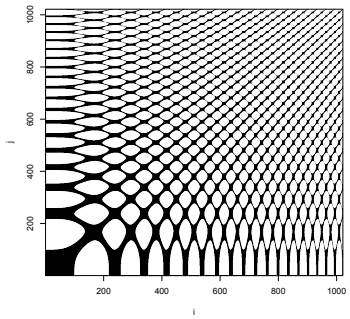
STATISTICAL DATA ANALYSIS: RECURRENCE PLOT

GÜNTHER SAWITZKI



Contents

1. Background

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: June 5, 2015 from svn/cvs *Revision* : 169
gs@statlab.uni-heidelberg.de .

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

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

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

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

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

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

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

This is Takens' delay embedding state (re)construction **?**.

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

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

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

ToDo: add support for higher dimensional signals

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

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

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

Takens states are vectors in M dimensions. There are standard statistical techniques to visiualize aspects of M dimensional vectors, at least for not too high dimension. One is the draftsman's plot, a scatterplot matrix by marginals. For Takens states, this is implemented as `statepairs()`. The other is coplots, a variant of scatterplot matrix by marginals, conditionend by one or two additional variables. For Takens states, this is implemented as `statecoplots()`

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

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

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

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

ToDo: consider dimension-adjusted radius

1.2. **Recurrence Plots.** The next step, taken in **?** 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 **?** actually used a nearest neighbourhood environment to cover about 10 data points.

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

ToDo: support distance instead of 0/1 indicators

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

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

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

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

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

Input

TABLE 1. Recurrence Quantification Analysis (RQA)

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

```
laptime <- 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 colums is the imbedding dimension. If present, the *time.lag* attribute is the lag parameter, *id* an identification string for the basic data set.

ToDo: improve choice of alpha

Input

```
alpha=0.5
```

`statepairs` *Show marginal scatterplots of Takens states*

Usage.

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

Arguments.

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

ToDo: colour by time

Input

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

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

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

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

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

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

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

ToDo: extend for low dimensions, extend parameters

`statecoplot` *Show conditioning marginal scatterplots of Takens states*

Usage.

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

Arguments.

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

Input

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

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


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

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

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

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

Input

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

```
        return(takens)
}
```

Input

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

minor cosmetics added to recurrencePlotAux

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

Input

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

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

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

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

        more <- NULL

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

        #use components of neights if available
        embedding.dim <- attr(neighs, "embedding.dim")
        if (is.null(embedding.dim)) embedding.dim <- dim
        if (!is.null(dim)) {
                if (embedding.dim != dim) warning(paste("Embedding dim:", embedding.dim,
```

```
                " does not match dim argument=", dim))
                more <- paste(more," dim:",dim)
        }


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

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

 }
```

ToDo: improve feedback for data structures in *nonlinearTseries*

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

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

Needs improvement.

Input

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

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

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

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

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

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

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

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

3. Test Signals

We set up a small series of test signals. Some synthetic test signals are introduced here. More test cases used later are the Geyser data set (Section 7 on page 54) and two examples of heart rate data sets (Section 8 on page 76 and **??** on page ??) 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.

Input

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

For signal representation, we use a common layout.

Input

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

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

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

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

3.1. **Sinus.**

Input

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

See Figure 1 on the facing page.

3.2. **Uniform Random Numbers.**

Input

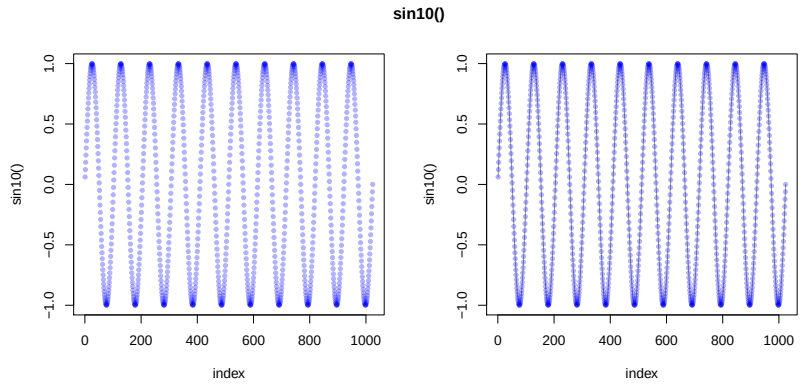


FIGURE 1. Test case: sin10. Signal and linear interpolation.

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

See Figure 2,

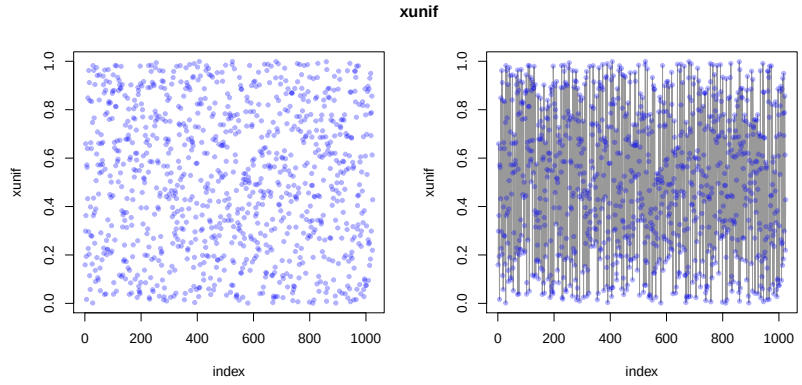


FIGURE 2. Test case: unif - uniform random numbers. Signal and linear interpolation.

3.3. **Chirp Signal.**

Input

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

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

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

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

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

See Figure 3,

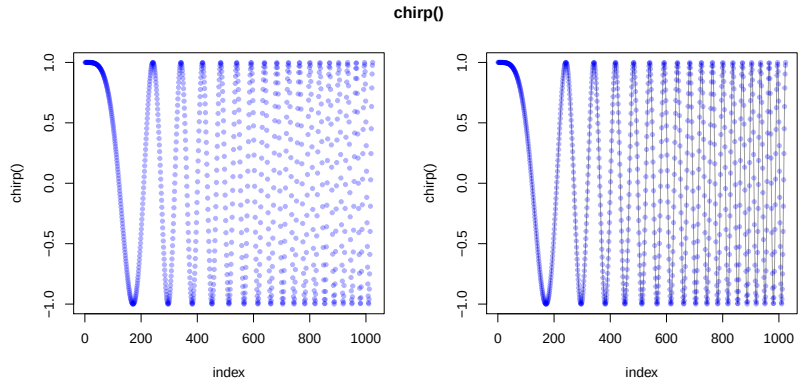


FIGURE 3. Test case: chirp signal. Signal and linear interpolation.

3.4. **Doppler signal.**

Input

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

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

See Figure 4 on the next page,

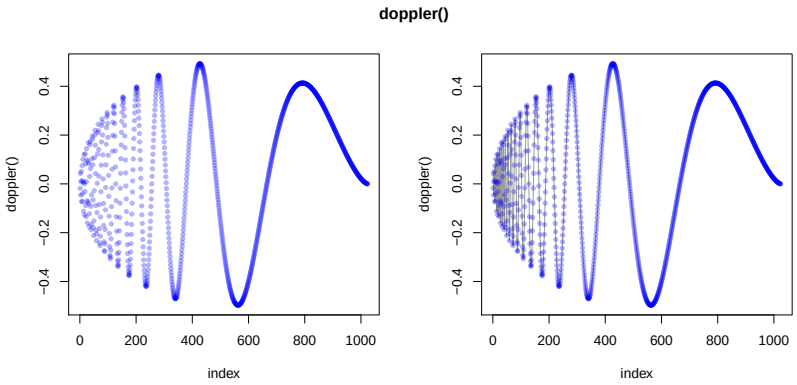


FIGURE 4. Test case: Doppler signal. Signal and linear interpolation.

4. nonlinearTseries Quick-Start

Input

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

4.1. **Sinus.**

Input

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

Output

```
tau.acf: 20
```

Input

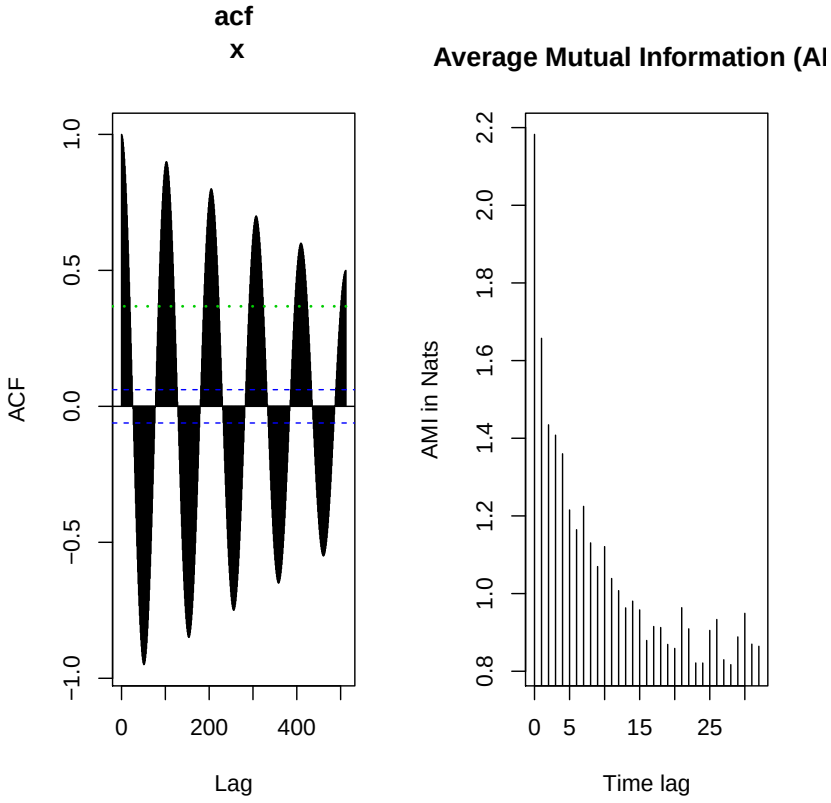
```
# taudelay estimation based on the mutual information function
tau.ami=NA
try(tau.ami <- timeLag(sinN, technique = "ami", do.plot = T,
main=paste("ami\n",deparse(substitute(x)))))
cat("tau.ami:",tau.ami)
```

Output

```
tau.ami: NA
```

Input

```
par(oldpar)
```



Note: fails. See Rnw source code.

4.2. **Uniform Random Numbers.**

Input

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

Output

```
tau.acf: 1
```

Input

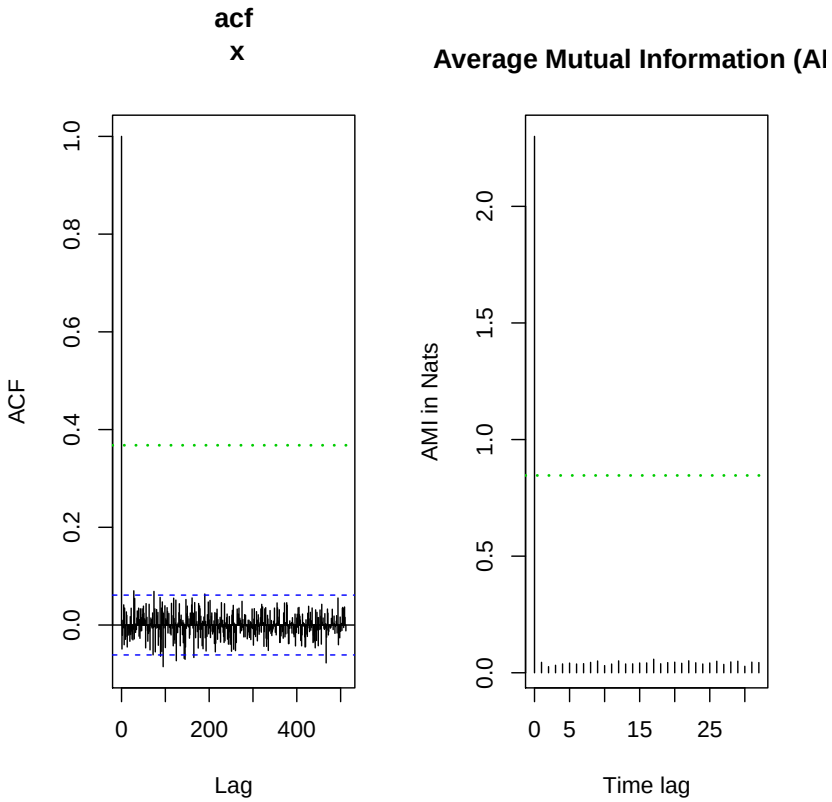
```
# taudelay estimation based on the mutual information function
tau.ami = timeLag(unifN, technique = "ami", do.plot = T,
main=paste("ami\n",deparse(substitute(x))))
cat("tau.ami:",tau.ami)
```

Output

```
tau.ami: 1
```

Input

```
par(oldpar)
```



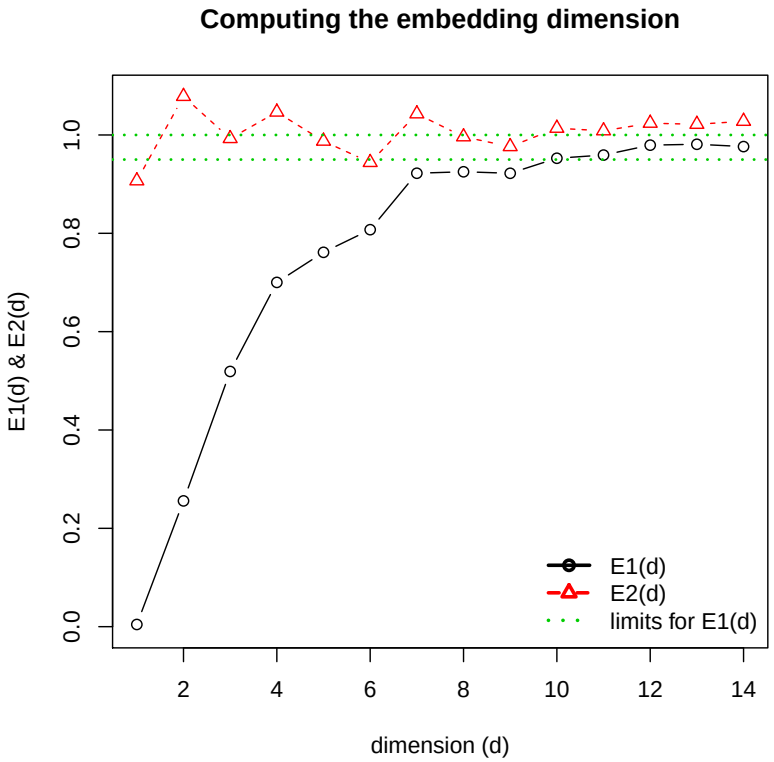
Input

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

Output

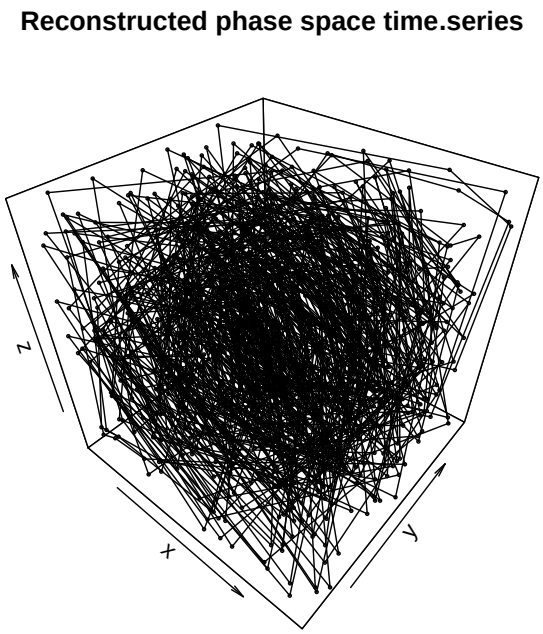
```
Estimated embedding dim: 10
```

Input



Input

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



4.3. **Chirp.**

Input

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

Output

```
tau.acf: 11
```

Input

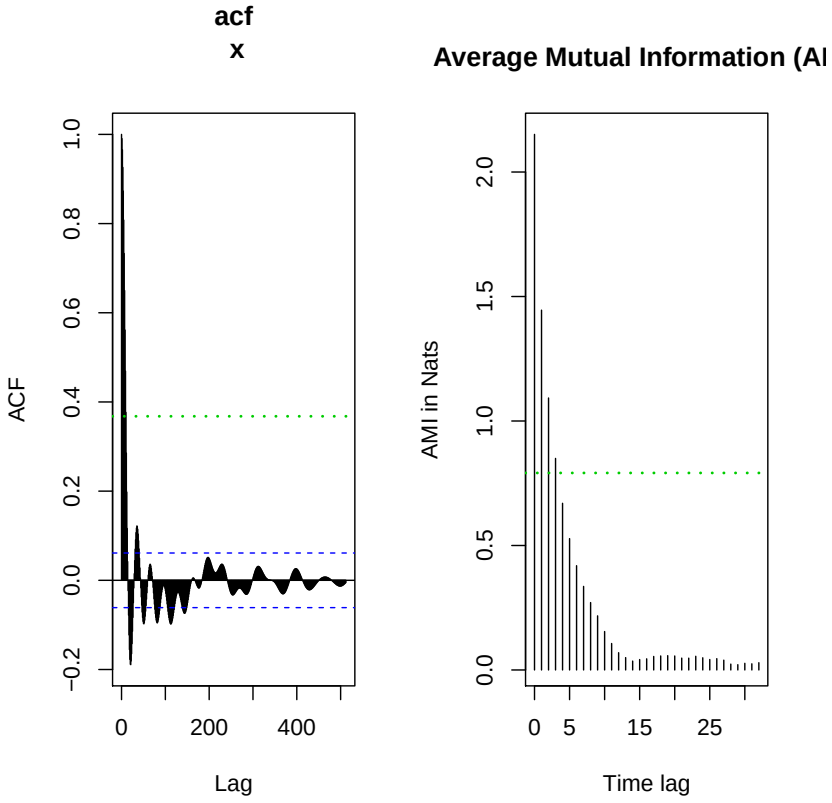
```
# taudelay estimation based on the mutual information function
tau.ami = timeLag(chirpN, technique = "ami", do.plot = T,
main=paste("ami\n",deparse(substitute(x))))
cat("tau.ami:",tau.ami)
```

