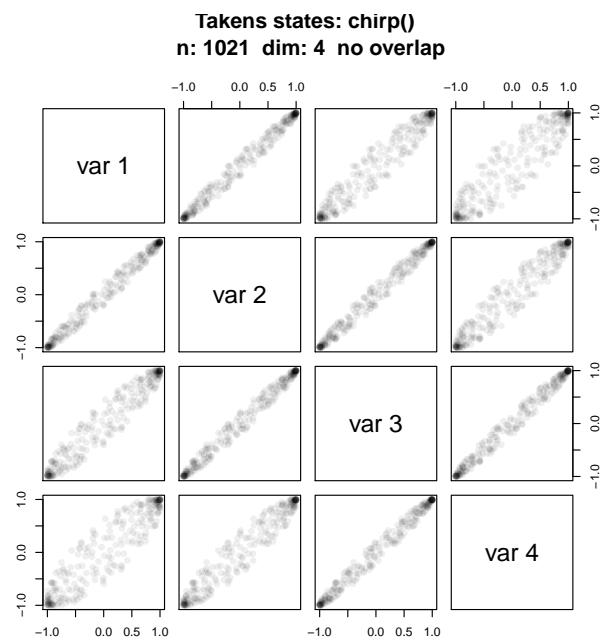


FIGURE 15. Takens states. Test case: chirp signal. Time used: 1.174 sec-

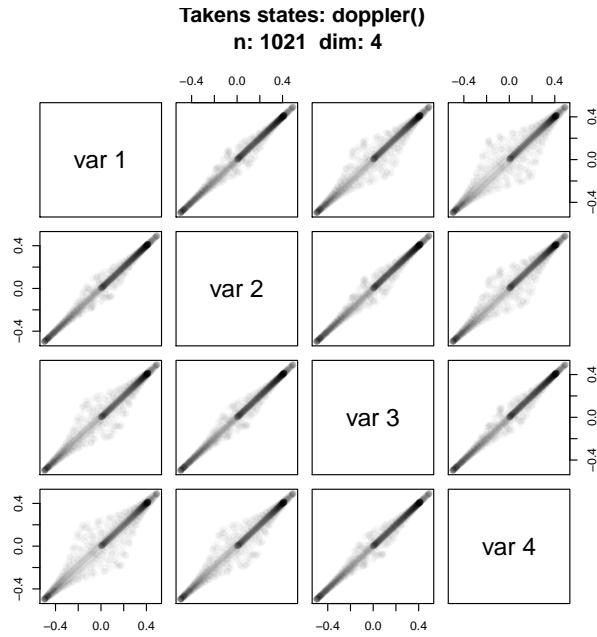


FIGURE 16. Takens states. Test case: Doppler. Note that  $2 - \dim$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.617 sec.

---

*Input*

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

---

See Figure 17 on the facing page.

---

*Input*

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

---

See Figure 18 on the next page.

---

*Input*

```
stateecplot(dopplertakens) #4
```

---

See Figure 19 on page 24.

---

*Input*

```
stateecplot(dopplertakenslag16) #4
```

---

See Figure 20 on page 25.

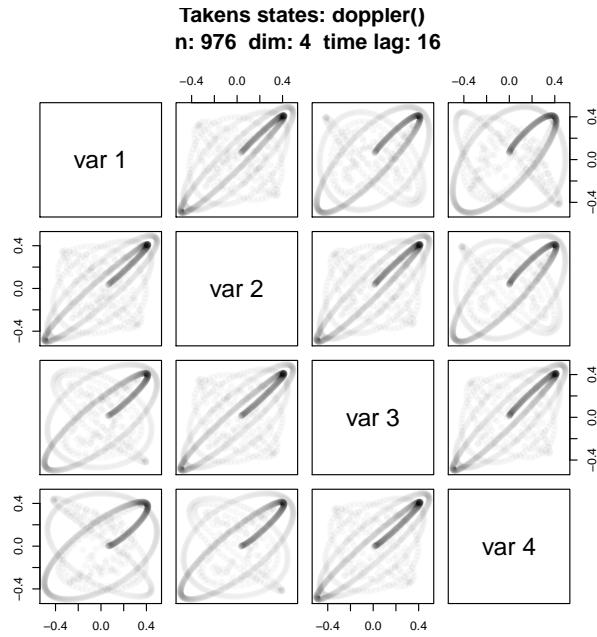


FIGURE 17. Takens states. Test case: doppler at time lag 16. Note that  $2-dim$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.692 sec.

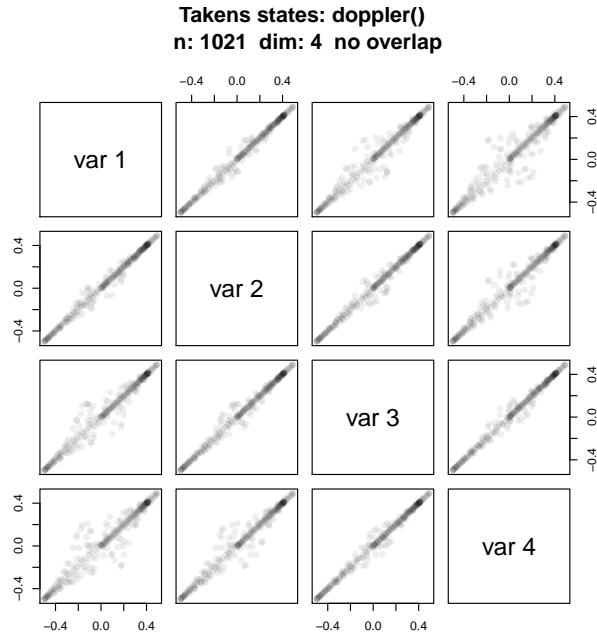


FIGURE 18. Takens states. Test case: doppler, no overlap. Note that  $2-dim$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.978 sec.

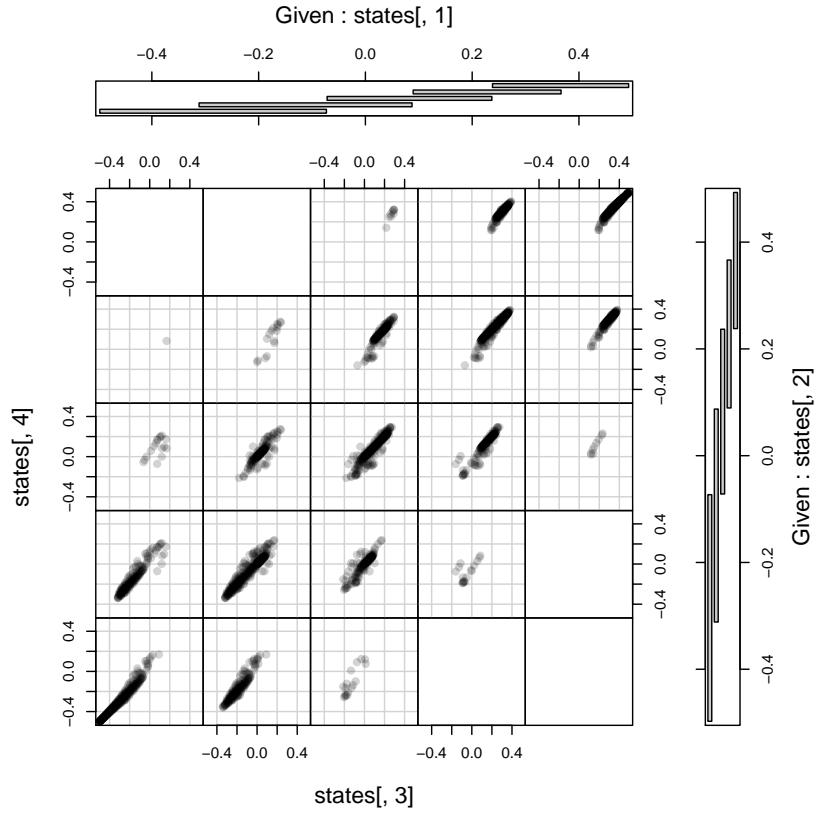


FIGURE 19. State coplot. Test case: Doppler. Note that  $2-dim$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.375 sec.

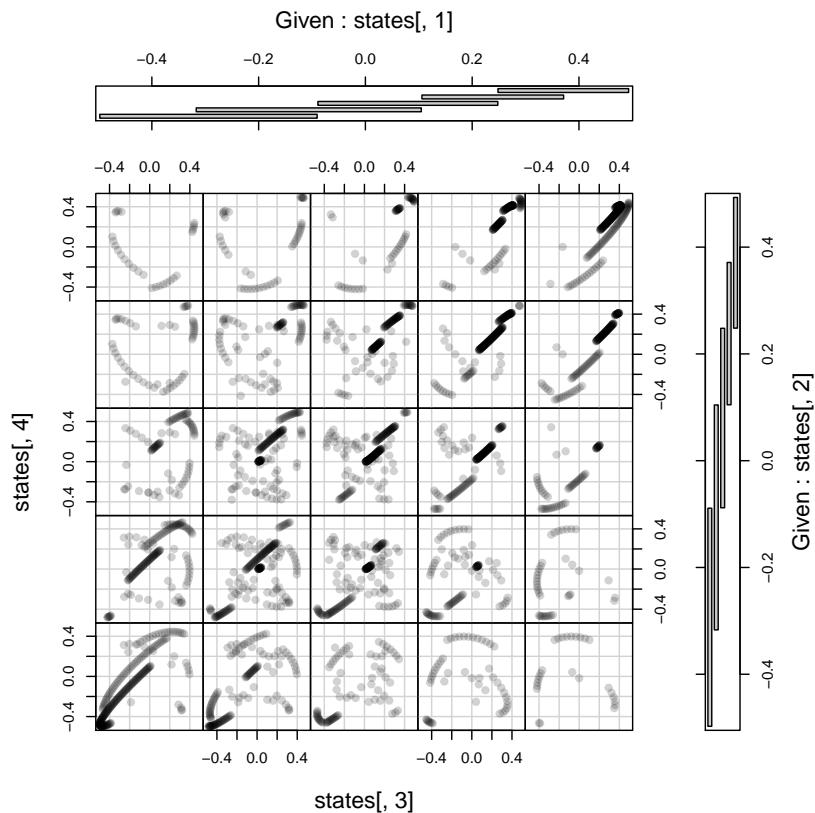


FIGURE 20. State coplot. Test case: Doppler, time lag 16. Note that  $2-dim$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.277 sec.

## 5. RECURRENCE PLOTS FOR TEST SIGNALS

### 5.0.1. Sinus Recurrence Plots.

---

```

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

---

See Figure 21.

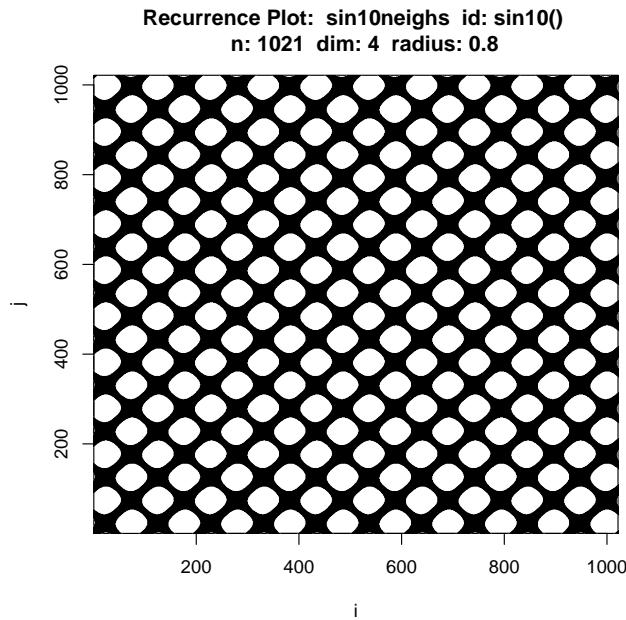


FIGURE 21. Recurrence plot. Test case: sinus curve. Time used: 3.885 sec.

---

```

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

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

---

See Figure 22 on the next page.

---

```

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

---

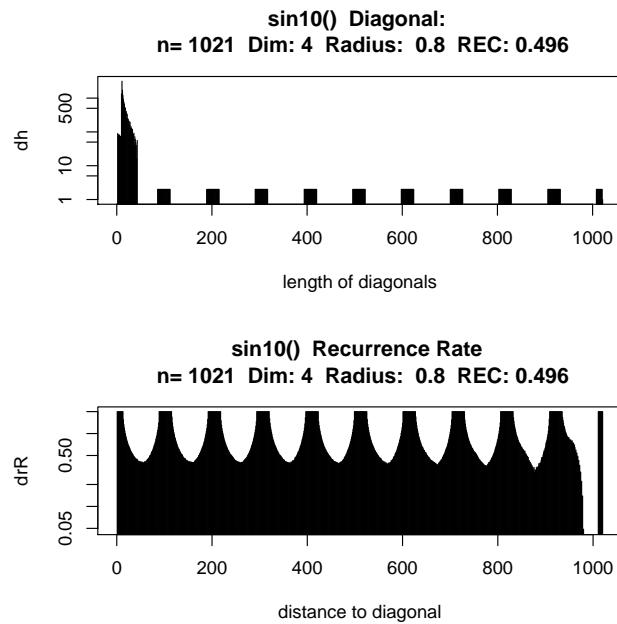


FIGURE 22. RQA. Test case: sinus curve. Time used: 0.821 sec.

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

See Figure 23.

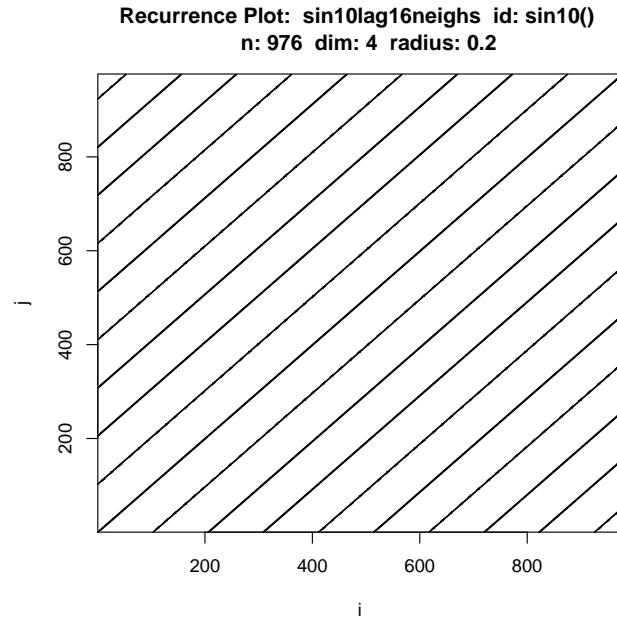


FIGURE 23. Recurrence plot. Recurrence Plot. Test case: sinus curve, time lag 16. Time used: 1.837 sec.

---

*Input*

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

---

*Output*

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

See Figure 24.

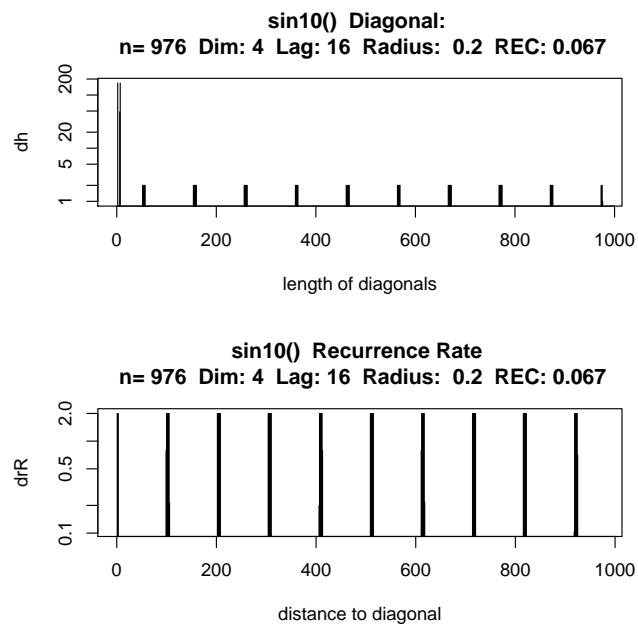


FIGURE 24. RQA. Test case: sinus curve, time lag 16. Time used: 2.246 sec.

5.0.2. *Uniform Random Numbers Recurrence Plots.*

---

*Input*

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

---

See Figure 25.

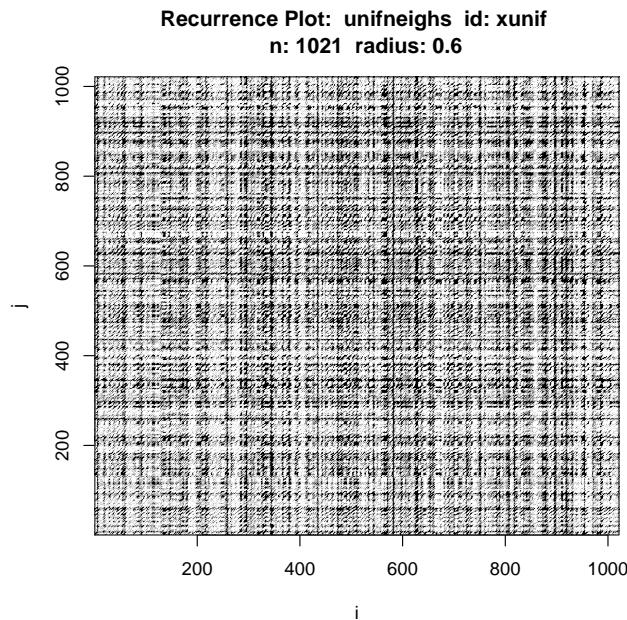


FIGURE 25. Recurrence plot. Test case: uniform random numbers. Time used: 3.306 sec.

---

*Input*

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

---

*Output*

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

---

See Figure 26 on the next page.

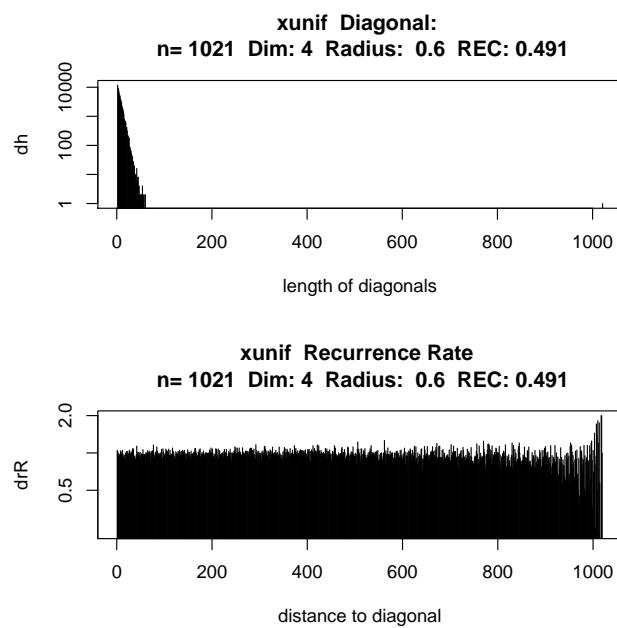


FIGURE 26. RQA. Test case: uniform random numbers, radius=0.6.  
Time used: 4.171 sec.

5.0.3. *Chirp Signal Recurrence Plots.*

---

*Input*

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

---

See Figure 27.

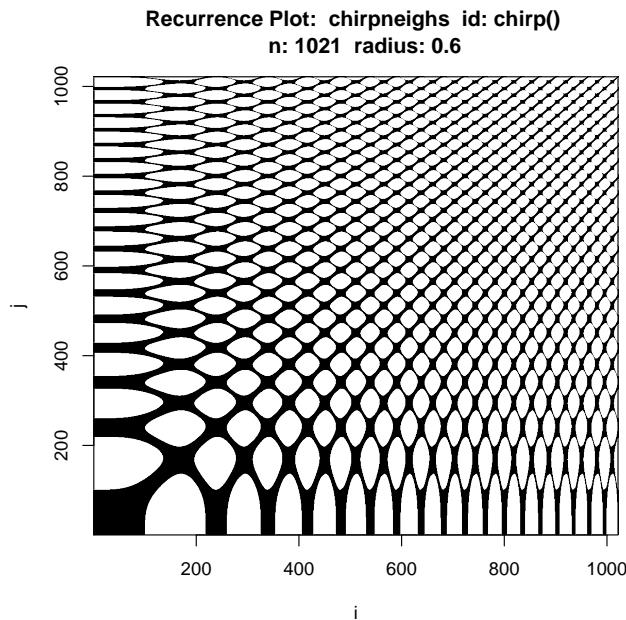


FIGURE 27. Recurrence plot. Test case: chirp signal. Time used: 2.289 sec.

---

*Input*

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

---

*Output*

