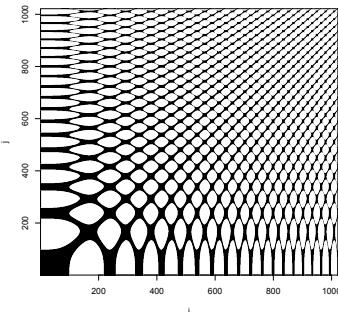


# STATISTICAL DATA ANALYSIS: RECURRENCE PLOT

GÜNTHER SAWITZKI



## CONTENTS

1. Background	2
1.1. Recurrence Plots	3
1.2. Recurrence Quantification Analysis	3
2. R Setup	3
2.1. Local Bottleneck	6
2.2. Recurrence Plots	7
3. Test Cases	9
3.1. Doppler signal	12
4. Takens' Recurrence States	13
5. Applied Recurrence Plots	24
5.1. Sinus	24
5.2. Uniform random	27
5.3. Chirp Signal	28
5.4. Doppler	30
6. Case Study: Geyser data	33
6.1. Geyser Eruptions	33
6.2. Geyser Eruptions: Comparison by Dimension	43
6.3. Geyser Waiting	43
6.4. Geyser - linearized	46
6.5. Geyser Eruptions linearized	47
7. Case Study: HRV data example.beats	54
7.1. RHRV: example.beats - Comparison by Dimension	56
7.2. RHRV: example.beats - Hart Rate Variation	61
7.3. RHRV: example.beats - Variation: Comparison by Dimension	62
8. Case Study: HRV data example2.beats	68
8.1. RHRV: example2.beats - Comparison by Dimension	70
8.2. RHRV: example2.beats - Hart Rate Variation	75
8.3. RHRV: example2.beats - Variation: Comparison by Dimension	76
References	82

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 13, 2014 from svn/cvs Revision : 159

gs@statlab.uni-heidelberg.de .

## 1. BACKGROUND

1.0.1. *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.

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

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

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

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

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

**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 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 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 `statecolplots()`

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.

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

The Takens states may be stationary, that is asymptotically, that is 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 8 dimensions, a limit suggested by the print area.

**ToDo:** consider dimension-adjusted radius

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

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

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

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

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

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

**1.2. 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()
```

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.

```
# save.RNGseed
save.RNGkind
```

---

Output

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

---

Input

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

---

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)
```

We use

---

Input

```
library(nonlineartseries)
```

**ToDo:** improve  
choice of alpha

2.0.1. *Takens states*.

**statepairs***Show marginal scatterplots of Takens states**Usage.*

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

*Arguments.*

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

**ToDo:** colour by time

```
Input
statepairs <- function(states, main, rank=FALSE, nooverlap= FALSE, 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 (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 **stateplot()** is layed out to show the highest index as response.

**ToDo:** extend for low dimensions, extend parameters

```
Input
stateplot <- function(states, main, rank=FALSE, nooverlap= FALSE, 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 (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")}

coplot((states[,4]^states[,3]|states[,1]+ states[,2]), 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,
                                Input
                                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, "id") <- id
  return(takens)
}

```

---



---

```

local.findAllNeighbours <- nonlinearTseries:::findAllNeighbours
Input

```

---



---

```

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

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

  ntakens=length(neigs)

```

minor cosmetics  
added to recurrence-  
PlotAux  
**ToDo:** propagate  
parameters from  
`buildTakens` and  
`findAllNeighbours`  
in a slot of the result,  
instead of using explicit parameters in  
`recurrencePlotAux`.

```

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

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

more <- NULL

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

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

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

```

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

**ToDo:** improve to a full *show* method

```
showrqa <- function(takens, dim=NULL, radius,
                      digits=3,
                      do.hist = TRUE, log = "")  
{  
  xxrqa <- rqa(takens=takens, radius=radius)  
  cat(paste(deparse(substitute(takens)), " n=", dim(takens)[1],  
            " Dim:", dim(takens)[2], "\n"))  
  cat(paste("Radius:", radius,  
            " Recurrence coverage REC:", round(xxrqa[1]$REC, digits),  
            " log(REC)/log(R):", round(log(xxrqa[1]$REC)/log(radius), digits), "\n"))  
  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 zero counts
  barplot(xxrqa$diagonalHistogram,
    main=paste(deparse(substitute(takens)), "Diagonal",
      "\n n=", dim(takens)[1], " Dim:",
      dim(takens)[2], " Radius: ", radius),
    xlab="length of diagonals",
    log = log)

  xxrqa$diagonalHistogram[xxrqa$diagonalHistogram==0] <- NA # hack for zero counts
  barplot(xxrqa$recurrenceRate,
    main=paste(deparse(substitute(takens)), "Recurrence Rate",
      "\n n=", dim(takens)[1], " Dim:",
      dim(takens)[2], " Radius: ", radius),
    xlab="distance to diagonal",
    log = log)
  par(oldpar)
}
invisible(xxrqa)
}

```

### 3. TEST CASES

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 33) and two examples of heart rate data sets ( Section 7 on page 54 and Section 8 on page 68) 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)
}
```

---

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

See Figure 1 on the following page.

---

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

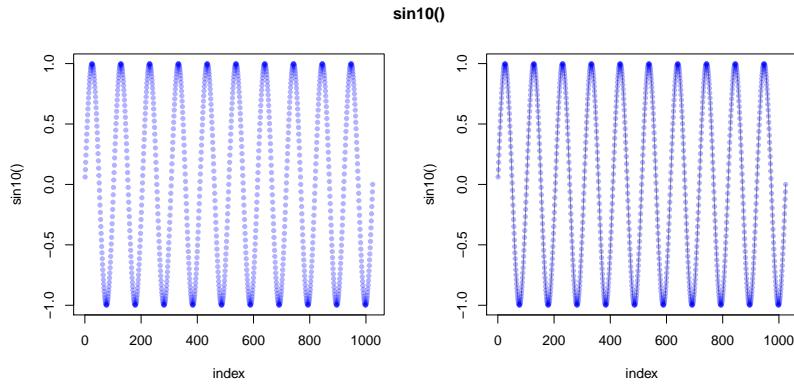


FIGURE 1. Test case: sin10. Signal and linear interpolation.

See Figure 2,

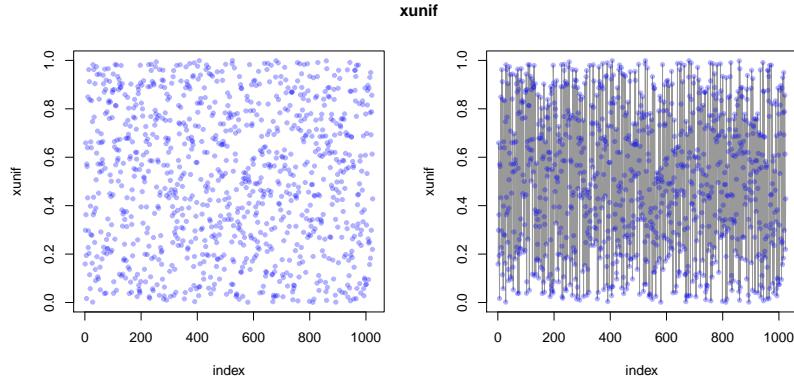


FIGURE 2. Test case: unif - uniform random numbers. Signal and linear interpolation.

---

*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 <- pi*(f1/f0)^2/(t1-t0)^2
        b <- 2*pi*f0
        cos(a*t^(t1-t0) + 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,

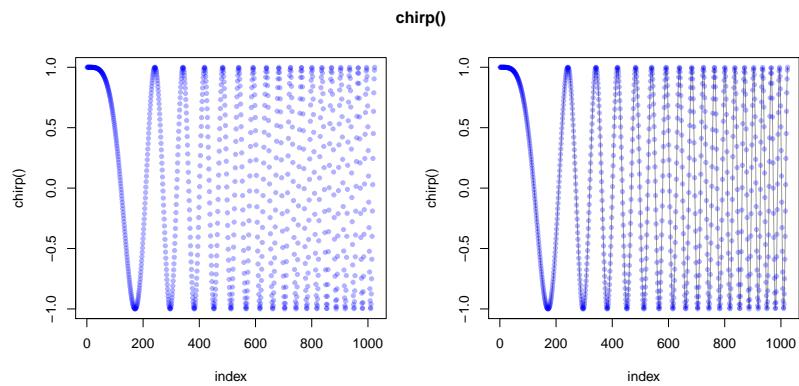


FIGURE 3. Test case: chirp signal. Signal and linear interpolation.

### 3.1. Doppler signal.

---

*Input*

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

---

*Input*

```
plotsignal(doppler())
```

---

See Figure 4,

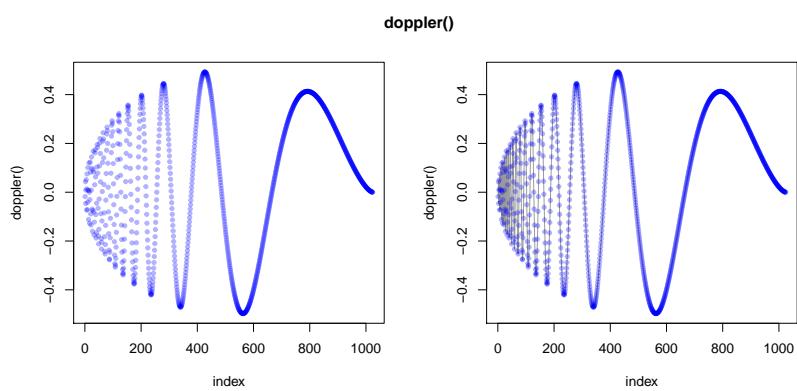


FIGURE 4. Test case: Doppler signal. Signal and linear interpolation.

## 4. TAKENS' RECURRENCE STATES

---

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

---

See Figure 5.

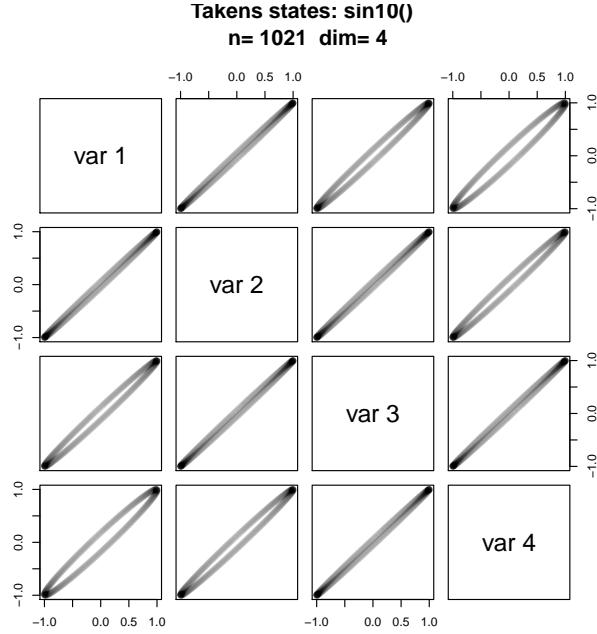


FIGURE 5. Takens states. Test case: sinus. Note that  $2d$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.614 sec.

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 following page.

---

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

---

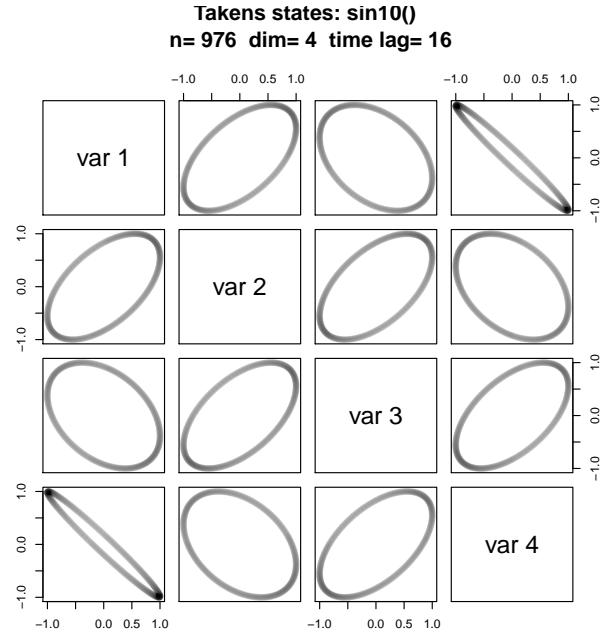


FIGURE 6. Takens states. Test case: sinus at time lag 16. Note that  $2d$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.629 sec.

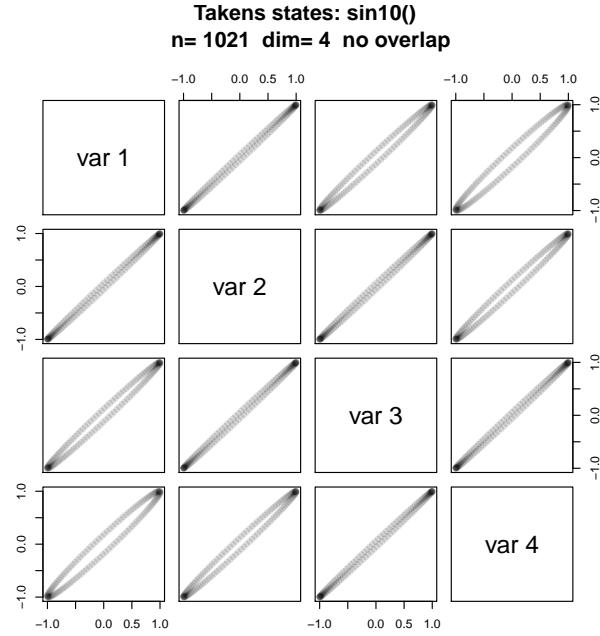


FIGURE 7. Takens states. Test case: sinus, no overlap. Note that  $2d$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.887 sec.

See Figure 8.

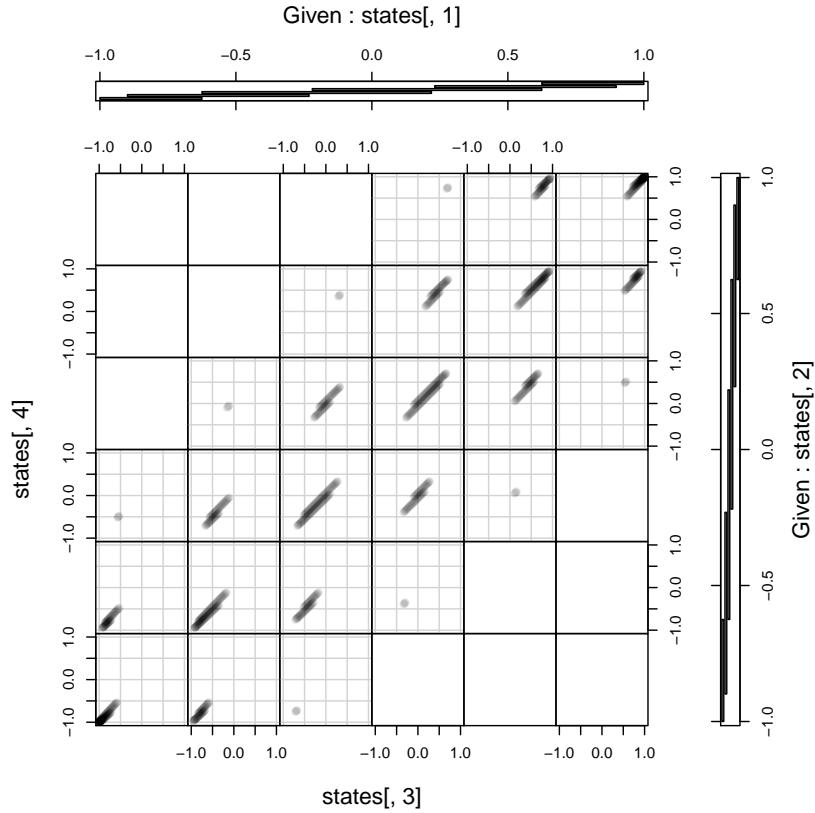


FIGURE 8. State coplot. Test case: sinus. Note that  $2d$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.273 sec.

---

*Input*

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

---

See Figure 9 on the following page.

---

*Input*

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

---

See Figure 10 on the next page.

---

*Input*

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

---

See Figure 11 on page 17.

---

*Input*

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

---

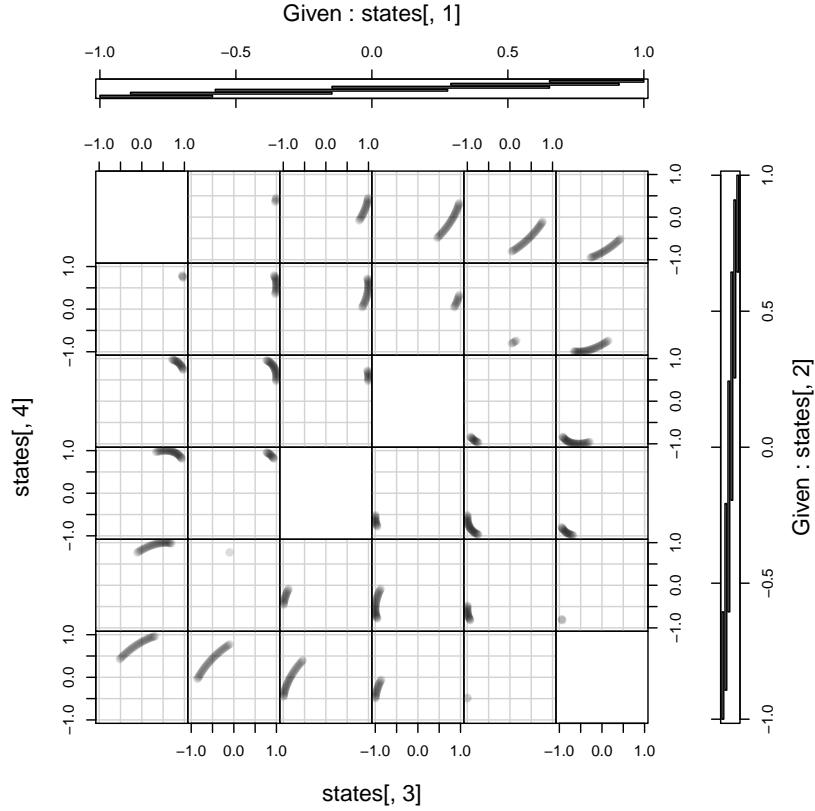


FIGURE 9. State coplot. Test case: sinus, time lag 16. Note that  $2d$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.26 sec.

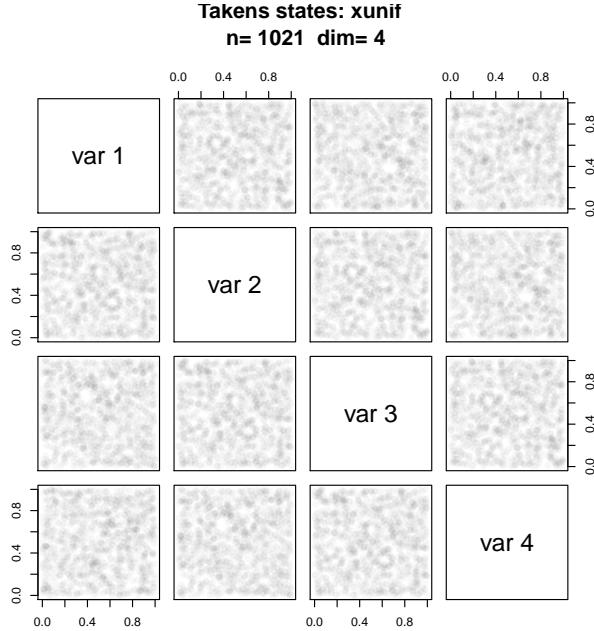


FIGURE 10. Takens states. Test case: uniform random numbers. Time used: 0.682 sec.