Output

```
tau.ami: 4
```

Input

```
par(oldpar)
```



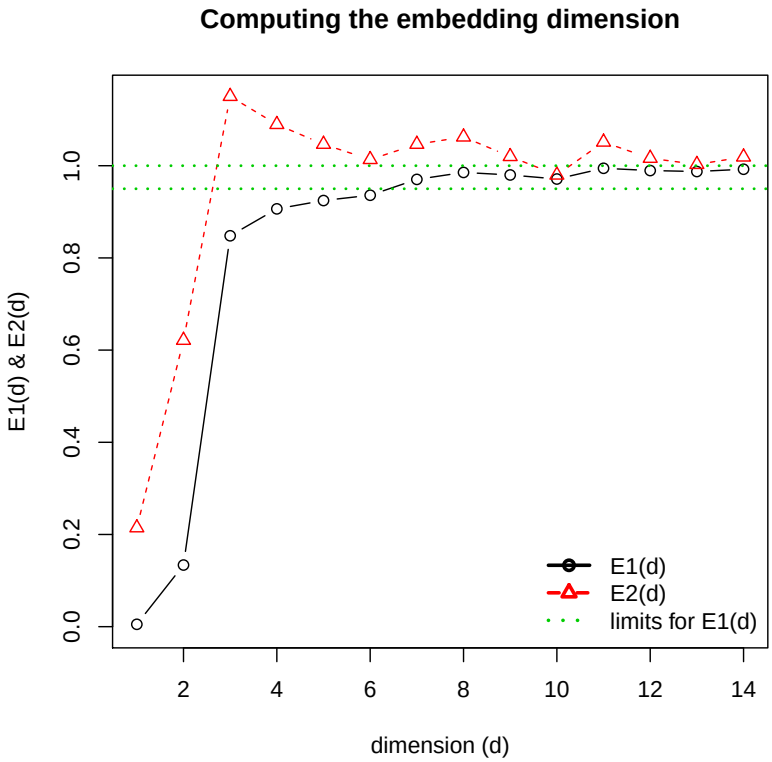
Input

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

Output

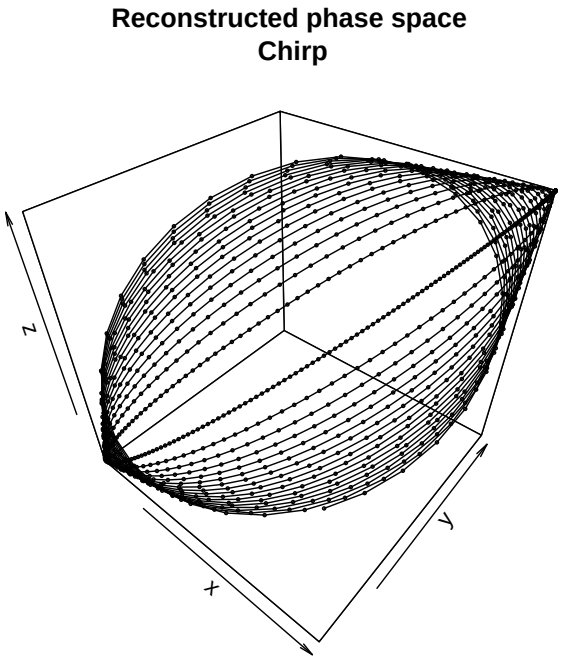
```
Estimated embedding dim: 7
```

Input



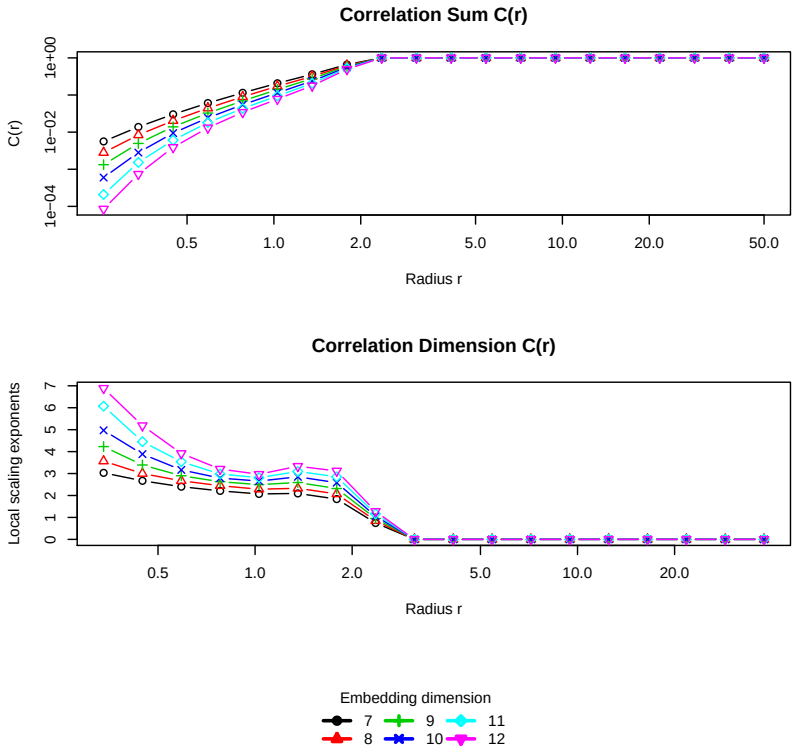
Input

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



Input

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



Input

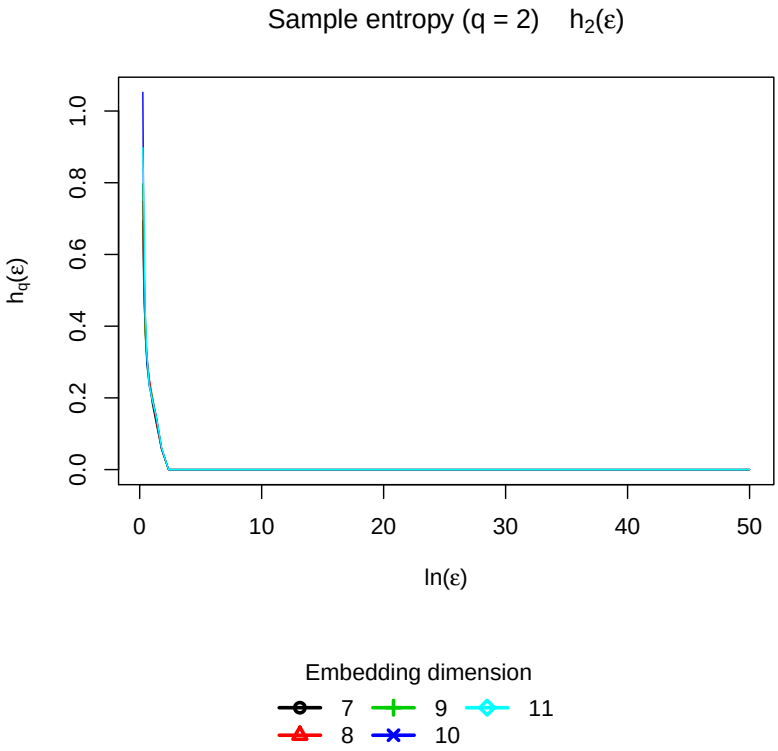
```
oldpar <- par(mfrow=c(1,2))
se = sampleEntropy(cd, do.plot =T)
se.est = estimate(se, do.plot = T,
regression.range = c(8,15))
cat("Sample entropy estimate: ", mean(se.est), "\n")
```

Output

```
Sample entropy estimate:  0
```

Input

```
par(oldpar)
```



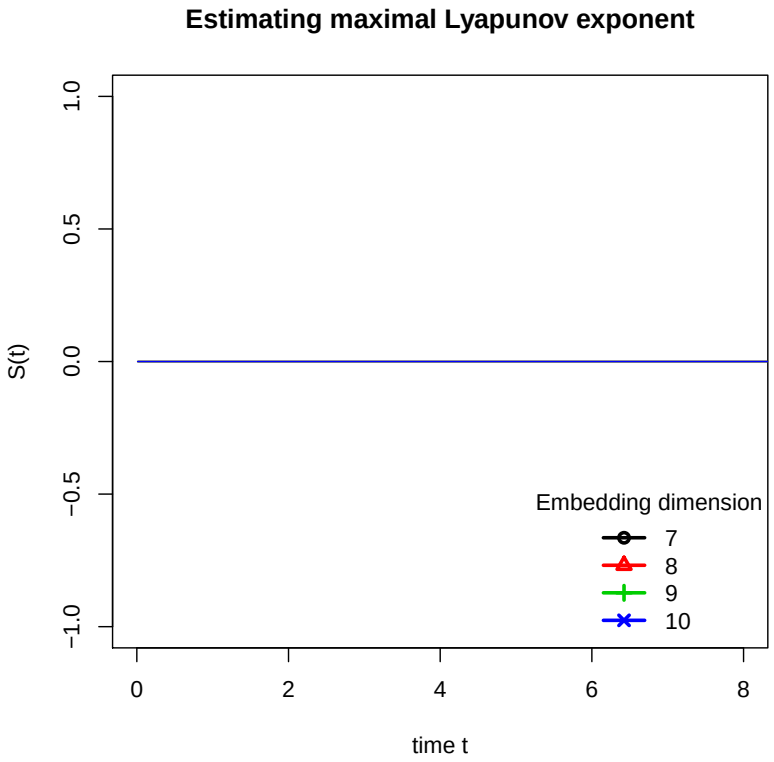
Input

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

Output

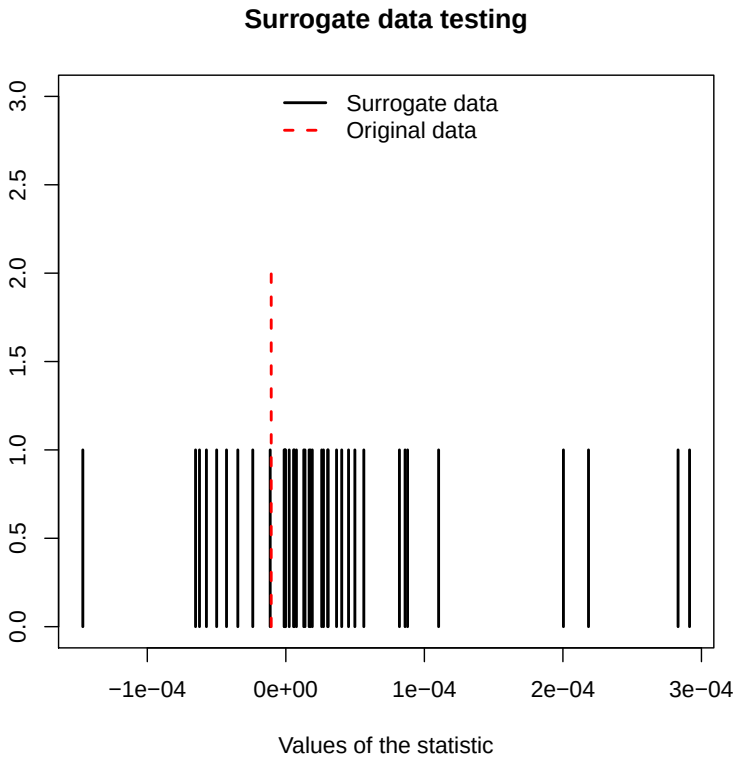
```
estimate: 0
```

Input



Input

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


Surrogate data testing
Surrogate data
Original data
0.0
0.5
1.0
1.5
2.0
2.5
3.0
−1e−04
0e+00
1e−04
2e−04
3e−04
Values of the statistic

4.4. **Doppler.**

Input

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

Output

```
tau.acf: 45
```

Input

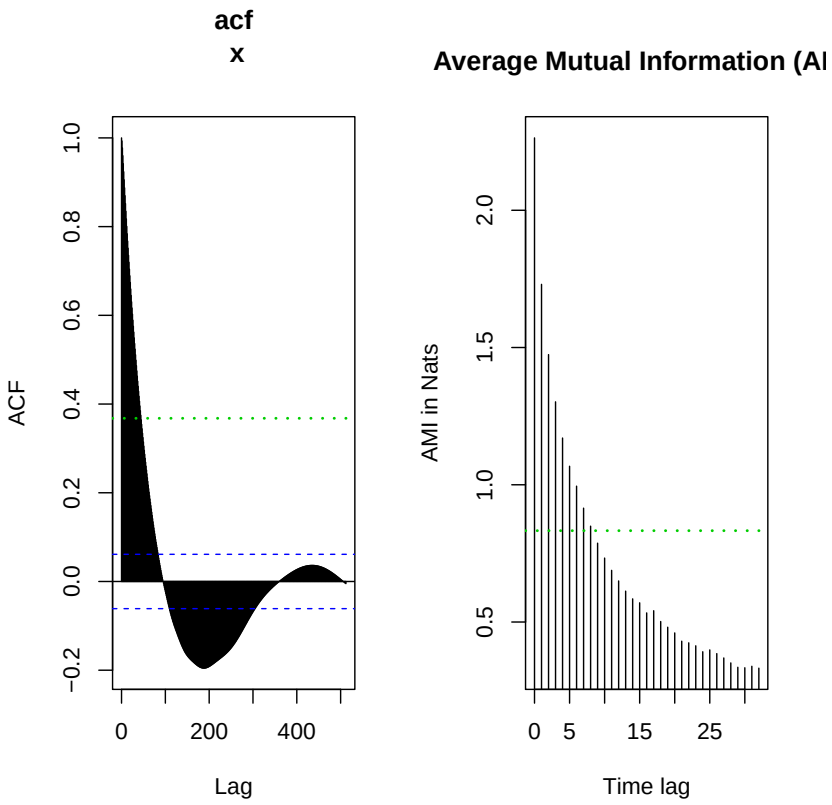
```
# taudelay estimation based on the mutual information function
tau.ami = timeLag(dopplerN, technique = "ami", do.plot = T,
main=paste("ami\n",deparse(substitute(x))))
cat("tau.ami:",tau.ami)
```