```
chirp() n: 1021 Dim: 4
Radius: 0.6 Recurrence coverage REC: 0.341 log(REC)/log(R): 2.108
Determinism: 0.988 Laminarity: 0.998
DIV: 0.001
Trend: 0 Entropy: 3.254
Diagonal lines max: 1020 Mean: 12.496 Mean off main: 12.46
Vertical lines max: 125 Mean: 14.721
```

---

See Figure 28 on the next page.

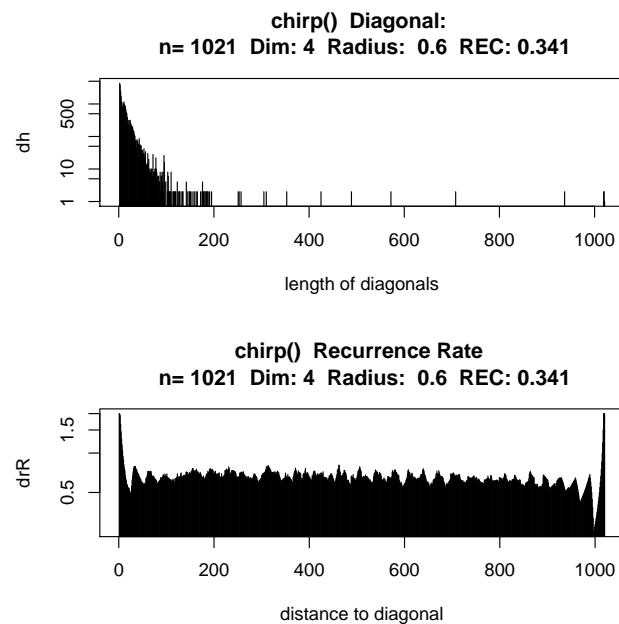


FIGURE 28. RQA. Test case: chirp signal. Time used: 2.932 sec.

5.0.4. *Doppler Recurrence Plots.*


---

*Input*

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

---



---

*Input*

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

---

See Figure 29.

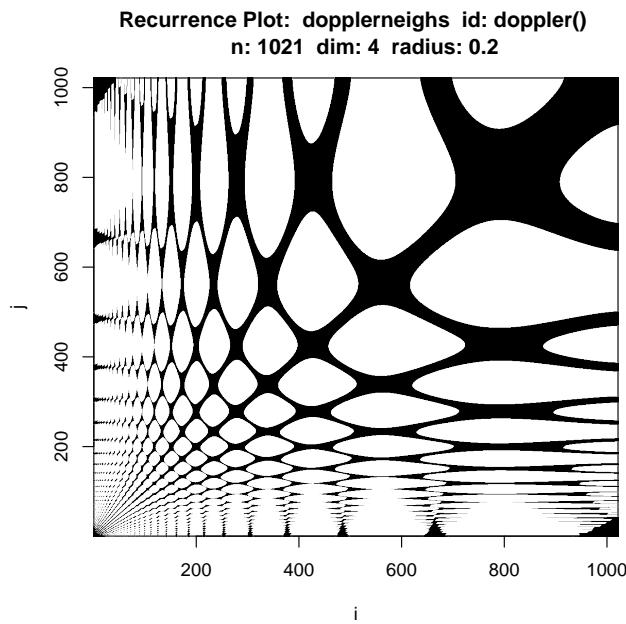


FIGURE 29. Recurrence plot. Test case: Doppler. Time used: 1.635 sec.

---

*Input*

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

---

*Output*

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

---

See Figure 30 on the next page.

---

*Input*

```
dopplerlag16neighs <- local.findAllNeighbours(dopplertakenslag16, radius=0.2)
save(dopplerlag16neighs, file="dopplerlag16neighs.Rdata")
```

---

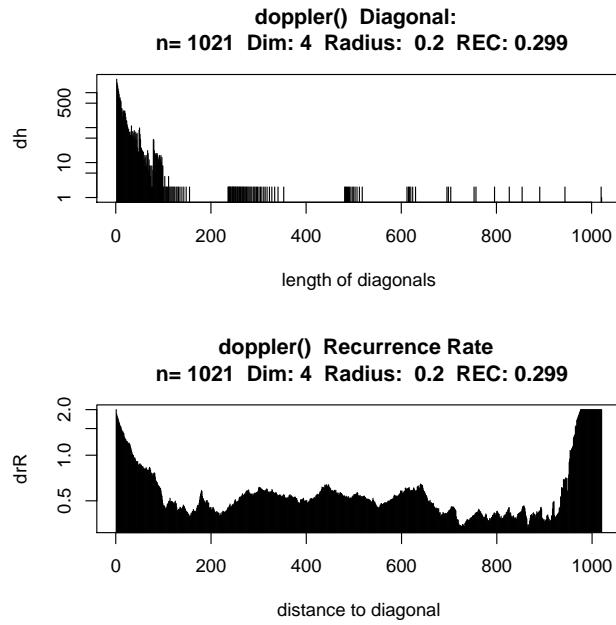


FIGURE 30. RQA. Test case: Doppler. Time used: 0.579 sec.

```
# load(file="dopplerlag16neighs.RData")
local.recurrencePlotAux(dopplerlag16neighs, dim=4, radius=0.2)
```

See Figure 31.

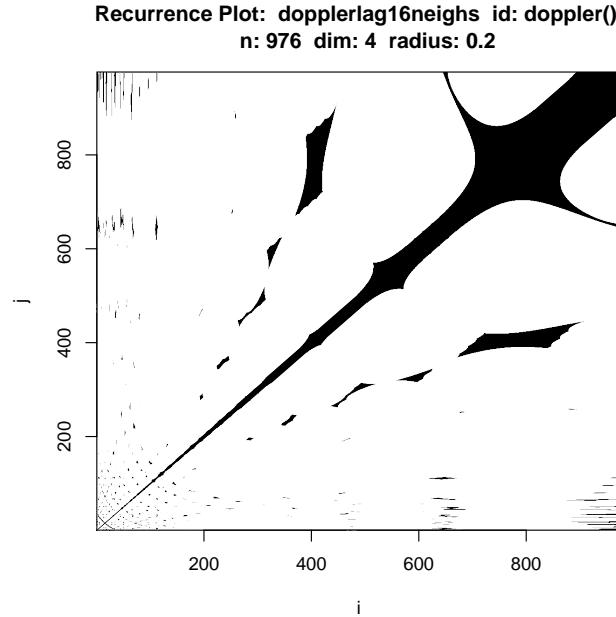


FIGURE 31. Recurrence plot. Test case: Doppler. Time used: 1.253 sec.

---

*Input*

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

---

*Output*

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

See Figure 32.

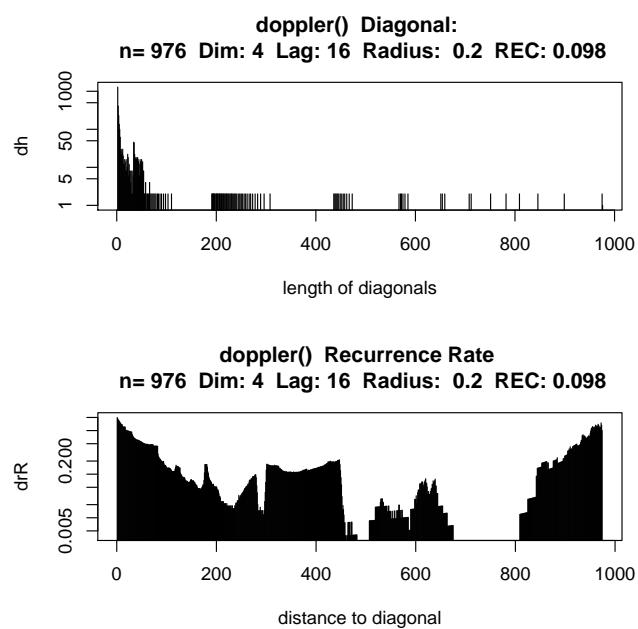


FIGURE 32. RQA. RQA. Test case: Doppler. Time used: 1.684 sec.

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

**ToDo:** Geyser extended to two-dimensional data in `geyserlin`. Experimental only. Check.

## 6. CASE STUDY: GEYSER DATA

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

The data structure asks for a variant of the recurrence plot, adapted to a two dimensional series.

---

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

Input

### 6.1. Geyser Eruption Durations.

---

```
plotSignal(geyser$duration)
```

Input

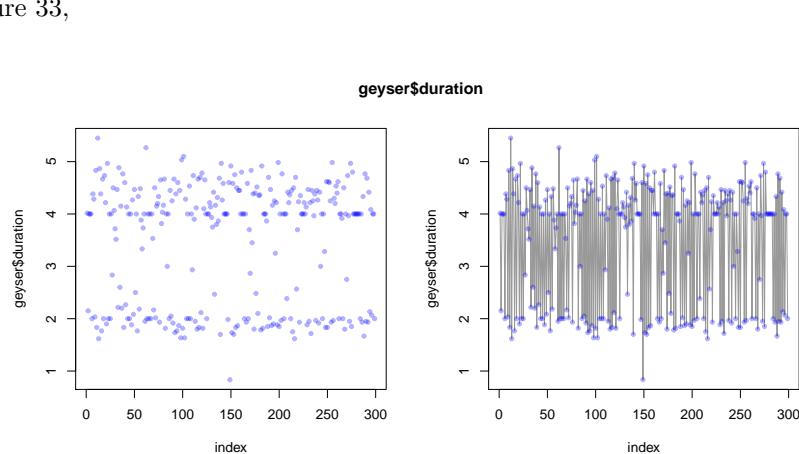


FIGURE 33. Example case: Old Faithful Geyser eruption durations. Signal and linear interpolation.

---

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

Input

See Figure 34 on the next page.

---

```
statecoplot(eruptionstakens4) #dim=4
```

Input

See Figure 35 on the facing page.

---

```
statepairs(eruptionstakens4, nooverlap=TRUE) #dim=4
```

Input

See Figure 36 on page 38.

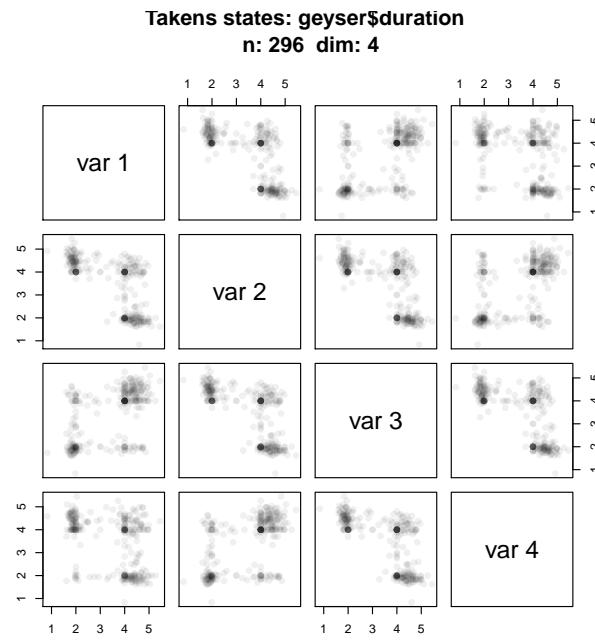


FIGURE 34. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Time used: 0.27 sec.

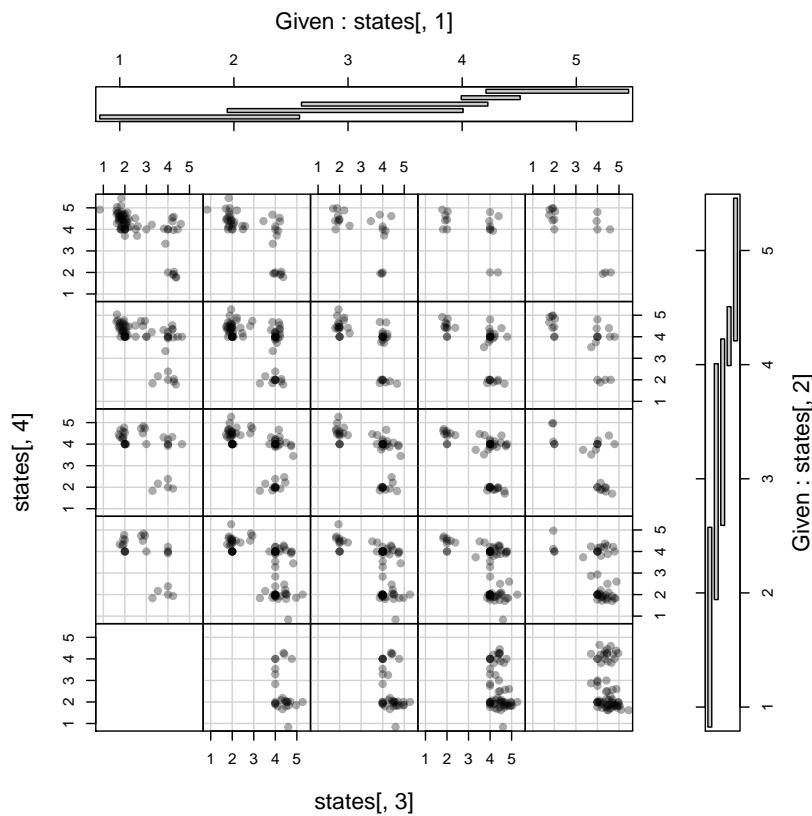


FIGURE 35. State coplot. Example case: Old Faithful Geyser eruption durations. Time used: 0.435 sec.

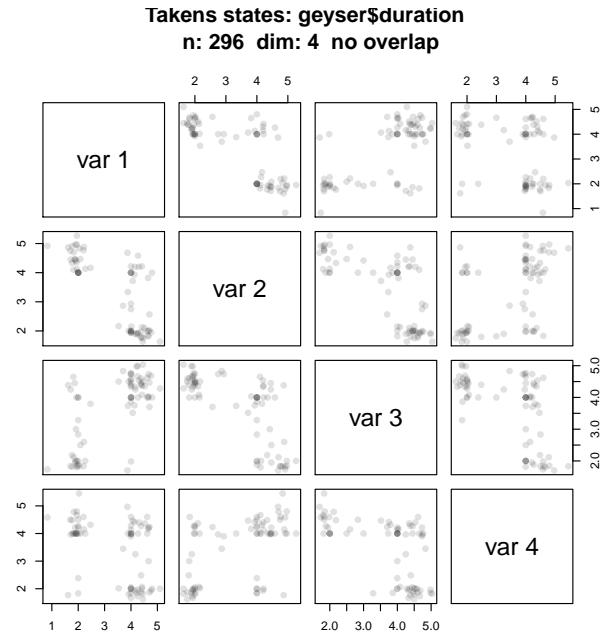


FIGURE 36. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Time used: 0.576 sec.

---

Input

```
eruptionsneighs4 <- local.findAllNeighbours(eruptionstakens4, radius=3*0.8)
save(eruptionsneighs4, file="eruptionsneighs4.RData")
```

---



---

Input

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

---

See Figure 37 on the facing page.

---

Input

```
showrqa(eruptionstakens4, radius=3*0.8)
```

---



---

Output

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

---

See Figure 38 on the next page.

### 6.1.1. Geyser eruption durations. Dim=2.

---

Input

---

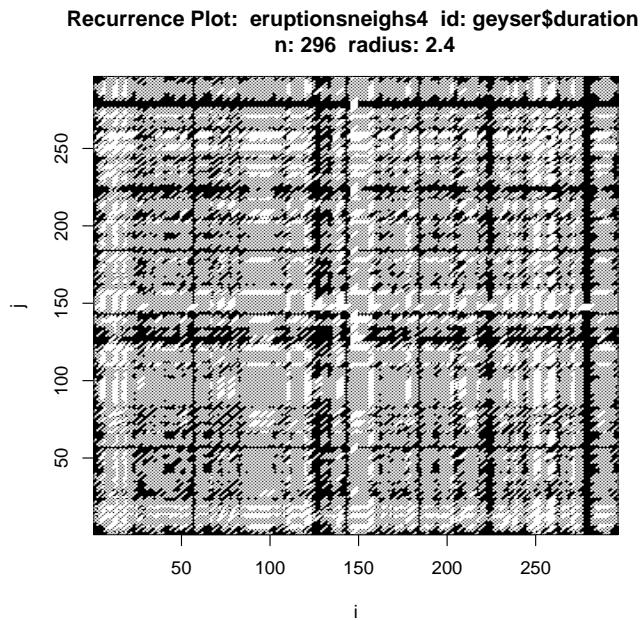


FIGURE 37. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.269 sec.

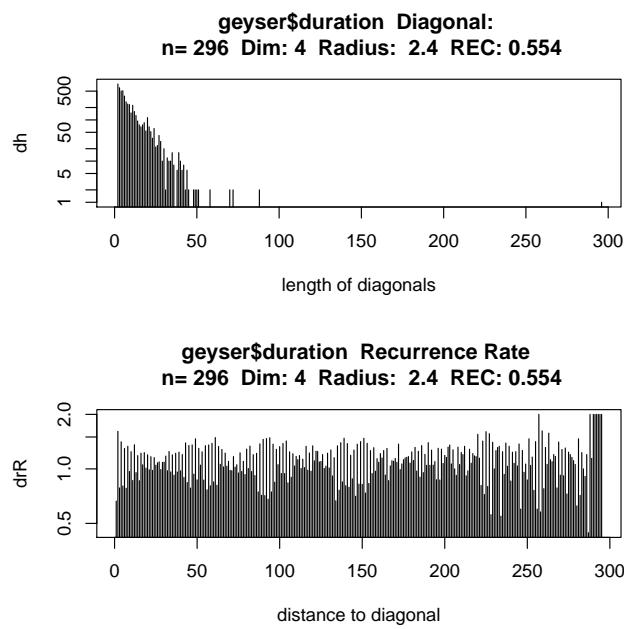


FIGURE 38. RQA. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.43 sec.

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

See Figure 39.

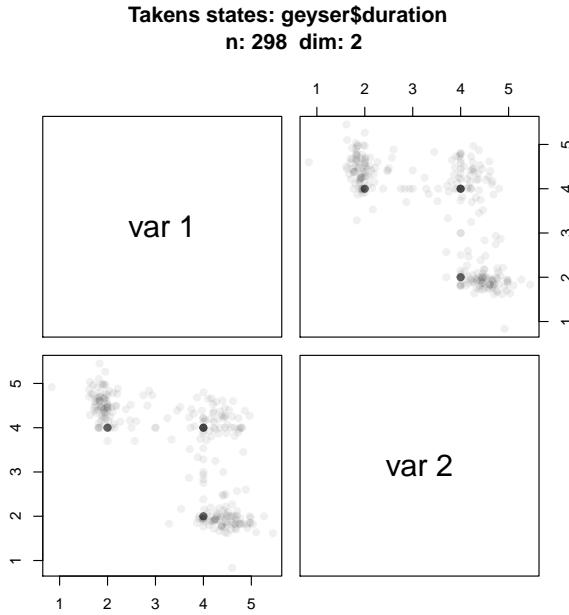


FIGURE 39. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=2. Time used: 0.105 sec.

---

*Input*

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

---

See Figure 40 on the next page

---

*Input*

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

---

See Figure 41 on the facing page.

---

*Input*

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

---

*Output*