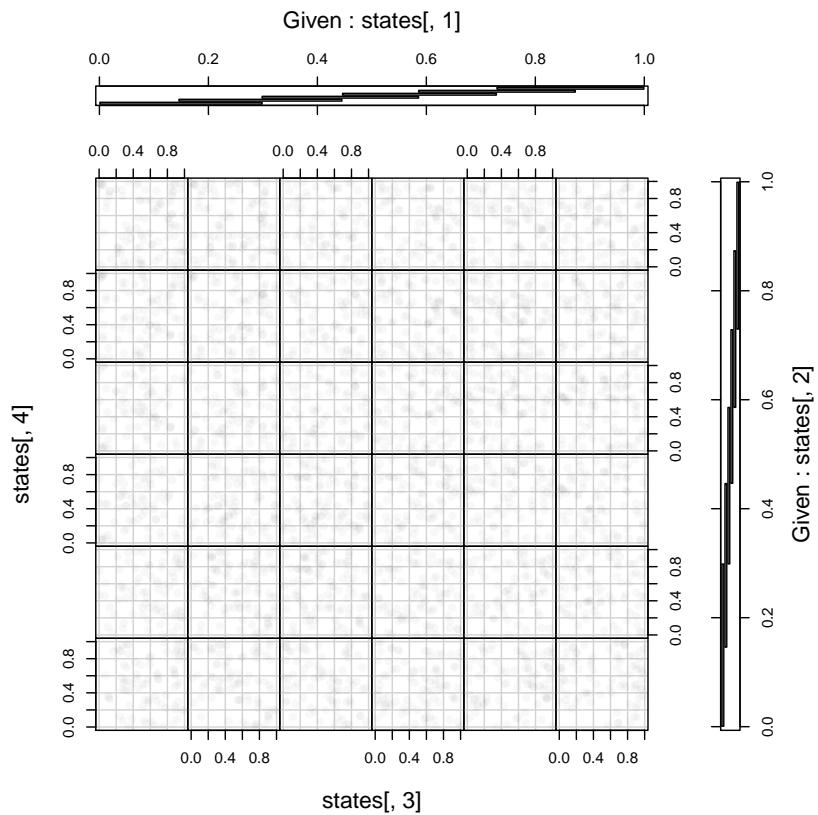


FIGURE 11. State coplot. Test case: uniform random numbers. Time used: 1.076 sec.

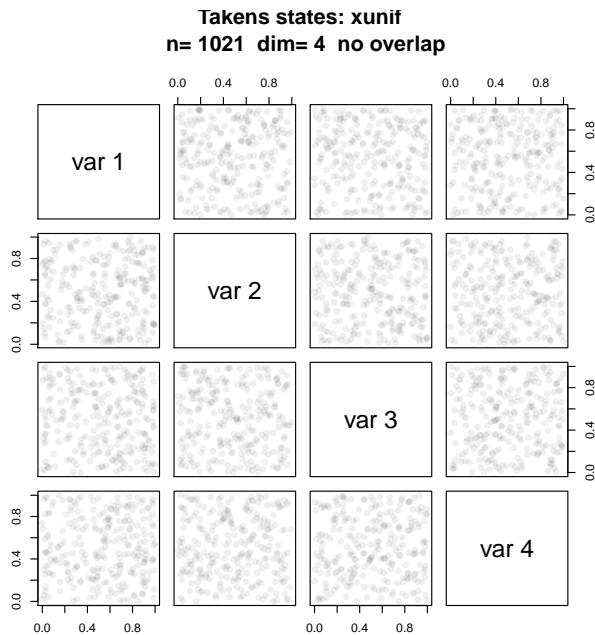


FIGURE 12. Takens states. Test case: uniform random numbers. Time used: 1.343 sec.

See Figure 12 on the previous page.

---

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

---

See Figure 13.

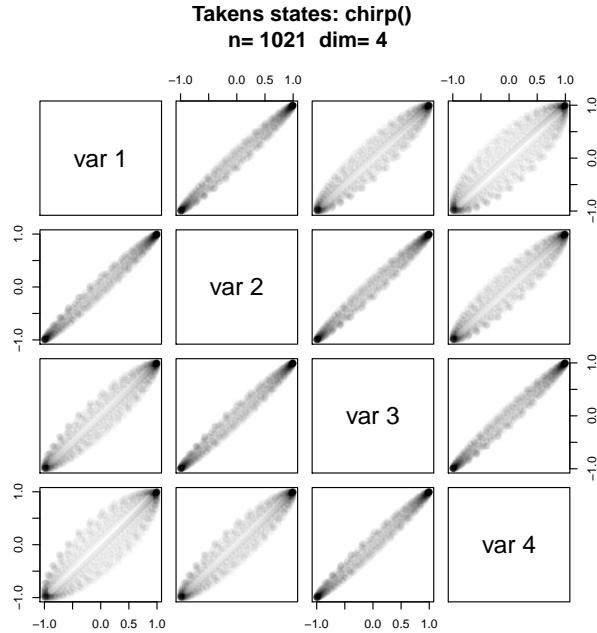


FIGURE 13. Takens states. Test case: chirp signal. Time used: 0.701 sec.

---

```
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 20.

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.

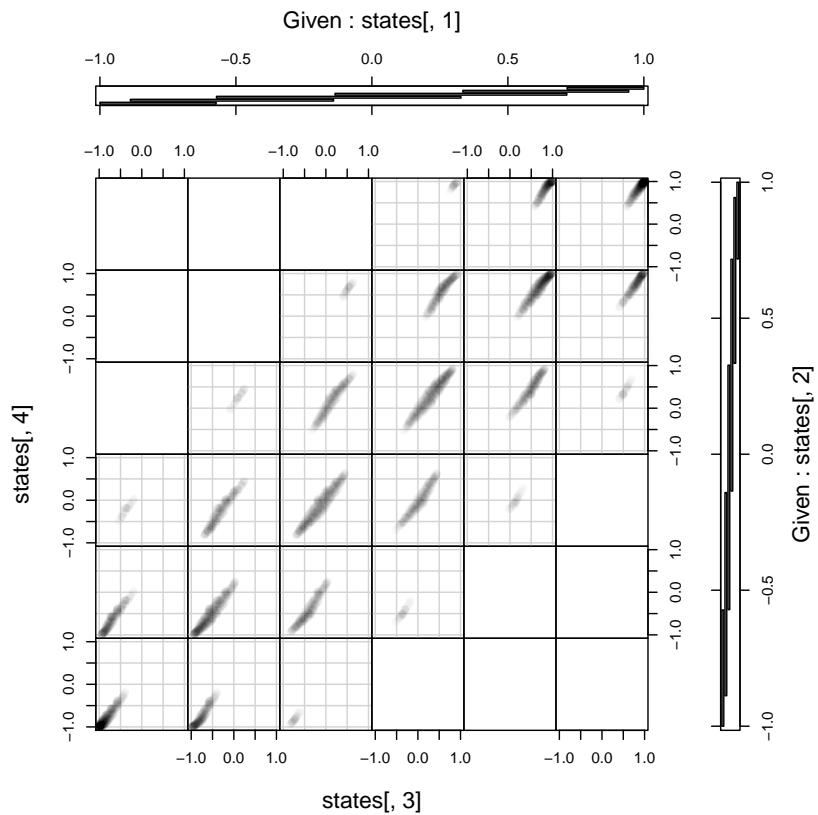


FIGURE 14. State coplot. Test case: chirp signal. Time used: 1.004 sec.

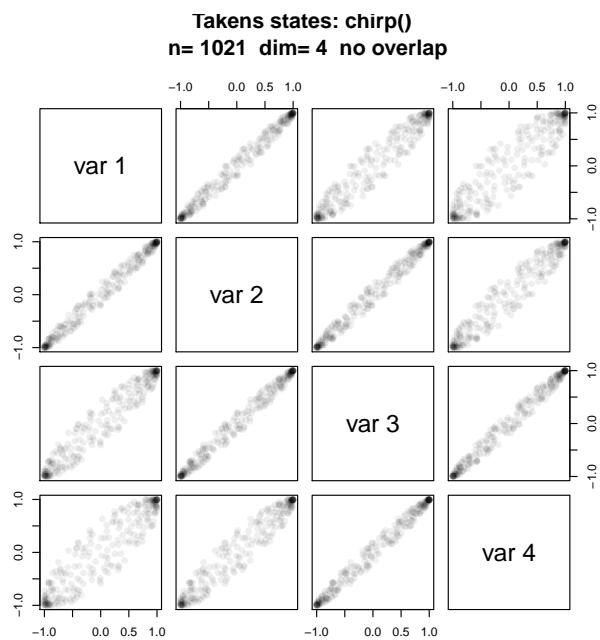


FIGURE 15. Takens states. Test case: chirp signal. Time used: 1.246 sec-

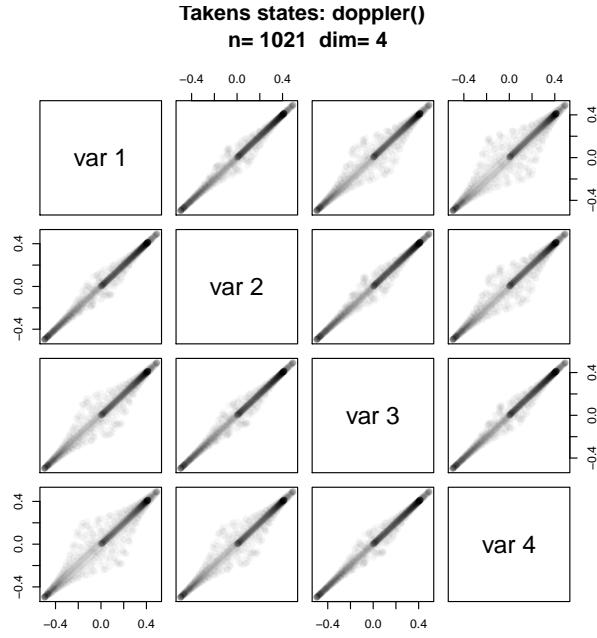


FIGURE 16. Takens states. Test case: Doppler. Note that  $2d$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.67 sec.

---

*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*

```
statecplot(dopplertakens) #4
```

See Figure 19 on page 22.

---

*Input*

```
statecplot(dopplertakenslag16) #4
```

See Figure 20 on page 23.

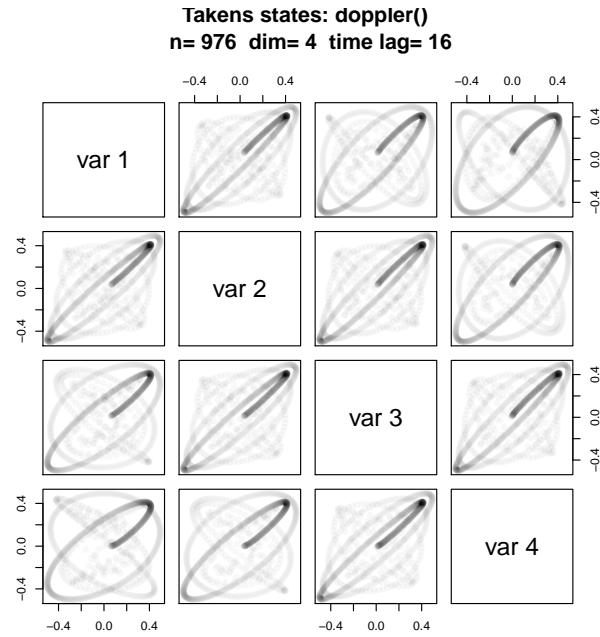


FIGURE 17. Takens states. Test case: doppler at time lag 16. Note that 2d marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.631 sec.

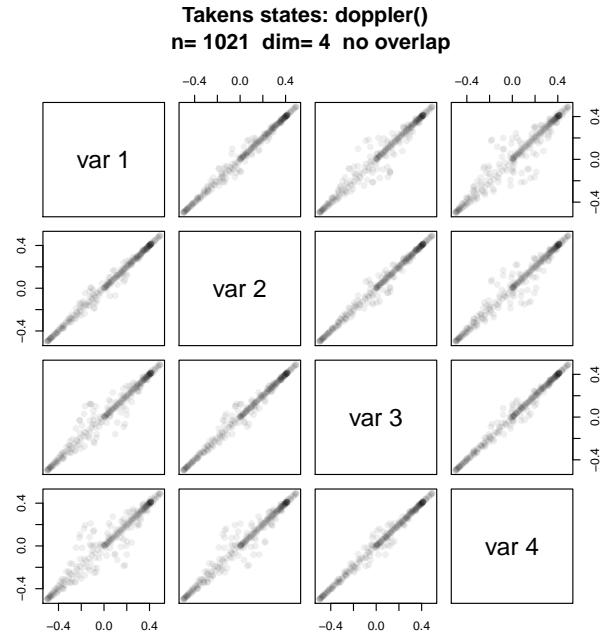


FIGURE 18. Takens states. Test case: doppler, no overlap. Note that 2d marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.939 sec.

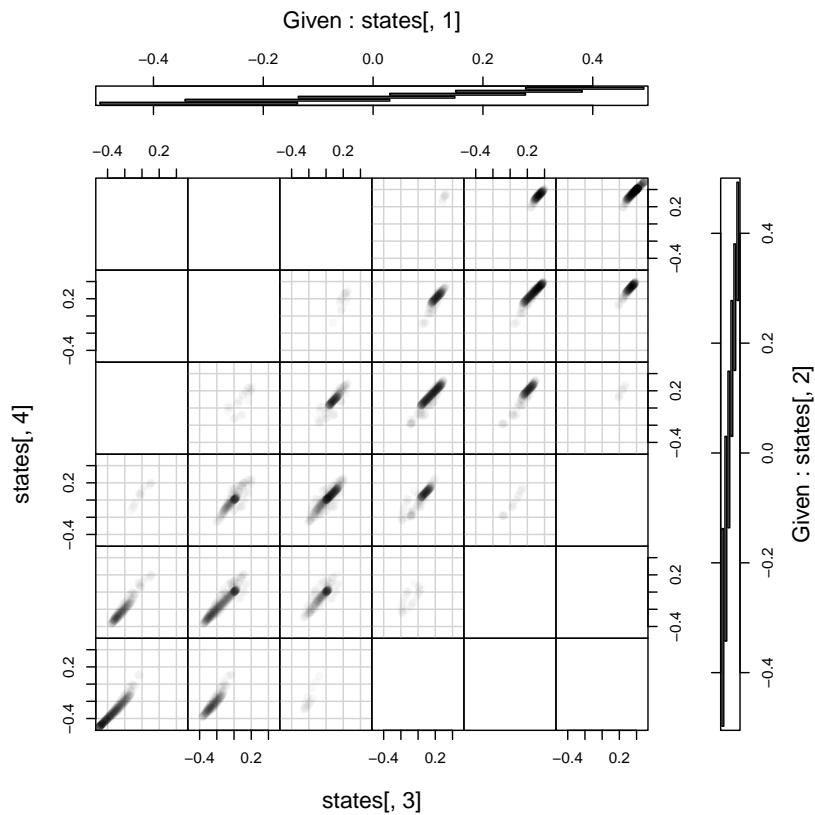


FIGURE 19. State coplot. Test case: Doppler. Note that  $2d$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.283 sec.

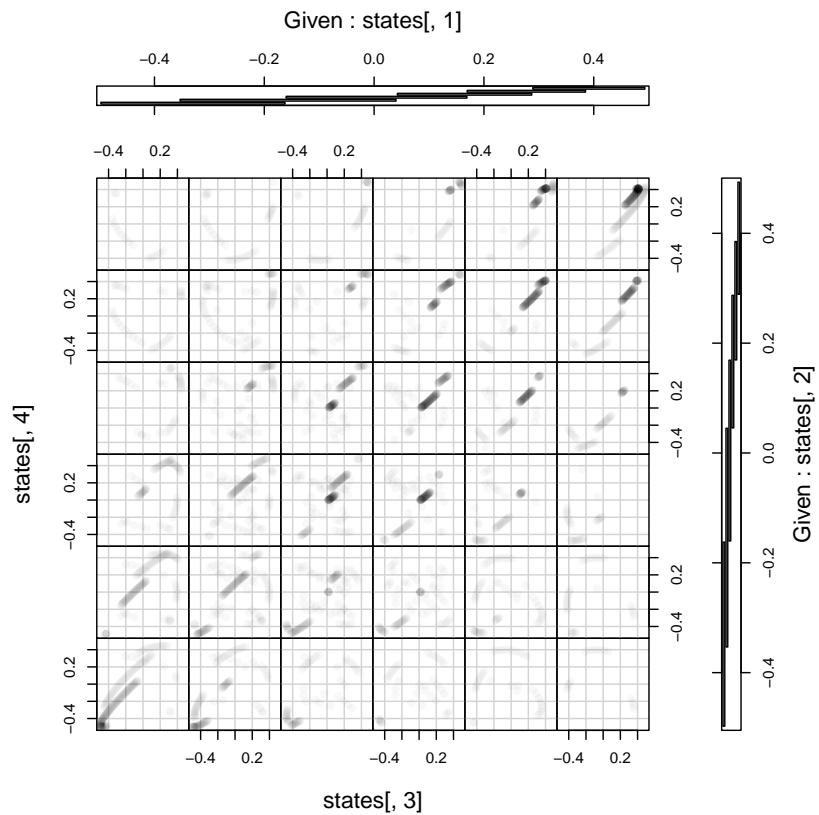


FIGURE 20. State coplot. Test case: Doppler, time lag 16. Note that  $2d$  marginal views of 1-dimensional circles in  $d$  space generally appear as ellipses. Time used: 0.369 sec.

## 5. APPLIED RECURRENCE PLOTS

### 5.1. Sinus.

---

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

---



---

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

---

See Figure 21.

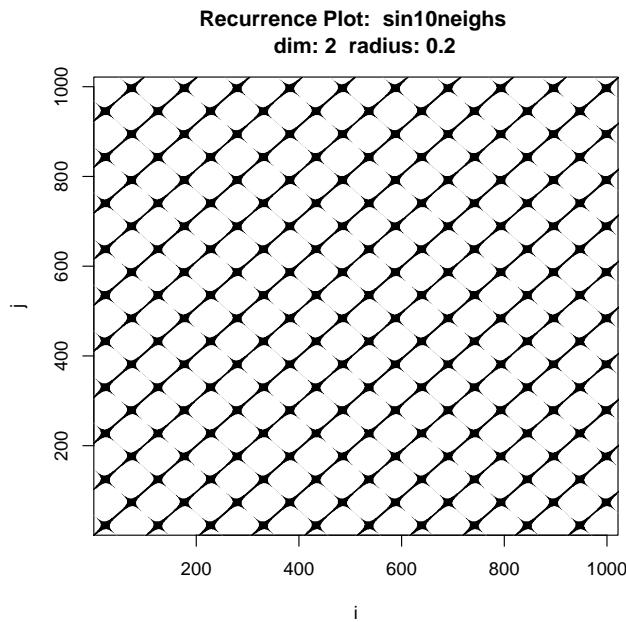


FIGURE 21. Recurrence plot. Recurrence Plot. Test case: sinus curves.  
Time used: 2.254 sec.

---

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

---

```
Output
sintakens n= 1021 Dim: 4
Radius: 0.2 Recurrence coverage REC: 0.136 log(REC)/log(R): 1.241
Determinism: 0.96 Laminarity: 0.982
DIV: 0.001
Trend: 0 Entropy: 2.74
Diagonal lines max: 1020 Mean: 14.263 Mean off main: 14.157
Vertical lines max: 25 Mean: 10.251
```

---

See Figure 22 on the next page.

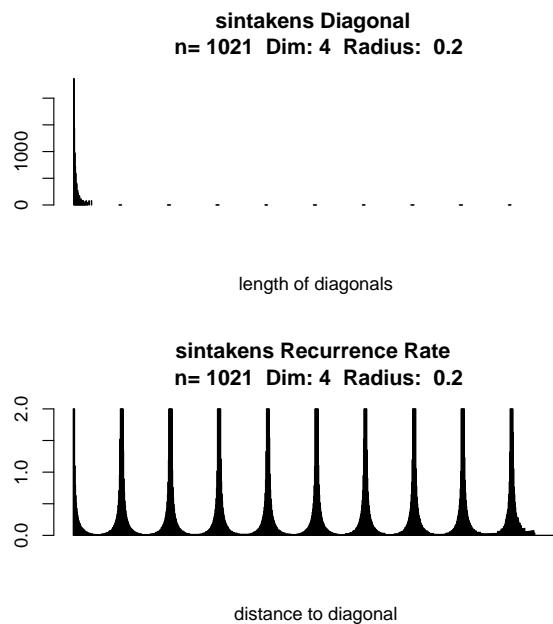


FIGURE 22. Recurrence plot. RQA. Test case: sinus curves. Time used: 0.462 sec.

---

*Input*

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

---



---

*Input*

```
load(file="sin10lag16neighs.RData")
local.recurrencePlotAux(sin10lag16neighs, dim=2, radius=0.2)
```

---

See Figure 23 on the following page.

---

*Input*

```
showrqa(sintakenslag16, radius=0.2)
```

---

*Output*

```
sintakenslag16 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 on the next page.