Output

```
tau.ami: 9
```

Input

```
par(oldpar)
```



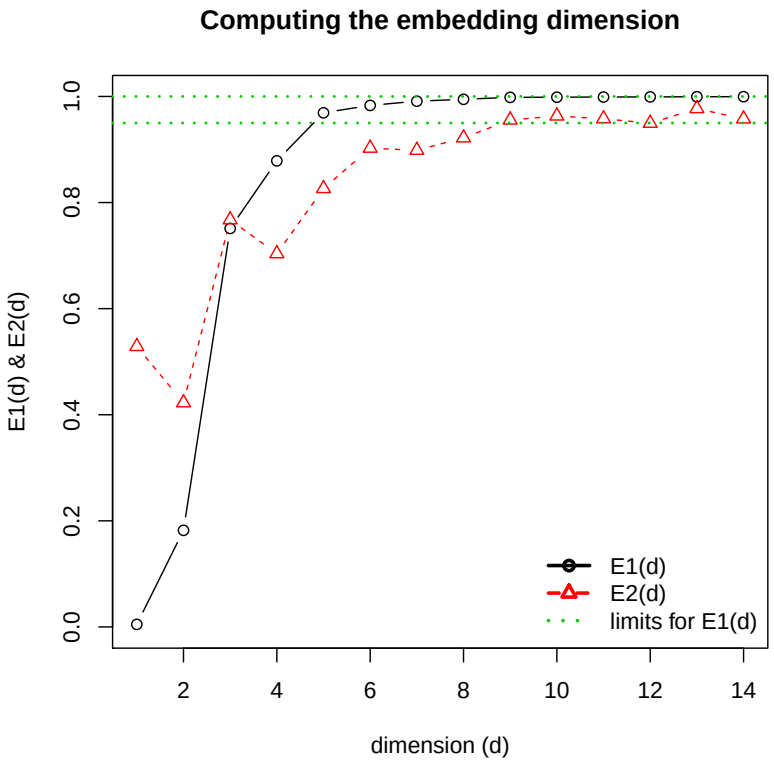
Input

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

Output

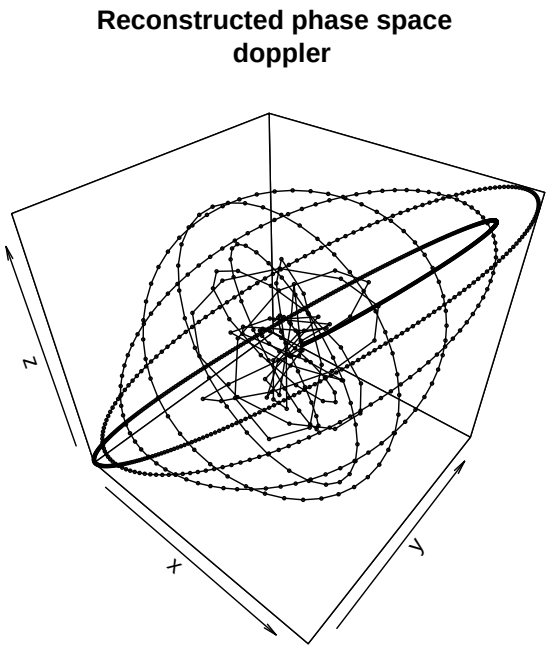
```
Estimated embedding dim: 5
```

Input



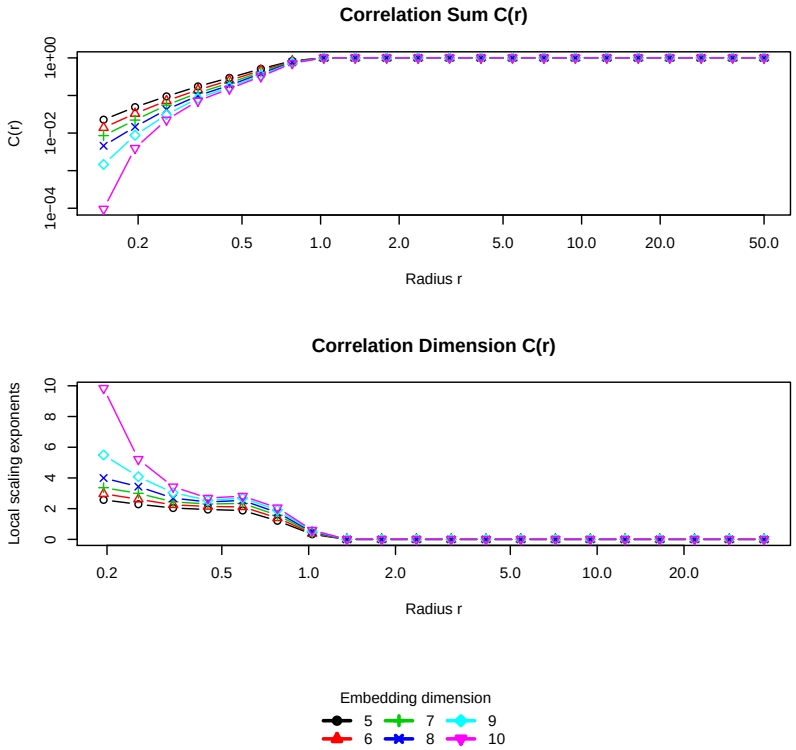
Input

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



Input

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



Input

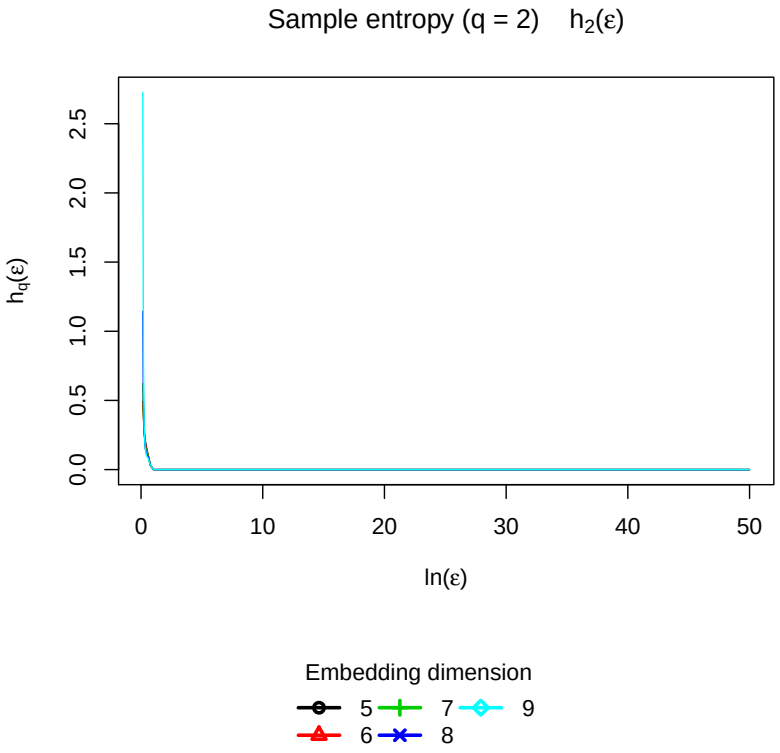
```
oldpar <- par(mfrow=c(1,2))
se = sampleEntropy(cd, do.plot = T)
se.est = estimate(se, do.plot = T,
regression.range = c(8,15))
cat("Sample entropy estimate: ", mean(se.est), "\n")
```

Output

```
Sample entropy estimate:  0
```

Input

```
par(oldpar)
```



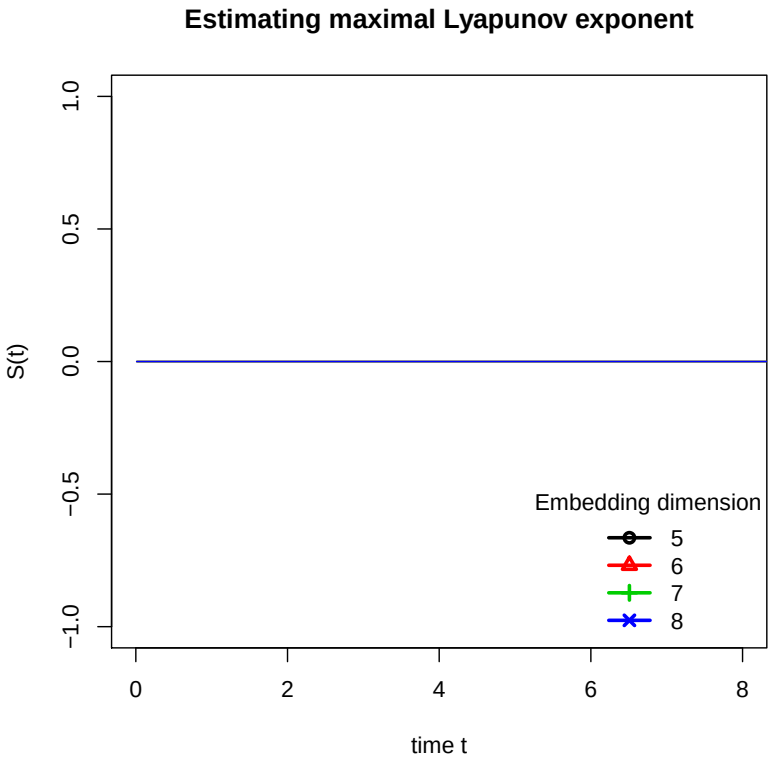
Input

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

Output

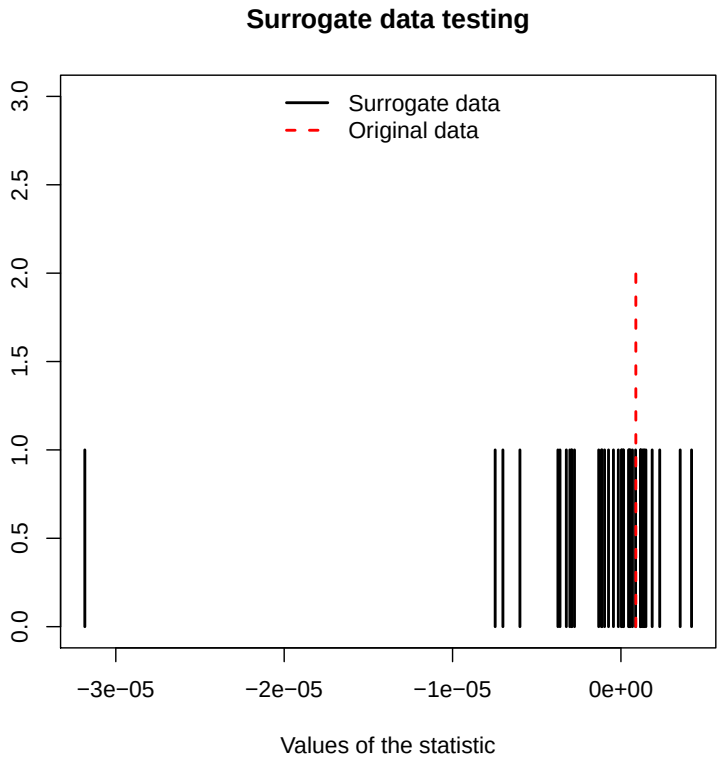
```
estimate: 0
```

Input



Input

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


Surrogate data testing
Surrogate data
Original data
0.0
0.5
1.0
1.5
2.0
2.5
3.0
−3e−05
−2e−05
−1e−05
0e+00
Values of the statistic

5. Takens' States for Test Signals

Input

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

See Figure 5.

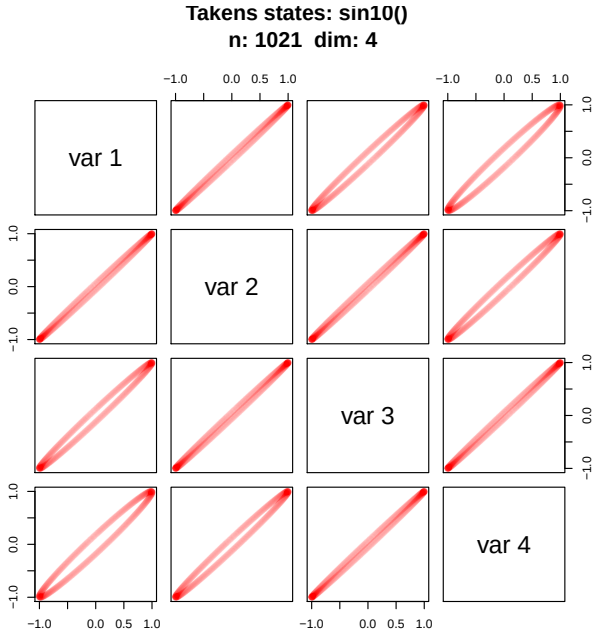


Figure 5. Takens states. Test case: sinus. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.676 sec.

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 following page.

Input

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

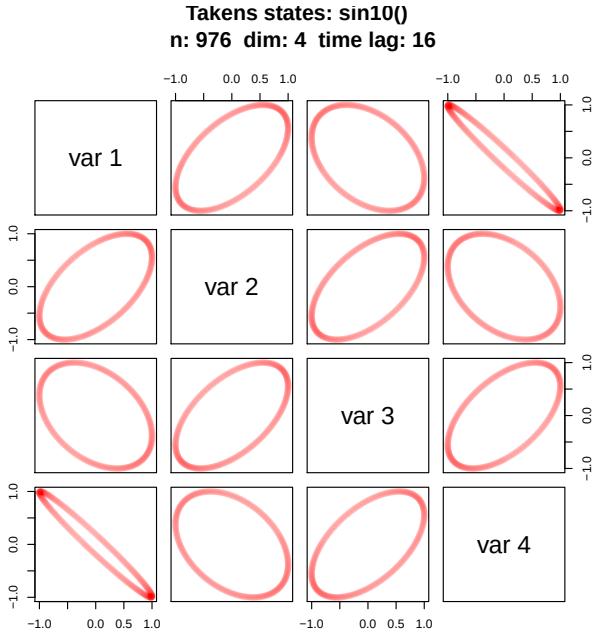


FIGURE 6. Takens states. Test case: sinus at time lag 16. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.608 sec.

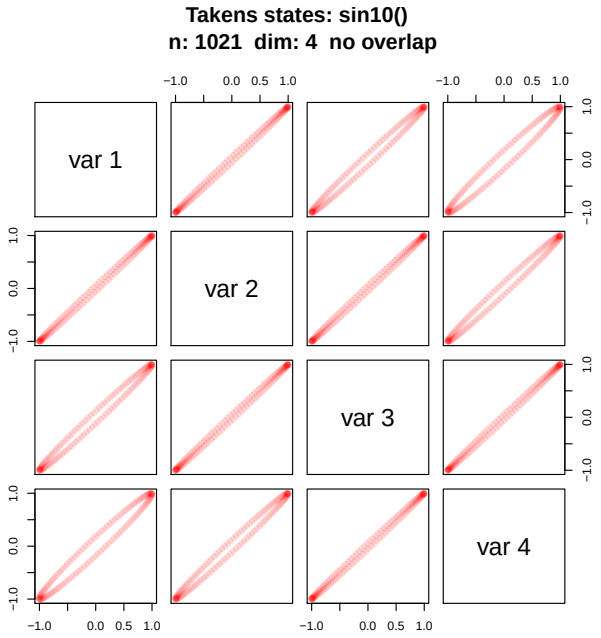


FIGURE 7. Takens states. Test case: sinus, no overlap. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.786 sec.

See Figure 8.

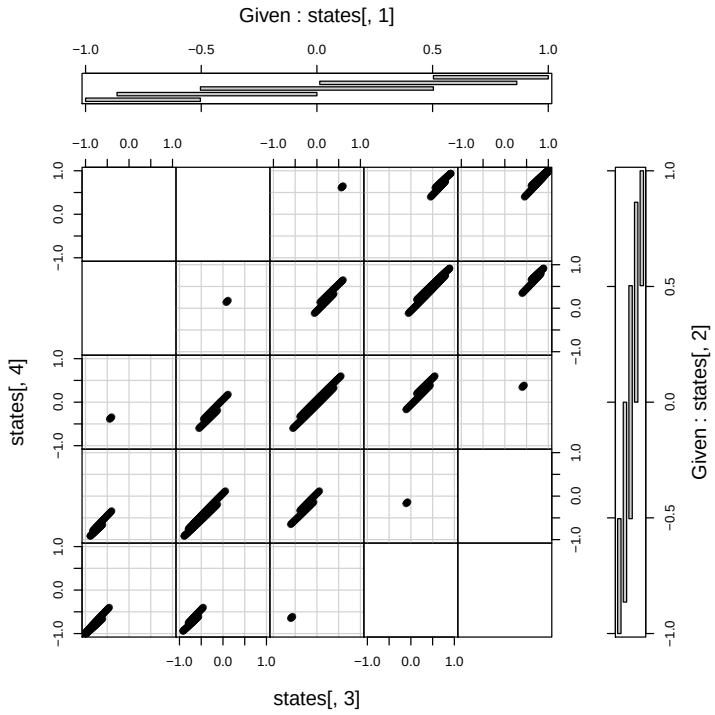


FIGURE 8. State coplot. Test case: sinus. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.225 sec.

Input

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

See Figure 9 on the following page.

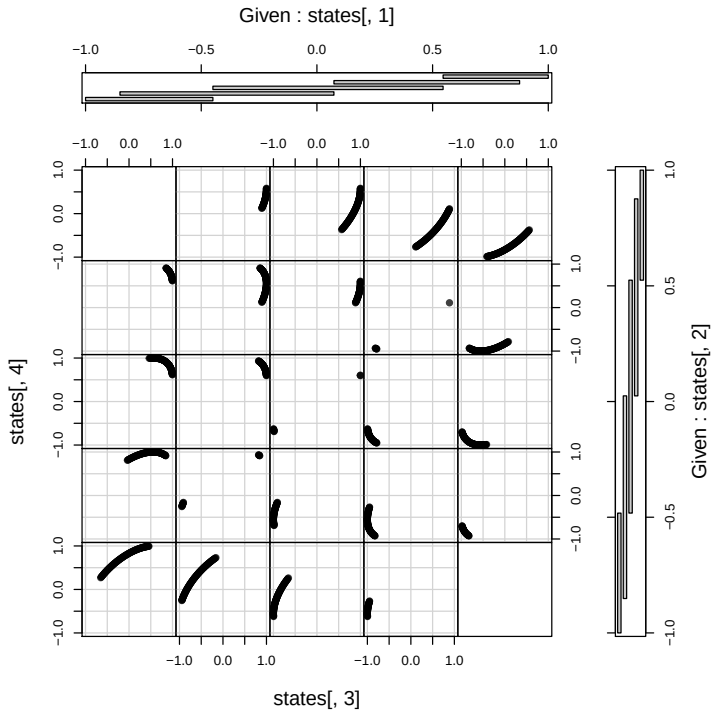


FIGURE 9. State coplot. Test case: sinus, time lag 16. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.188 sec.