```
geyser$duration n: 298 Dim: 2
Radius: 0.8 Recurrence coverage REC: 0.274 log(REC)/log(R): 5.806
Determinism: 0.892 Laminarity: 0.205
DIV: 0.045
Trend: 0 Entropy: 1.691
Diagonal lines max: 22 Mean: 3.644 Mean off main: 3.595
Vertical lines max: 7 Mean: 3.457
```

---

See Figure 42 on page 42.

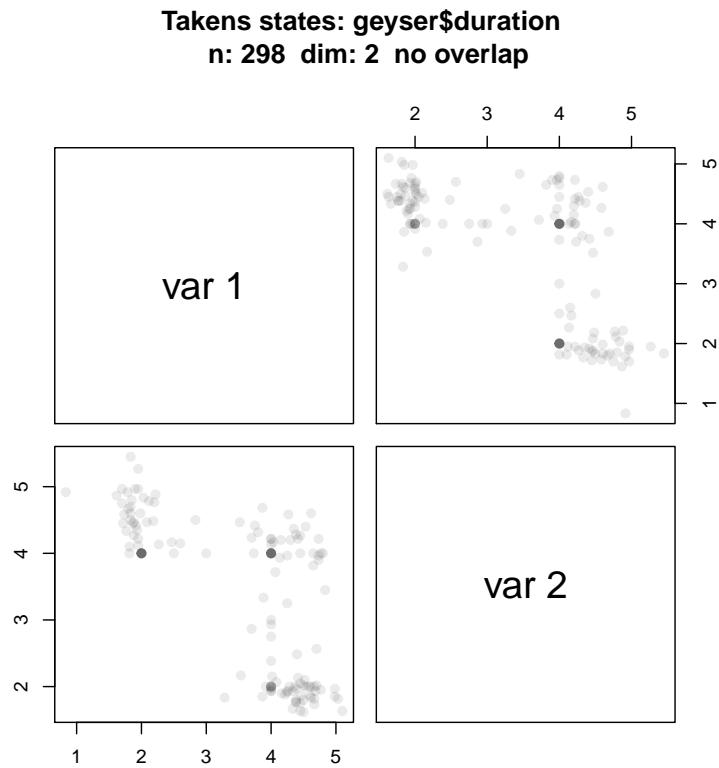


FIGURE 40. Example case: Old Faithful Geyser eruption durations.  
 Dim=2. Time used: 0.205 sec.

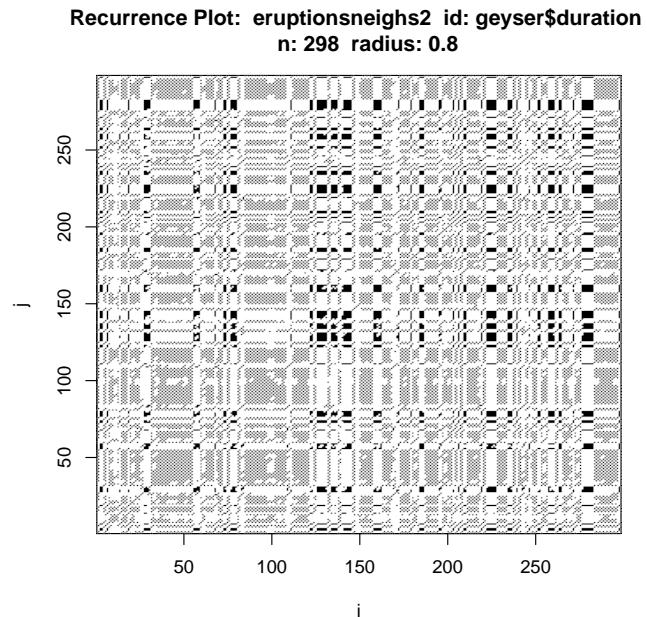


FIGURE 41. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=2. Time used: 0.247 sec.

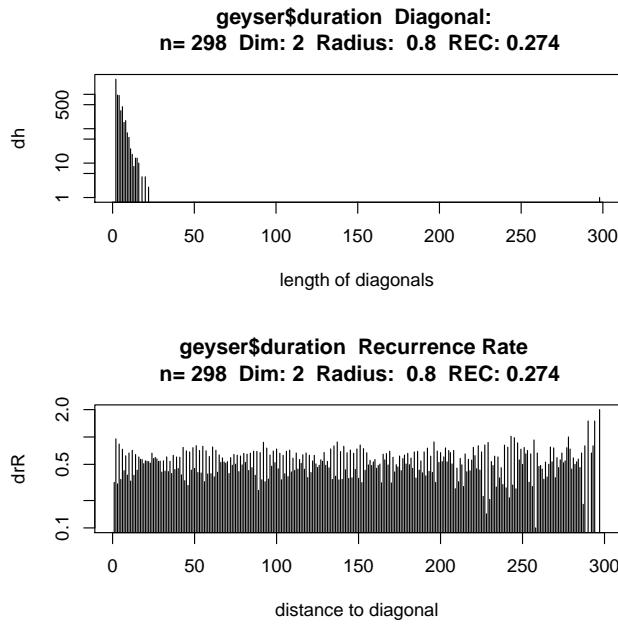


FIGURE 42. RQA. Example case: Old Faithful Geyser eruption durations. Dim=2. Time used: 0.397 sec.

#### 6.1.2. Geyser eruptions. Dim=4.

---

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

---

See Figure 43 on the facing page.

---

```
Input
statepairs(eruptionstakens4, nooverlap=TRUE) #dim=4
```

---

See Figure 44 on the next page.

---

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

---



---

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

---

See Figure 45 on page 44.

---

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

---

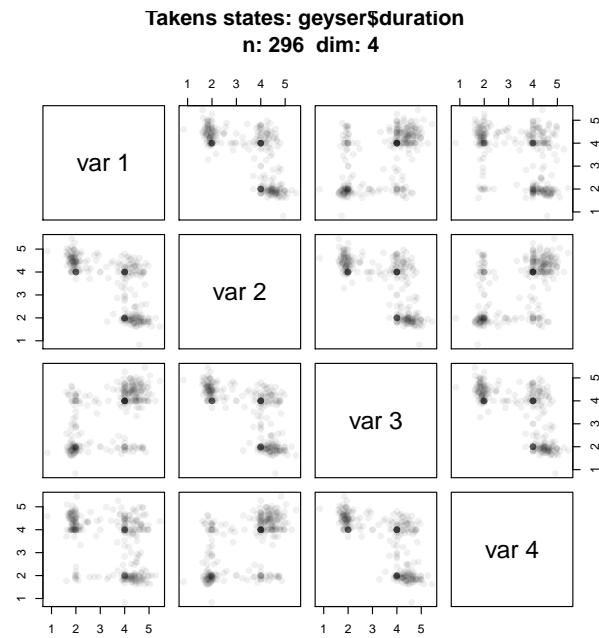


FIGURE 43. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.244 sec.

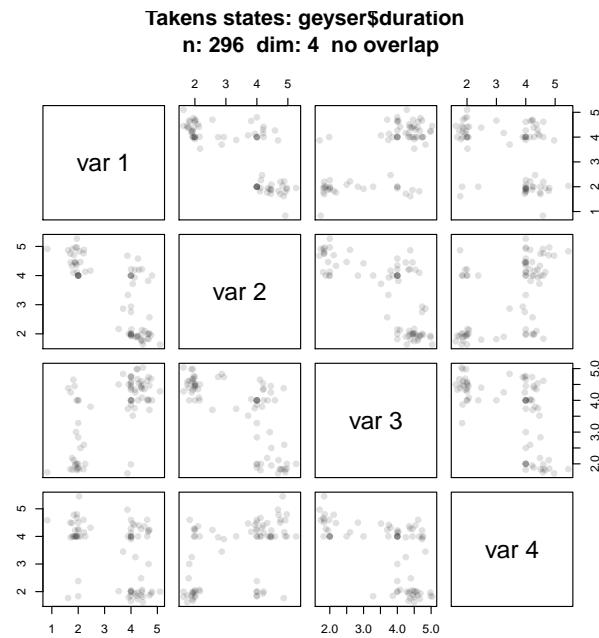


FIGURE 44. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.391 sec.

---

Output

---

```
geyser$duration n: 296 Dim: 4
Radius: 0.8 Recurrence coverage REC: 0.112 log(REC)/log(R): 9.822
Determinism: 0.903 Laminarity: 0.076
DIV: 0.05
```

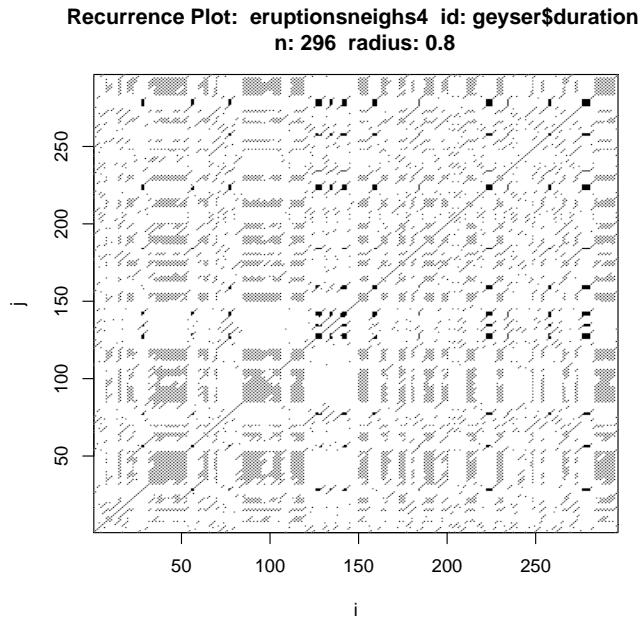


FIGURE 45. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.138 sec.

```
Trend: 0 Entropy: 1.779
Diagonal lines max: 20 Mean: 3.919 Mean off main: 3.79
Vertical lines max: 5 Mean: 3.041
```

See Figure 46.

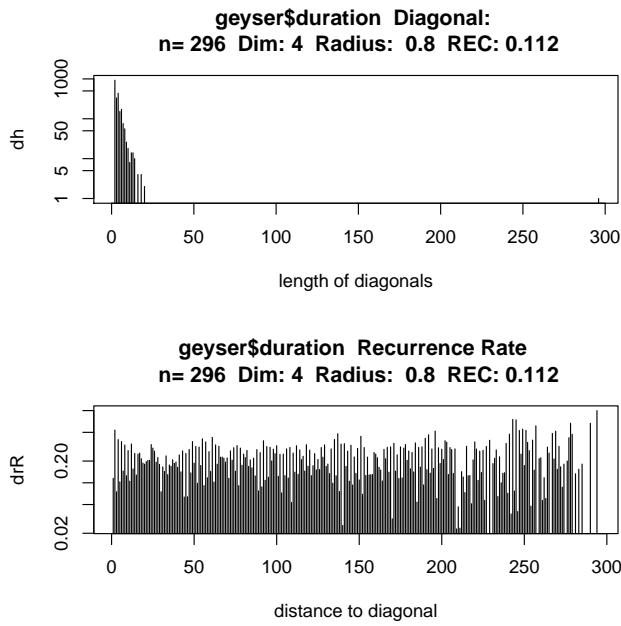


FIGURE 46. Recurrence plot. .

Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.279 sec.

### 6.1.3. Geyser eruption durations. Dim=8.

---

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

---

See Figure 47.

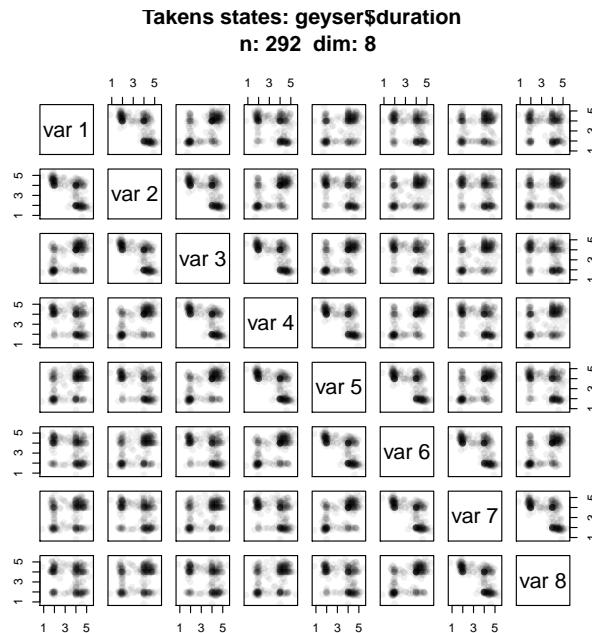


FIGURE 47. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 0.831 sec.

---

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

---

See Figure 48 on the next page.

---

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

---

See Figure 49 on page 47.

---

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

---

**Takens states: geyser\$duration  
n: 292 dim: 8 no overlap**

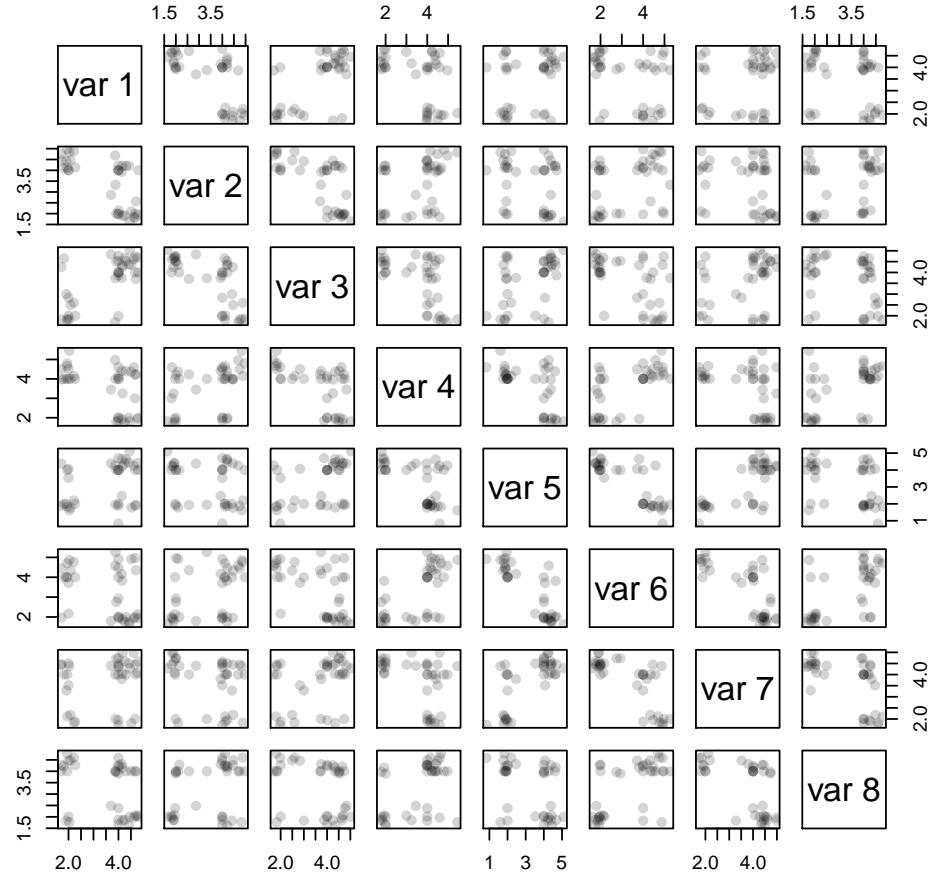


FIGURE 48. Example case: Old Faithful Geyser eruption durations.  
Dim=8. Time used: 1.086 sec.

---

Output

---