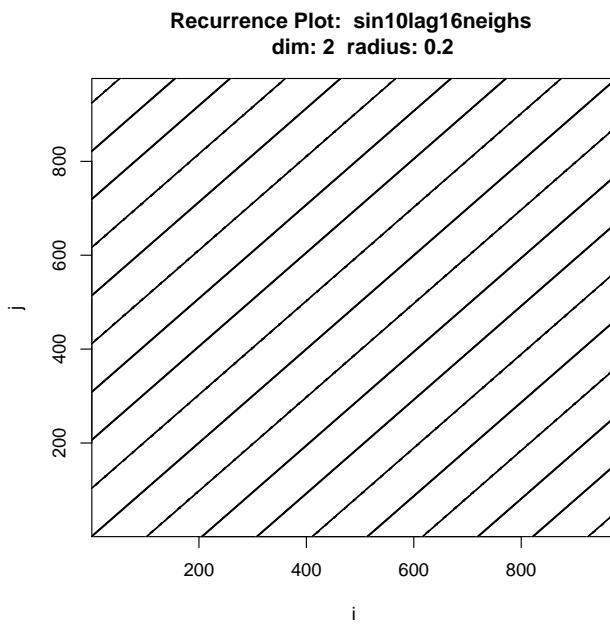


FIGURE 23. Recurrence plot. Recurrence Plot. Test case: sinus curves, time lag 16. Time used: 0.884 sec.

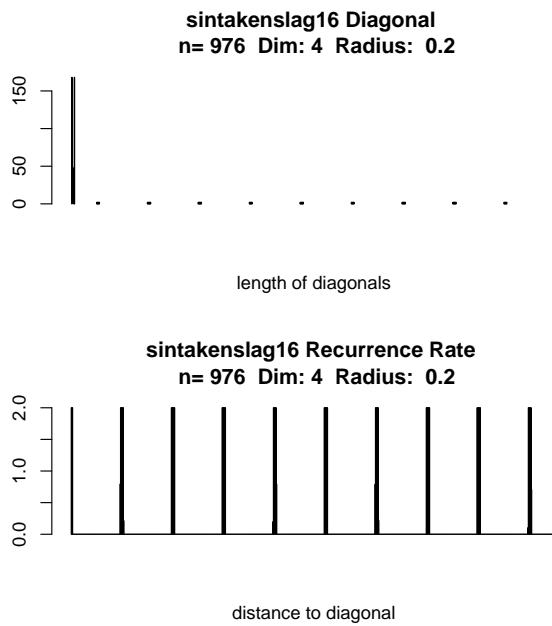


FIGURE 24. Recurrence plot. RQA. Test case: sinus curves. Time used: 0.31 sec.

### 5.2. Uniform random.

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

See Figure 25.

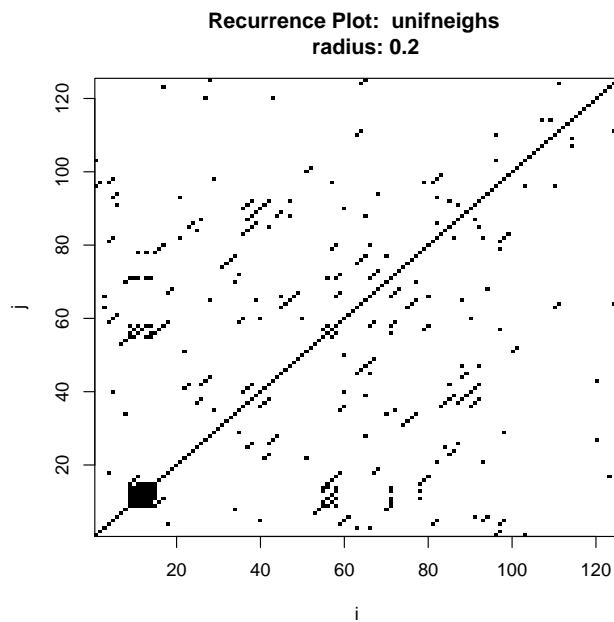


FIGURE 25. Recurrence plot. Recurrence Plot. Test case: uniform random numbers. Time used: 0.086 sec.

### 5.3. Chirp Signal.

---

*Input*

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

---



---

*Input*

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

---

See Figure 26.

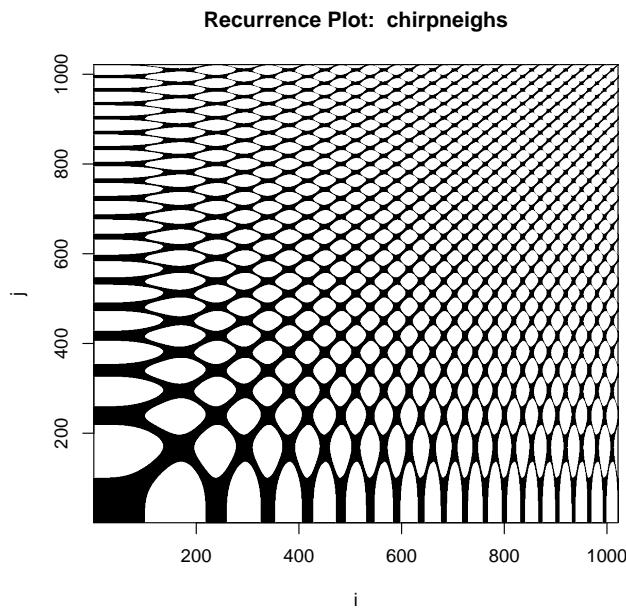


FIGURE 26. Recurrence plot. Recurrence Plot. Test case: chirp signal.  
Time used: 2.038 sec.

---

*Input*

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

---

*Output*

```
chirptakens 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 27 on the facing page.

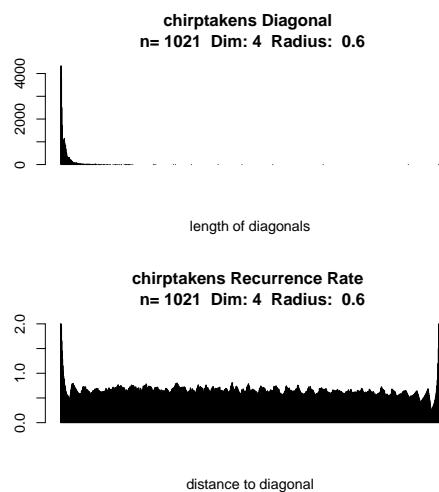


FIGURE 27. Recurrence Plot RQA. Test case: chirp signal. Time used: 2.663 sec.

#### 5.4. Doppler.

---

*Input*

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

---



---

*Input*

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

---

See Figure 28.

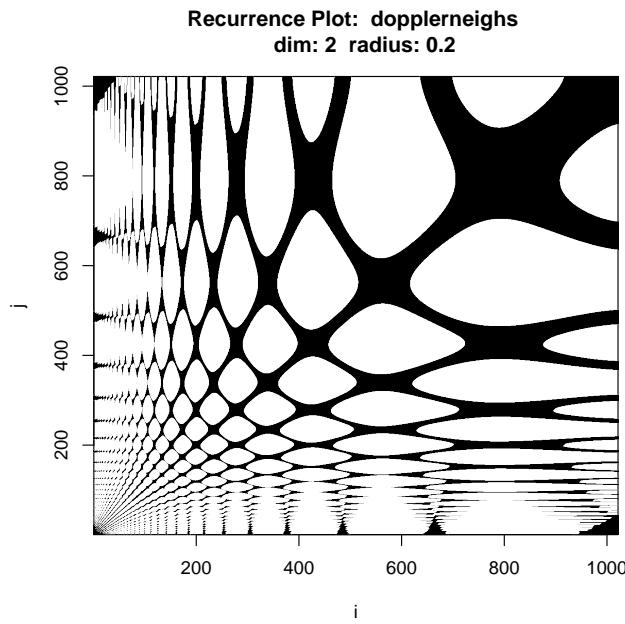


FIGURE 28. Recurrence plot. Recurrence Plot. Test case: Doppler.  
Time used: 1.627 sec.

---

*Input*

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

---



---

*Output*

```
dopplertakens 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 29 on the facing page.

<print=FALSE, echo=FALSE> chunk.time.start<- proc.time()

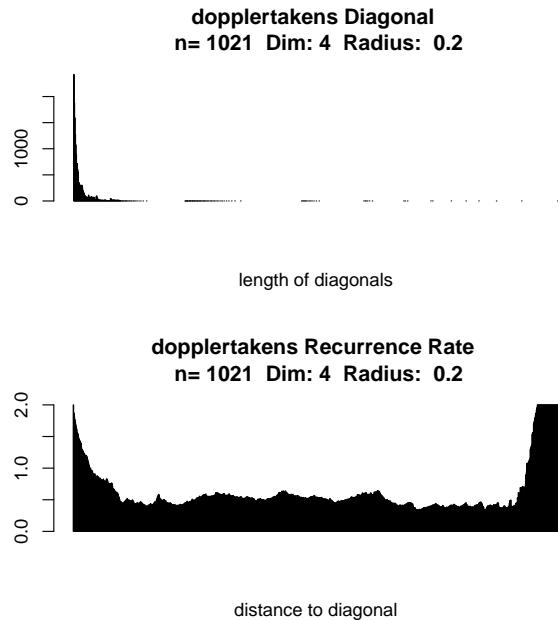


FIGURE 29. Recurrence plot. RQA. Test case: Doppler. Time used: 0.573 sec.

---

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

---

*load(file="dopplerlag16neighs.RData")* *Input*  
*local.recurrencePlotAux(dopplerlag16neighs, dim=2, radius=0.2)*

---

See Figure 30 on the next page.

---

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

---

*dopplertakenslag16 n= 976 Dim: 4* *Output*  
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 31 on the following page.

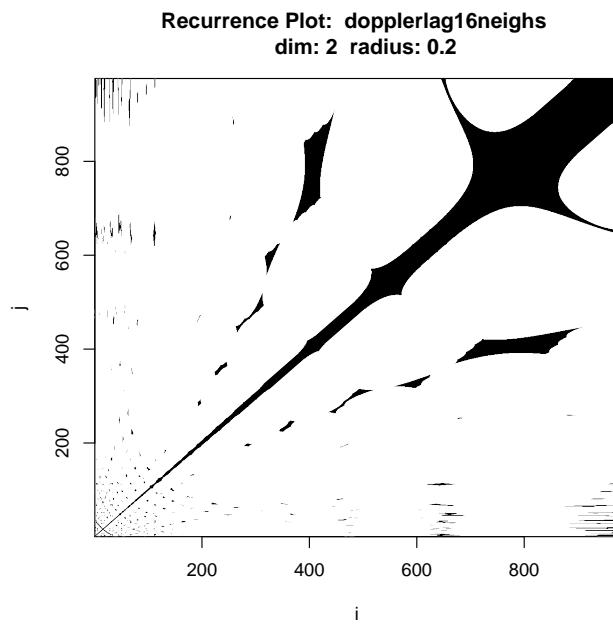


FIGURE 30. Recurrence plot. Recurrence Plot. Test case: Doppler. Time used: 1.066 sec.

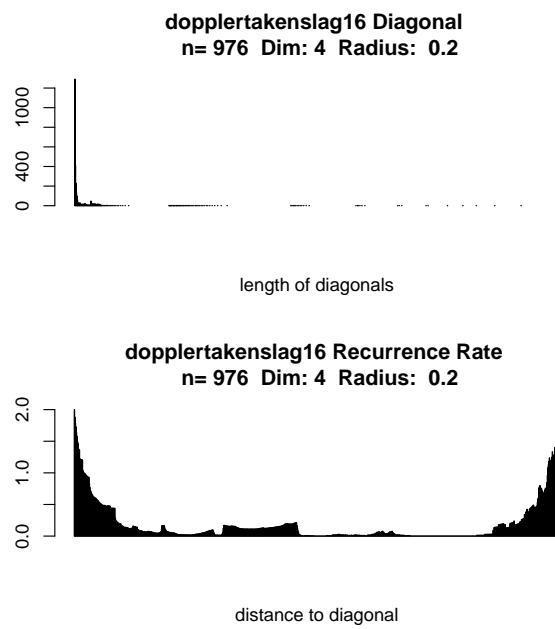


FIGURE 31. Recurrence plot. RQA. Test case: Doppler. Time used: 0.431 sec.

## 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

---

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

Input

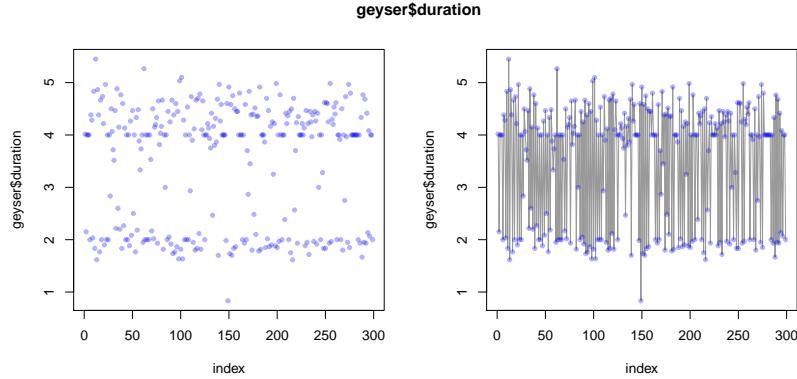


FIGURE 32. Example case: Old Faithful Geyser eruptions. Signal and linear interpolation.

---

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

See Figure 42 on page 39.

---

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

See Figure 34 on the next page.

---

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

See Figure 43 on page 40.

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

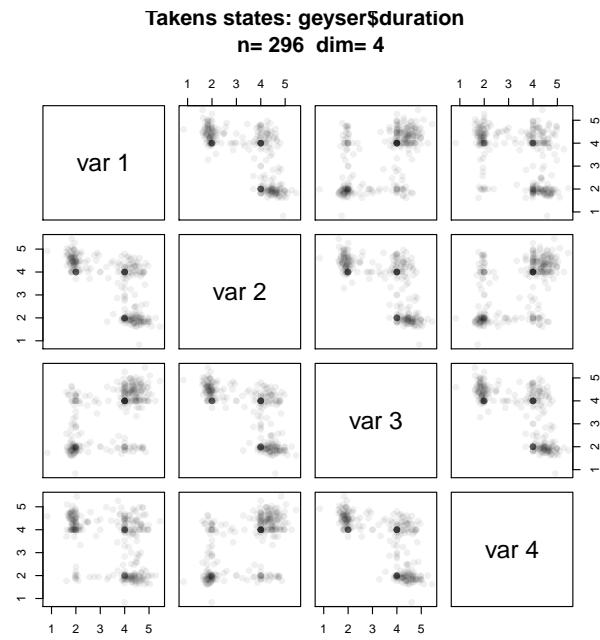


FIGURE 33. Recurrence plot. Example case: Old Faithful Geyser eruptions. Time used: 0.271 sec.

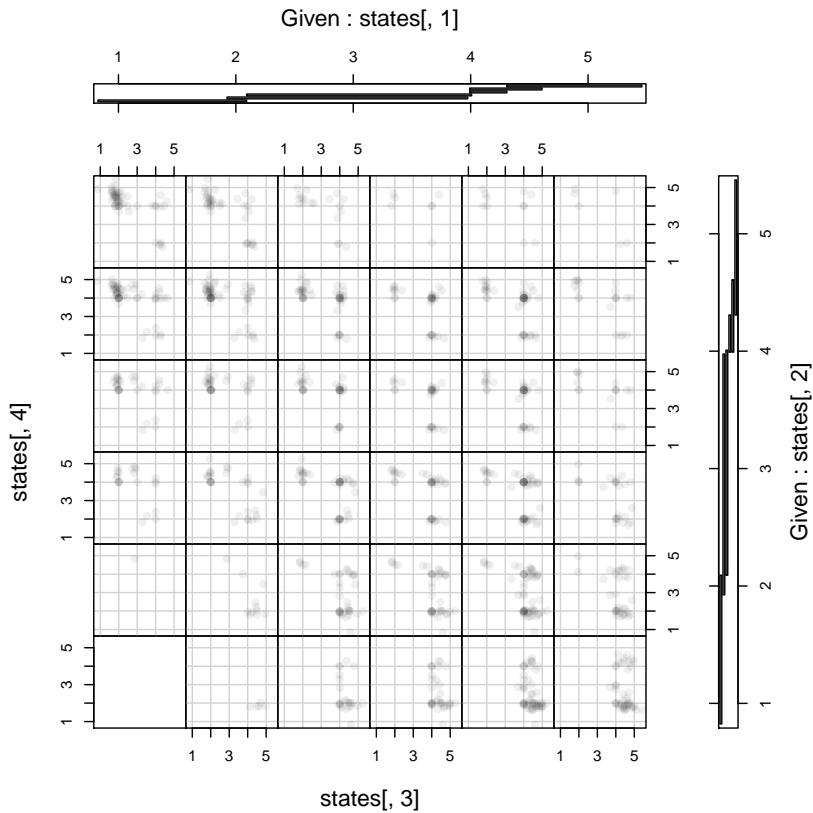


FIGURE 34. State coplot. Example case: Old Faithful Geyser eruptions. Time used: 0.488 sec.

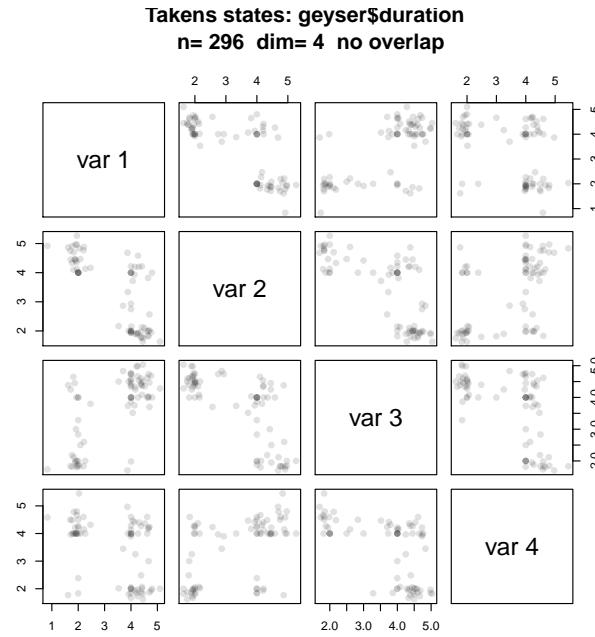


FIGURE 35. Recurrence plot. Example case: Old Faithful Geyser eruptions. Time used: 0.621 sec.