Input

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

See Figure 10.

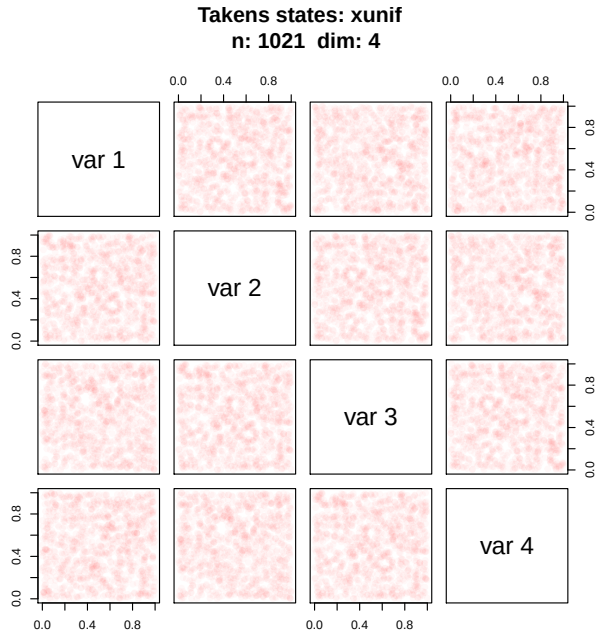


FIGURE 10. Takens states. Test case: uniform random numbers. Time used: 0.699 sec.

Input

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

See Figure 11 on the following page.

Input

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

See Figure 12 on the next page.

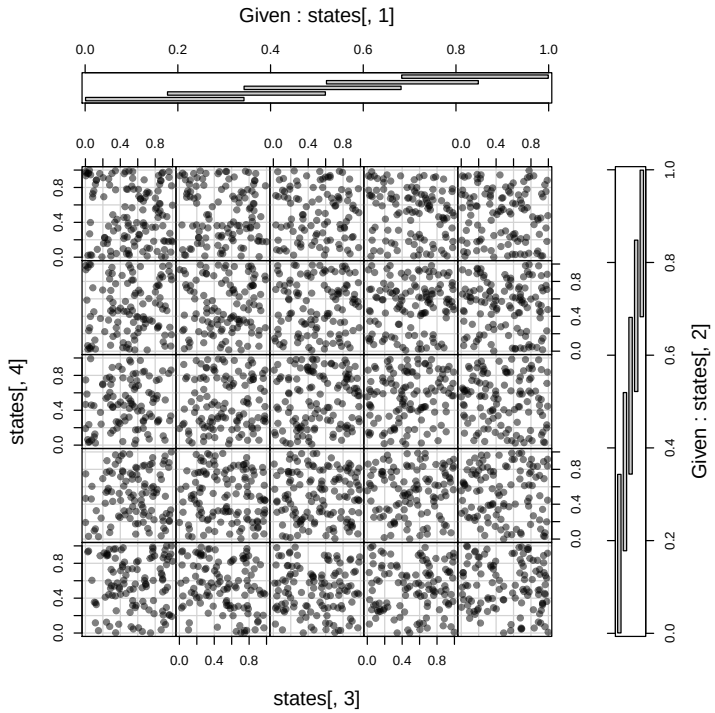


FIGURE 11. State coplot. Test case: uniform random numbers. Time used: 0.922 sec.

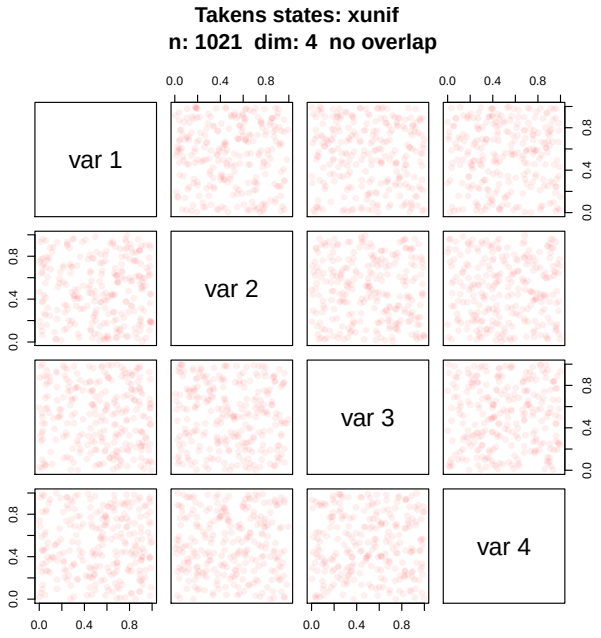


FIGURE 12. Takens states. Test case: uniform random numbers. Time used: 1.152 sec.

Input

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

See Figure 13.

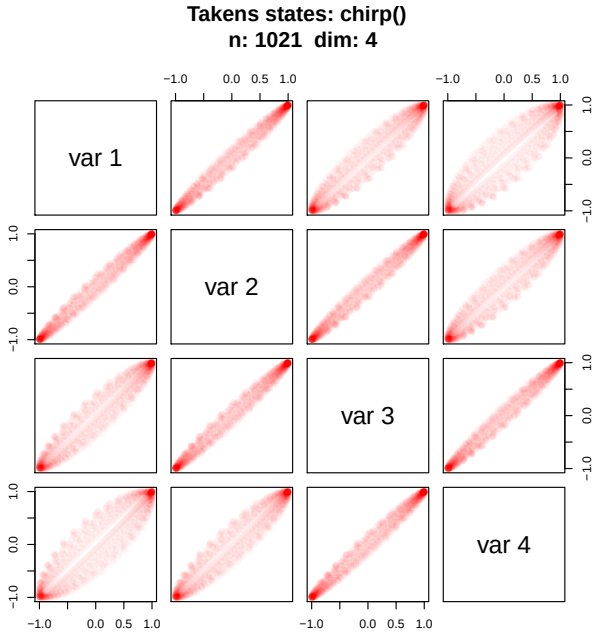


FIGURE 13. Takens states. Test case: chirp signal. Time used: 0.62 sec.

Input

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

See Figure 14 on the next page.

Input

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

See Figure 15 on the following page.

Input

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

See Figure 16 on page 41.

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.

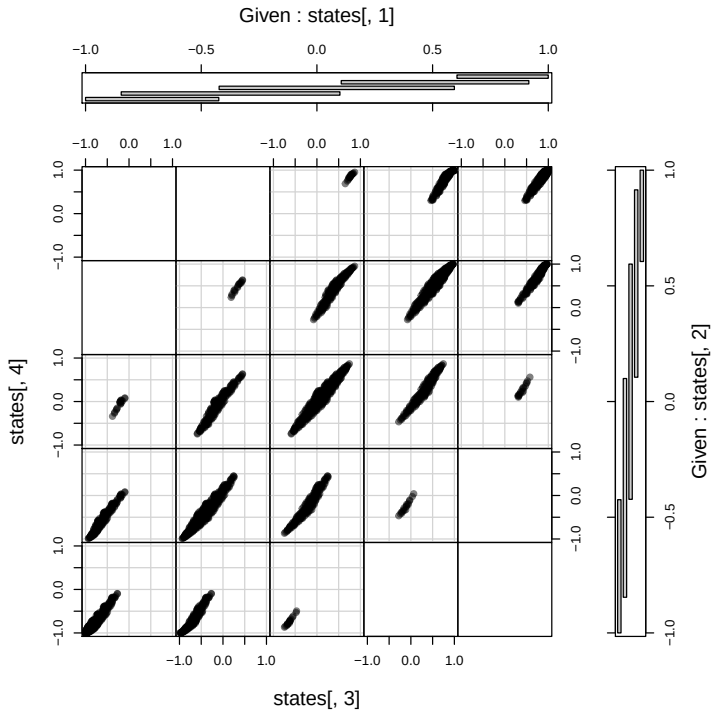


FIGURE 14. State coplot. Test case: chirp signal. Time used: 0.874 sec.

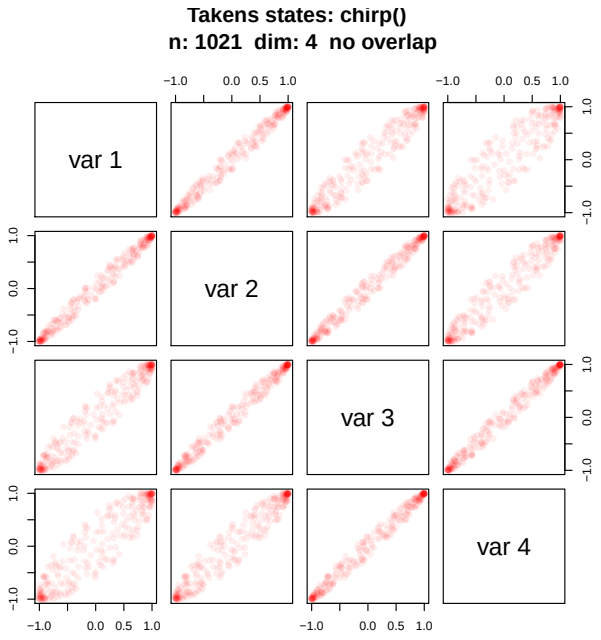


FIGURE 15. Takens states. Test case: chirp signal. Time used: 1.097 sec-

Input

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

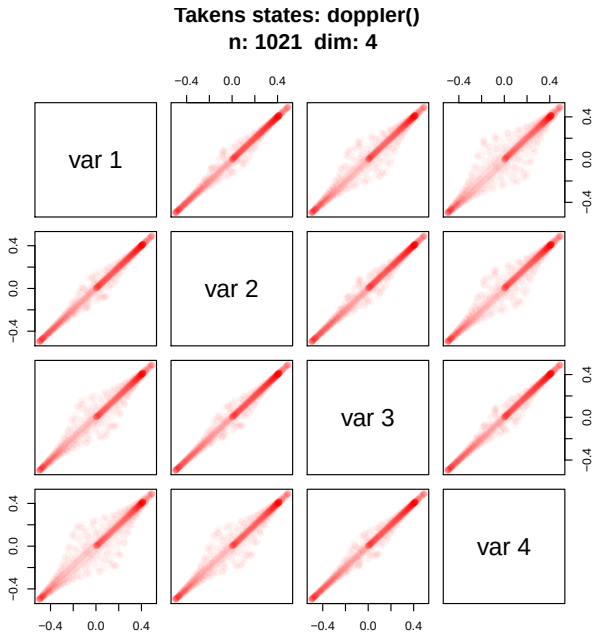


FIGURE 16. Takens states. Test case: Doppler. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.635 sec.

See Figure 17.

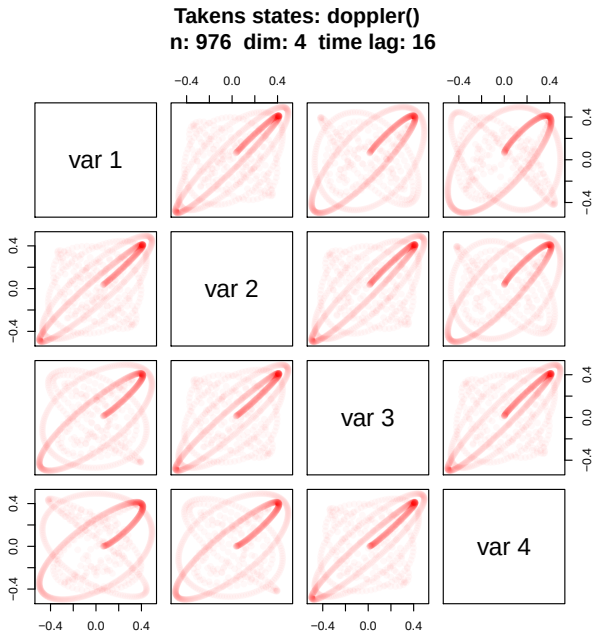


FIGURE 17. Takens states. Test case: doppler at time lag 16. Note that $2 - dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.604 sec.

Input

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

See Figure 18.

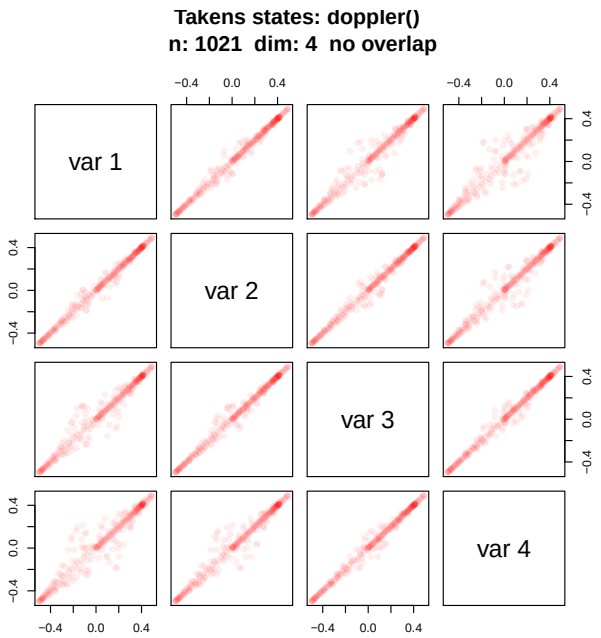


FIGURE 18. Takens states. Test case: doppler, no overlap. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.83 sec.

Input

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

See Figure 19 on the facing page.

Input

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

See Figure 20 on the next page.

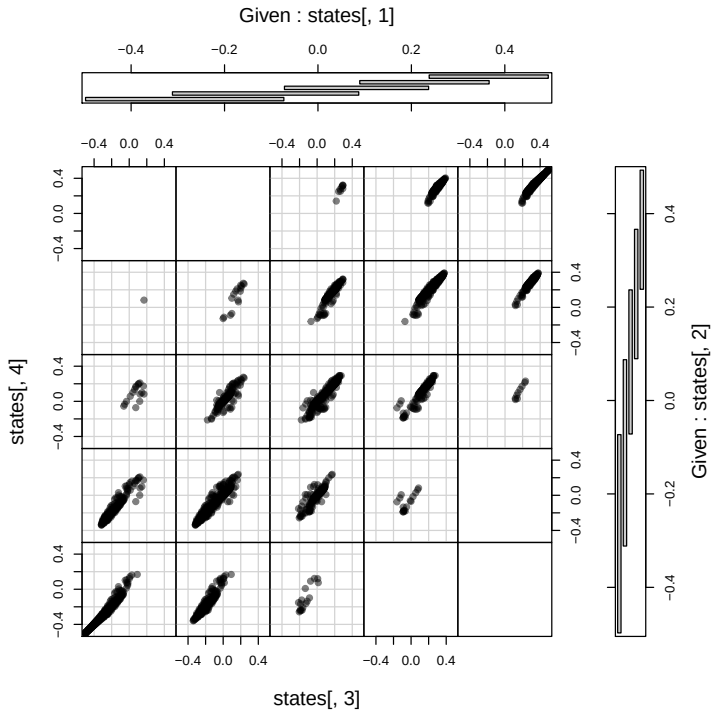


FIGURE 19. State coplot. Test case: Doppler. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.245 sec.

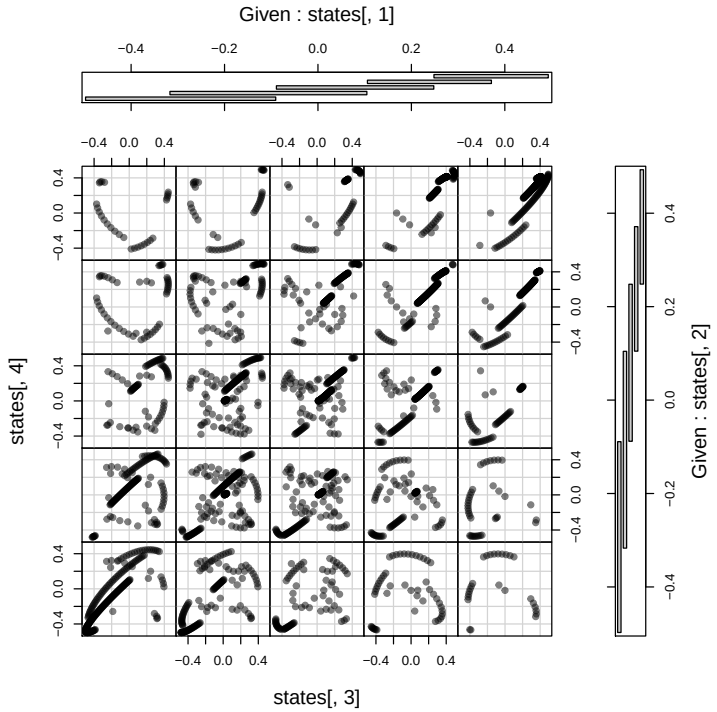


FIGURE 20. State coplot. Test case: Doppler, time lag 16. Note that $2-dim$ marginal views of 1-dimensional circles in d space generally appear as ellipses. Time used: 0.258 sec.