```

geyser$duration n: 292 Dim: 8
Radius: 2.6 Recurrence coverage REC: 0.494 log(REC)/log(R): -0.738
Determinism: 0.991 Laminarity: 0.5
DIV: 0.011
Trend: 0 Entropy: 3.338
Diagonal lines max: 93 Mean: 12.635 Mean off main: 12.55
Vertical lines max: 36 Mean: 4.073

```

---

See Figure 50 on the next page.

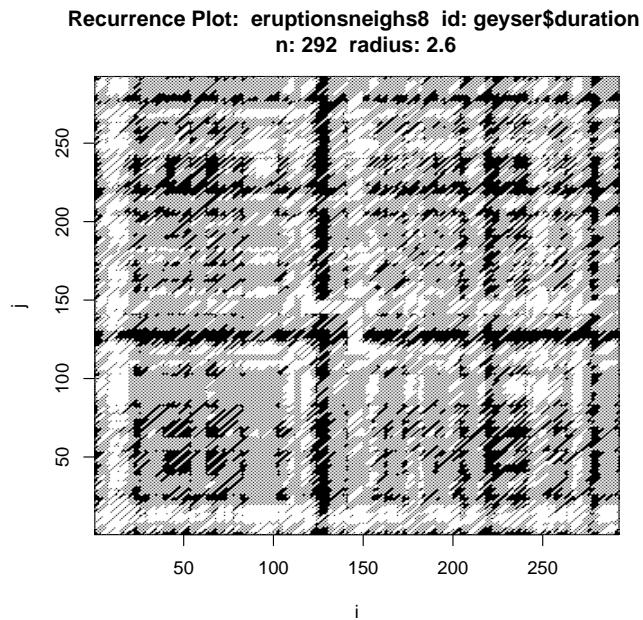


FIGURE 49. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 0.299 sec.

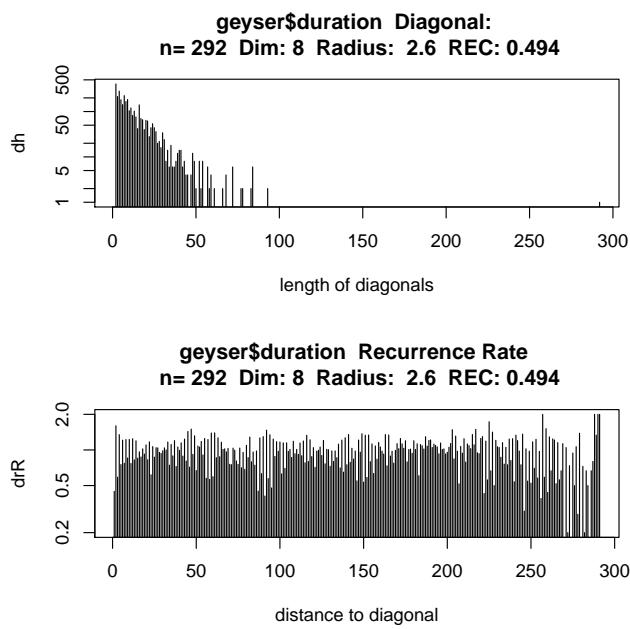


FIGURE 50. RQA. Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 0.451 sec.

6.1.4. *Geyser eruption durations: Comparison by Dimension.* For comparison, recurrence plots for the Geyser data with varying dimension are in Figure 51

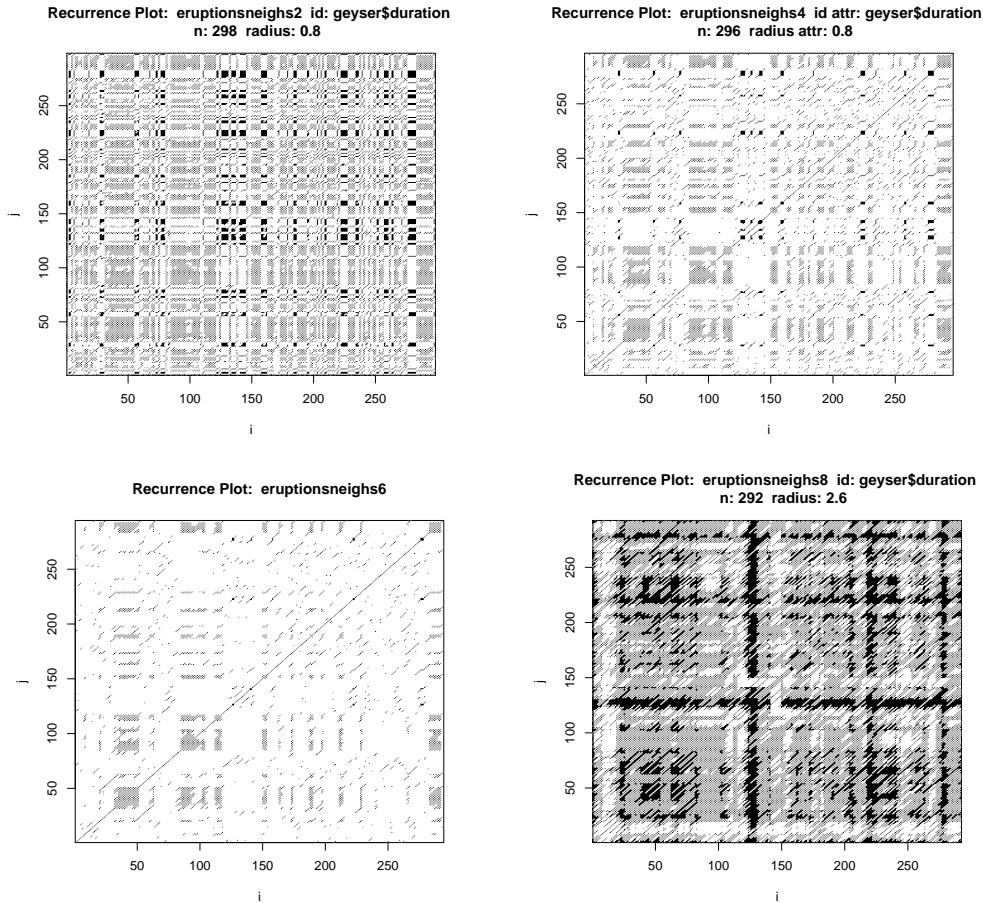


FIGURE 51. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=2, 4, 6, 8.

## 6.2. Geyser Waiting.

---

`plotsignal(geyser$waiting)`

*Input*

---

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

See Figure 52 on the next page.

---

`Input`

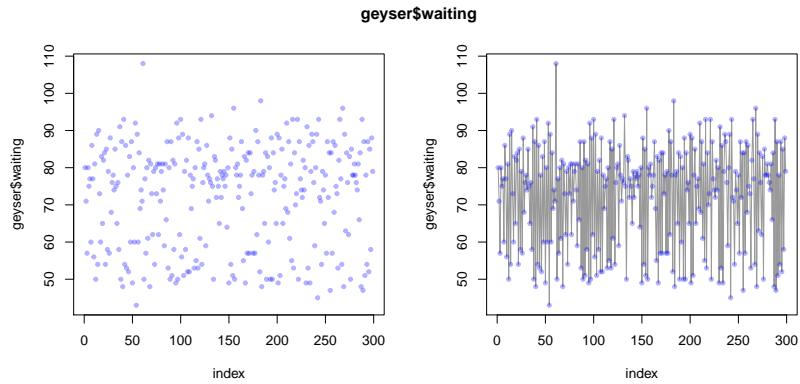


FIGURE 52. Example case: Old Faithful Geyser waiting. Signal and linear interpolation. Time used: 0.653 sec.

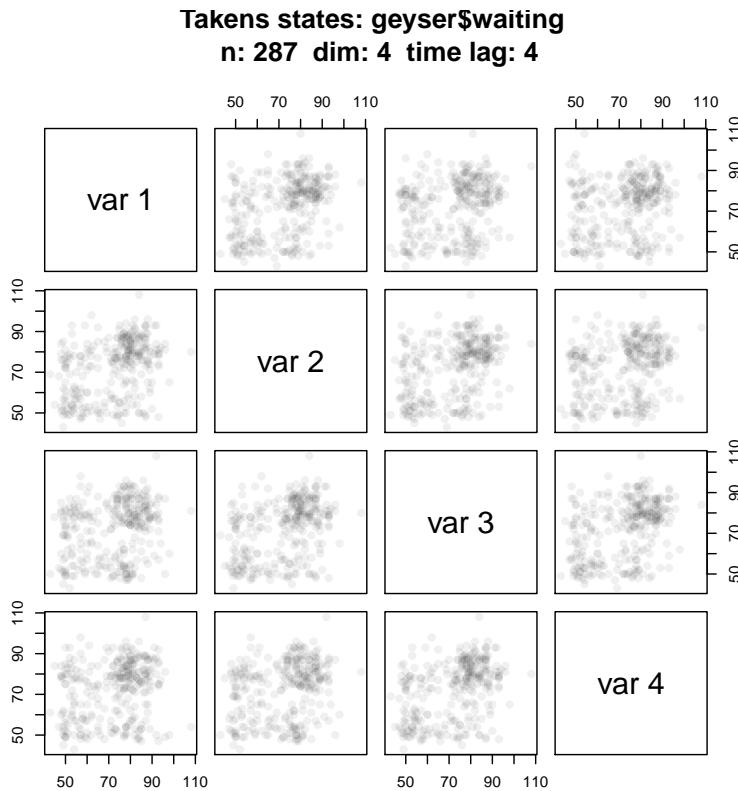


FIGURE 53. Example case: Old Faithful Geyser waiting. Time used: 0.262 sec.

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

---

*Input*

---

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

---

*Output*

---

```
geyser$waiting n: 287 Dim: 4
Radius: 16 Recurrence coverage REC: 0.137 log(REC)/log(R): -0.718
Determinism: 0.382 Laminarity: 0.053
DIV: 0.053
Trend: 0 Entropy: 1.002
Diagonal lines max: 19 Mean: 2.878 Mean off main: 2.688
Vertical lines max: 3 Mean: 2.315
```

See Figure 54.

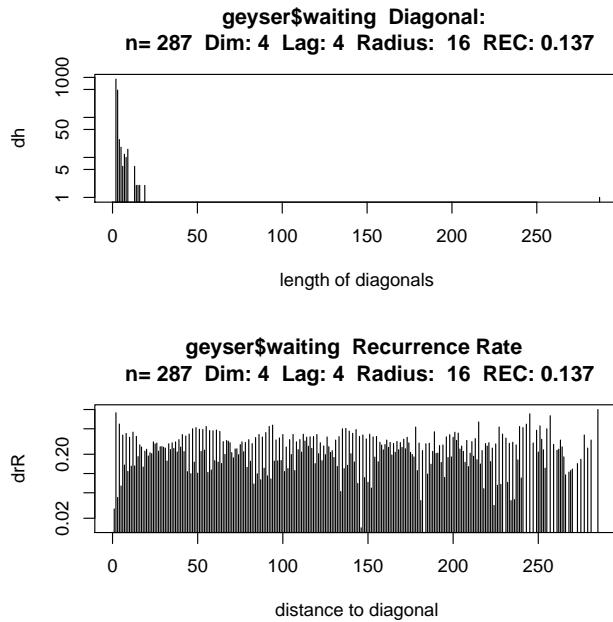


FIGURE 54. RQA. Example case: Old Faithful Geyser waiting. Time used: 0.147 sec.

---

*Input*

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

---

See Figure 55 on the facing page.

**6.3. Geyser - linearized.** The Geyser data is a bivariate series and asks for bivariate Takens states and recurrence plots. This is a first crude attempt and need improvement.

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

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

---

*Input*

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

---

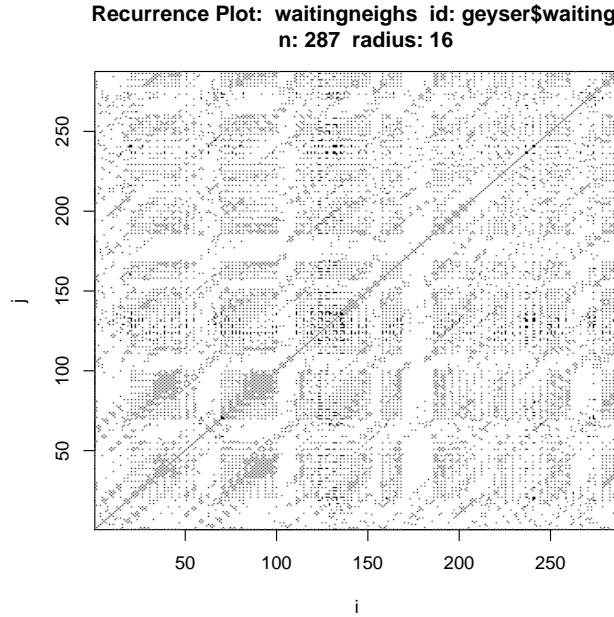


FIGURE 55. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser waiting. Time used: 0.3 sec.

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

---

*Input*

```
plotsignal(geyserlin)
```

---

See Figure 56.

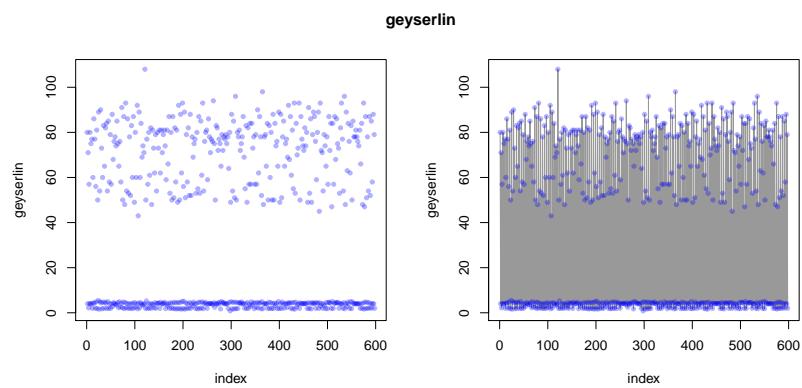


FIGURE 56. Example case: Old Faithful Geyser eruptions. Signal and linear interpolation.

---

*Input*

```
glineruptionstakens4 <-  
local.buildTakens( time.series=geyserlin,
```

---

```
embedding.dim=4, time.lag=2)
statepairs(glineruptionstakens4) #dim=4
```

See Figure 57.

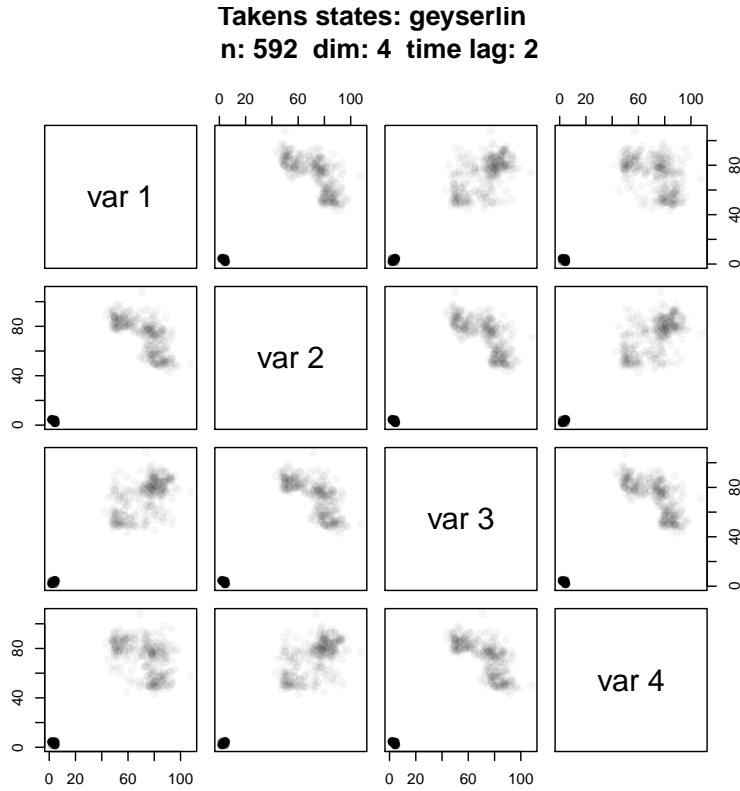


FIGURE 57. Example case: Old Faithful Geyser eruptions. Time used: 0.395 sec.

---

*Input*

```
glineruptionsneighs4 <- local.findAllNeighbours(glineruptionstakens4, radius=0.8)
save(glineruptionsneighs4, file="glineruptionsneighs4.RData")
```

---



---

*Input*

```
showrqa(glineruptionstakens4, radius=0.8)
```

---



---

*Output*

```
geyserlin n: 592 Dim: 4
Radius: 0.8 Recurrence coverage REC: 0.029 log(REC)/log(R): 15.901
Determinism: 0.059 Laminarity: 0
DIV: Inf
Trend: 0 Entropy: 0
Diagonal lines max: 0 Mean: 592 Mean off main: NaN
Vertical lines max: 0 Mean: 0
```

---

See Figure 58 on the next page.

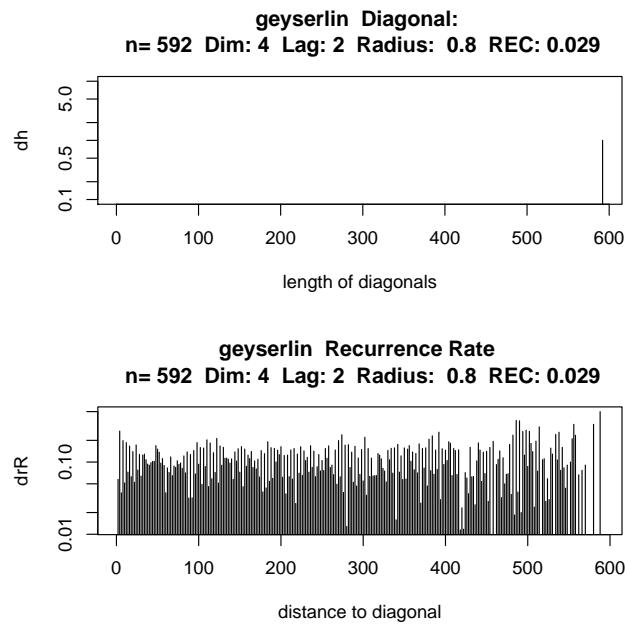


FIGURE 58. Recurrence plot. Old Faithful Geyser - both. Dim=4. Time used: 0.215 sec.

---

*Input*

```
load(file="glineruptionsneighs4.RData")
local.recurrencePlotAux(glineruptionsneighs4)
```

See Figure 59.

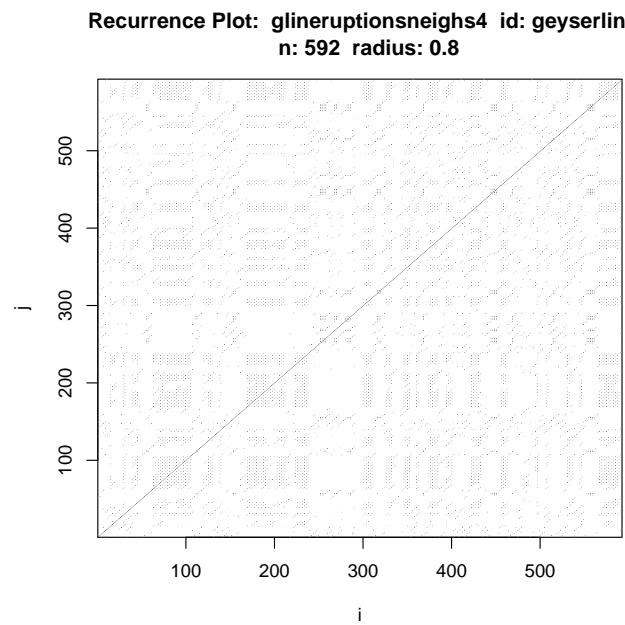


FIGURE 59. Recurrence plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.497 sec.

6.3.1. *Geyser eruptions - linearized. Dim=2.*

---

*Input*

```
glineruptonestakens2 <-
  local.buildTakens(time.series=geyserlin,
    embedding.dim=2, time.lag=2)
statepairs(glineruptonestakens2) #dim=2
```

---

See Figure 60.

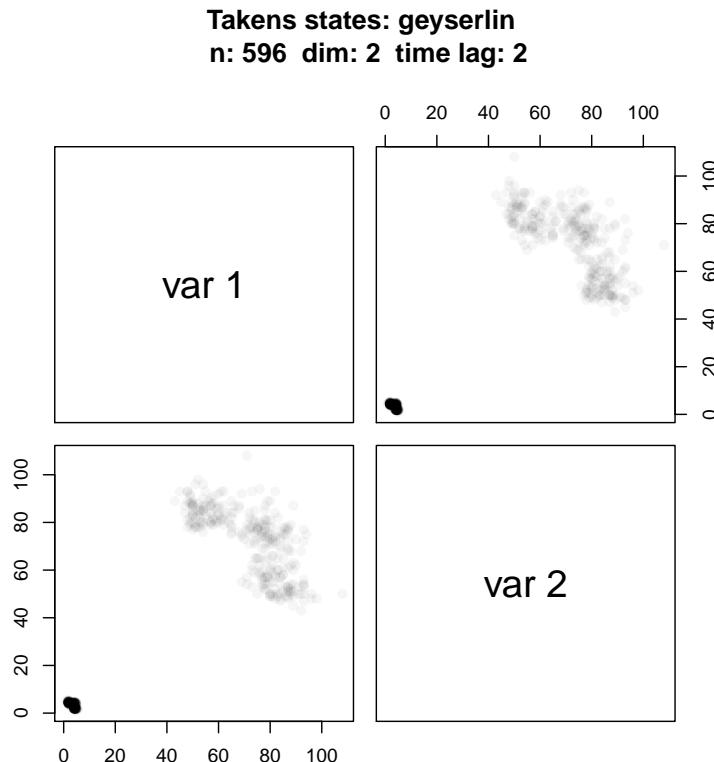


FIGURE 60. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.124 sec.

---

*Input*