---

Input

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

---



---

Input

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

---

See Figure 44 on page 41.

---

Input

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

---



---

Output

```
eruptionstakens4 n= 296 Dim: 4
Radius: 0.8 Recurrence coverage REC: 0.112 log(REC)/log(R): 9.822
Determinism: 0.903 Laminarity: 0.076
DIV: 0.05
Trend: 0 Entropy: 1.779
Diagonal lines max: 20 Mean: 3.919 Mean off main: 3.79
Vertical lines max: 5 Mean: 3.041
```

---

See Figure 45 on page 41.

### 6.1.1. Geyser eruptions. Dim=2.

---

Input

---

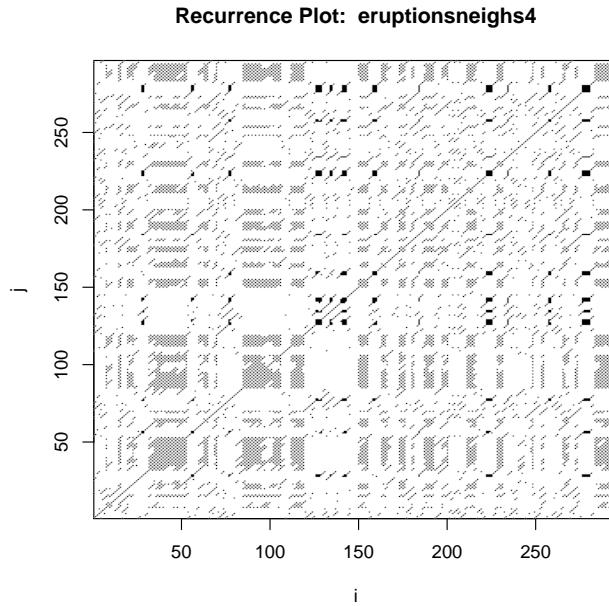


FIGURE 36. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.182 sec.

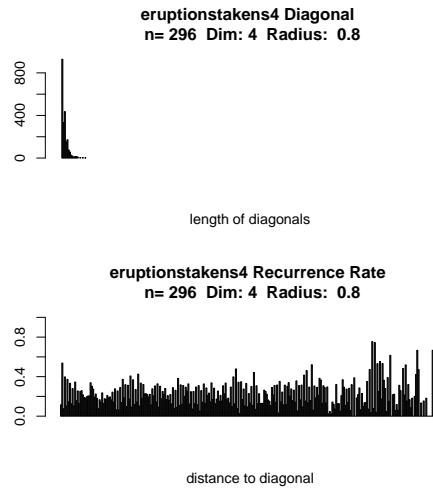


FIGURE 37. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.299 sec.

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

See Figure 38 on the next page.

---

*Input*  
statepairs(eruptionstakens2, nooverlap=TRUE) #dim=2

---

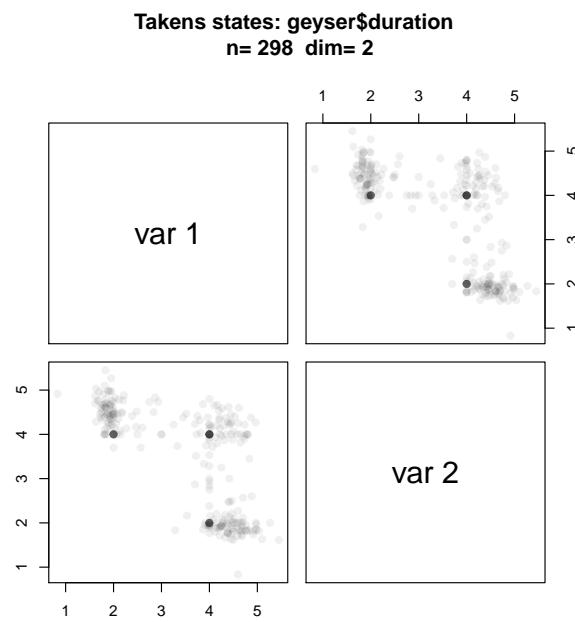


FIGURE 38. Recurrence plot. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.1 sec.

See Figure 39 on the following page

---

Input

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

---



---

Input

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

---

See Figure 40 on the next page.

---

Input

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

---

Output

```
eruptionstakens2 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 41 on page 39.

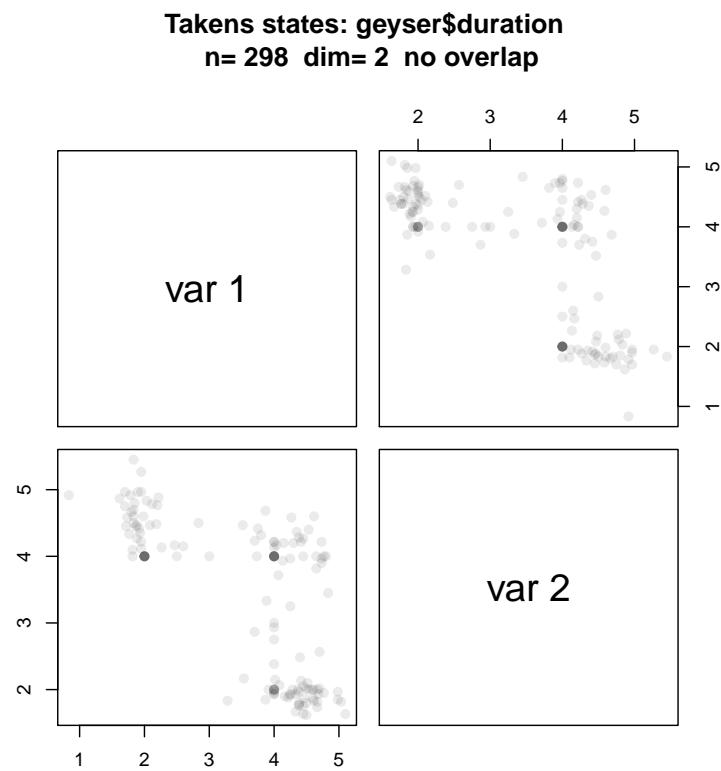


FIGURE 39. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.184 sec.

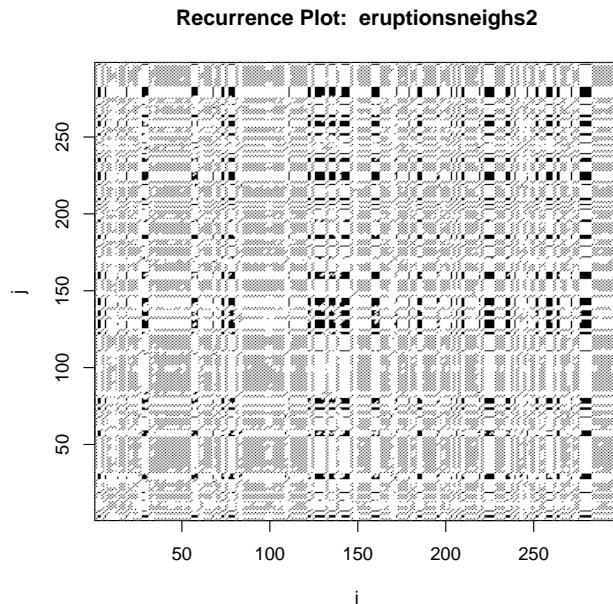


FIGURE 40. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser Geyser eruptions. Dim=2. Time used: 0.206 sec.

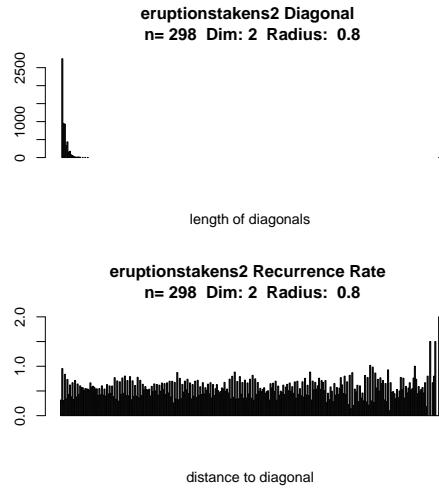


FIGURE 41. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.361 sec.

#### 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 42.

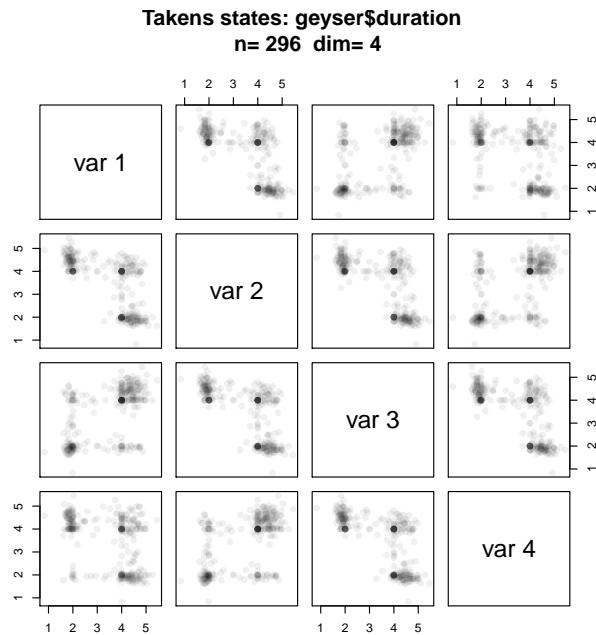


FIGURE 42. Recurrence plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.274 sec.

---

*Input*

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

---

See Figure 43.

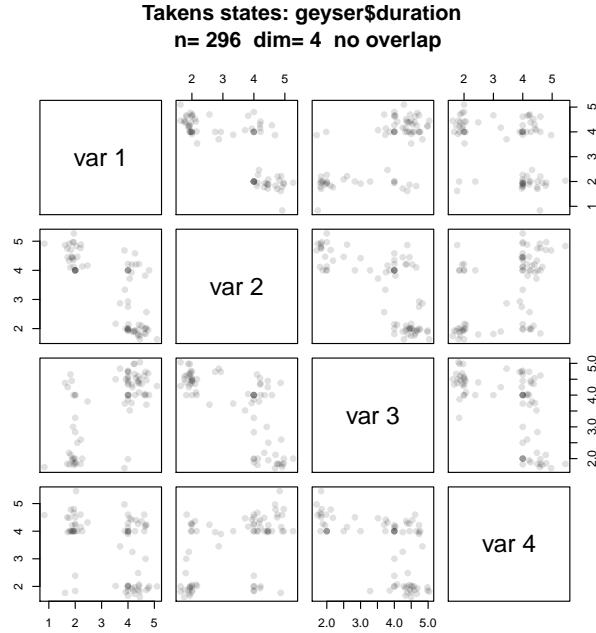


FIGURE 43. Recurrence plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.418 sec.

---

*Input*

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

---



---

*Input*

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

---

See Figure 44 on the next page.

Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.147 sec.

---

*Input*

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

---



---

*Output*

```
eruptionstakens4 n= 296 Dim: 4
Radius: 0.8 Recurrence coverage REC: 0.112 log(REC)/log(R): 9.822
Determinism: 0.903 Laminarity: 0.076
DIV: 0.05
Trend: 0 Entropy: 1.779
Diagonal lines max: 20 Mean: 3.919 Mean off main: 3.79
Vertical lines max: 5 Mean: 3.041
```

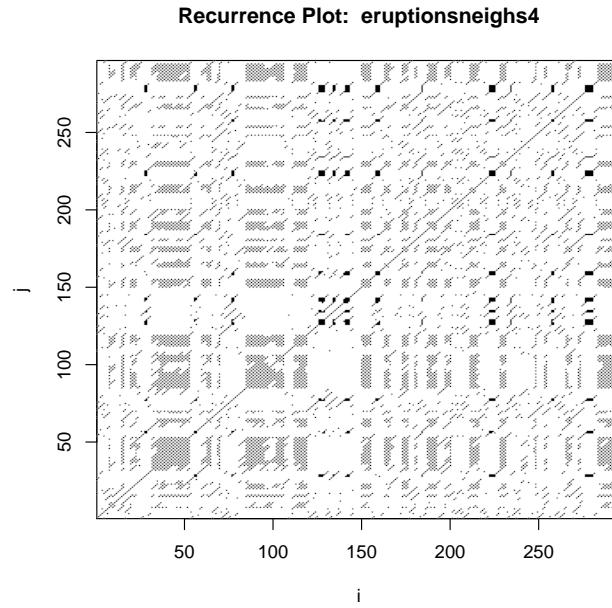


FIGURE 44. Recurrence plot. .

See Figure 45.

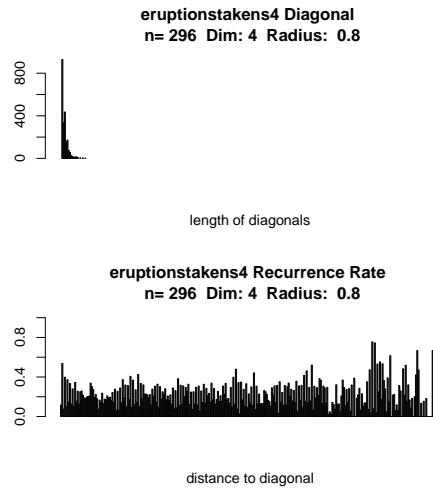


FIGURE 45. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.269 sec.

### 6.1.3. Geyser eruptions. Dim=8.

---

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

---

See Figure 46 on the next page.

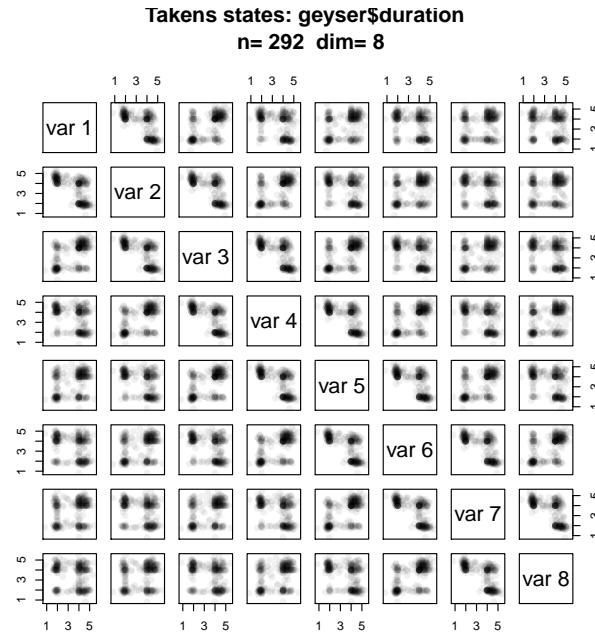


FIGURE 46. Recurrence plot. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.865 sec.

---

*Input*

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

---

See Figure 47 on the facing page.

---

*Input*

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

---



---

*Input*

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

---

See Figure 48 on page 44.

---

*Input*