6. Recurrence Plots for Test Signals

6.0.1. *Sinus Recurrence Plots.*

Input

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

See Figure 21.

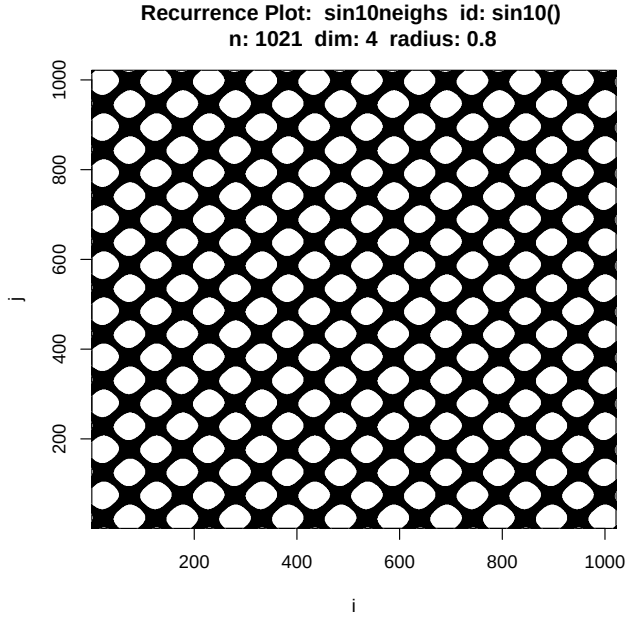


Figure 21. Recurrence plot. Test case: sinus curve. Time used: 2.704 sec.

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")
```

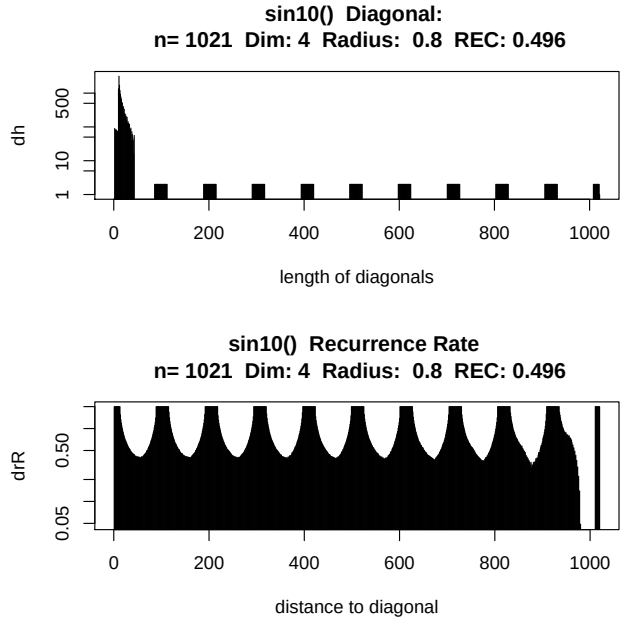


FIGURE 22. RQA. Test case: sinus curve. Time used: 0.72 sec.

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

See Figure 23.

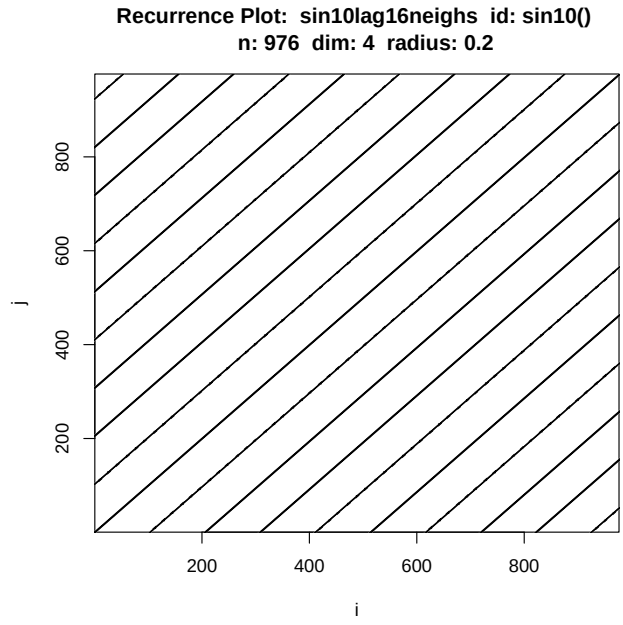


FIGURE 23. Recurrence plot. Recurrence Plot. Test case: sinus curve, time lag 16. Time used: 1.583 sec.

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.

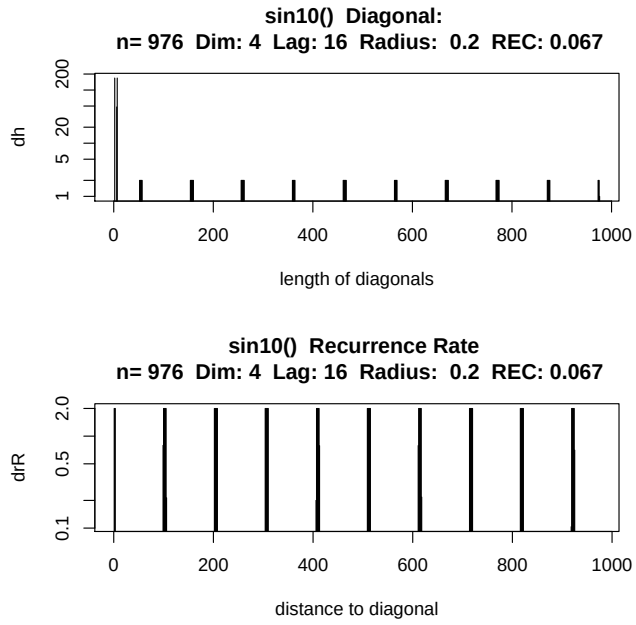


FIGURE 24. RQA. Test case: sinus curve, time lag 16. Time used: 1.777 sec.

6.0.2. *Uniform Random Numbers Recurrence Plots.*

Input

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

See Figure 25.

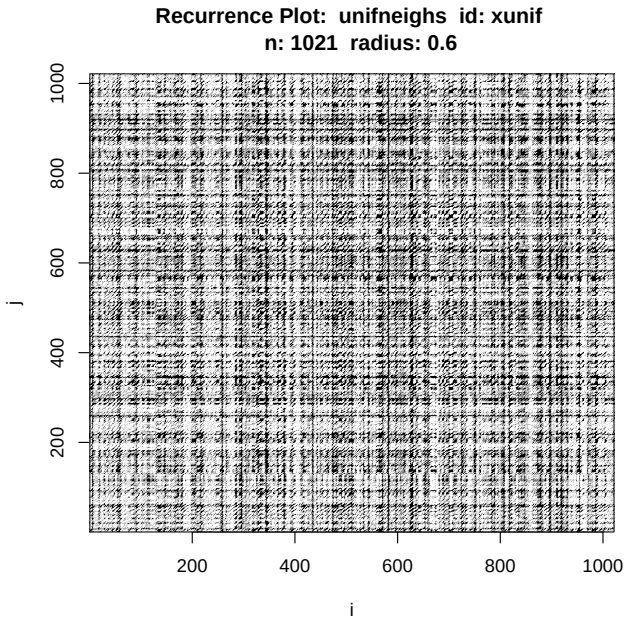


FIGURE 25. Recurrence plot. Test case: uniform random numbers. Time used: 3.054 sec.

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.

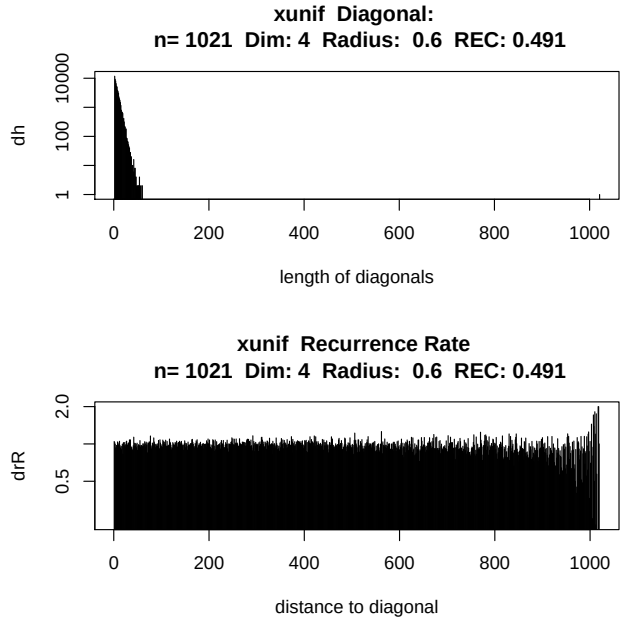


FIGURE 26. RQA. Test case: uniform random numbers, radius=0.6. Time used: 4.02 sec.

6.0.3. *Chirp Signal Recurrence Plots.*

Input

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

See Figure 27.

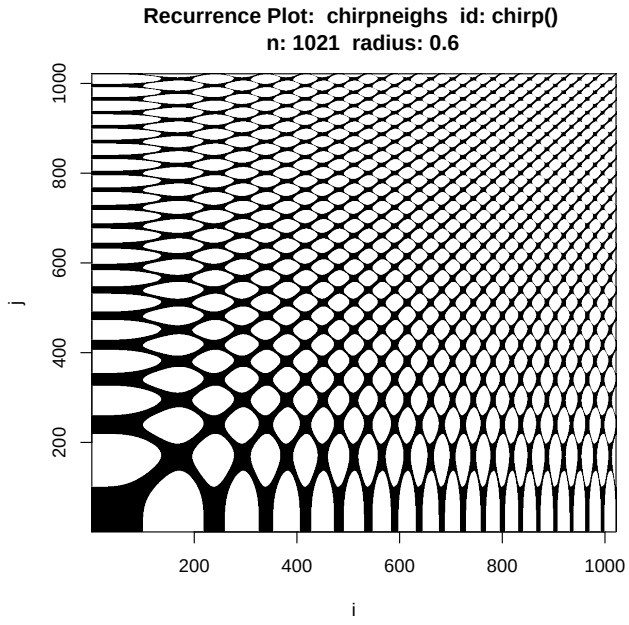


FIGURE 27. Recurrence plot. Test case: chirp signal. Time used: 2.141 sec.

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.

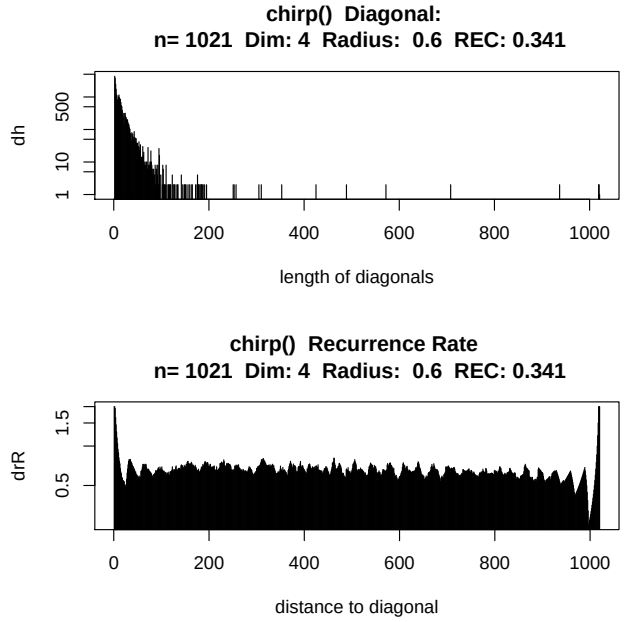


FIGURE 28. RQA. Test case: chirp signal. Time used: 2.627 sec.

6.0.4. *Doppler Recurrence Plots.*

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

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

See Figure 29.

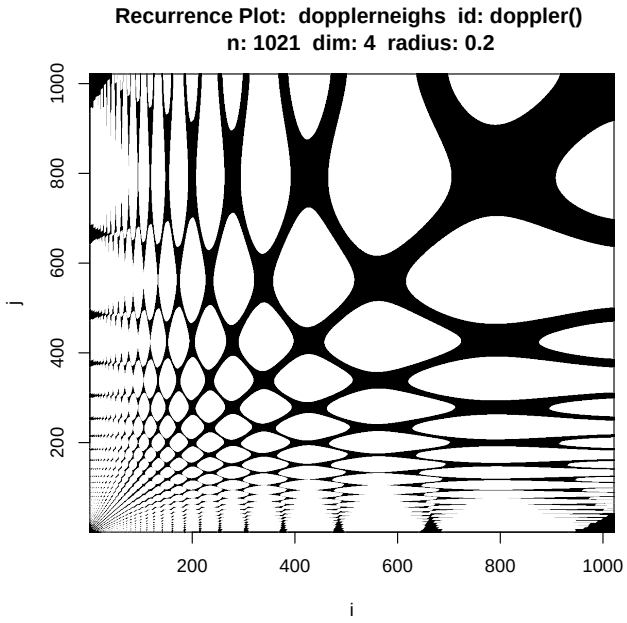


FIGURE 29. Recurrence plot. Test case: Doppler. Time used: 1.766 sec.

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")
```

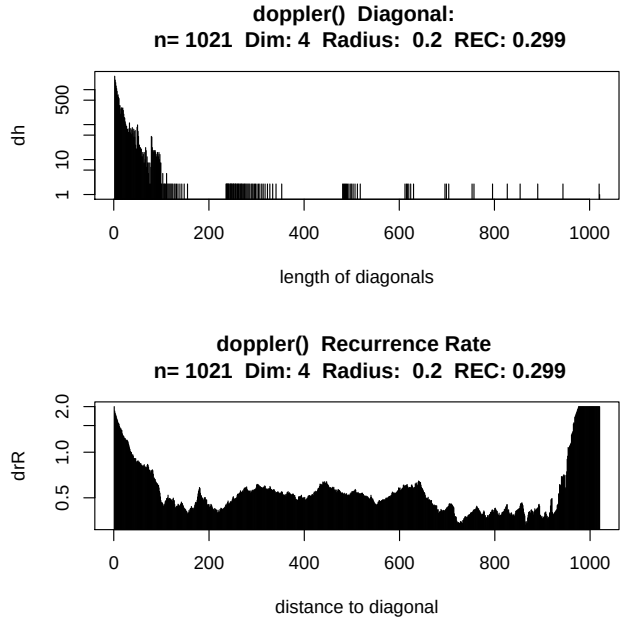


FIGURE 30. RQA. Test case: Doppler. Time used: 0.407 sec.

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

See Figure 31.

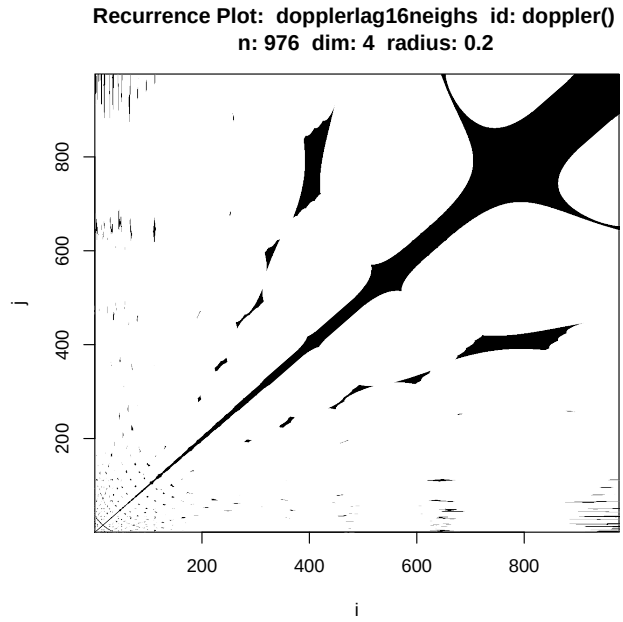


FIGURE 31. Recurrence plot. Test case: Doppler. Time used: 1.094 sec.

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.

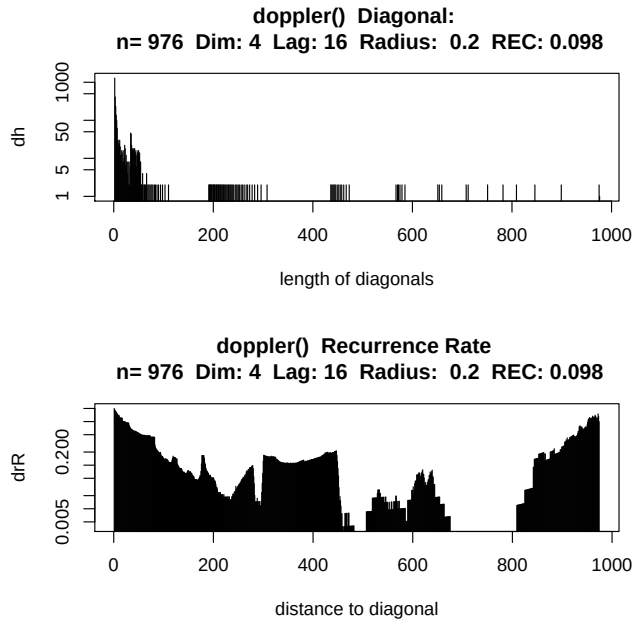


FIGURE 32. RQA. RQA. Test case: Doppler. Time used: 1.307 sec.