```
glineruptonestakens2 <- local.findAllNeighbours(glineruptonestakens2, radius=0.8)
save(glineruptonestakens2, file="glineruptonestakens2.RData")
#load(file="glineruptonestakens2.RData")
local.recurrencePlotAux(glineruptonestakens2)
```

---

See Figure 61 on the facing page.

6.3.2. *Geyser eruptions - linearized. Dim=8.*

---

*Input*

```
glineruptonestakens8 <- local.buildTakens( time.series=geyserlin,
  embedding.dim=8,time.lag=2)
statepairs(glineruptonestakens8) #dim=8
```

---

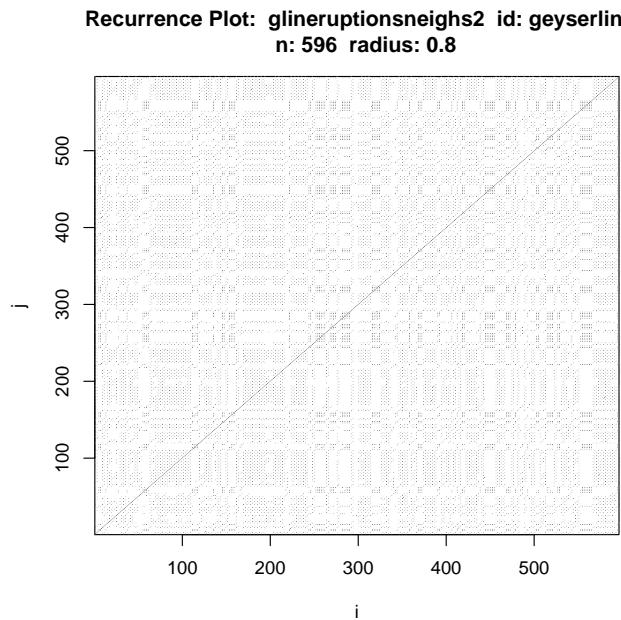


FIGURE 61. Recurrence plot. Example case: Old Faithful Geyser eruptions linearized. Dim=2. Time used: 0.458 sec.

See Figure 62 on the next page

---

*Input*  
`glineruptionsneighs8 <- local.findAllNeighbours(glineruptionsneighs8,  
 radius=0.8)  
 save(glineruptionsneighs8, file="glineruptionsneighs8.RData")`

---



---

*Input*  
`load(file="glineruptionsneighs8.RData")  
 local.recurrencePlotAux(glineruptionsneighs8)`

---

See Figure 63 on page 57.

**Takens states: geyserlin  
n: 584 dim: 8 time lag: 2**

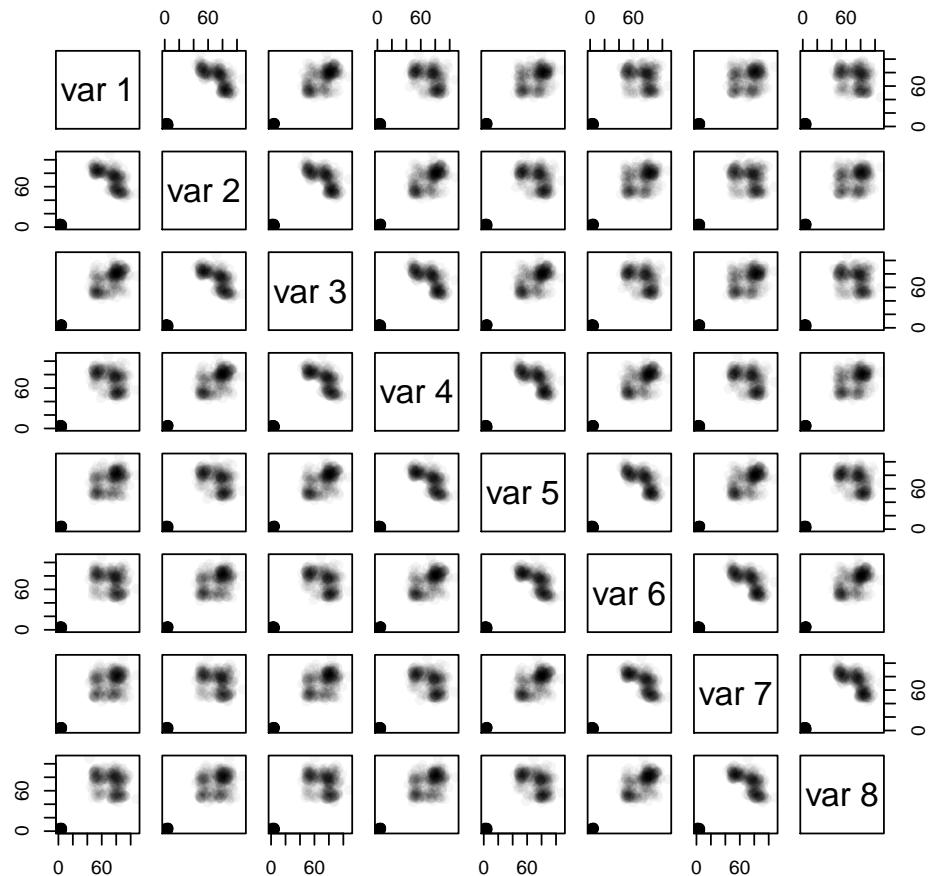


FIGURE 62. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 1.402 sec.

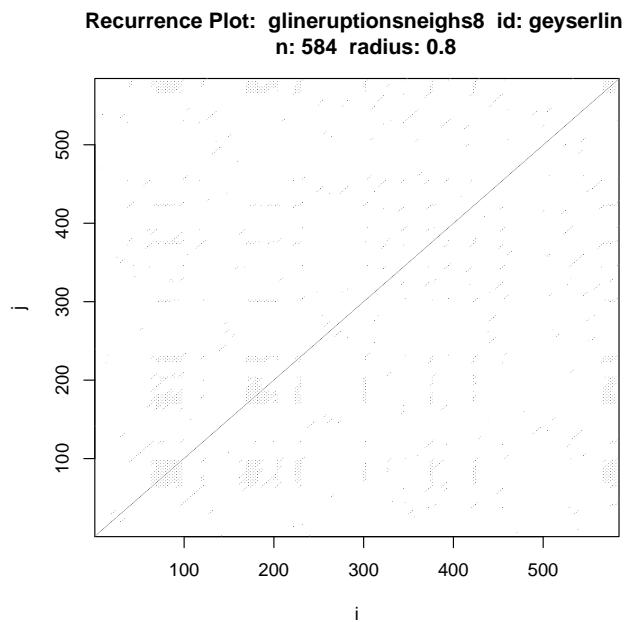


FIGURE 63. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.195 sec.

Only 1024 data points used in recurrence plots in this section

## 7. CASE STUDY: HRV DATA EXAMPLE.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
#####
hrv.data = CreateHRVData()
hrv.data = SetVerbose(hrv.data, TRUE )
#####
### code chunk number 3: loading
#####
hrv.data = LoadBeatAscii(hrv.data, "example.beats",
  RecordPath = "/users/gs/projects/rforge/rhrv/tutorial/beatsFolder")
```

---

*Output*

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

---

*Input*

```
# RecordPath = "beatsFolder")

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

---

*Output*

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

---

*Input*

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

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

See Figure 64 on the next page.

---

*Input*

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

See Figure 65 on the facing page.

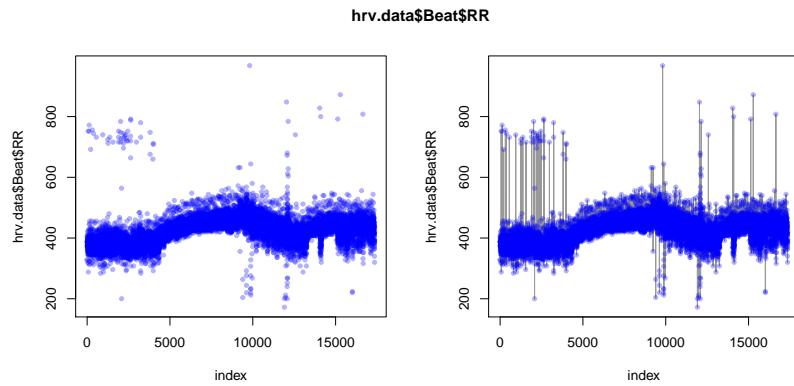


FIGURE 64. RHRV tutorial example.beats. Signal and linear interpolation.

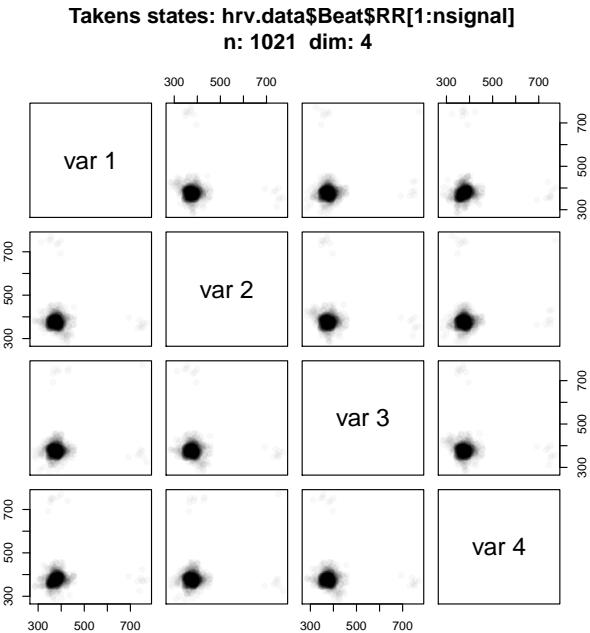


FIGURE 65. Recurrence plot. RHRV tutorial example.beats. Time used: 0.539 sec.

---

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

---

See Figure 66 on the next page.

---

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

---

See Figure 67 on page 61.

See Figure 68 on page 61.

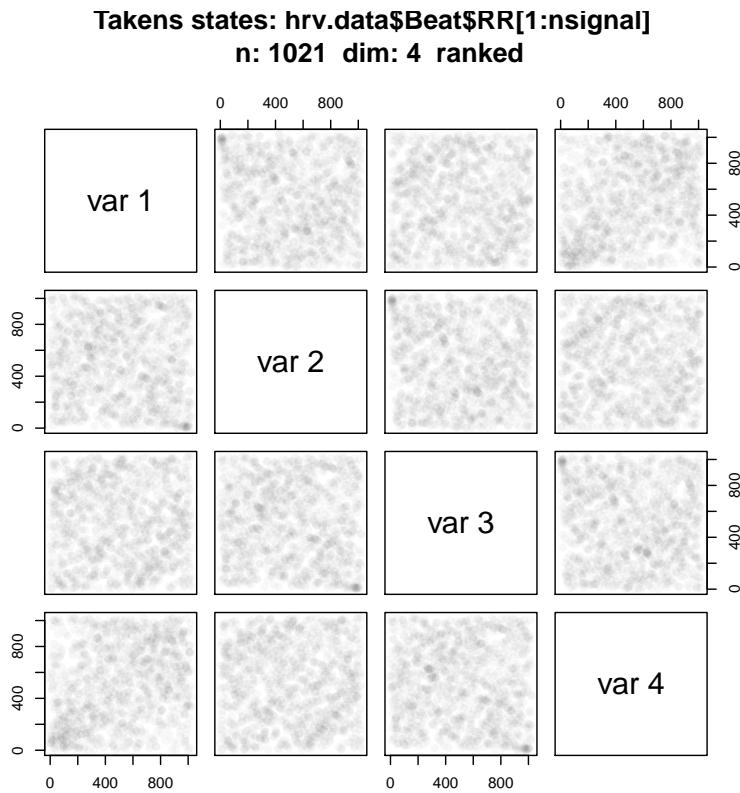


FIGURE 66. RHRV tutorial example.beats. Ranked data. Time used:  
1.176 sec.

---

*Input*

```
hrvRRneighs4 <- local.findAllNeighbours(hrvRRTakens4, radius=16)
save(hrvRRneighs4, file="hrvRRneighs4.Rdata")
```

Time used: 0.131 sec.

---

*Input*

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

See Figure 69 on page 62.

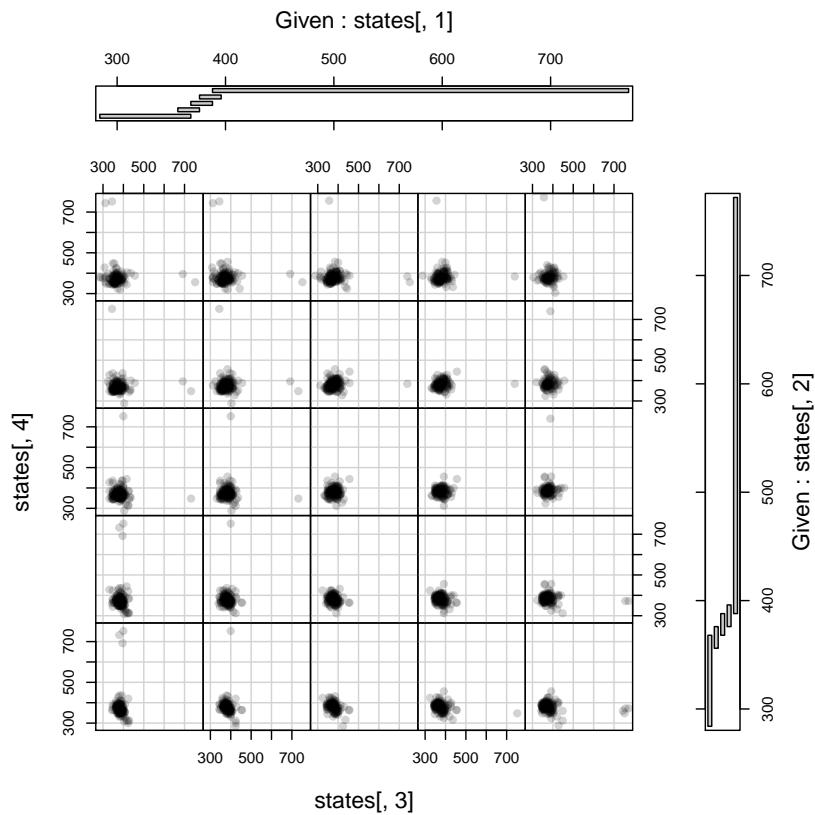


FIGURE 67. State coplot. RHRV tutorial example.beats. Time used:  
0.259 sec.

**Takens states: hrv.data\$Beat\$RR[1:nsignal]  
n= 1021 dim= 4**

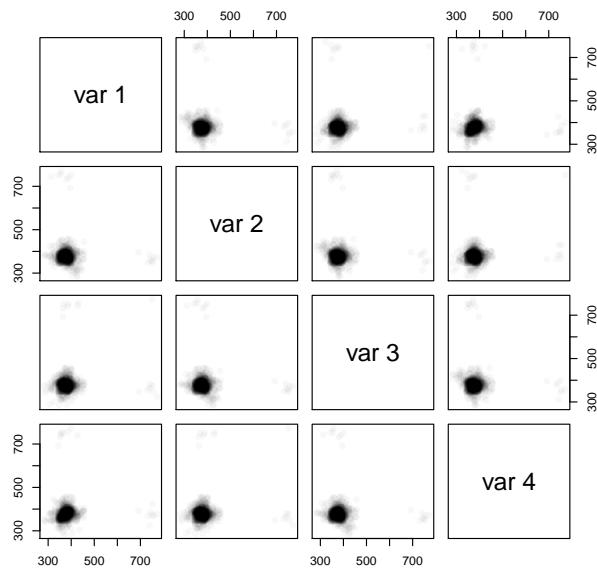


FIGURE 68. Recurrence plot. RHRV tutorial example.beats. Time used:  
0.001 sec.

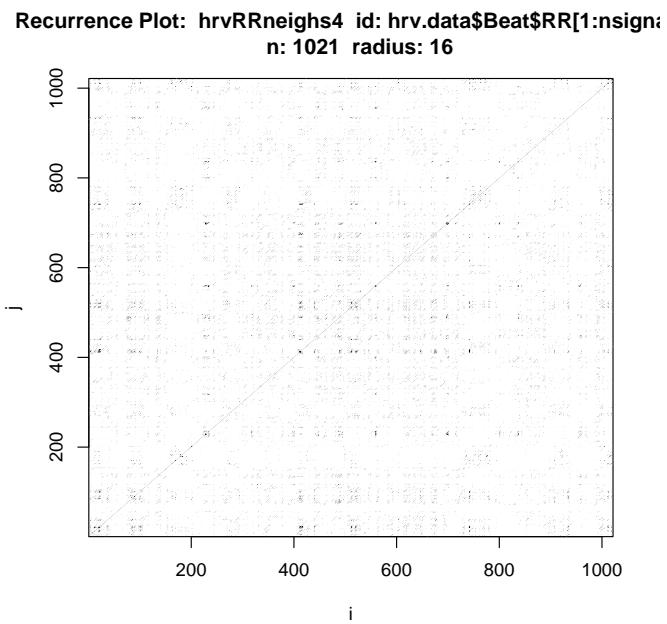


FIGURE 69. Recurrence plot. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=4. Time used: 0.869 sec.

### 7.0.3. RHRV: example.beats - Comparison by Dimension.

---

*Input*

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

---

*Output*

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

See Figure 70.

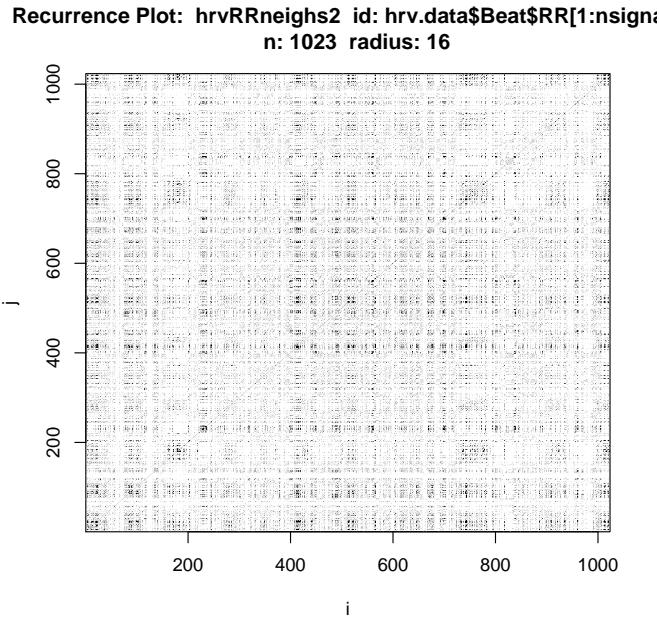


FIGURE 70. Recurrence plot. Dim=2. Time used: 2.294 sec.

---

*Input*

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

---

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

Dim=6. Time used: 0.917 sec.

---

*Input*

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

---

Dim=8. Time used: 1.187 sec.

---

*Input*

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

---

Dim=12. Time used: 3.225 sec.

---

*Input*

```
hrvRRtakens16 <- local.buildTakens(
  time.series=hrv.data$Beat$RR[1:nignal],
  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: 1.109 sec.