```
showrqa(eruptionstakens8, radius=0.8)
```

---

*Output*

```
eruptionstakens8 n= 292 Dim: 8
Radius: 0.8 Recurrence coverage REC: 0.024 log(REC)/log(R): 16.799
Determinism: 0.924 Laminarity: 0
DIV: 0.062
Trend: 0 Entropy: 1.8
Diagonal lines max: 16 Mean: 4.583 Mean off main: 3.871
Vertical lines max: 0 Mean: 0
```

---

See Figure 49 on page 44.

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

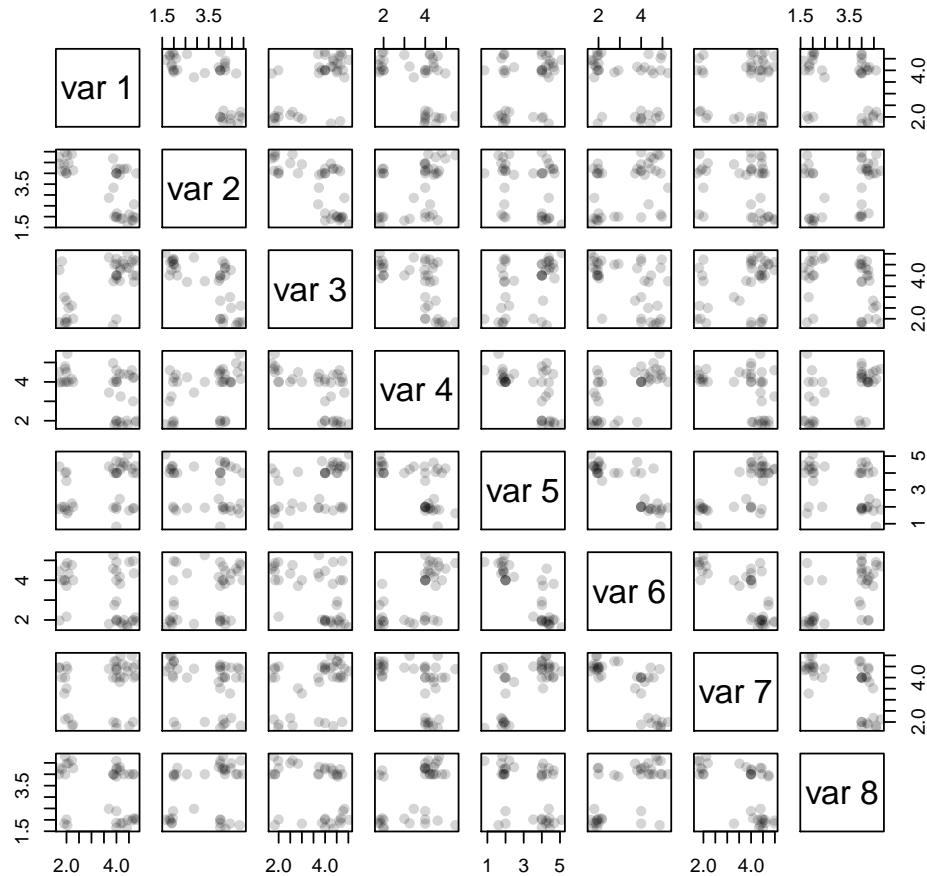


FIGURE 47. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 1.14 sec.

6.2. **Geyser Eruptions: Comparison by Dimension.** For comparison, recurrence plots for the Geyser data with varying dimension are in Figure 50 on page 45

### 6.3. Geyser Waiting.

---

*Input*

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

See Figure 51 on page 45.

---

*Input*

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

See Figure 52 on page 46.

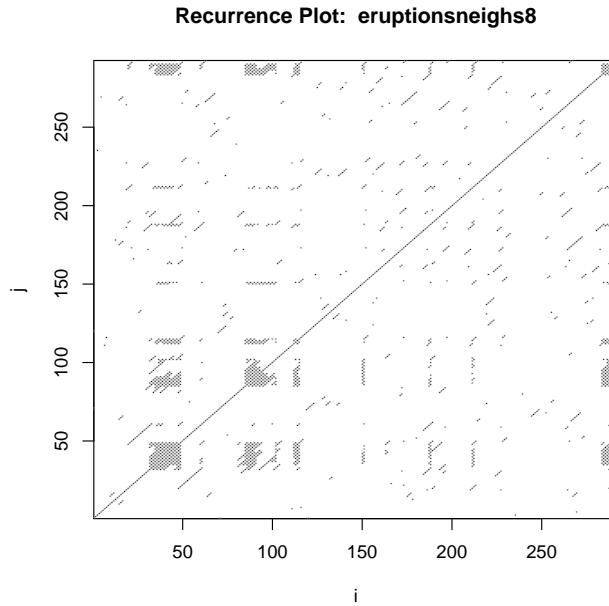


FIGURE 48. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.111 sec.

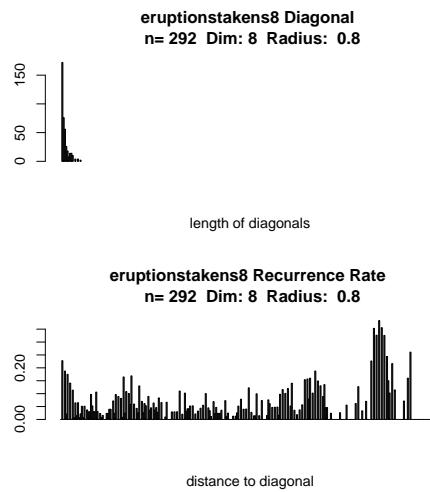


FIGURE 49. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.245 sec.

---

Input

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

---

Input

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

---

Output

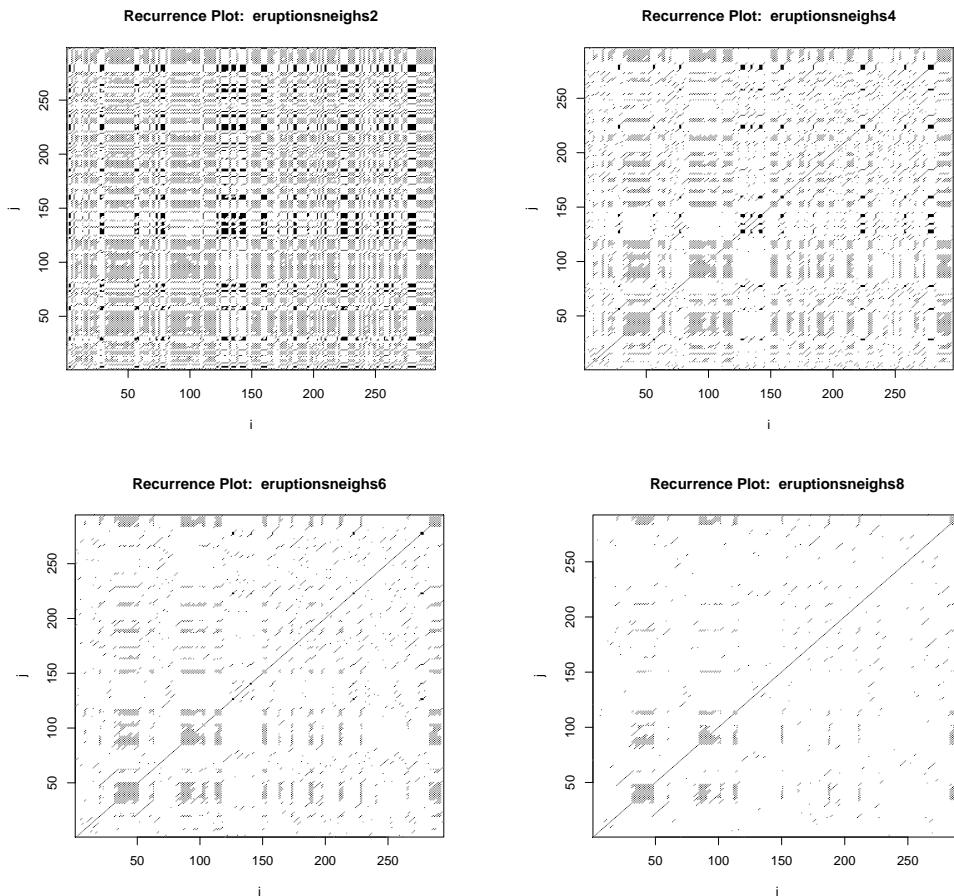


FIGURE 50. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=2, 4, 6, 8.

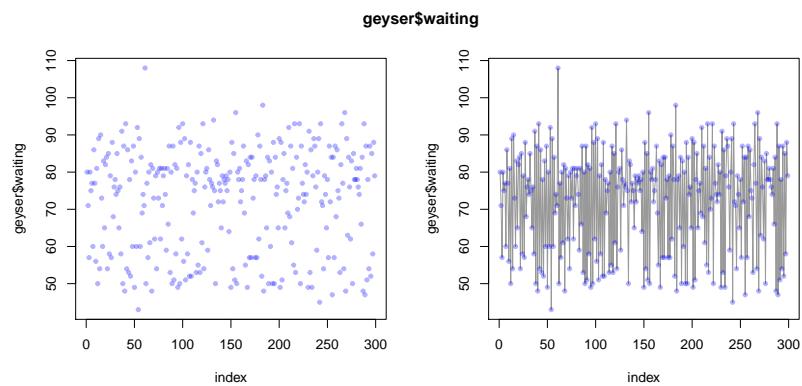


FIGURE 51. Example case: Old Faithful Geyser waiting. Signal and linear interpolation. Time used: 0.432 sec.

```

waitingtakens 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

```

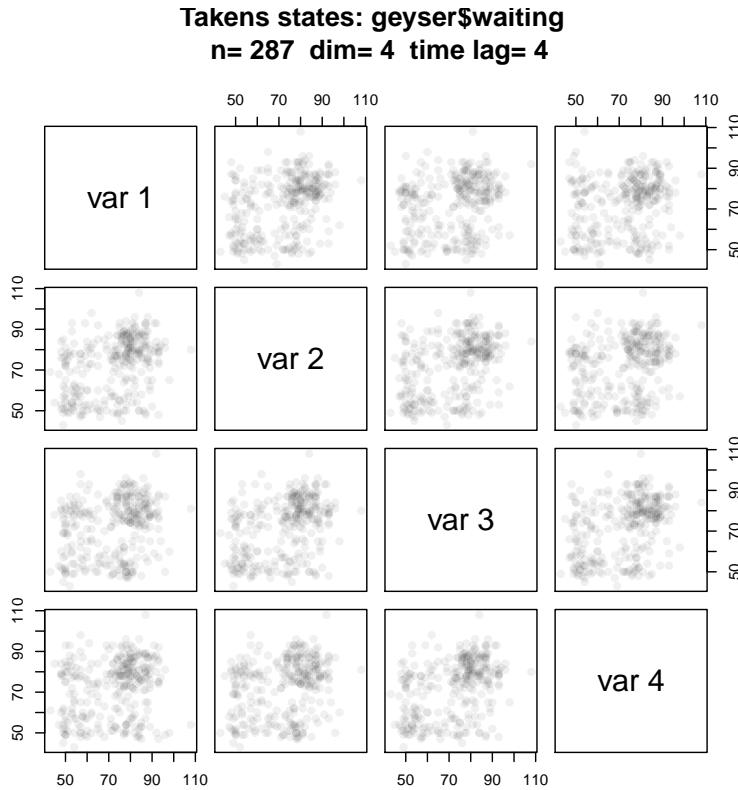


FIGURE 52. Example case: Old Faithful Geyser waiting. Time used: 0.266 sec.

```
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 53 on the facing page.

---

Input

---

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

See Figure 54 on the next page.

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

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

---

Input

---

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

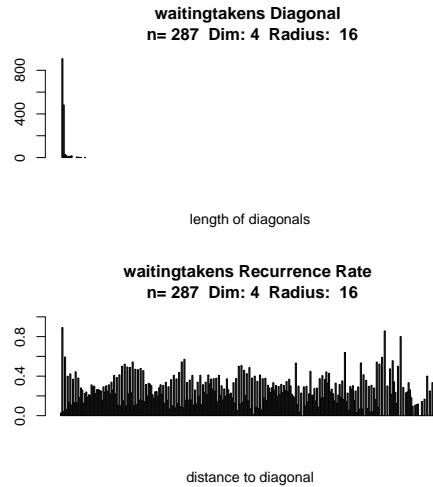


FIGURE 53. Recurrence Plot RQA. Example case: Old Faithful Geyser waiting. Time used: 0.123 sec.

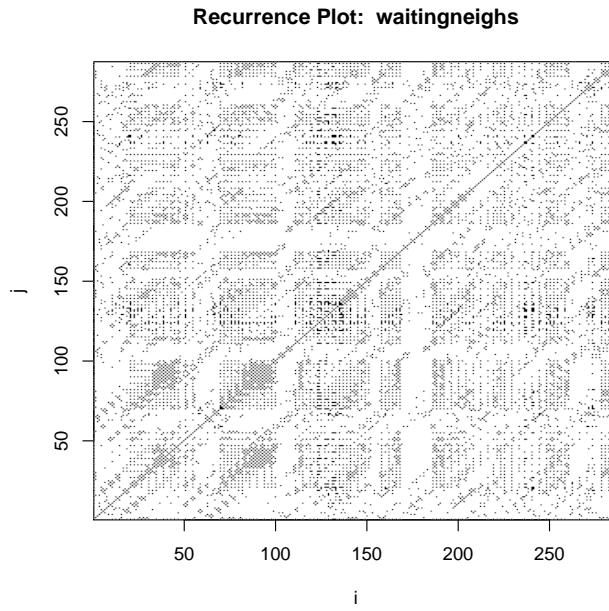


FIGURE 54. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser waiting. Time used: 0.286 sec.

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.

### 6.5. Geyser Eruptions linearized.

---

*Input*

---

```
plotsignal(geyserlin)
```

See Figure 55.

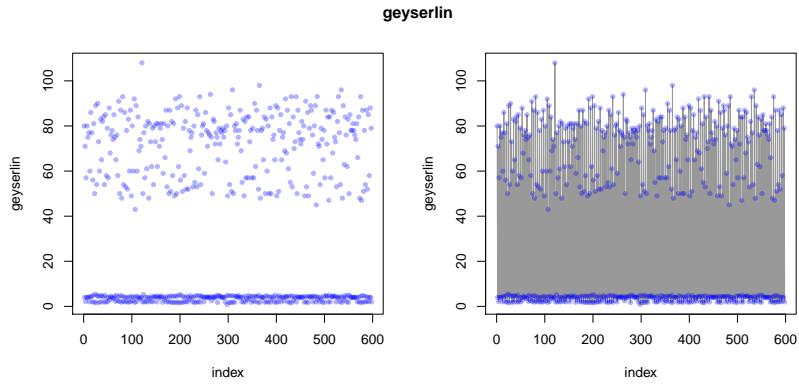


FIGURE 55. Example case: Old Faithful Geyser eruptions. Signal and linear interpolation.

---

Input

```
glineruptionstakens4 <-
  local.buildTakens( time.series=geyserlin,
                     embedding.dim=4, time.lag=2)
statepairs(glineruptionstakens4) #dim=4
```

---

See Figure 56 on the next page.

---

Input

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

---



---

Input

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

---

Output

```
glineruptionstakens4 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 57 on the facing page.

---

Input

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

---

See Figure 58 on page 50.

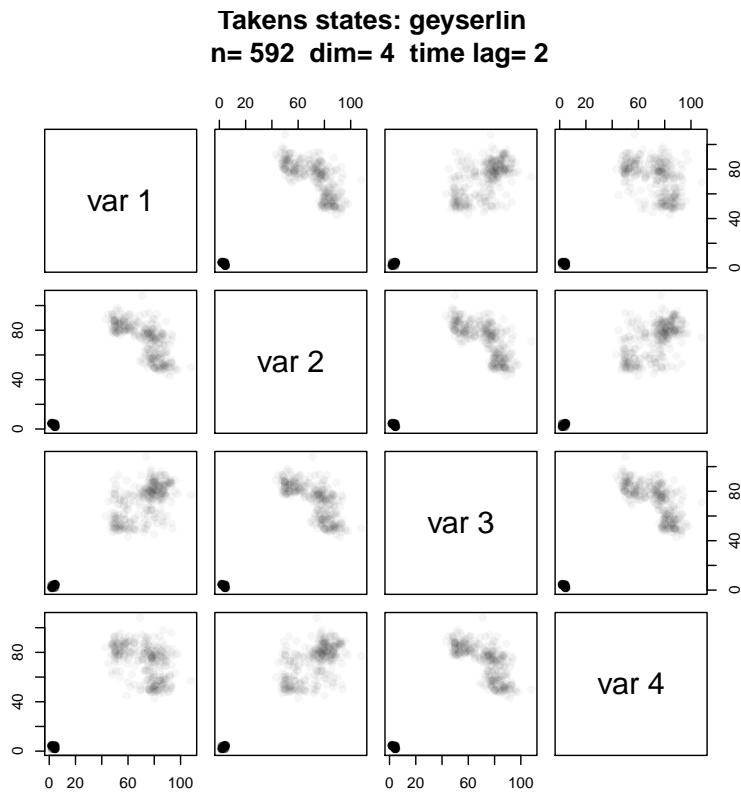


FIGURE 56. Example case: Old Faithful Geyser eruptions. Time used: 0.441 sec.

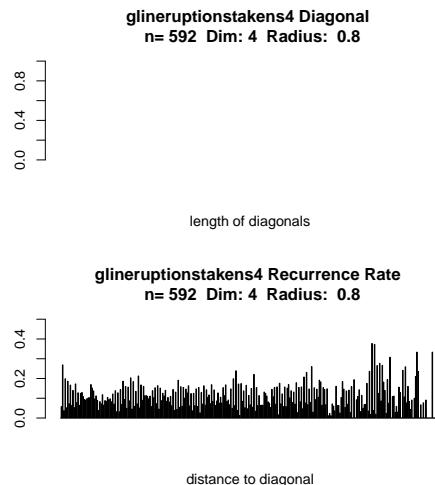


FIGURE 57. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.231 sec.

#### 6.5.1. Geyser eruptions - linearized. Dim=2.

---

Input

---

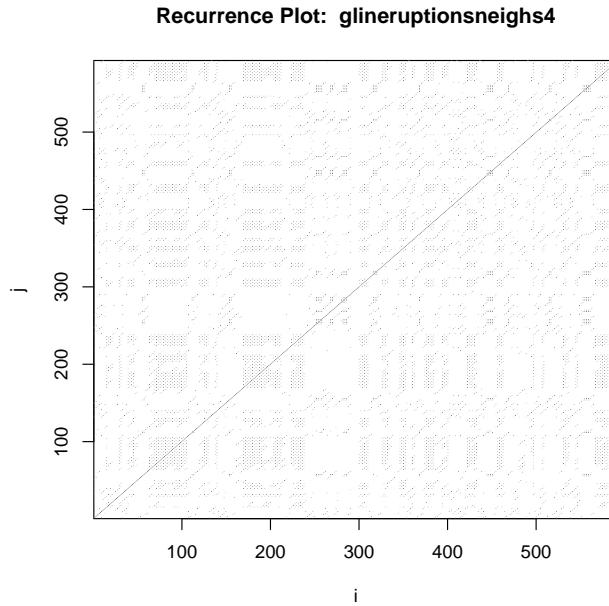


FIGURE 58. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.548 sec.

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

See Figure 59 on the facing page.

---

*Input*

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

---



---

*Input*

```
load(file="glineruptionsneighs2.RData")
local.recurrencePlotAux(glineruptionsneighs2)
```

---

See Figure 60 on the next page.

#### 6.5.2. Geyser eruptions - linearized. Dim=8.

---

*Input*

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

---

See Figure 61 on page 52

---

*Input*

---

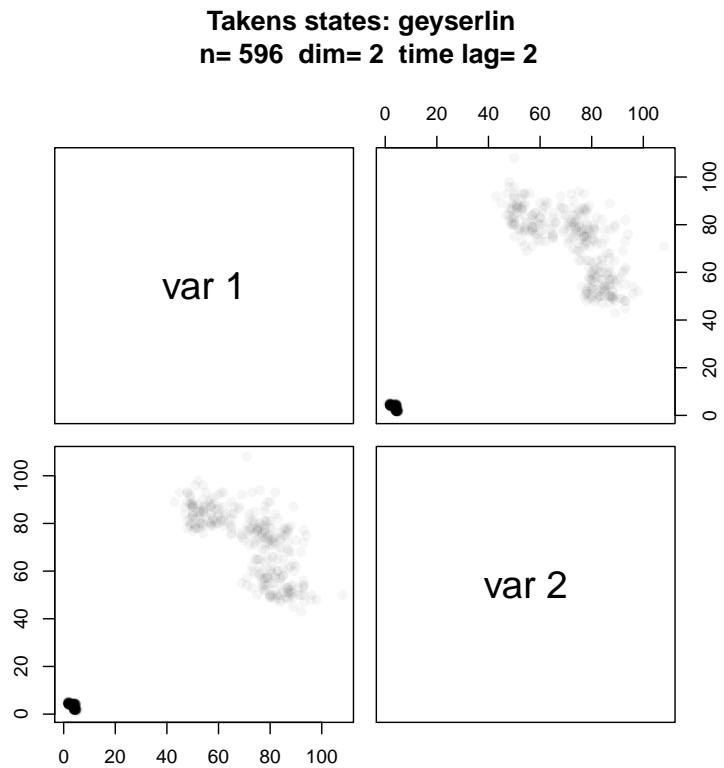


FIGURE 59. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.149 sec.

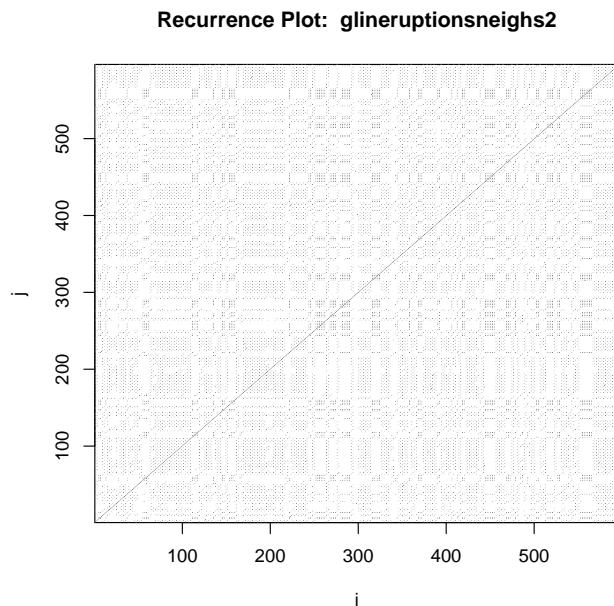


FIGURE 60. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser eruptions linearized. Dim=2. Time used: 0.341 sec.

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

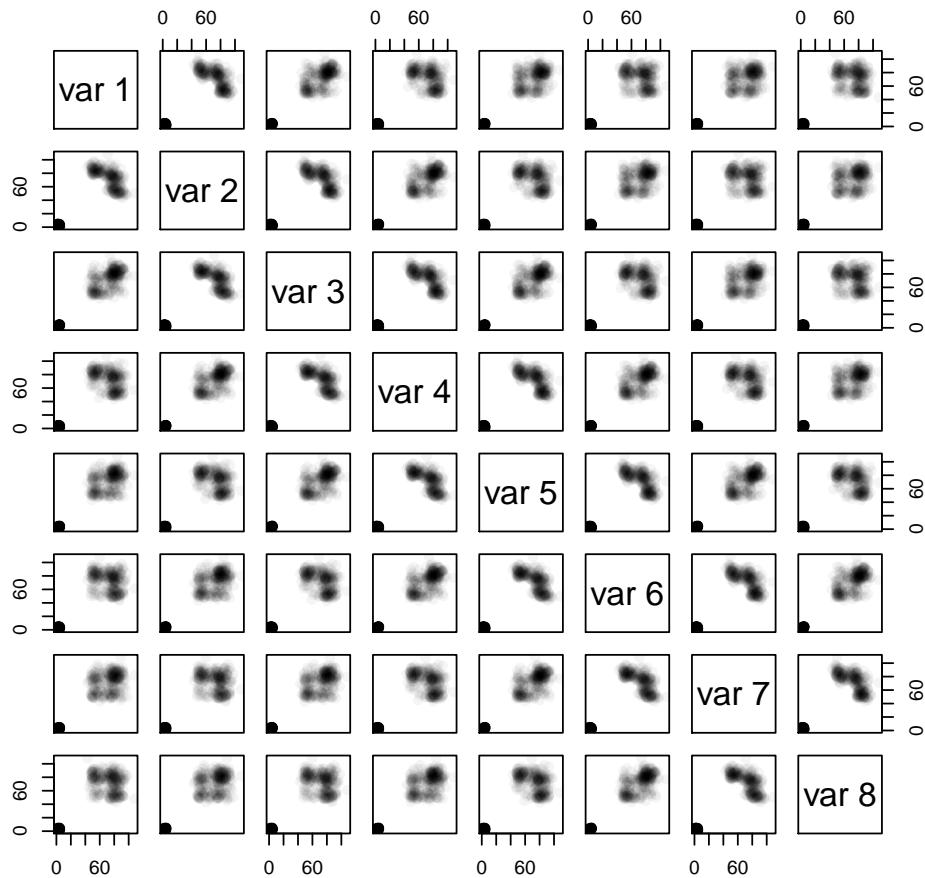


FIGURE 61. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 1.485 sec.

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

---

*Input*

---

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

See Figure 62 on the next page.

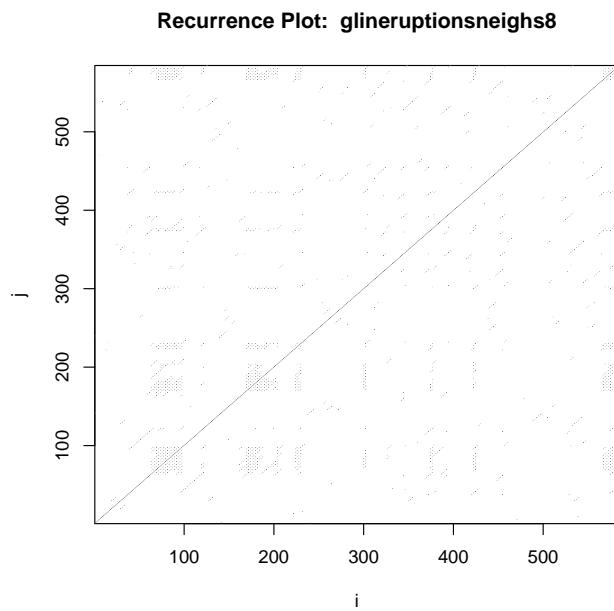


FIGURE 62. Recurrence plot. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.227 sec.

## 7. CASE STUDY: HRV DATA EXAMPLE.BEATS

Only 1024 data points used in this section