7. Case Study: Geyser data

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

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

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

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

Input

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

7.1. **Geyser Eruption Durations.**

Input

```
plotsignal(geyser$duration)
```

See Figure 33,

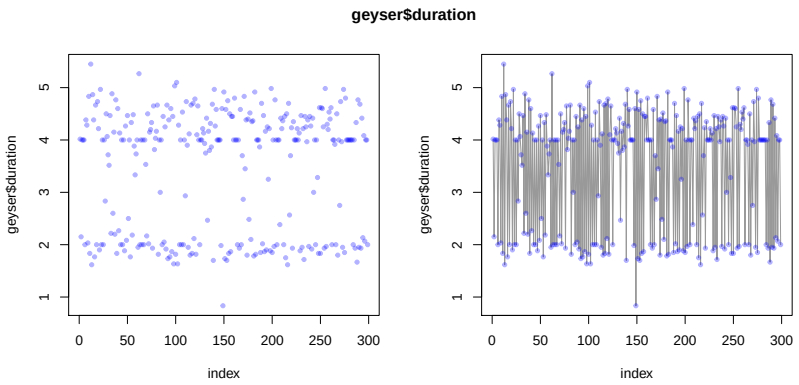


Figure 33. Example case: Old Faithful Geyser eruption durations. Signal and linear interpolation.

Input

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

See Figure 34 on the next page.

Input

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

See Figure 35 on the facing page.

Input

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

See Figure 36 on page 56.

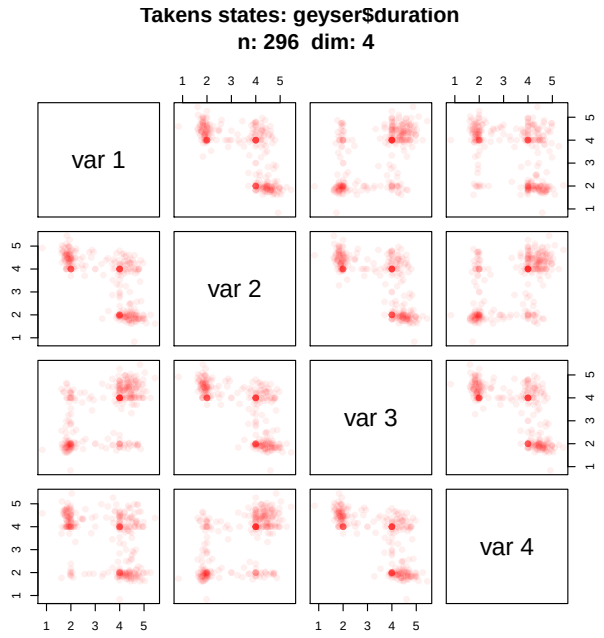


FIGURE 34. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Time used: 0.217 sec.

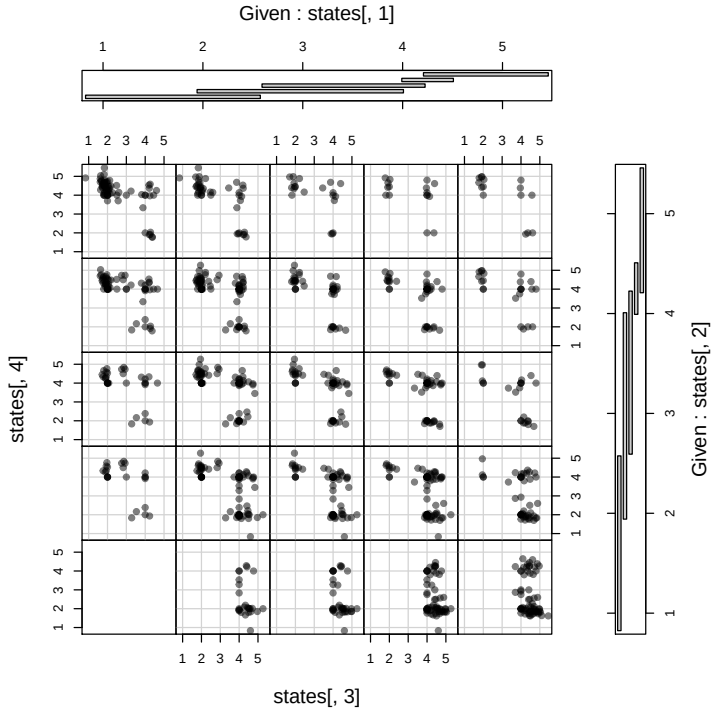


FIGURE 35. State coplot. Example case: Old Faithful Geyser eruption durations. Time used: 0.366 sec.

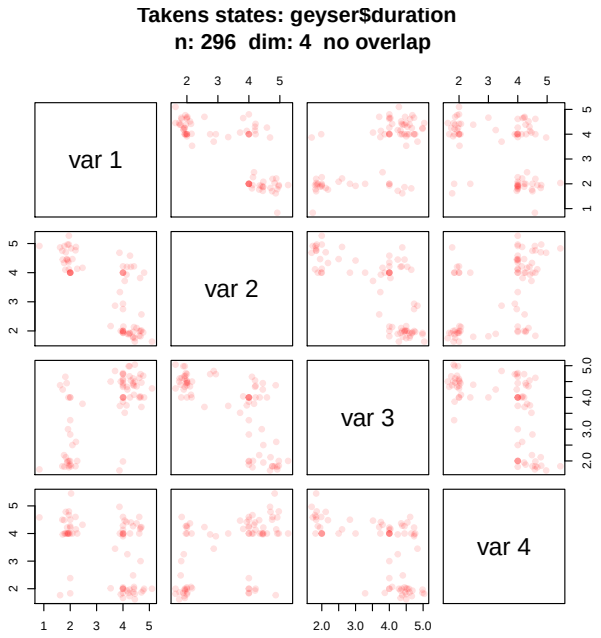


FIGURE 36. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Time used: 0.468 sec.

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.

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

Input

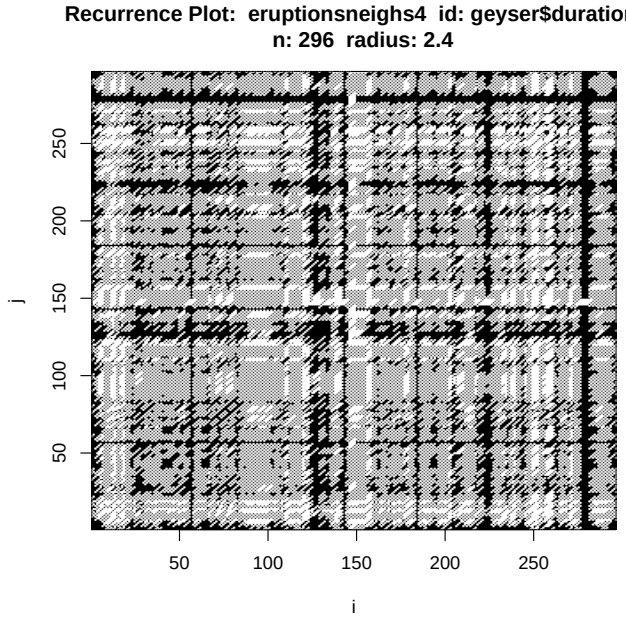


FIGURE 37. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.236 sec.

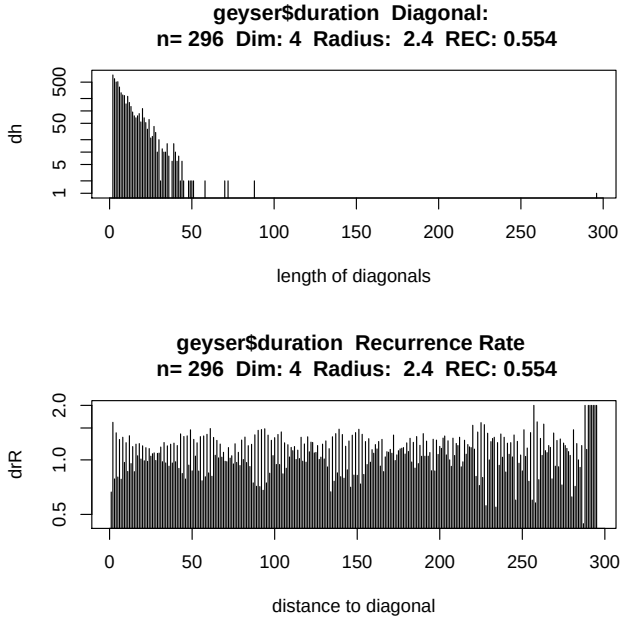


FIGURE 38. RQA. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.357 sec.

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

See Figure 39.

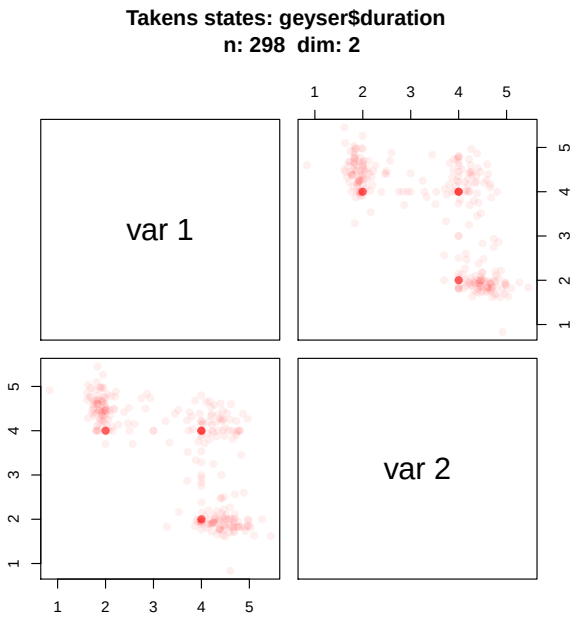


FIGURE 39. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=2. Time used: 0.068 sec.

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 60.

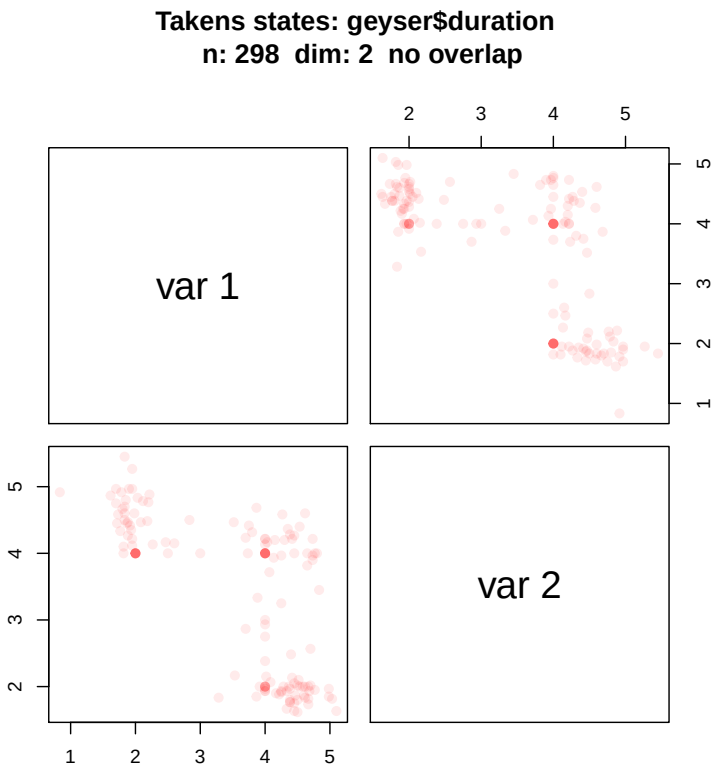


FIGURE 40. Example case: Old Faithful Geyser eruption durations. Dim=2. Time used: 0.138 sec.

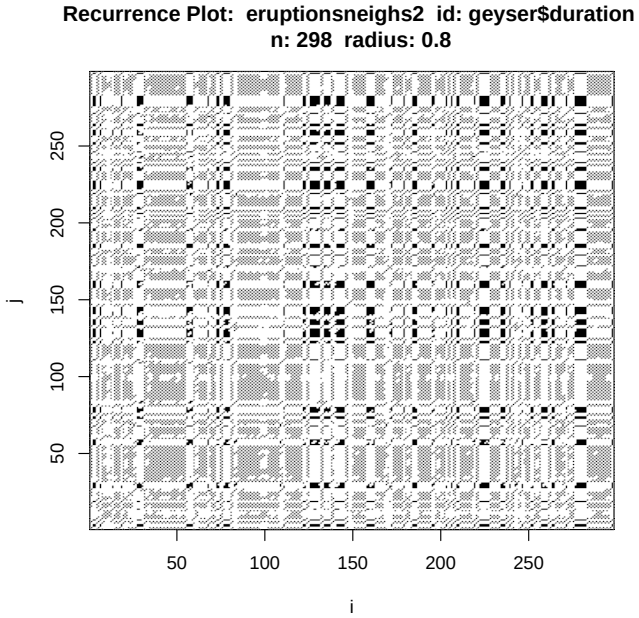


FIGURE 41. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=2. Time used: 0.191 sec.

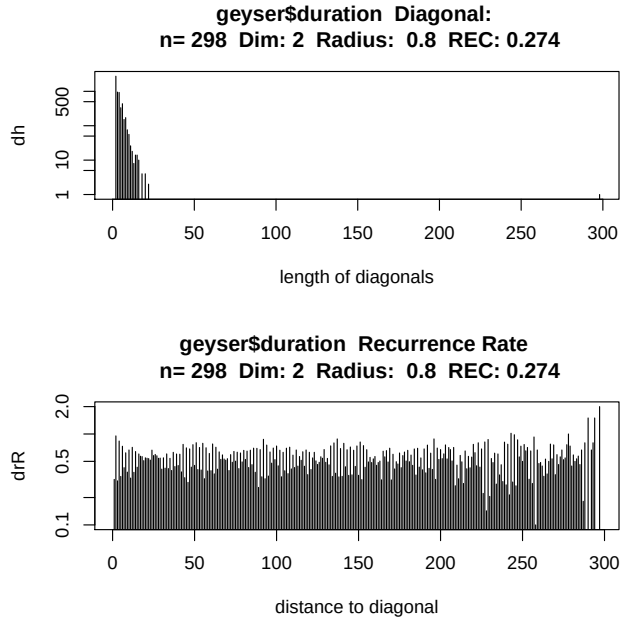


FIGURE 42. RQA. Example case: Old Faithful Geyser eruption durations. Dim=2. Time used: 0.274 sec.