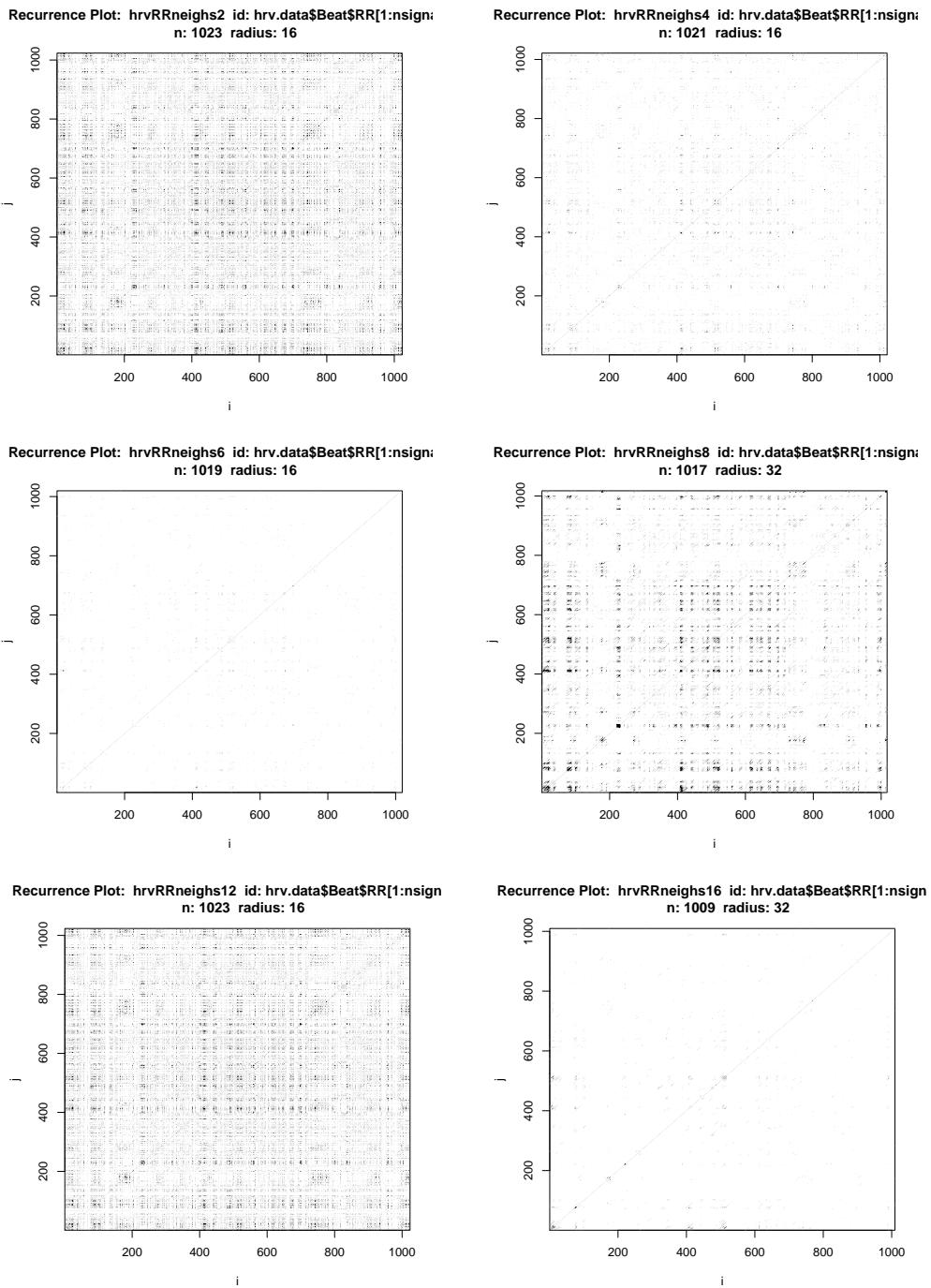


FIGURE 71. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 1.11 sec.

**ToDo:** This is an experimental proposal

7.1. **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.

```
# source('/users/gs/projects/rforge/rhrv/pkg/R/BuildNIHR2.R', chdir = TRUE) _____ Input _____
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 72,

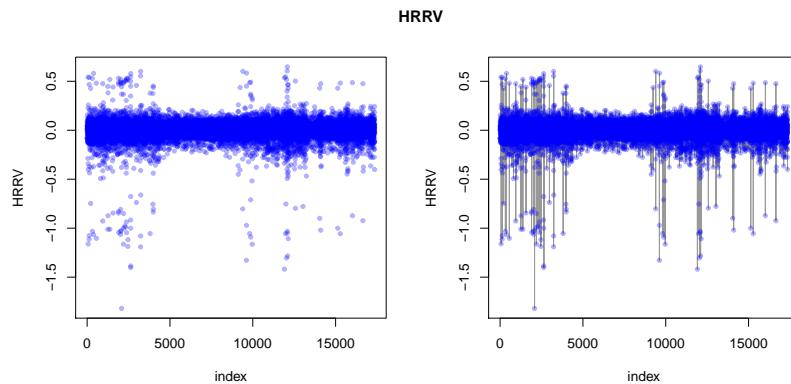


FIGURE 72. RHRV tutorial example.beats. HRRV Signal and linear interpolation.

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 73 on the next page

---

*Input*

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

See Figure 74 on page 69

**ToDo:** findAllNeighbours does not handle NAs

---

*Input*

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

Time used: 0.224 sec.

---

*Input*

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

**ToDo:** check. There seem to be strange artefacts.

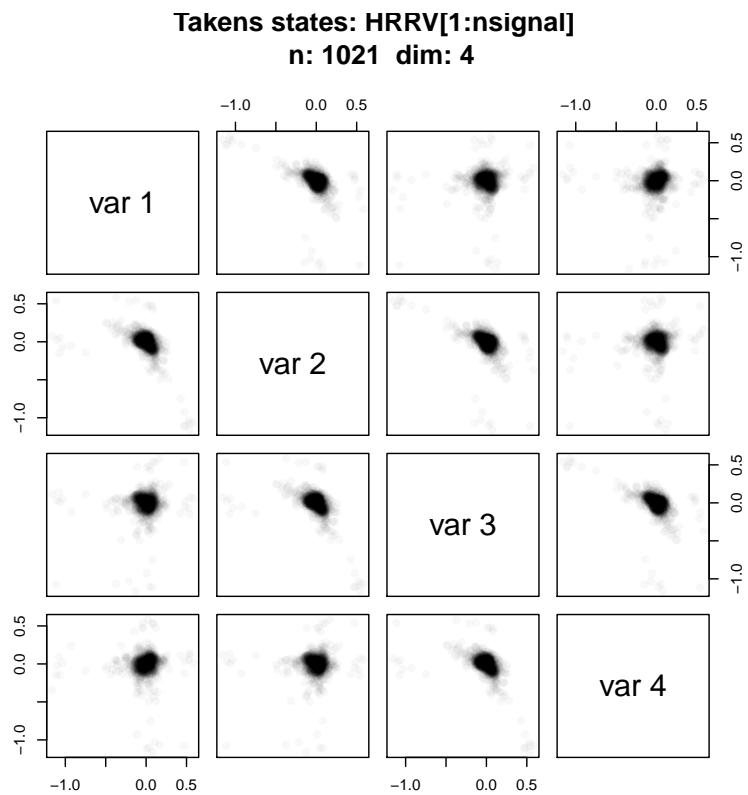


FIGURE 73. RHRV tutorial example.beats. HRRV Time used: 0.662 sec.

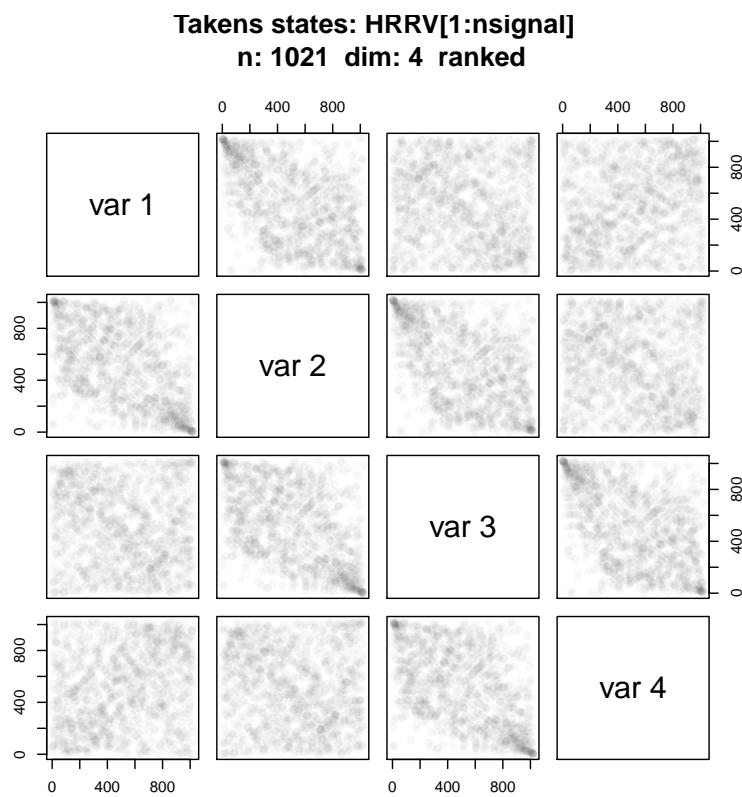


FIGURE 74. RHRV tutorial example.beats. Ranked HRRV data. Time used: 1.341 sec.

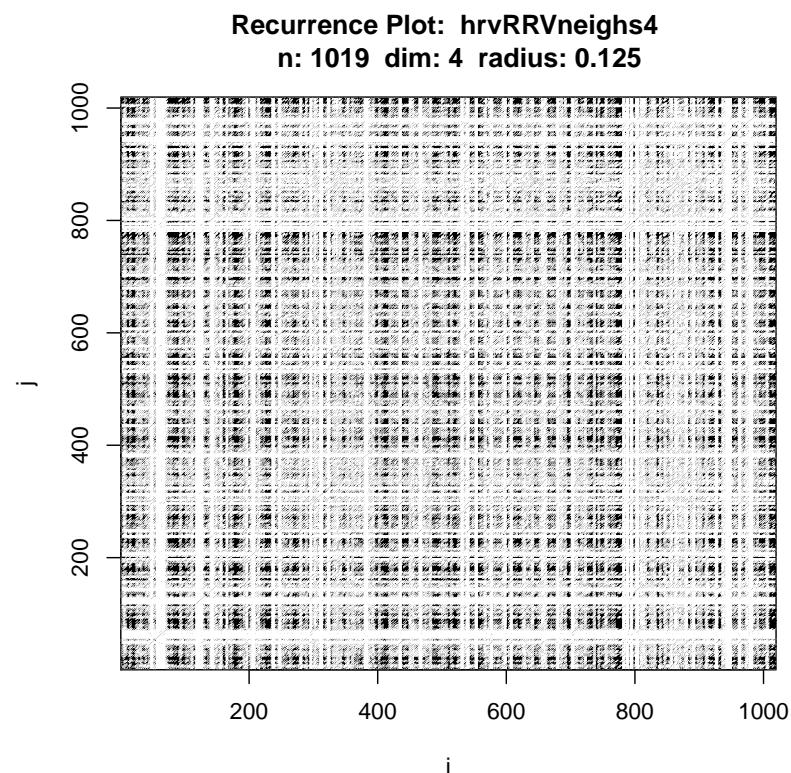


FIGURE 75. Recurrence Plot. Example case: RHRV tutorial example.beats. HRRV Dim=4. Time used: 2.506 sec.

### 7.1.1. RHRV: example.beats - RR Variation: Comparison by Dimension.

---

*Input*

```
hrvRRVtakens2 <- local.buildTakens( time.series=HRRV[1:nSignal],
    embedding.dim=2, time.lag=1)
hrvRRVneighs2 <- local.findAllNeighbours(hrvRRVtakens2[-(1:2),],
    radius=0.125)
save(hrvRRVneighs2, file="hrvRRVneighs2.Rdata")
# load(file="hrvRRVneighs2.RData")
local.recurrencePlotAux(hrvRRVneighs2, dim=2, radius=0.125)
showrqa(hrvRRVtakens2[-(1:2),], radius=0.125, do.hist=FALSE)
```

---

*Output*

---

```
hrvRRVtakens2[-(1:2), ] n: 1021 Dim: 2
Radius: 0.125 Recurrence coverage REC: 0.515 log(REC)/log(R): 0.319
Determinism: 0.939 Laminarity: 0.81
DIV: 0.027
Trend: 0 Entropy: 2.346
Diagonal lines max: 37 Mean: 5.373 Mean off main: 5.362
Vertical lines max: 48 Mean: 3.998
```

---

Dim=2. Time used: 4.824 sec.

---

*Input*

```
hrvRRVtakens6 <- local.buildTakens( time.series=HRRV[1:nSignal],
    embedding.dim=6, time.lag=1)
hrvRRVneighs6 <- local.findAllNeighbours(hrvRRVtakens6[-(1:2),], radius=0.125)
save(hrvRRVneighs6, file="hrvRRVneighs6.Rdata")
# load(file="hrvRRVneighs6.RData")
local.recurrencePlotAux(hrvRRVneighs6, dim=6, radius=0.125)
showrqa(hrvRRVtakens6[-(1:2),], radius=0.125, do.hist=FALSE)
```

---

*Output*

---

```
hrvRRVtakens6[-(1:2), ] n: 1017 Dim: 6
Radius: 0.125 Recurrence coverage REC: 0.179 log(REC)/log(R): 0.827
Determinism: 0.943 Laminarity: 0.451
DIV: 0.03
Trend: 0 Entropy: 2.305
Diagonal lines max: 33 Mean: 5.243 Mean off main: 5.213
Vertical lines max: 22 Mean: 2.887
```

---

Dim=6. Time used: 2.648 sec.

---

*Input*

```
hrvRRVtakens8 <- local.buildTakens( time.series=HRRV[1:nSignal],
    embedding.dim=8, time.lag=1)
hrvRRVneighs8 <- local.findAllNeighbours(hrvRRVtakens8[-(1:2),], radius=0.125)
save(hrvRRVneighs8, file="hrvRRVneighs8.Rdata")
# load(file="hrvRRVneighs8.RData")
local.recurrencePlotAux(hrvRRVneighs8, dim=8, radius=0.125)
showrqa(hrvRRVtakens8[-(1:2),], radius=0.125, do.hist=FALSE)
```

---

We should expect the breathing rhythm, so a time lag in the order of 10 is to be expected. **To Do:** fix default setting for radius. Eckmann uses nearest neighbours with NN=10

```

hrvRRVtakens8[-(1:2), ] n: 1015 Dim: 8
Radius: 0.125 Recurrence coverage REC: 0.105 log(REC)/log(R): 1.084
Determinism: 0.943 Laminarity: 0.337
DIV: 0.032
Trend: 0 Entropy: 2.329
Diagonal lines max: 31 Mean: 5.361 Mean off main: 5.308
Vertical lines max: 17 Mean: 2.715

```

Dim=8. Time used: 1.949 sec.

---

*Input*

---

```

hrvRRVtakens12 <-
  local.buildTakens( time.series=HRRV[1:nseries],
                     embedding.dim=12, time.lag=1)
hrvRRVneighs12 <-
  local.findAllNeighbours(hrvRRVtakens12[-(1:2), ], radius=3/16)
save(hrvRRVneighs12, file="hrvRRVneighs12.Rdata")
# load(file="hrvRRVneighs12.RData")
local.recurrencePlotAux(hrvRRVneighs12, dim=12, radius=3/16)
showrqa(hrvRRVtakens12[-(1:2), ], radius=3/16, do.hist=FALSE)

```

---

*Output*

---

```

hrvRRVtakens12[-(1:2), ] n: 1011 Dim: 12
Radius: 0.1875 Recurrence coverage REC: 0.235 log(REC)/log(R): 0.865
Determinism: 0.988 Laminarity: 0.635
DIV: 0.015
Trend: 0 Entropy: 3.075
Diagonal lines max: 68 Mean: 9.775 Mean off main: 9.733
Vertical lines max: 52 Mean: 3.656

```

Dim=12: Time used: 3.079 sec.

---

*Input*

---

```

hrvRRVtakens16 <- local.buildTakens( time.series=HRRV[1:nseries],
                                         embedding.dim=16, time.lag=1)
hrvRRVneighs16 <- local.findAllNeighbours(hrvRRVtakens16[-(1:2), ], radius=3/16)
save(hrvRRVneighs16, file="hrvRRVneighs16.Rdata")
# load(file="hrvRRVneighs16.RData")
local.recurrencePlotAux(hrvRRVneighs16, dim=16, radius=3/16)
showrqa(hrvRRVtakens16[-(1:2), ], radius=3/16, do.hist=FALSE)

```

---

*Output*

---

```

hrvRRVtakens16[-(1:2), ] n: 1007 Dim: 16
Radius: 0.1875 Recurrence coverage REC: 0.147 log(REC)/log(R): 1.144
Determinism: 0.99 Laminarity: 0.527
DIV: 0.016
Trend: 0 Entropy: 3.163
Diagonal lines max: 64 Mean: 10.594 Mean off main: 10.523
Vertical lines max: 40 Mean: 3.114

```

Dim=16. Time used: 2.436 sec.

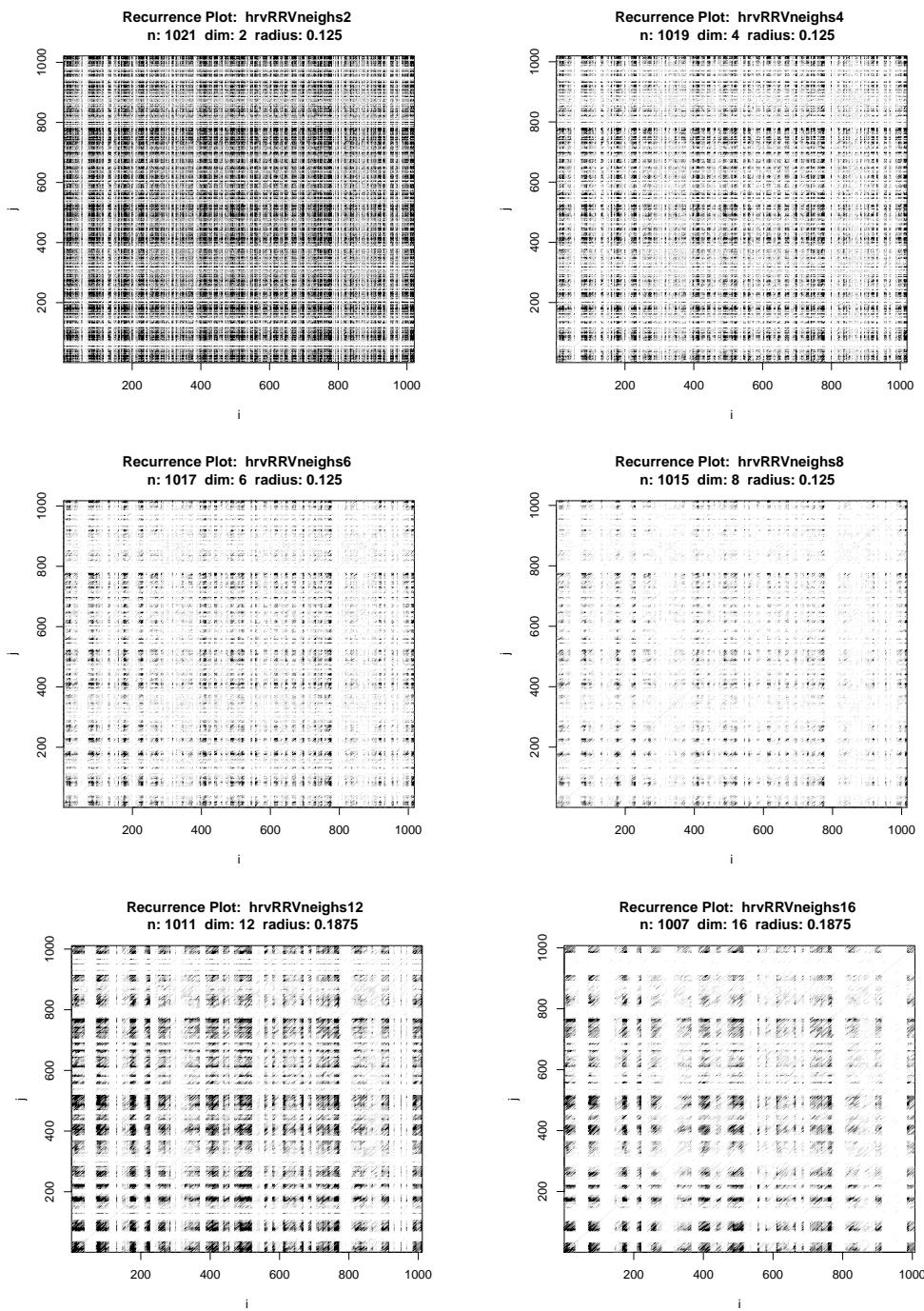


FIGURE 76. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 2.436 sec.

## 8. 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")
```

---

*Output*

```
** Loading beats positions for record: example2.beats **
Path: /users/gs/projects/rforge/rhrv/tutorial/beatsFolder
Scale: 1
Removed 2437 duplicated beats
Date: 01/01/1900
Time: 00:00:00
Number of beats: 2437
```

---

*Input*

```
#     RecordPath = "beatsFolder")
```

---

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

---

*Output*

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

---

*Input*

---

*Input*

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

See Figure 77 on the next page.