```

library(RHRV)                                     Input
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")
```

---

```

** 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
```

---

```

#      RecordPath = "beatsFolder")                                     Input
```

```

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

---

```

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

---

```

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

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

See Figure 63 on the facing page.

```

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

See Figure 67 on page 57.

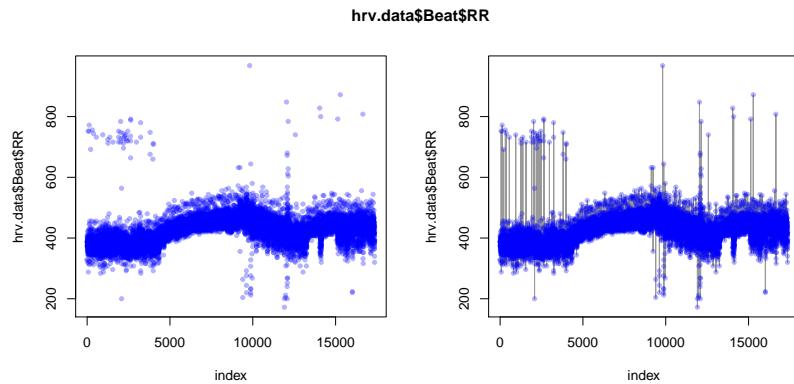


FIGURE 63. RHRV tutorial example.beats. Signal and linear interpolation.

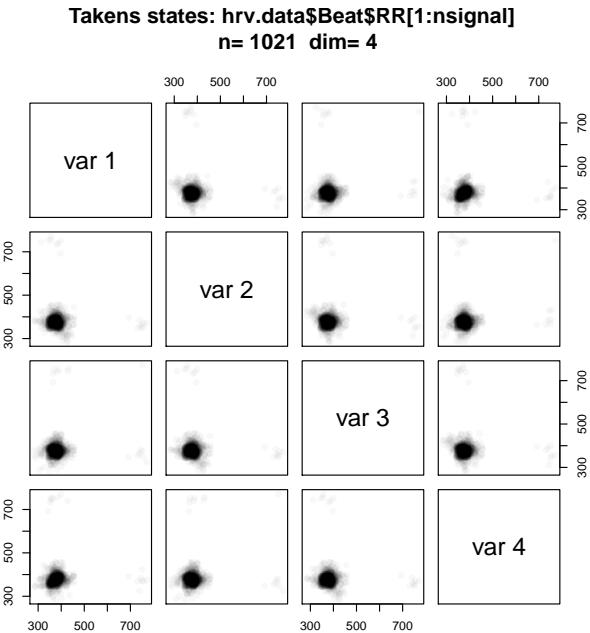


FIGURE 64. Recurrence plot. RHRV tutorial example.beats. Time used: 0.636 sec.

---

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

---

See Figure 65 on the following page.

---

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

---

See Figure 66 on page 57.

See Figure 67 on page 57.

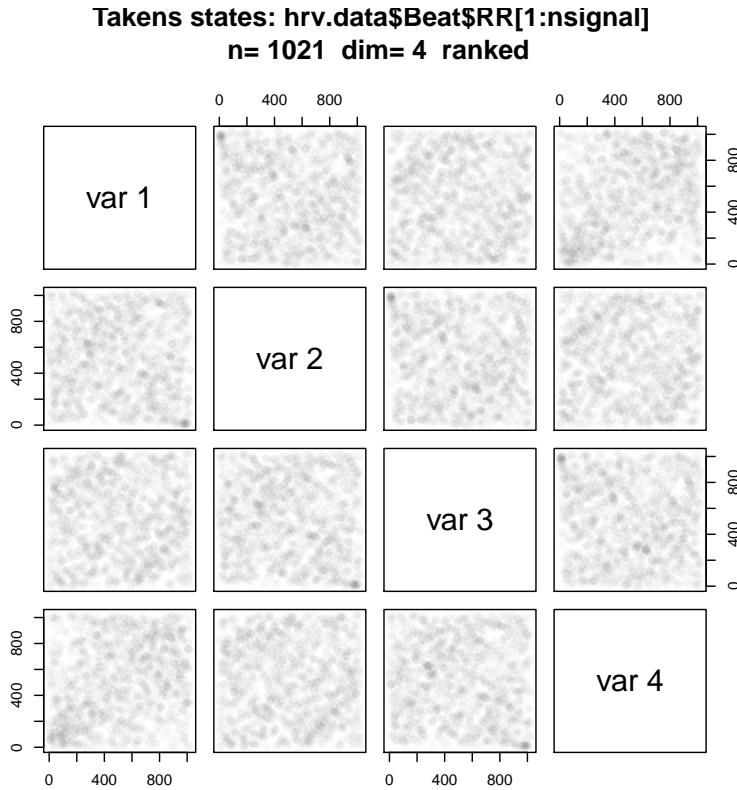


FIGURE 65. RHRV tutorial example.beats. Ranked data. Time used: 1.335 sec.

---

Input

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

Time used: 0.253 sec.

---

Input

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

See Figure 68 on page 58.

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

---

Input

```
hrvRRTakens2 <- local.buildTakens( time.series=hrv.data$Beat$RR[1:nsignal],
embedding.dim=2,time.lag=1)
hrvRRneighs2 <- local.findAllNeighbours(hrvRRTakens2, radius=16)
save(hrvRRneighs2, file="hrvRRneighs2.Rdata")
```

Time used: 0.15 sec.

---

Input

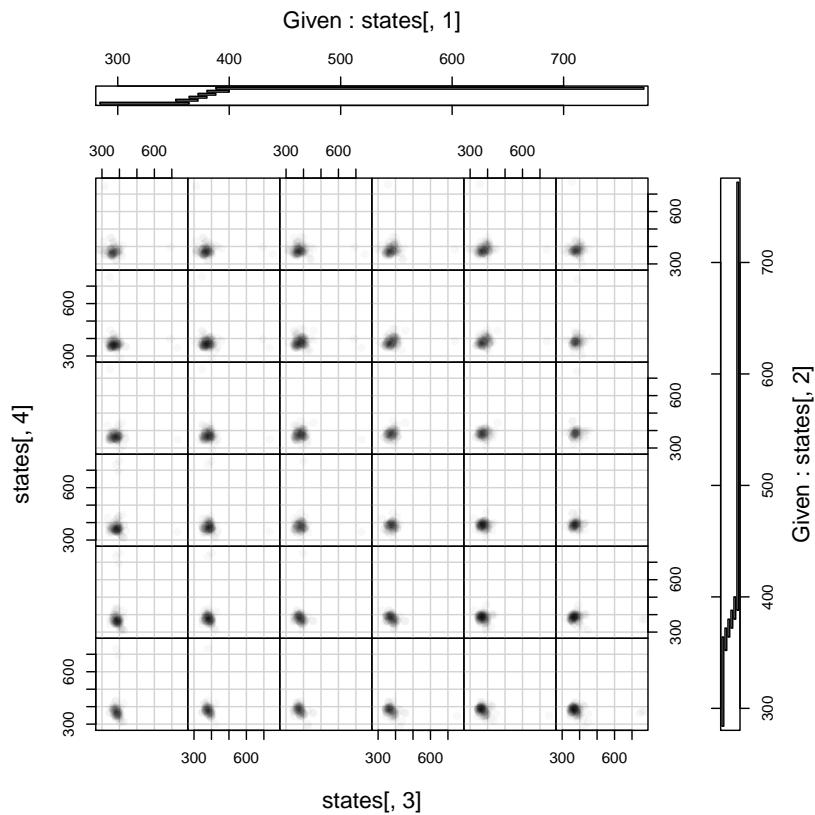


FIGURE 66. State coplot. RHRV tutorial example.beats. Time used: 0.332 sec.

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

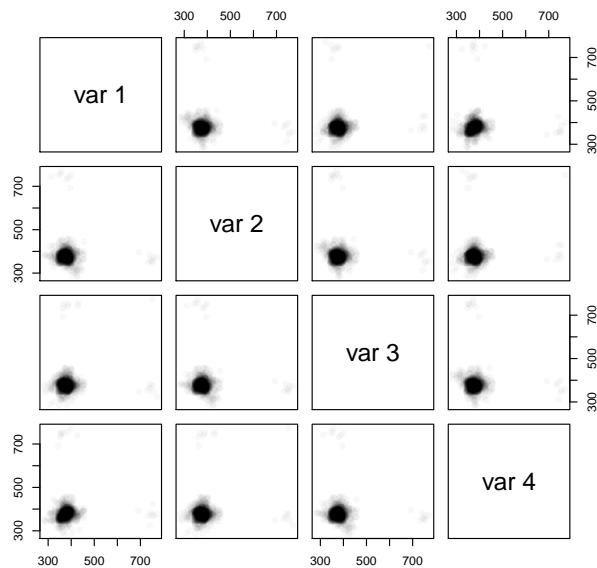


FIGURE 67. Recurrence plot. RHRV tutorial example.beats. Time used: 0.001 sec.

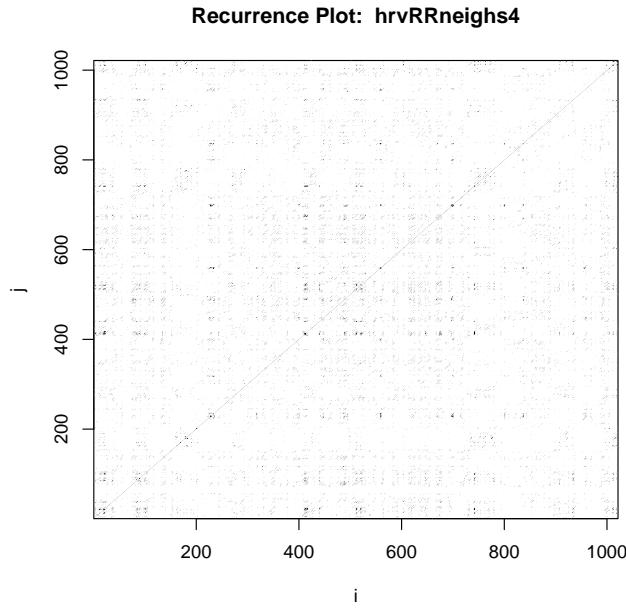


FIGURE 68. Recurrence plot. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=4. Time used: 0.774 sec.

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

See Figure 69 on page 60. Time used: 1.786 sec.

---

*Input*

```
hrvRRTakens6 <- local.buildTakens( time.series=hrv.data$Beat$RR[1:nignal],
                                     embedding.dim=6,time.lag=1)
hrvRRneighs6 <-local.findAllNeighbours(hrvRRTakens6, radius=16)
save(hrvRRneighs6, file="hrvRRneighs6.Rdata")
```

Time used: 0.242 sec.

---

*Input*

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

Dim=6. Time used: 0.818 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")
```

Time used: 0.356 sec.

---

*Input*

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

Dim=8. Time used: 1.17 sec.

---

```
Input
hrvRRTakens12 <- local.buildTakens( time.series=hrv.data$Beat$RR[1:nsignal],
                                         embedding.dim=2,time.lag=1)
hrvRRneighs12 <-local.findAllNeighbours(hrvRRTakens12, radius=16)
save(hrvRRneighs12, file="hrvRRneighs12.Rdata")
```

Time used: 1.451 sec.

---

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

Time used: 1.7 sec.

---

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

Time used: 2.052 sec.

---

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

Time used: 0.807 sec.

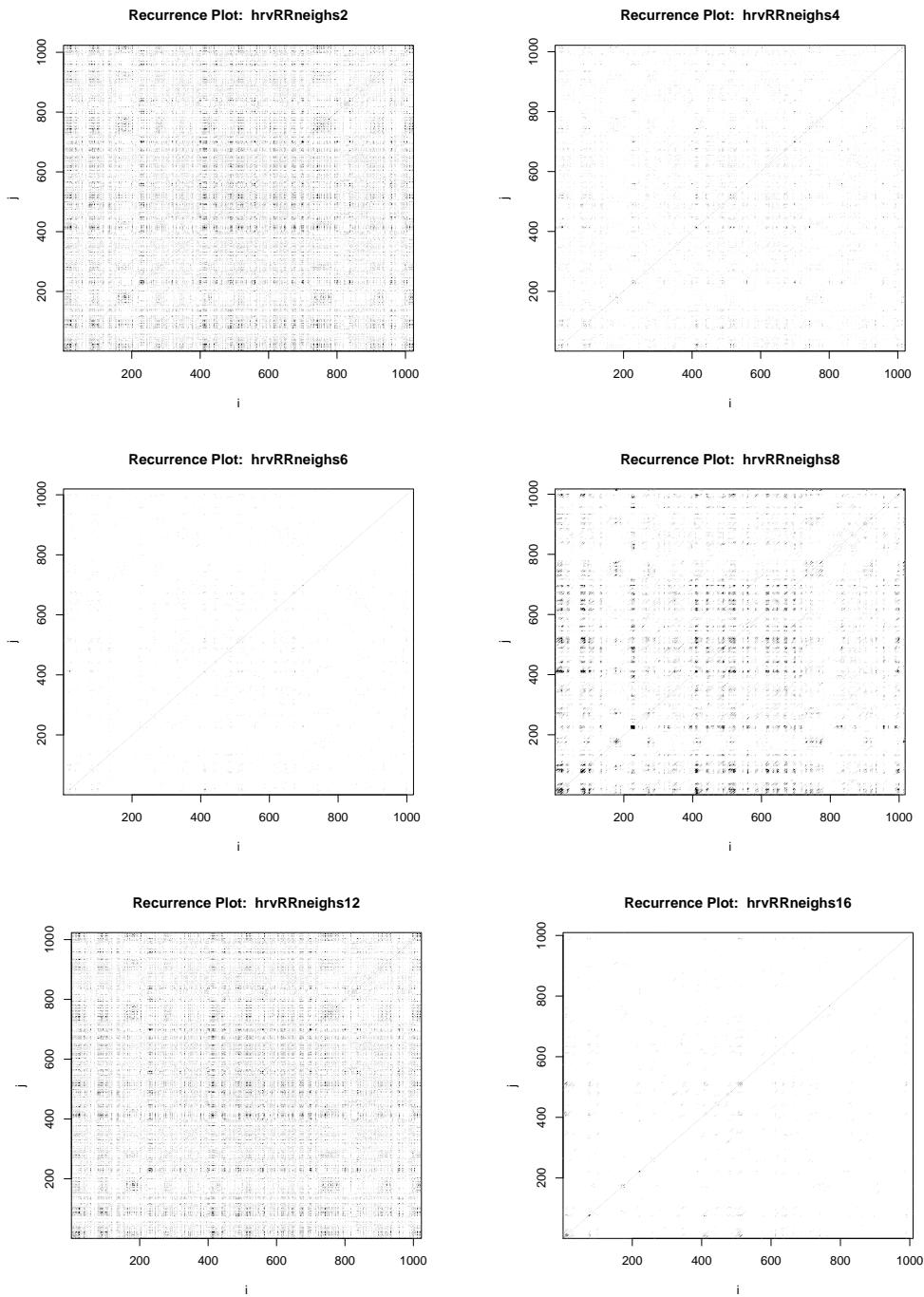


FIGURE 69. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 0.808 sec.

**7.2. 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.

**ToDo:** This is an experimental proposal

---

*Input*

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

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

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

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

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

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

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

  return(HRVData)
}
```

differences for HRV

---



---

*Input*

```
hrv.data <- BuildNIHR(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 70,

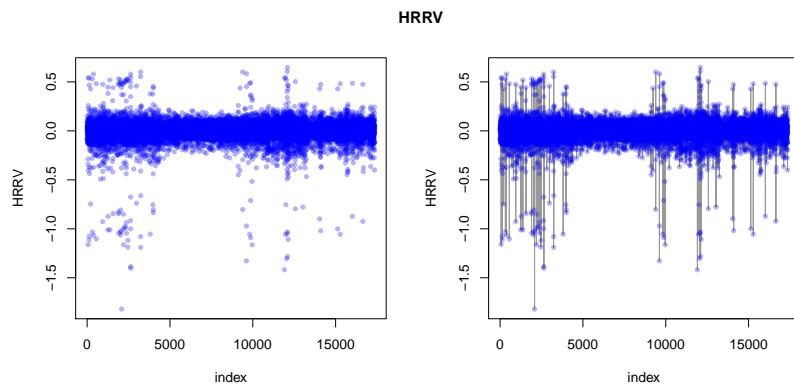


FIGURE 70. RHRV tutorial example.beats. HRRV Signal and linear interpolation.

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 71 on the facing page

---

*Input*

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

---

**ToDo:** findAll-  
Neighbours does not  
handle NAs

See Figure 72 on page 64

---

*Input*

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

---

Time used: 0.351 sec.