7.1.2. *Geyser eruptions. Dim=4.*

Input

```
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.

Input

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

Input

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

See Figure 45 on page 62.

Input

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

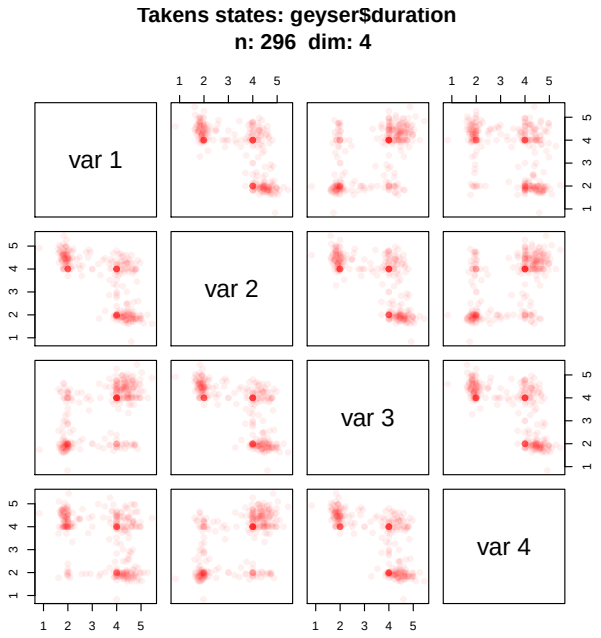


FIGURE 43. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.23 sec.

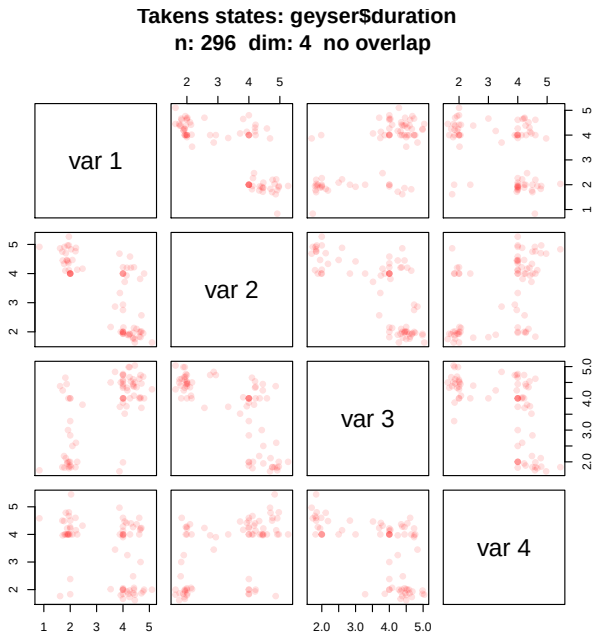


FIGURE 44. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.349 sec.

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
```

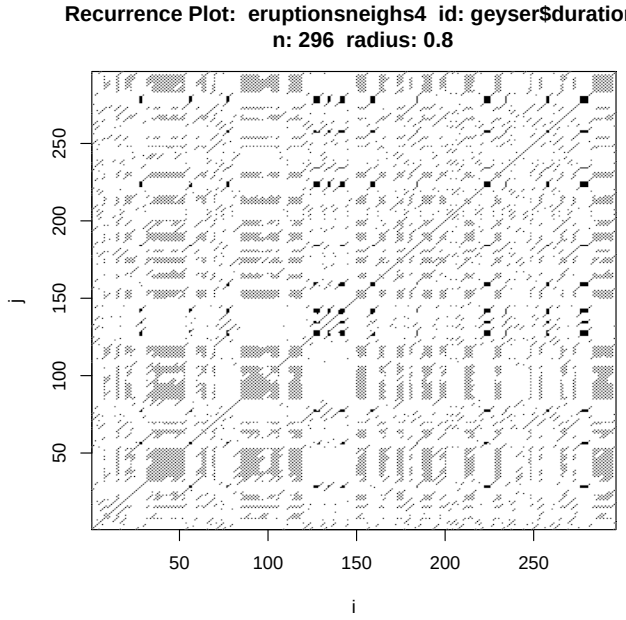


FIGURE 45. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.119 sec.

```
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.

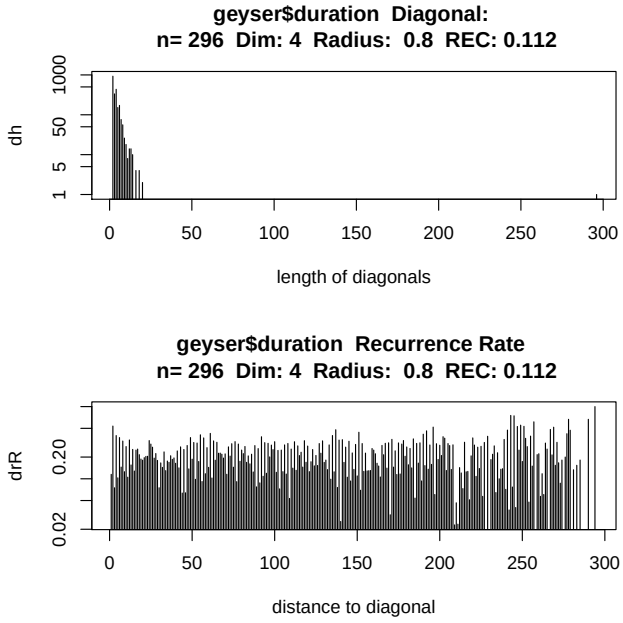


FIGURE 46. Recurrence plot. .

Example case: Old Faithful Geyser eruption durations. Dim=4. Time used: 0.209 sec.

7.1.3. *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.

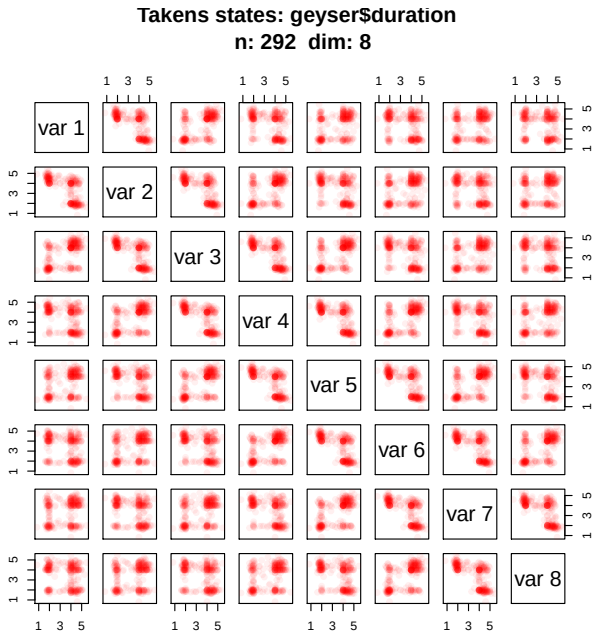


FIGURE 47. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 0.894 sec.

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 65.

Input

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

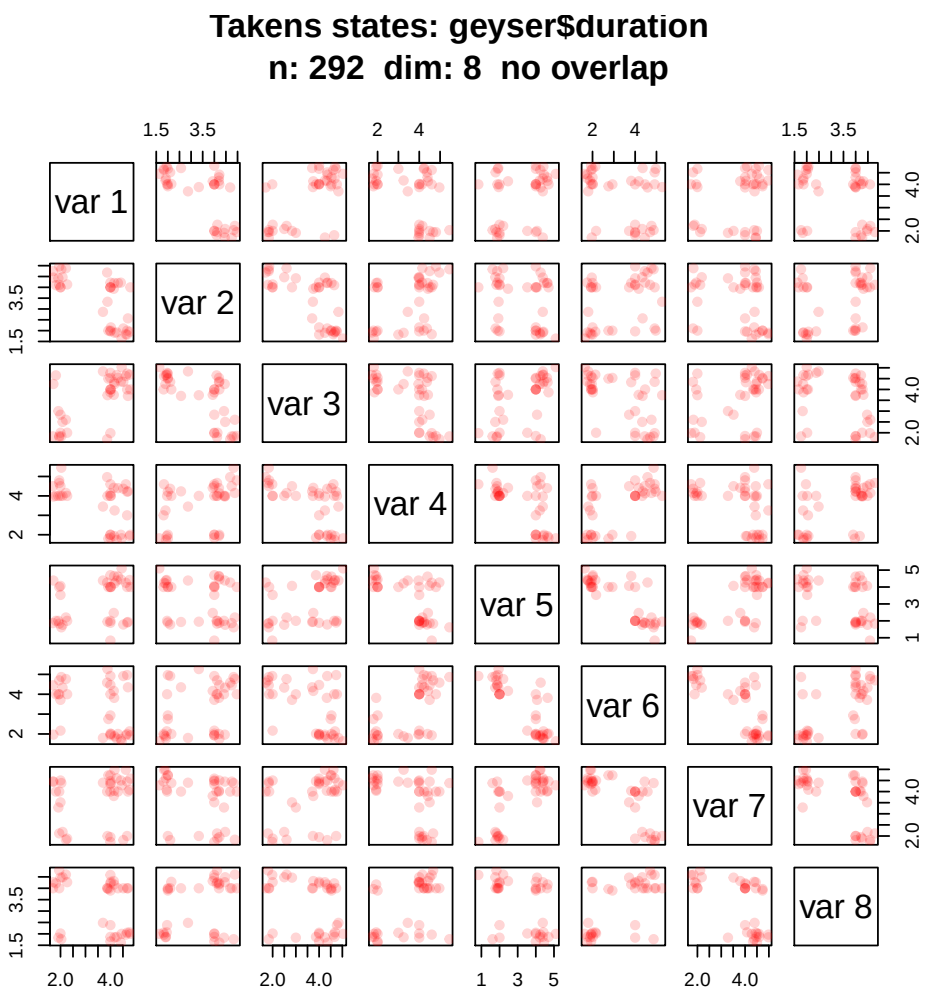


FIGURE 48. Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 1.133 sec.

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.

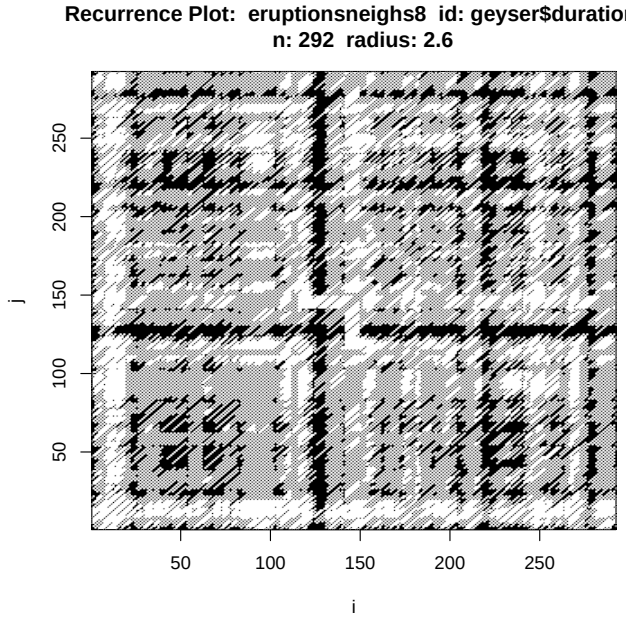


FIGURE 49. Recurrence plot. Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 0.298 sec.

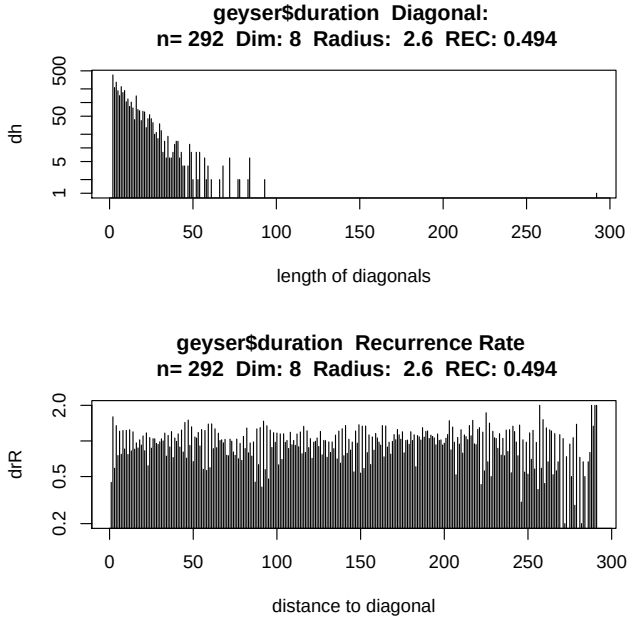


FIGURE 50. RQA. Example case: Old Faithful Geyser eruption durations. Dim=8. Time used: 0.401 sec.

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

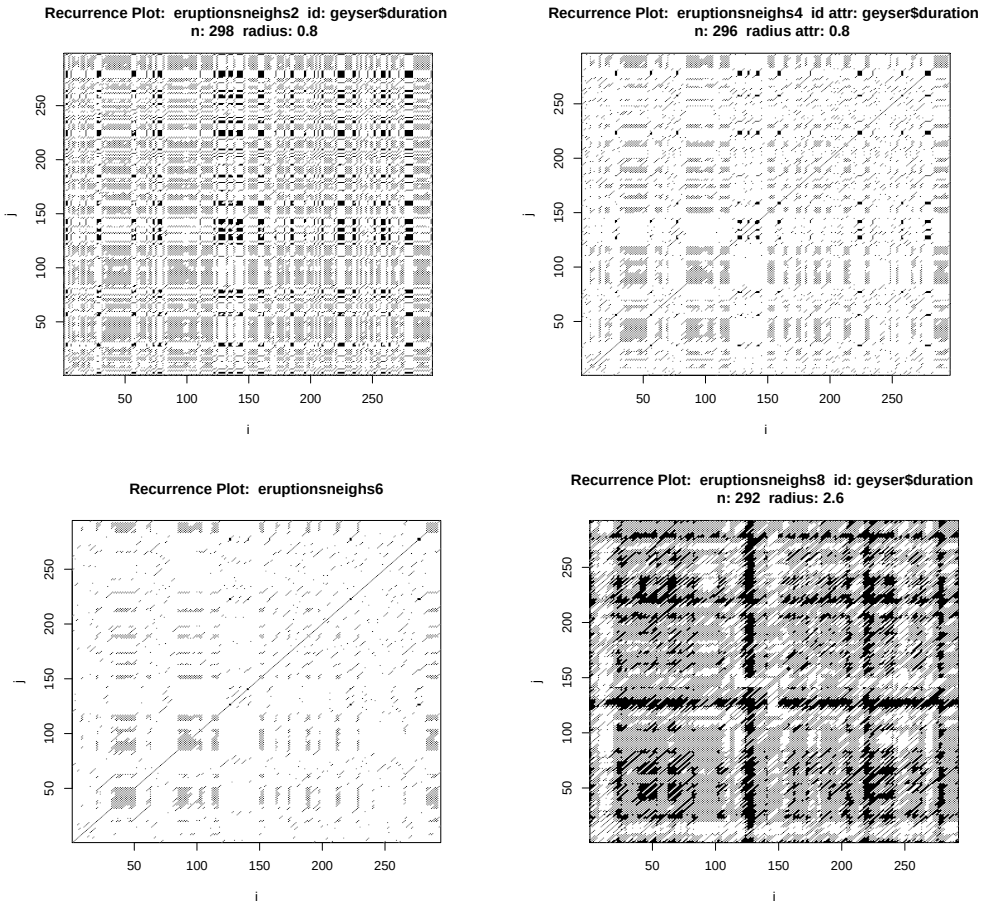


FIGURE 51. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=2, 4, 6, 8.

7.2. **Geyser Waiting.**

Input

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

See Figure 52 on the next page.

Input

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

See Figure 53 on the facing page.

Input

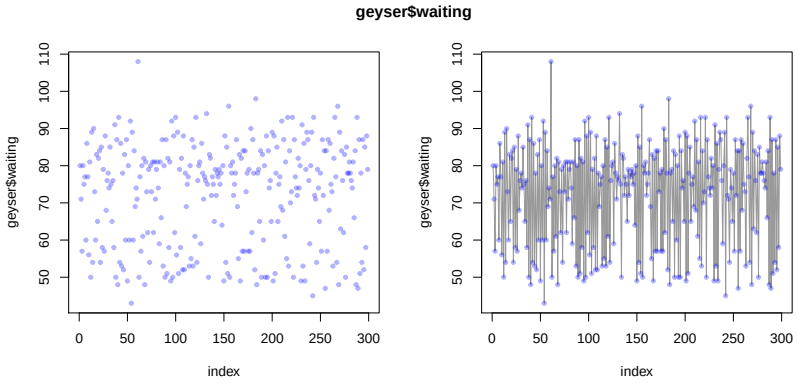


FIGURE 52. Example case: Old Faithful Geyser waiting. Signal and linear interpolation. Time used: 0.5 sec.

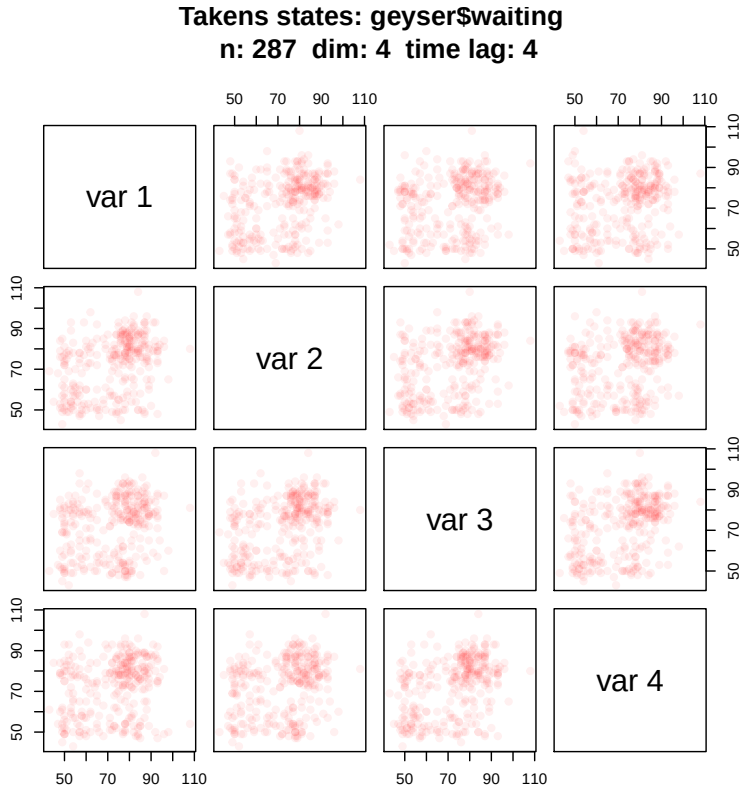


FIGURE 53. Example case: Old Faithful Geyser waiting. Time used: 0.235 sec.

```
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.