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

---

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

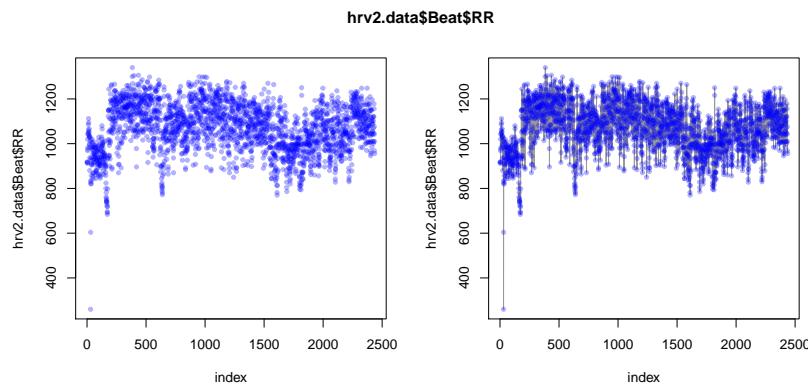


FIGURE 77. RHRV tutorial example2.beats. Signal and linear interpolation.

See Figure 78.

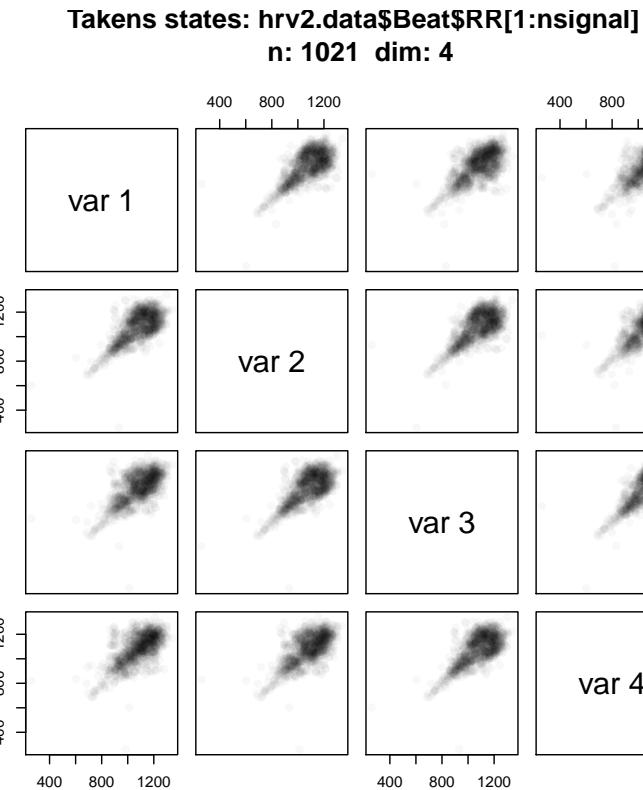


FIGURE 78. RHRV tutorial example2.beats. Time used: 0.661 sec.

---

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

---

See Figure 79 on the next page.

---

*Input*

---

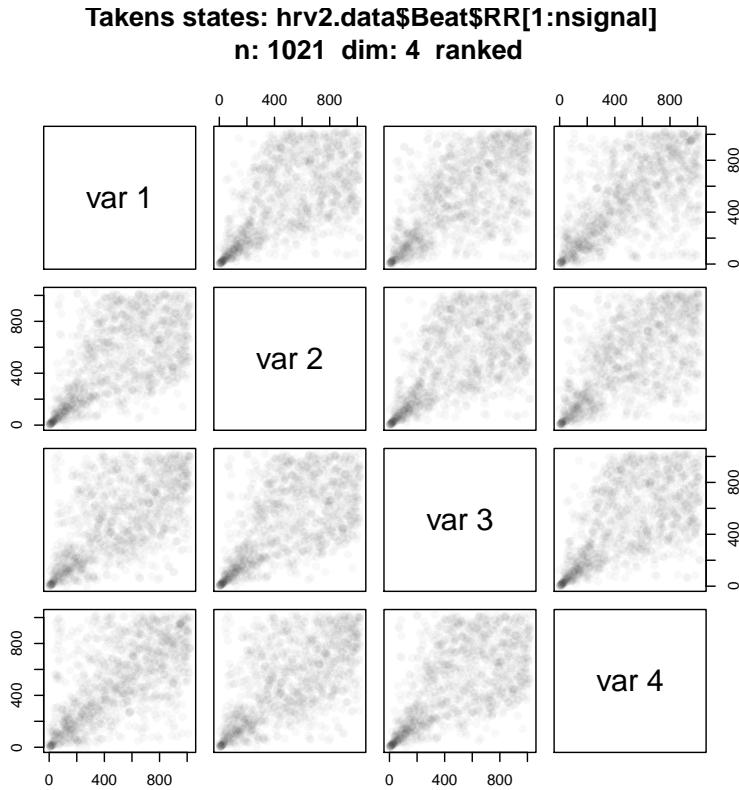


FIGURE 79. RHRV tutorial example2.beats. Ranked data. Time used: 1.323 sec.