---

*Input*

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

---

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

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

**ToDo:** fix default setting for radius.

### 7.3. RHRV: example.beats - Variation: Comparison by Dimension.

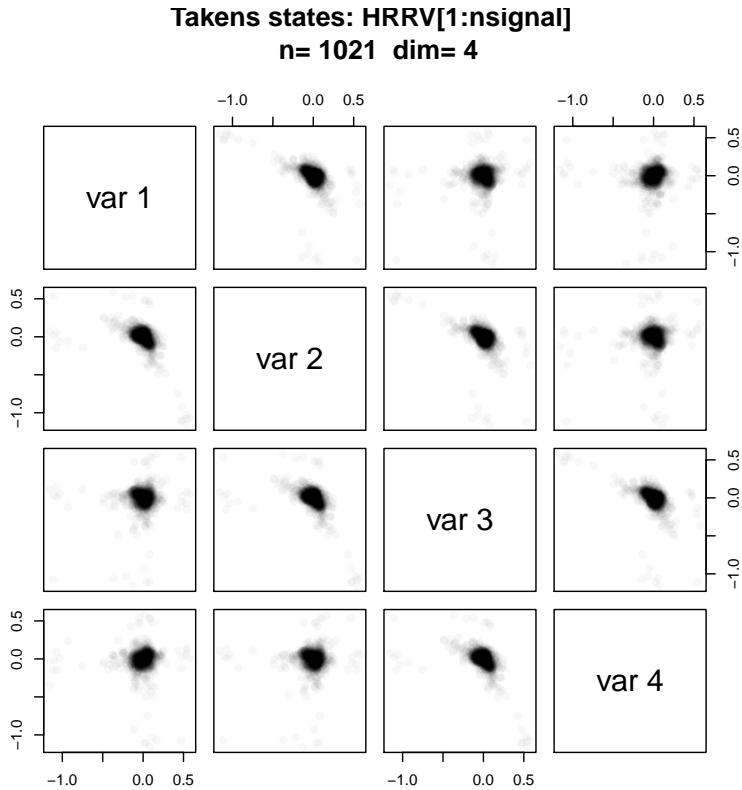


FIGURE 71. RHRV tutorial example.beats. HRRV Time used: 0.726 sec.

---

*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")
```

Time used: 0.591 sec.

---

*Input*

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

---

*Output*

```
hrvRRVtakens2[-(1:2), ] n= 1021 Dim: 2
Radius: 0.125 Recurrence coverage REC: 0.515 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
```

---

*Input*

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

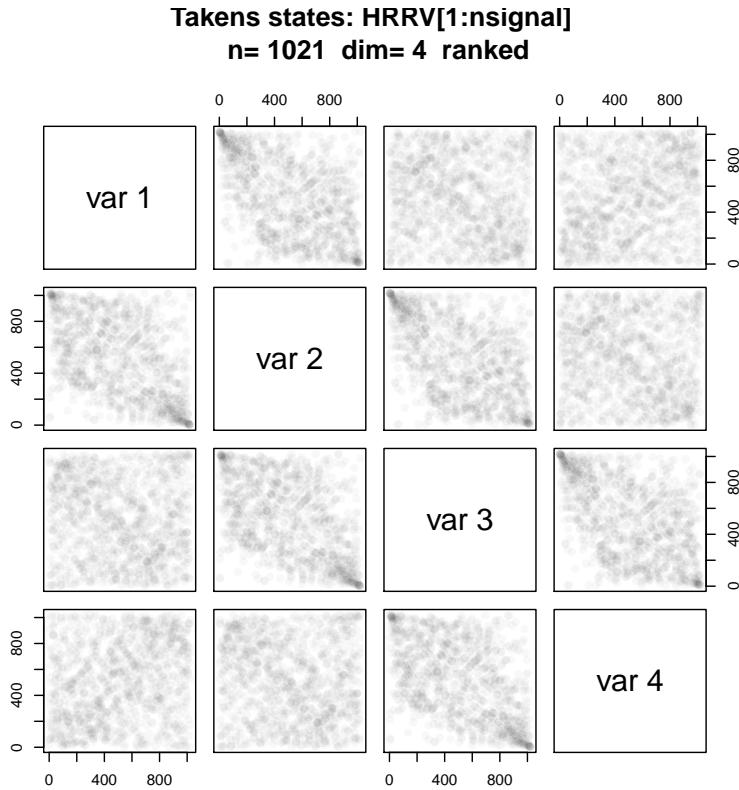


FIGURE 72. RHRV tutorial example.beats. Ranked HRRV data. Time used: 1.411 sec.

Time used: 3.881 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")
```

---

Time used: 0.515 sec.

---

*Input*

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

---

*Output*

```
hrvRRVtakens6[-(1:2), ] n= 1017 Dim: 6
Radius: 0.125 Recurrence coverage REC: 0.179 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
```

---



---

*Input*

---

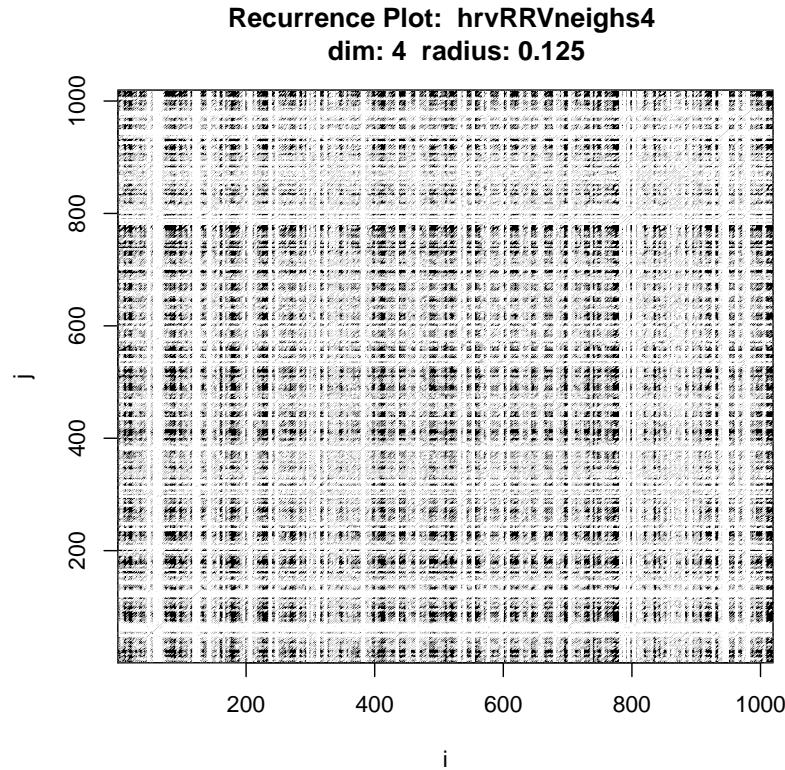


FIGURE 73. Recurrence Plot. Example case: RHRV tutorial example.beats. HRRV Dim=4. Time used: 2.285 sec.

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

Dim=6. Time used: 1.743 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")
```

---

Time used: 0.472 sec.

---

*Input*

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

---



---

*Output*

```
hrvRRVtakens8[-(1:2), ] n= 1015 Dim: 8
Radius: 0.125 Recurrence coverage REC: 0.105 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
```

---

---

*Input*

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

---

Dim=8. Time used: 1.245 sec.

---

*Input*

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

---

Time used: 1.73 sec.

---

*Input*

```
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
```

---

*Input*

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

---

Time used: 1.981 sec.

---

*Input*

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

---

Time used: 2.458 sec.

---

*Input*

```
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
```

---

*Input*

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

---

Time used: 1.381 sec.

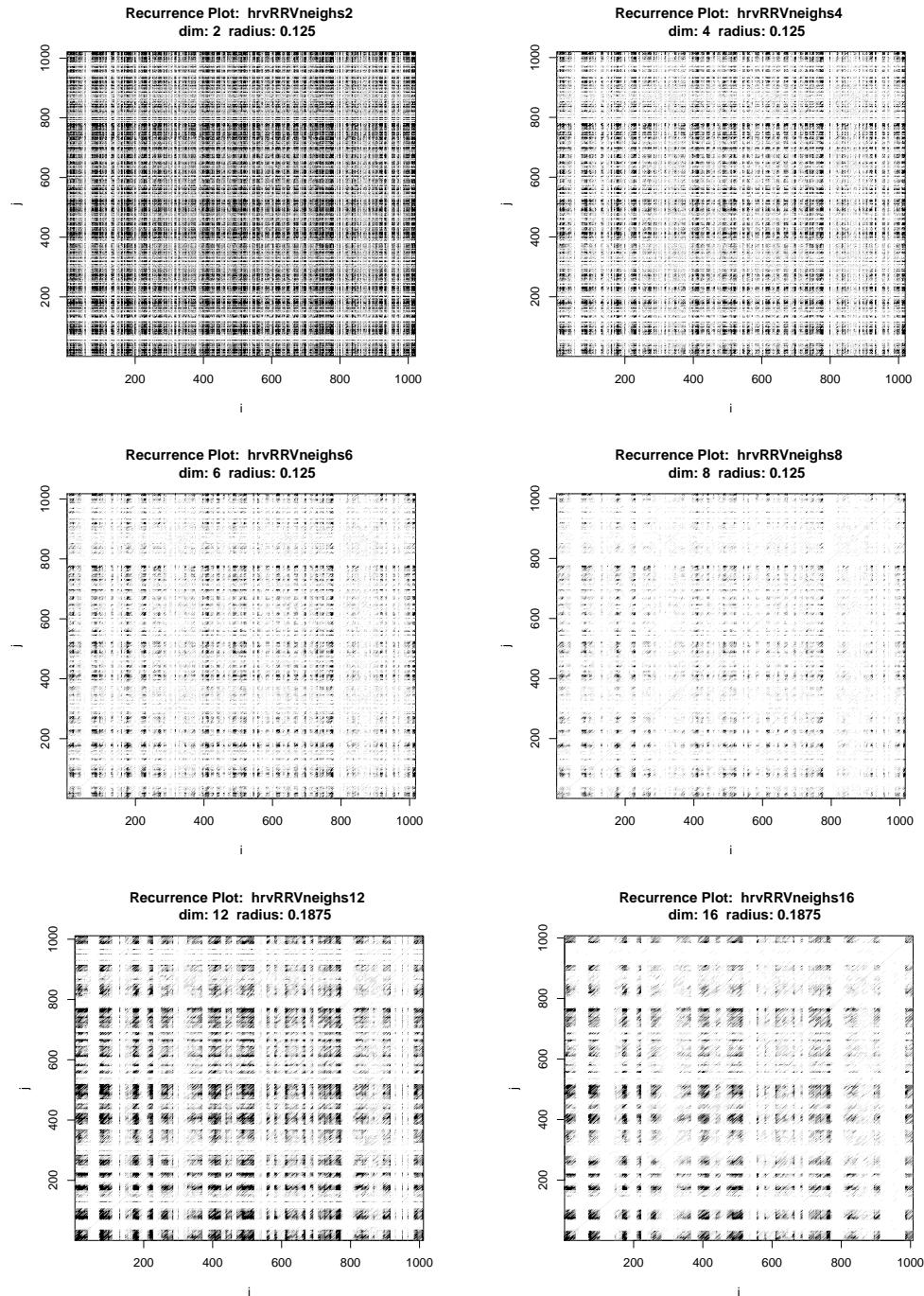


FIGURE 74. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 1.382 sec.

## 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*

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

---

See Figure 75 on the facing page.

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

---

*Input*

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

---

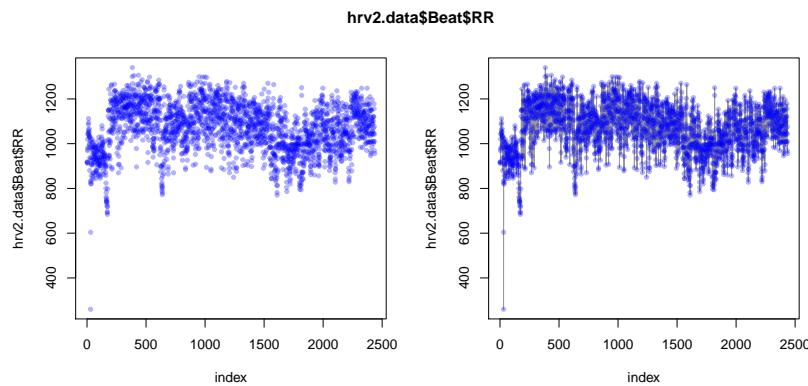
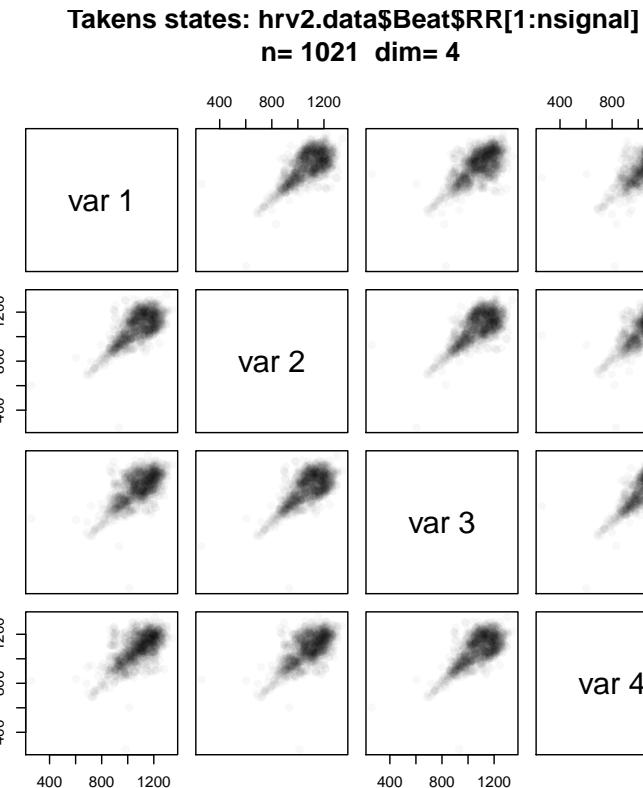


FIGURE 75. RHRV tutorial example2.beats. Signal and linear interpolation.

See Figure 76.



Only 1024 data points used in this plot

FIGURE 76. RHRV tutorial example2.beats. Time used: 0.659 sec.

---

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

---

See Figure 77 on the following page.

---

*Input*

---

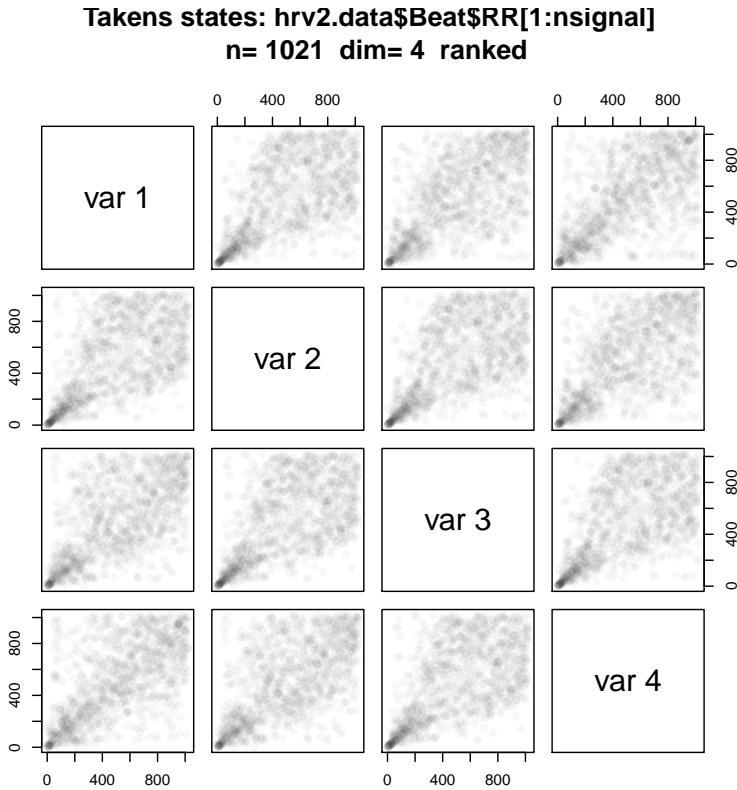


FIGURE 77. RHRV tutorial example2.beats. Ranked data. Time used: 1.36 sec.

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

See Figure 78 on the next page.

---

*Input*

```
hrv2RRneighs4 <- local.findAllNeighbours(hrv2RRtakens4, radius = 48)
save(hrv2RRneighs4, file="hrv2RRneighs4.Rdata")
```

---

Time used: 0.431 sec.

---

*Input*

```
load(file="hrv2RRneighs4.RData")
local.recurrencePlotAux(hrv2RRneighs4)
```

---

See Figure 79 on page 72.

### 8.1. RHRV: example2.beats - 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=32)
save(hrv2RRneighs2, file="hrv2RRneighs2.Rdata")
```

---

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

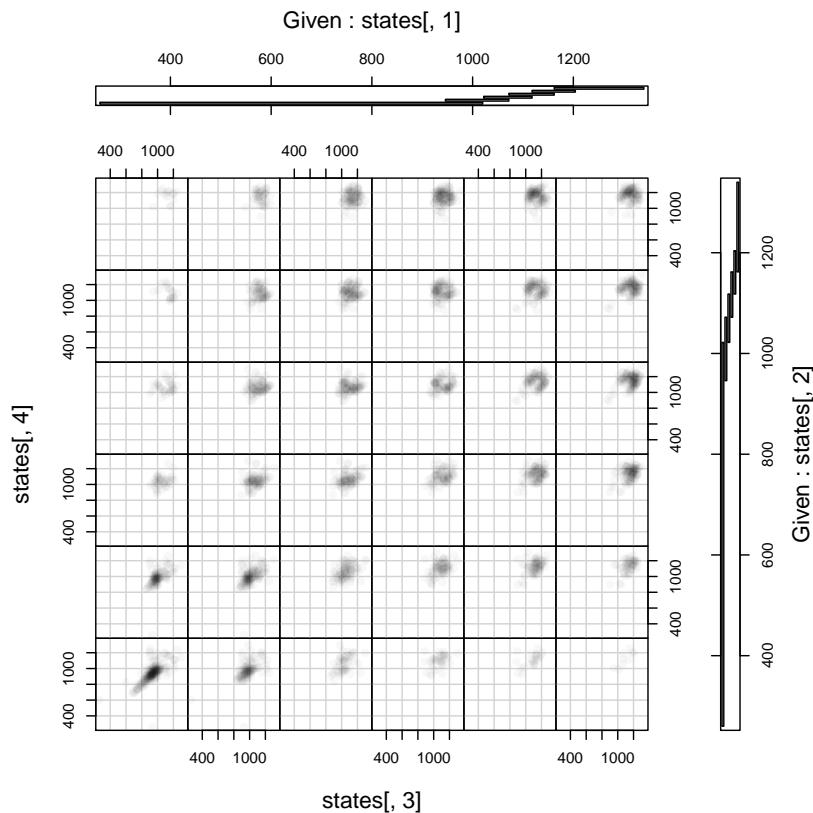


FIGURE 78. State coplot. RHRV tutorial example2.beats. Time used: 0.304 sec.

Time used: 0.126 sec.

---

*Input*

```
load(file="hrv2RRneighs2.RData")
local.recurrencePlotAux(hrv2RRneighs2)
```

---

See Figure 80 on page 74. Time used: 0.902 sec.

---

*Input*

```
hrv2RRTakens6 <- local.buildTakens( time.series=hrv2.data$Beat$RR[1:nsignal],
embedding.dim=6,time.lag=1)
hrv2RRneighs6 <- local.findAllNeighbours(hrv2RRTakens6, radius=96)
save(hrv2RRneighs6, file="hrv2RRneighs6.Rdata")
```

---

Time used: 0.391 sec.