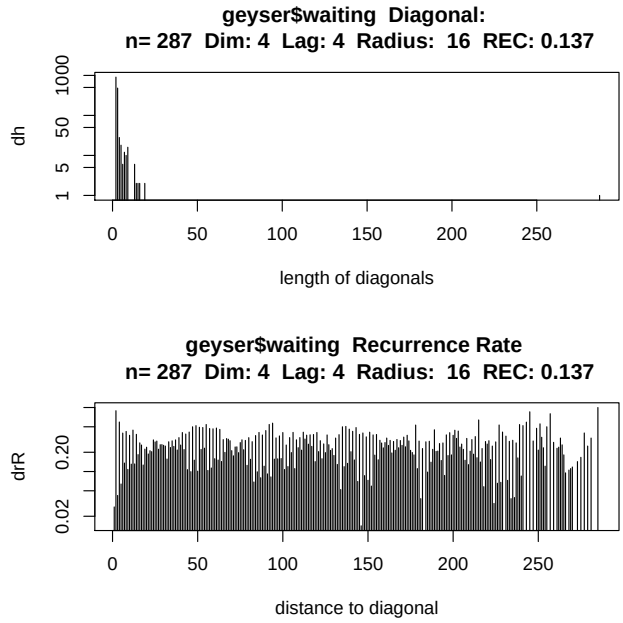


FIGURE 54. RQA. Example case: Old Faithful Geyser waiting. Time used: 0.074 sec.

Input

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

See Figure 55 on the facing page.

7.3. **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
```

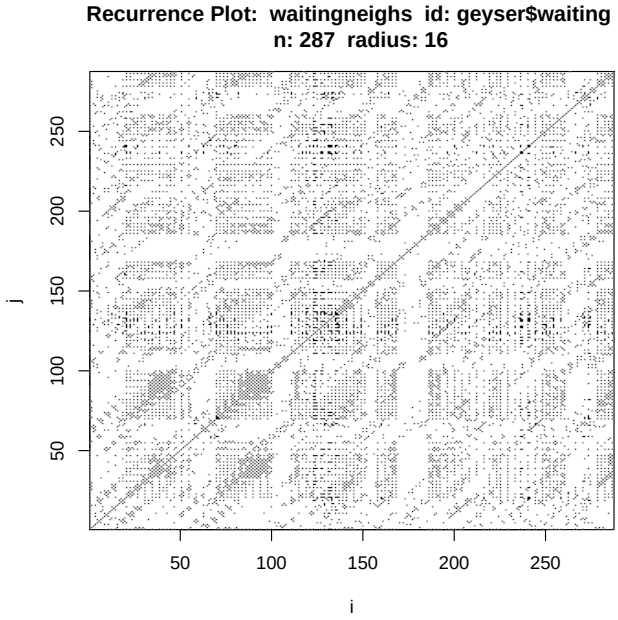


FIGURE 55. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser waiting. Time used: 0.206 sec.

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.

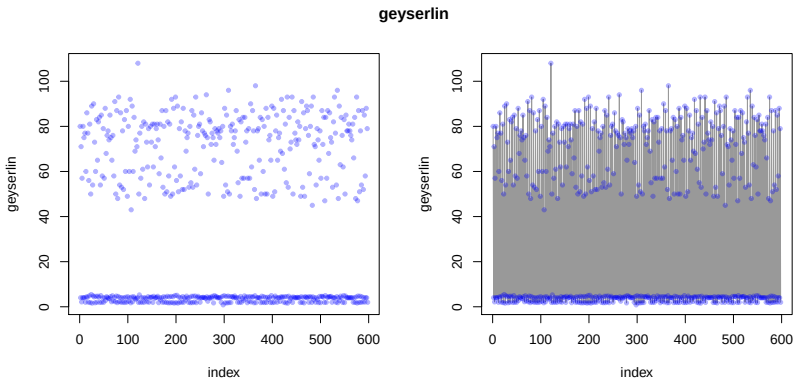


FIGURE 56. Example case: Old Faithful Geyser eruptions. Signal and linear interpolation.

Input

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

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

See Figure 57.

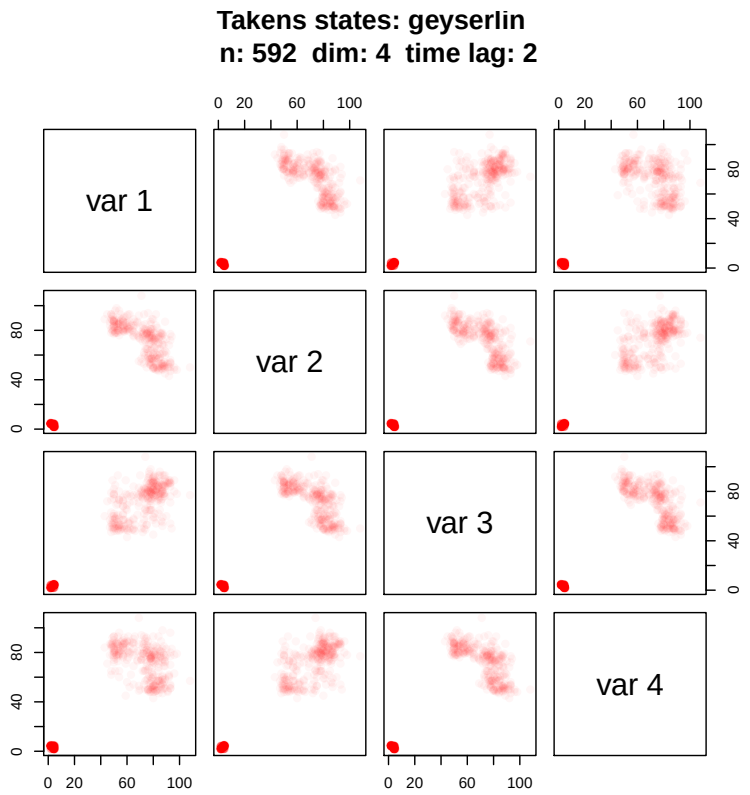


FIGURE 57. Example case: Old Faithful Geyser eruptions. Time used: 0.417 sec.

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.

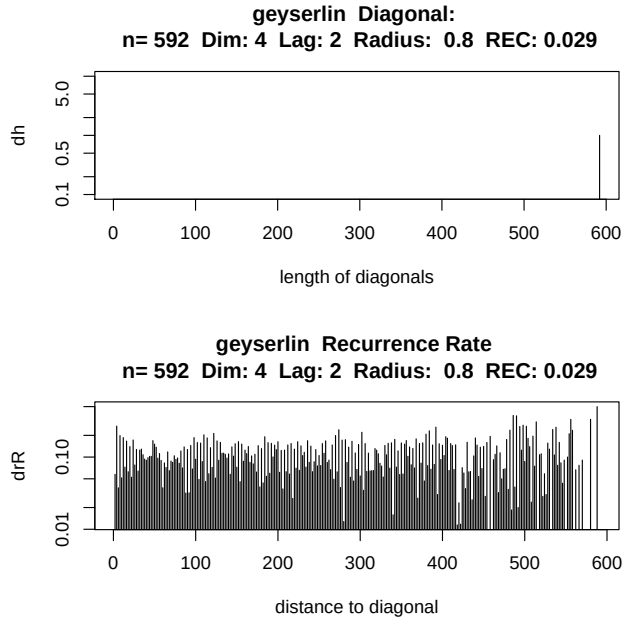


FIGURE 58. Recurrence plot. Old Faithful Geyser - both. Dim=4. Time used: 0.154 sec.

Input

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

See Figure 59.

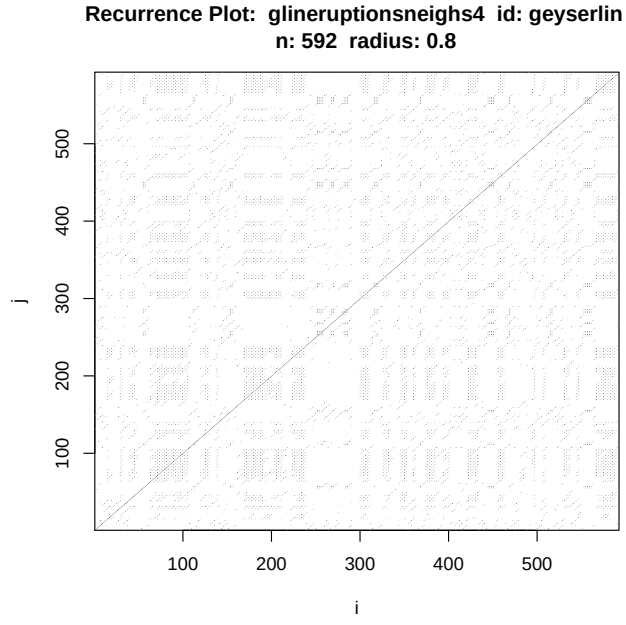


FIGURE 59. Recurrence plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.39 sec.

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

Input

```
glineruptionstakens2 <-
        local.buildTakens(time.series=geyserlin,
                embedding.dim=2, time.lag=2)
statepairs(glineruptionstakens2) #dim=2
```

See Figure 60.

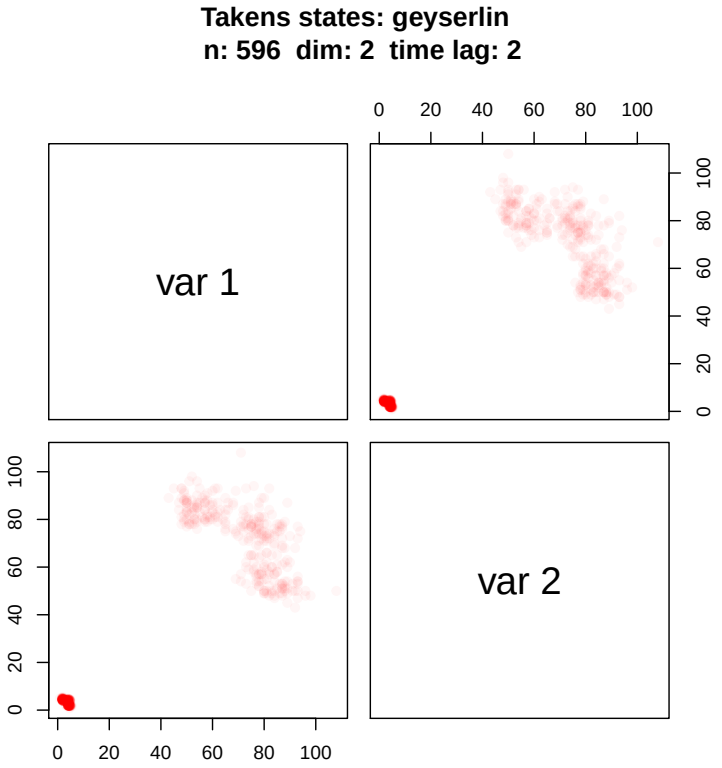


FIGURE 60. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.127 sec.

Input

```
glineruptionsneighs2 <- local.findAllNeighbours(glineruptionstakens2, radius=0.8)
save(glineruptionsneighs2, file="glineruptionsneighs2.RData")
#load(file="glineruptionsneighs2.RData")
local.recurrencePlotAux(glineruptionsneighs2)
```

See Figure 61 on the facing page.

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

Input

```
glineruptionstakens8 <- local.buildTakens( time.series=geyserlin,
        embedding.dim=8,time.lag=2)
statepairs(glineruptionstakens8) #dim=8
```

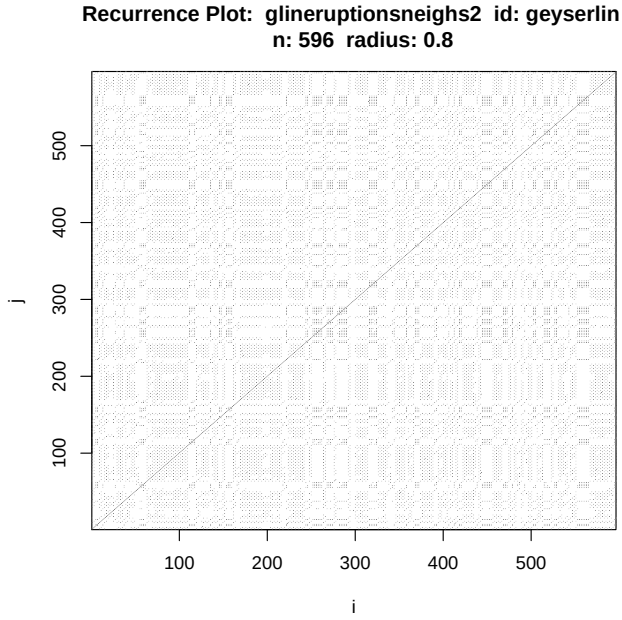


FIGURE 61. Recurrence plot. Example case: Old Faithful Geyser eruptions linearized. Dim=2. Time used: 0.369 sec.

See Figure 62 on the next page

Input

```
glineruptionsneighs8 <- local.findAllNeighbours(glineruptionstakens8,
        radius=0.8)
save(glineruptionsneighs8, file="glineruptionsneighs8.RData")
```

Input

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

See Figure 63 on page 75.

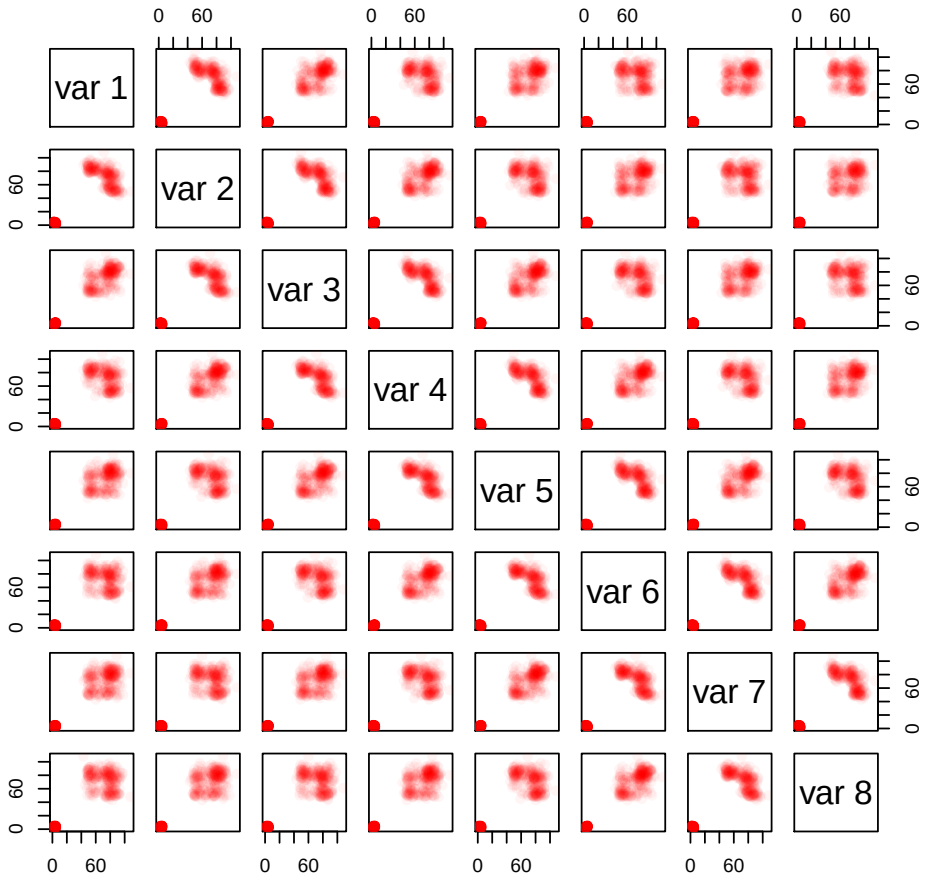


FIGURE 62. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 1.553 sec.

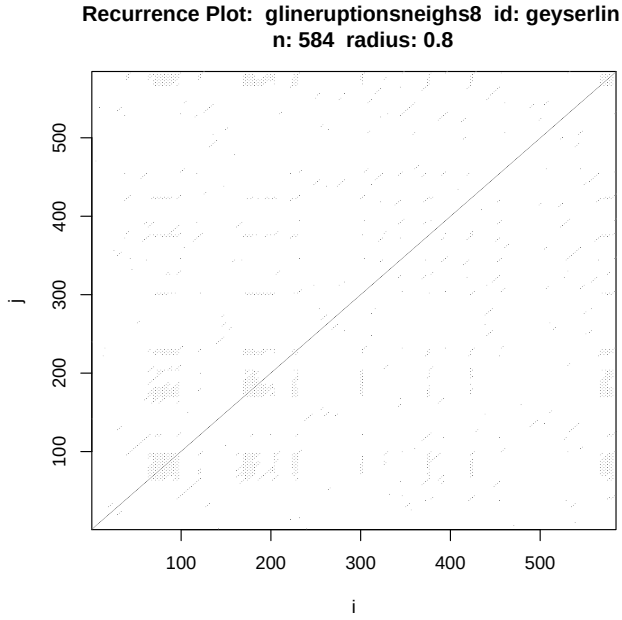


FIGURE 63. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.34 sec.

Only 1024 data points used in recurrence plots in this section

8. Case Study: HRV data example.beats

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

Günther Sawitzki
StatLab Heidelberg
Im Neuenheimer Feld 294
D 69120 Heidelberg