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

See Figure 80 on the facing 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=FALSE)
```

---



---

*Output*

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

---

Dim=4. Time used: 4.326 sec.

See Figure 81 on page 78.

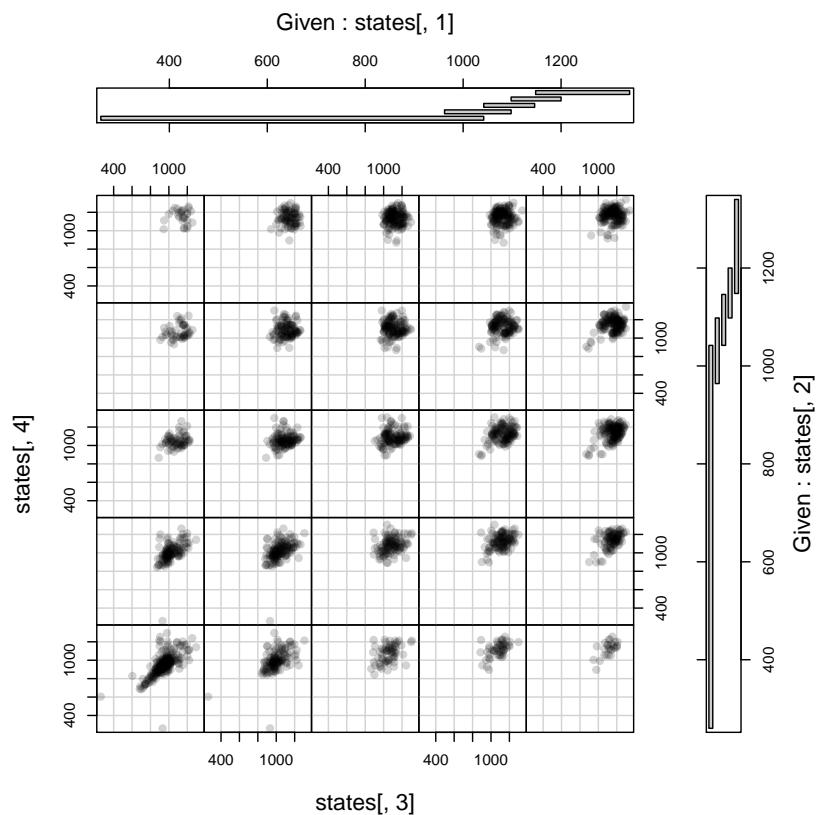


FIGURE 80. State coplot. RHRV tutorial example2.beats. Time used:  
0.263 sec.

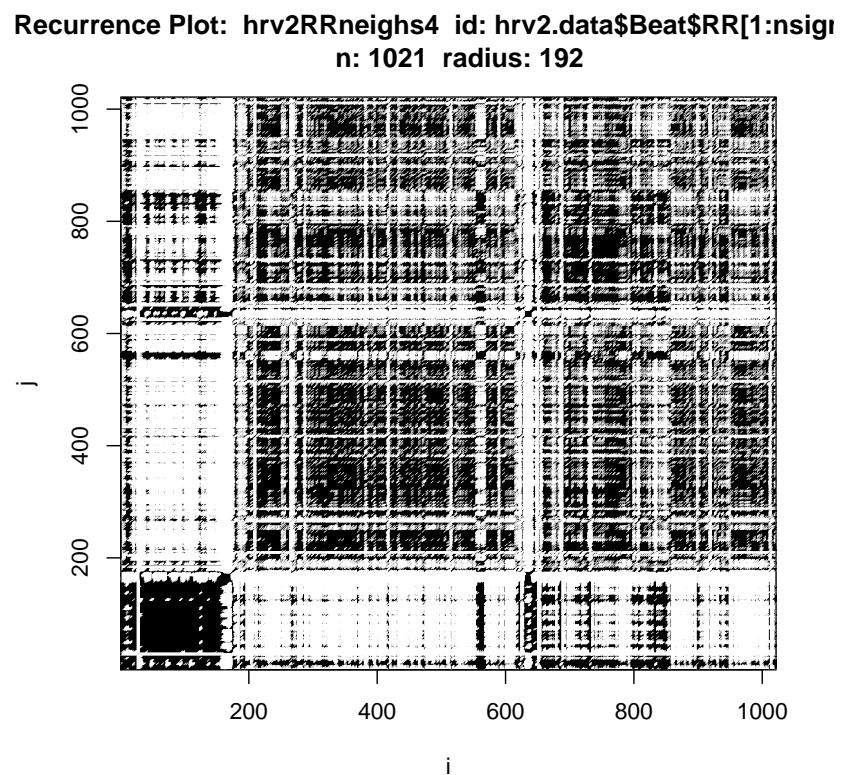


FIGURE 81. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=4. Time used: 4.326 sec.

### 8.0.2. RHRV: example2.beats, RR-intervals. Comparison by Dimension.

---

*Input*

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

---

*Output*

---

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

---

Dim=2. Time used: 4.425 sec.

See Figure 82 on page 81.

---

*Input*

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

---

*Output*

---

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

---

Dim=6. Time used: 4.519 sec.

See Figure 82 on page 81.

---

*Input*

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

---

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

---

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

---

Dim=8. Time used: 4.834 sec.

See Figure 82 on the facing page.

---

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

---



---

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

---

Dim=12. Time used: 4.392 sec.

---

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

---



---

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

---

Dim=16. Time used: 4.784 sec.

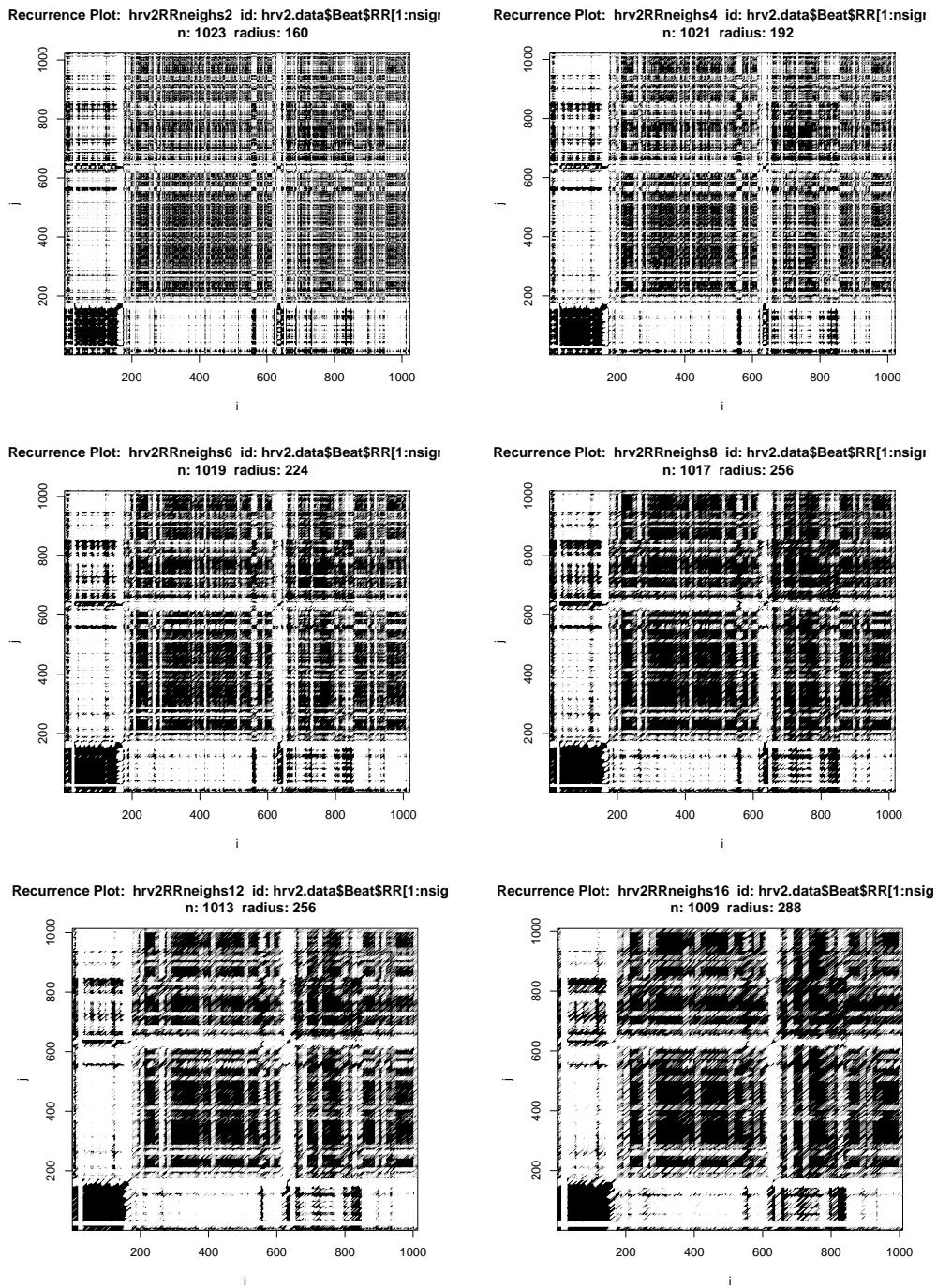


FIGURE 82. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 4.784 sec.

**ToDo:** Consider using differences  
differences for HRV

8.1. **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:

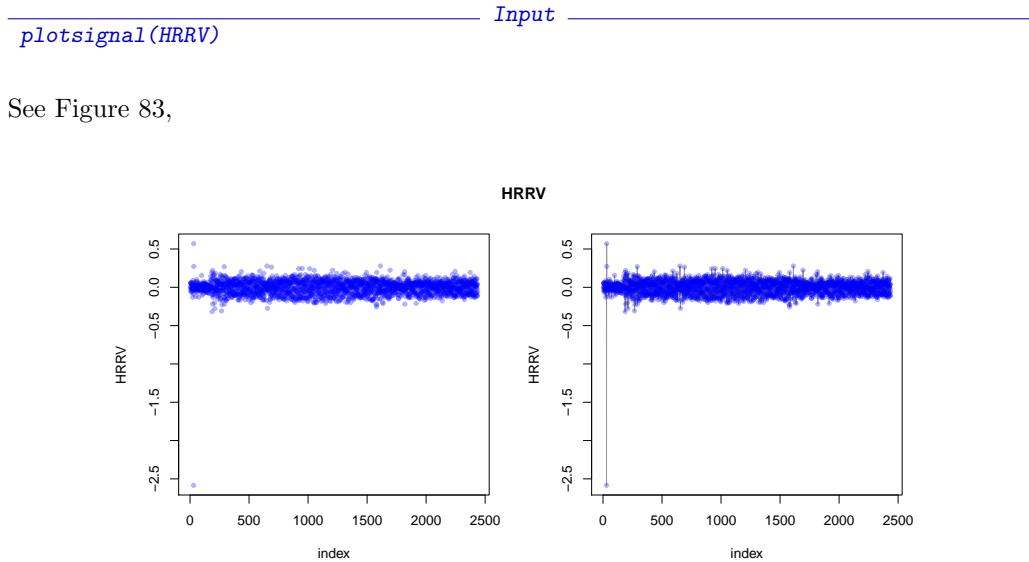


FIGURE 83. RHRV tutorial example2.beats. HRRV Signal and linear interpolation.

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 84 on the facing page.

---

*Input*

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

---

*Output*

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

---

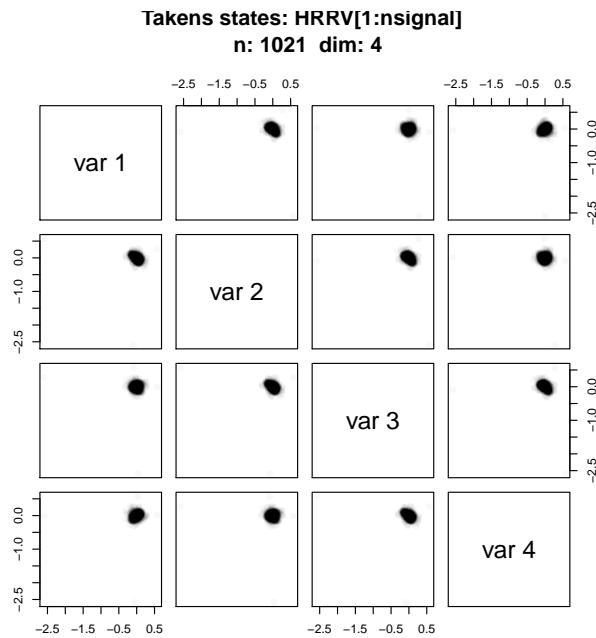


FIGURE 84. Recurrence plot. RHRV tutorial example2.beats. HRRV  
Time used: 0.666 sec.

See Figure 85 on the next page.

---

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

---

See Figure 86 on page 85

---

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

---

See Figure 87 on page 85.

---

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

---



---

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

---

See Figure 88 on page 86.

**ToDo:** `findAllNeighbours` does not handle NAs

---

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

---

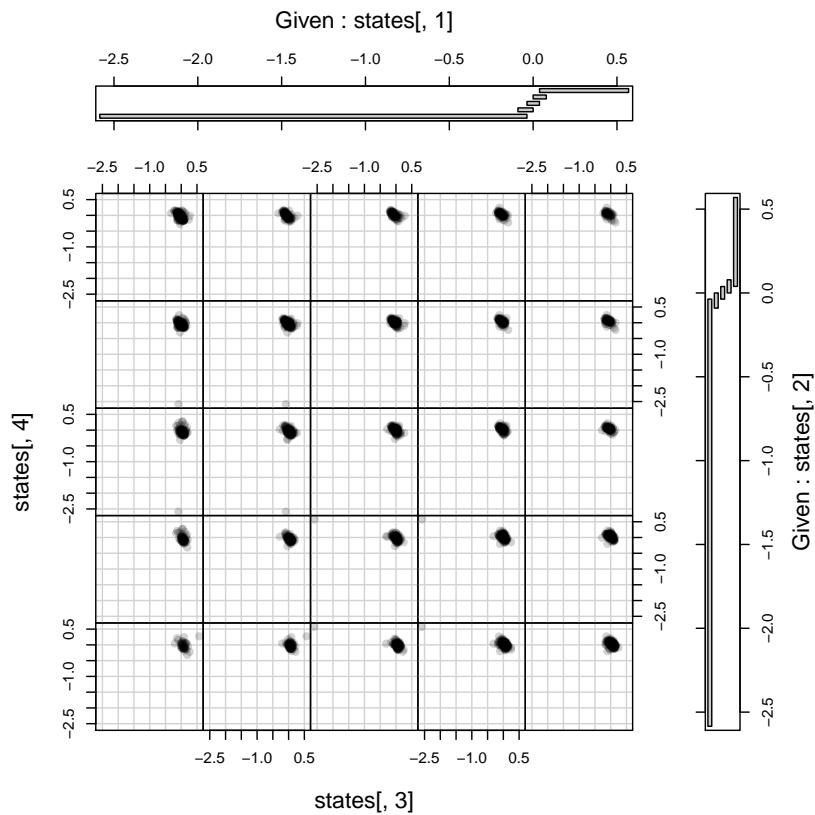


FIGURE 85. State coplot. RHRV tutorial example2.beats. HRRV. Time used: 0.953 sec.

Time used: 0.236 sec.

---

*Input*

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

**To Do:** check. There seem to be strange artefacts.

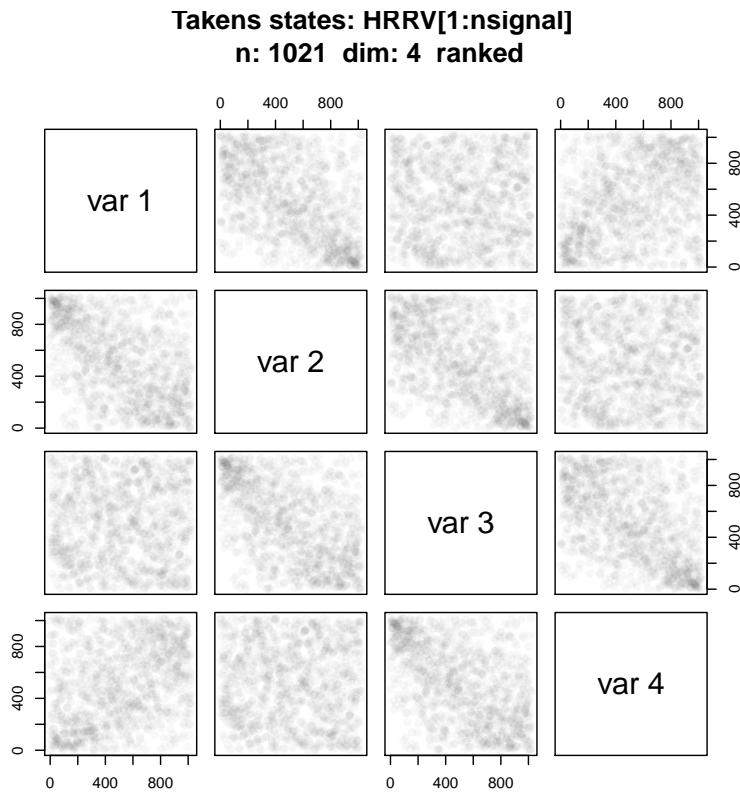


FIGURE 86. RHRV tutorial example2.beats. Ranked HRRV data. Time used: 1.603 sec.

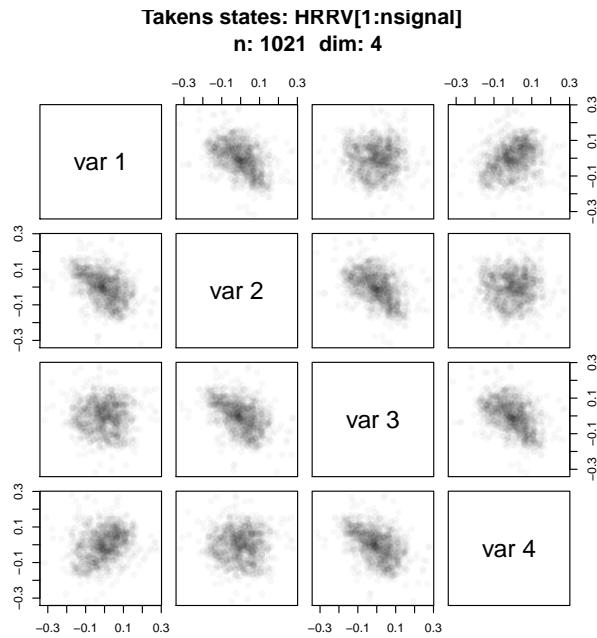


FIGURE 87. Recurrence plot. RHRV tutorial example2.beats. HRRV Time used: 2.299 sec.

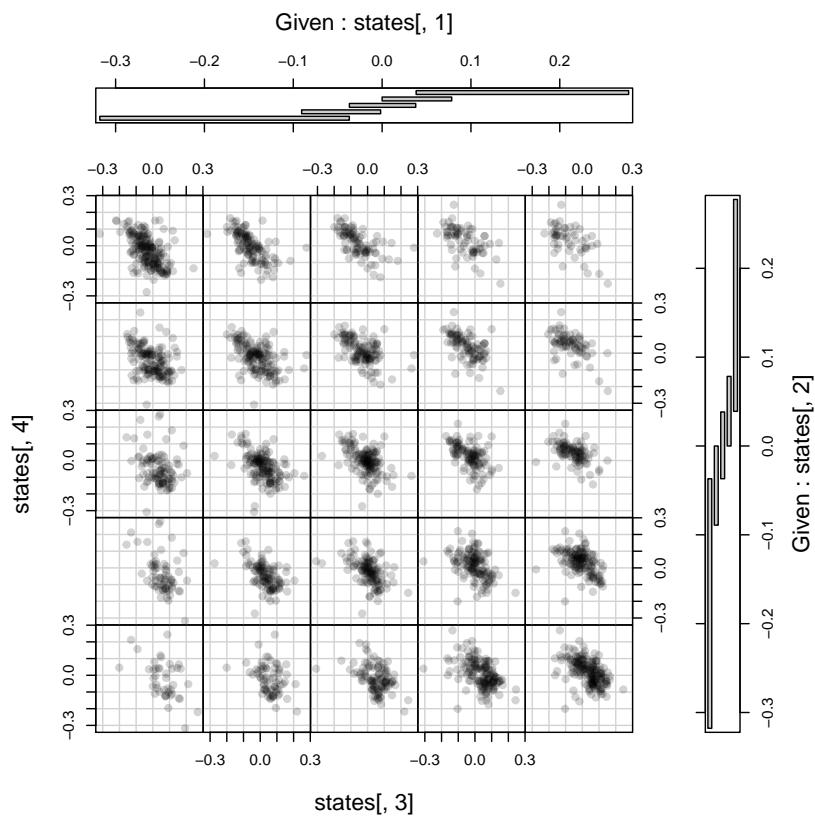


FIGURE 88. State coplot. RHRV tutorial example2.beats. HRRV. Time used: 2.603 sec.

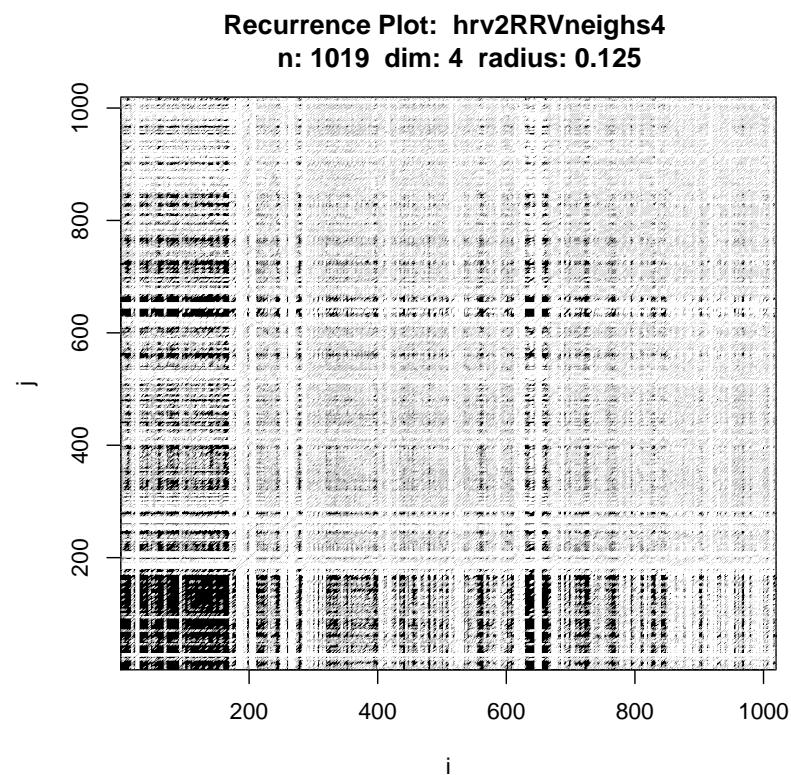


FIGURE 89. Recurrence Plot. Example case: RHRV tutorial example2.beats. HRRV Dim=4. Time used: 2.233 sec.

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

### 8.1.1. RHRV: example2.beats - RR Variation: Comparison by Dimension.

---

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

---



---

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

---

Time used: 4.383 sec.

---

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

---



---

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

---

Dim=6. Time used: 4.568 sec.

---

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

---



---

*Output*

---

```

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

```

Dim=8. Time used: 3.899 sec.

---

*Input*

```

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

```

---

*Output*

```

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

```

Dim=12. Time used: 4.322 sec.

---

*Input*

```

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

```

---

*Output*

```

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

```

Dim=16. Time used: 4.892 sec.

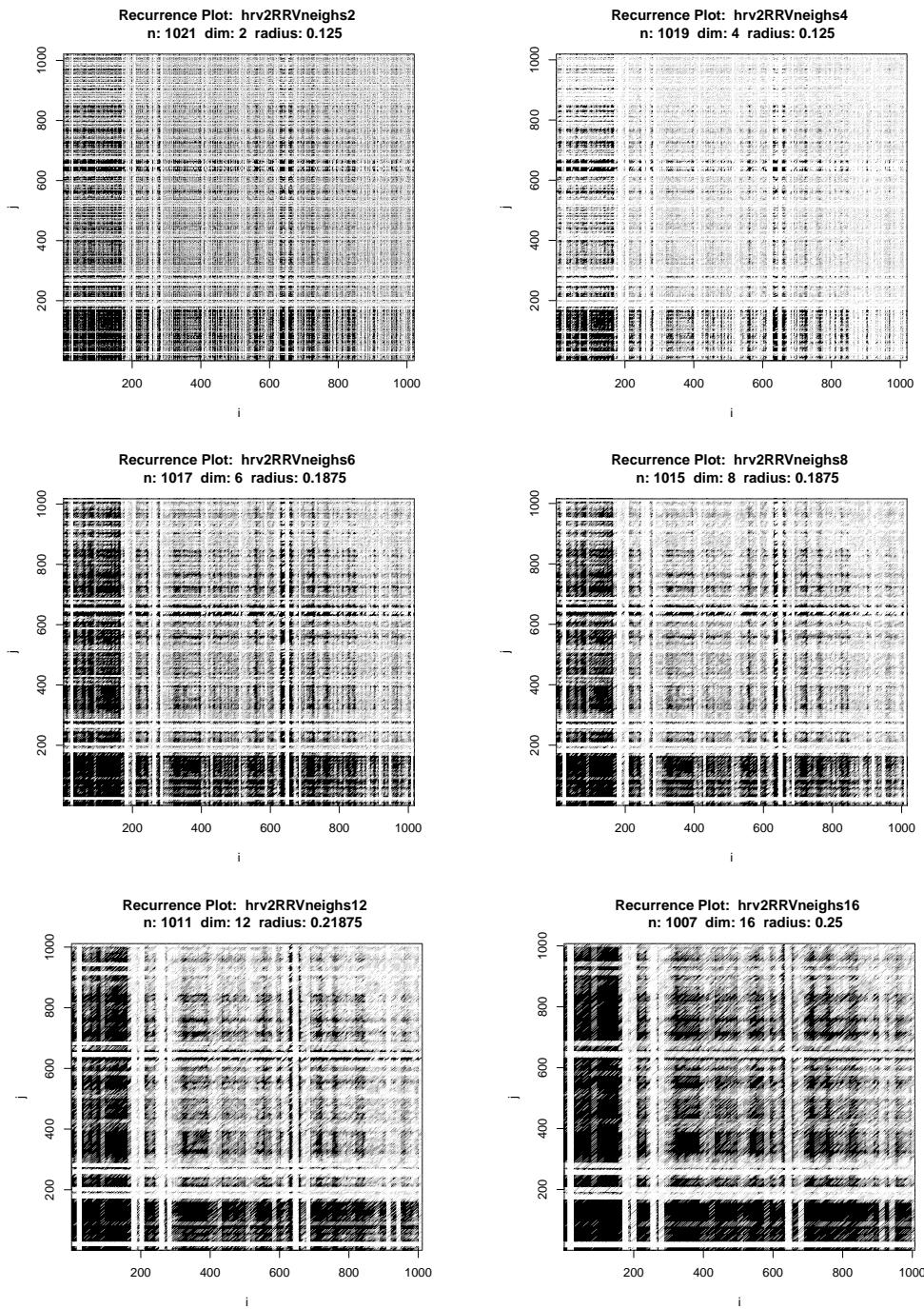


FIGURE 90. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 4.893 sec.

## REFERENCES

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

## INDEX

### ToDo

- 1: add support for higher dimensional signals, 2
- 1: consider dimension-adjusted radius, 3
- 1: support distance instead of 0/1 indicators, 3
- 1: the takTakensens state plot may be critically affected by outliers. Find a good rescaling., 3
- 2: colour by time, 5
- 2: extend for low dimensions, extend parameters, 5
- 2: improve choice of alpha, 4
- 2: improve feedback for data structures in *nonlinearTseries*, 8
- 2: improve to a full *show* method for class *rqa*., 8
- 2: propagate parameters from *buildTakens* and *findAllNeighbours* in a slot of the result, instead of using explicit parameters in *recurrencePlotAux*., 7
- 6: Geyser: extended to two-dimensional data in *geyserlin*. Experimental only. Check., 36
- 6: double check: *MASS:::geyser* should be used, not *faithful*, 36
- 7: This is an experimental proposal, 66
- 7: We have outliers at approximately 2\*RR. Could this be an artefact of preprocessing, filtering out too many impulses?, 58
- 7: check. There seem to be strange artefacts., 67
- 7: *findAllNeighbours* does not handle NAs, 67
- 7: fix default setting for radius. Eckmann uses nearest neighbours with NN=10, 71
- 8: Consider using differences, 82
- 8: We have outliers at approximately 2\*RR. Could this be an artefact of preprocessing, filtering out too many impulses?, 74
- 8: check. There seem to be strange artefacts., 84
- 8: *findAllNeighbours* does not handle NAs, 83
- 8: fix default setting for radius. Eckmann uses nearest neighbours with NN=10, 88

delay embedding, 2

Geyser, 36

heart rate, 58, 74

heart rate variation, 67, 82

hrv, 58, 74

recurrence plot, 2

recurrence quantification analysis, 3

RQA, 3

takens plot

hrv2, 75

Takens state, 1, 2

R session info:

Total Sweave time used: 132.757 sec. at Sat Aug 23 17:31:49 2014.

- R version 3.1.1 (2014-07-10), x86\_64-apple-darwin13.1.0
- Locale: en\_GB.UTF-8/en\_GB.UTF-8/en\_GB.UTF-8/C/en\_GB.UTF-8/en\_GB.UTF-8
- Base packages: base, datasets, graphics, grDevices, methods, stats, tcltk, utils
- Other packages: leaps 2.9, locfit 1.5-9.1, MASS 7.3-34, Matrix 1.1-5, mgcv 1.8-2, nlme 3.1-117, nonlinearTseries 0.2.2, rgl 0.94.1118, RHRV 4.0, sintro 0.1-3, tkrplot 0.0-23, TSA 1.01, tseries 0.10-32, waveslim 1.7.3
- Loaded via a namespace (and not attached): grid 3.1.1, lattice 0.20-29, quadprog 1.5-5, tools 3.1.1, zoo 1.7-12

L<sup>A</sup>T<sub>E</sub>X 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: 163 $
$Date: 2014-08-21 18:10:31 +0200 (Thu, 21 Aug 2014) $
$Name: 
$Author: gsawitzki $
```

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

GÜNTHER SAWITZKI  
 STATLAB HEIDELBERG  
 IM NEUENHEIMER FELD 294  
 D 69120 HEIDELBERG