---

*Input*

```
load(file="hrv2RRneighs6.RData")
local.recurrencePlotAux(hrv2RRneighs6)
```

---

Dim=6. Time used: 0.901 sec.

---

*Input*

---

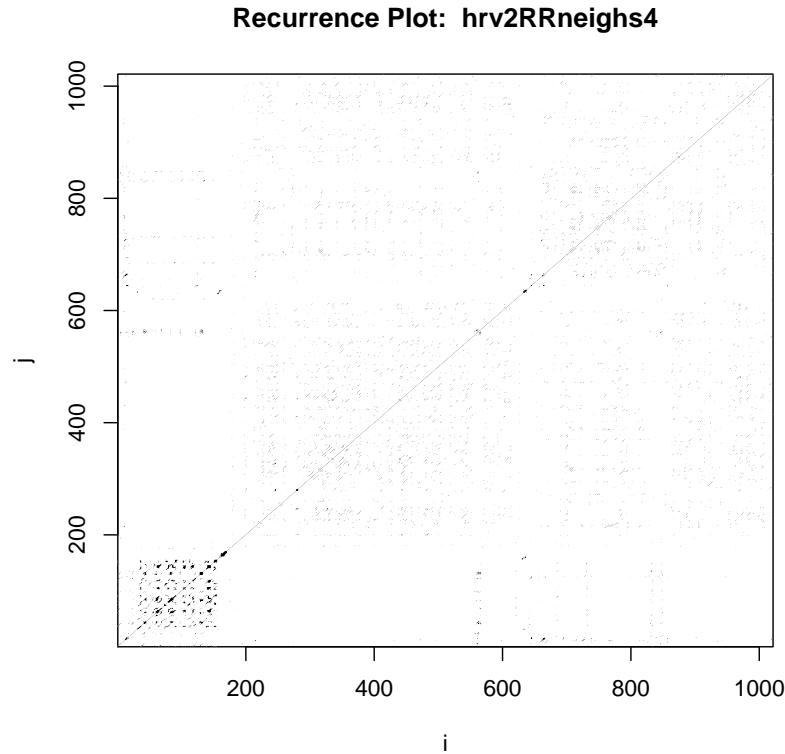


FIGURE 79. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=4. Time used: 0.862 sec.

```
hrv2RRtakens8 <- local.buildTakens( time.series=hrv2.data$Beat$RR[1:nSignal],
                                       embedding.dim=8, time.lag=1)
hrv2RRneighs8 <- local.findAllNeighbours(hrv2RRtakens8, radius=96)
save(hrv2RRneighs8, file="hrv2RRneighs8.Rdata")
```

Time used: 0.368 sec.

---

Input

---

```
load(file="hrv2RRneighs8.RData")
local.recurrencePlotAux(hrv2RRneighs8)
```

Dim=8. Time used: 0.68 sec.

---

Input

---

```
hrv2RRtakens12 <- local.buildTakens( time.series=hrv2.data$Beat$RR[1:nSignal],
                                         embedding.dim=2, time.lag=1)
hrv2RRneighs12 <- local.findAllNeighbours(hrv2RRtakens12, radius=96)
save(hrv2RRneighs12, file="hrv2RRneighs12.Rdata")
```

Time used: 1.11 sec.

---

Input

---

```
load(file="hrv2RRneighs12.RData")
local.recurrencePlotAux(hrv2RRneighs12)
```

Time used: 2.151 sec.

---

*Input*

```
hrv2RRtakens16 <- local.buildTakens(
  time.series=hrv2.data$Beat$RR[1:nSignal],
  embedding.dim=16,time.lag=1)
hrv2RRneighs16 <- local.findAllNeighbours(hrv2RRtakens16, radius=192)
save(hrv2RRneighs16, file="hrv2RRneighs16.Rdata")
```

---

Time used: 2.715 sec.

---

*Input*

```
load(file="hrv2RRneighs16.RData")
local.recurrencePlotAux(hrv2RRneighs16)
```

---

Time used: 1.3 sec.

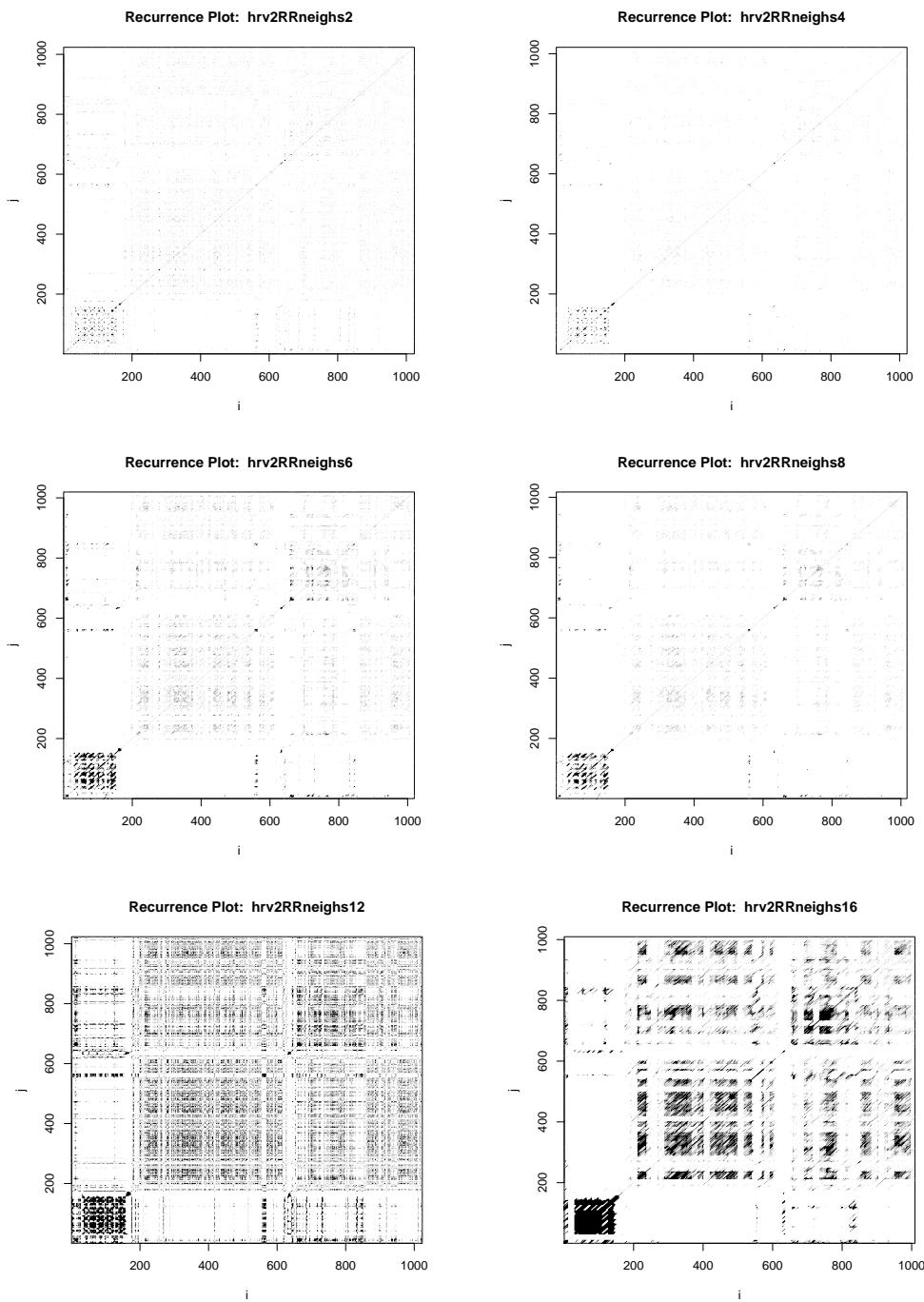


FIGURE 80. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 1.3 sec.

**8.2. 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.

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

---

*Input*

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

---

*Output*

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

---

*Input*

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

---

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

---

*Input*

```
plotsignal(HRRV)
```

---

See Figure 81,

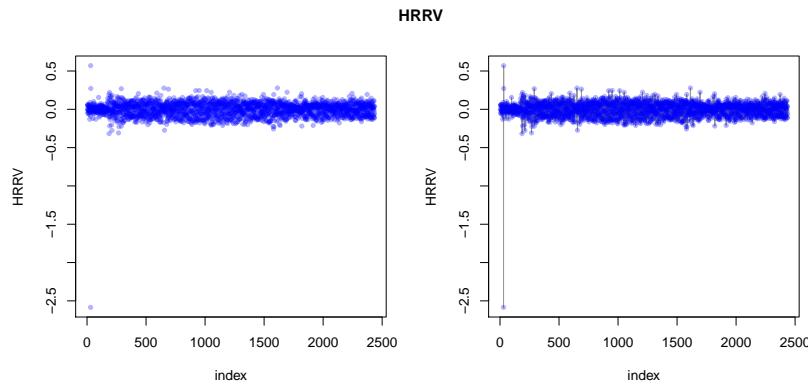


FIGURE 81. RHRV tutorial example2.beats. HRRV Signal and linear interpolation.

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 82 on the next page.

---

*Input*

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

---

*Output*

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

---

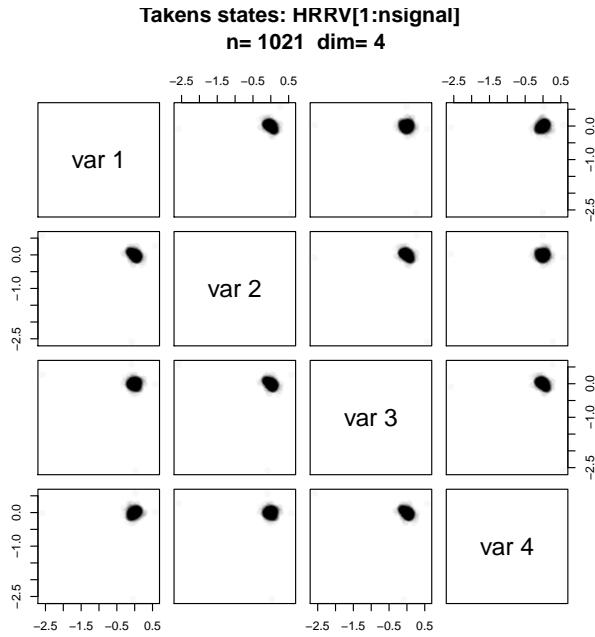


FIGURE 82. Recurrence plot. RHRV tutorial example2.beats. HRRV  
Time used: 0.697 sec.

See Figure 83 on the facing page.

---

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

---

**ToDo:** findAll-  
Neighbours does not  
handle NAs

See Figure 84 on page 78

---

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

---

Time used: 0.216 sec.

---

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

---

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

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

**ToDo:** fix default  
setting for radius.  
Eckmann uses near-  
est neighbours with  
NN=10

### 8.3. RHRV: example2.beats - Variation: Comparison by Dimension.

---

```
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")
```

---

Time used: 0.476 sec.

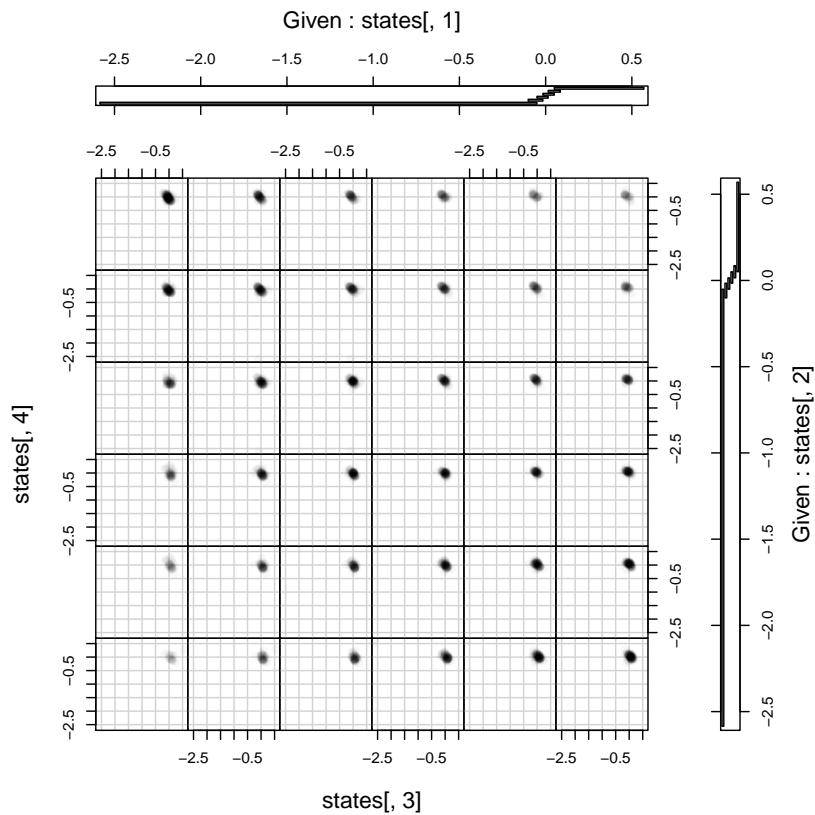


FIGURE 83. State coplot. RHRV tutorial example2.beats. HRRV. Time used: 1.042 sec.

---

*Input*  
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

---



---

*Input*  
load(file="hrv2RRVneighs2.RData")  
local.recurrencePlotAux(hrv2RRVneighs2, dim=2, radius=0.125)

---

Time used: 3.53 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)  
save(hrv2RRVneighs6, file="hrv2RRVneighs6.RData")

---

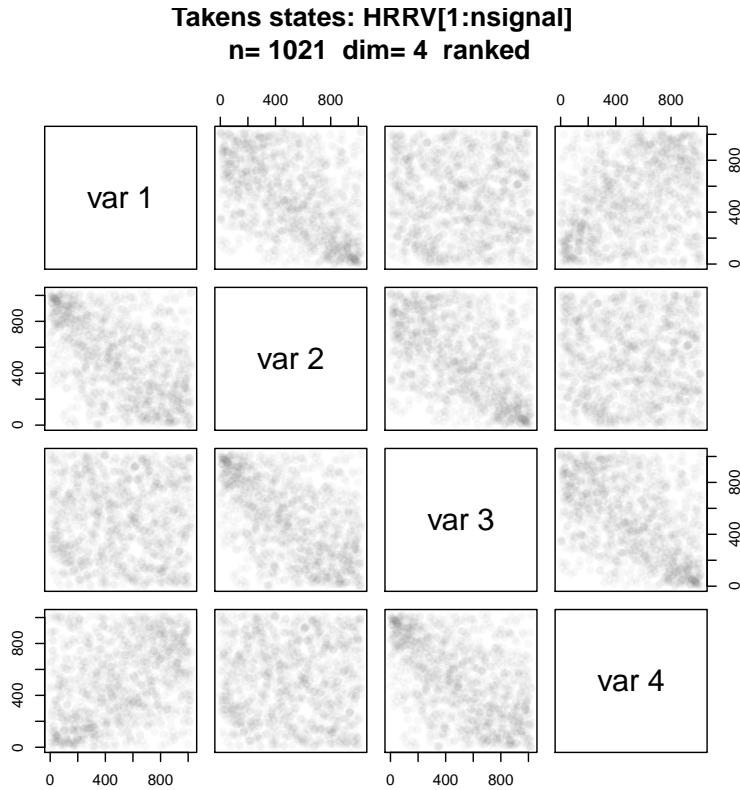


FIGURE 84. RHRV tutorial example2.beats. Ranked HRRV data. Time used: 1.724 sec.

Time used: 0.409 sec.

---

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

---

*Output*  
`hrv2RRVtakens6[-(1:2), ] n= 1017 Dim: 6`  
Radius: 0.125 Recurrence coverage REC: 0.187 log(REC)/log(R): 0.808  
Determinism: 0.959 Laminarity: 0.368  
DIV: 0.009  
Trend: 0 Entropy: 2.528  
Diagonal lines max: 113 Mean: 6.229 Mean off main: 6.195  
Vertical lines max: 73 Mean: 3.978

---

*Input*  
`load(file="hrv2RRVneighs6.RData")  
local.recurrencePlotAux(hrv2RRVneighs6, dim=6, radius=0.125)`

---

Dim=6. Time used: 1.685 sec.

---

*Input*  
`hrv2RRVtakens8 <- local.buildTakens( time.series=HRRV[1:nsignal],  
embedding.dim=8,time.lag=1)`

---

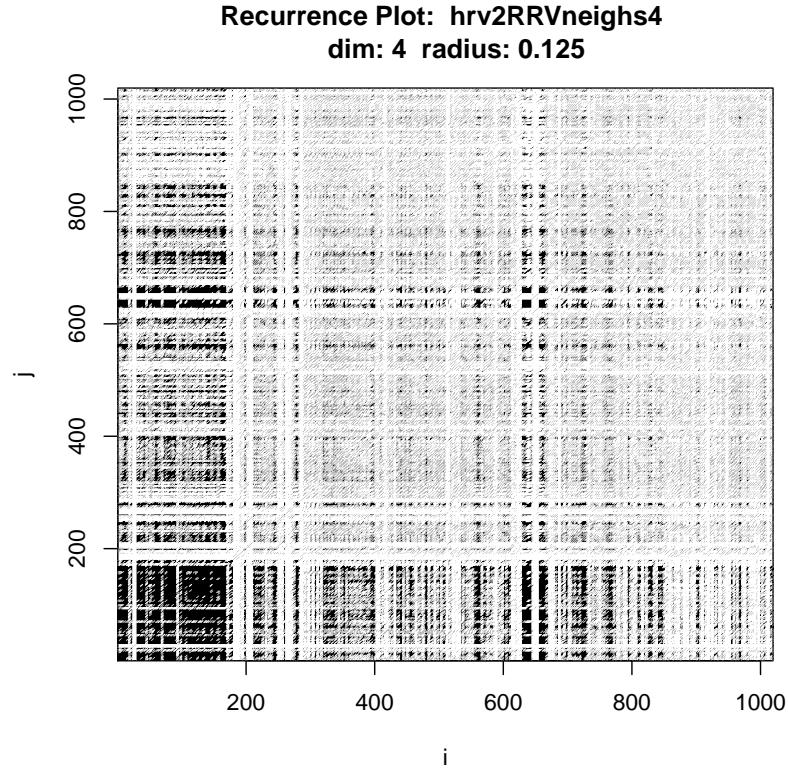


FIGURE 85. Recurrence Plot. Example case: RHRV tutorial example2.beats. HRRV Dim=4. Time used: 2.223 sec.

```
hrv2RRVneighs8 <- local.findAllNeighbours(hrv2RRVtakens8[-(1:2),], radius=0.125)
save(hrv2RRVneighs8, file="hrv2RRVneighs8.Rdata")
```

Time used: 0.5 sec.

---

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

---

```
showrqa(hrv2RRVtakens8[-(1:2),], radius=0.125, do.hist=FALSE)
```

---

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

---

```
hrv2RRVtakens8[-(1:2), ] n= 1015 Dim: 8
Radius: 0.125 Recurrence coverage REC: 0.122 log(REC)/log(R): 1.012
Determinism: 0.96 Laminarity: 0.335
DIV: 0.009
Trend: 0 Entropy: 2.57
Diagonal lines max: 111 Mean: 6.482 Mean off main: 6.428
Vertical lines max: 71 Mean: 4.039
```

---

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

---

```
load(file="hrv2RRVneighs8.RData")
local.recurrencePlotAux(hrv2RRVneighs8, dim=8, radius=0.125)
```

Dim=8. Time used: 1.192 sec.

---

*Input*

```
hrv2RRVtakens12 <-
  local.buildTakens( time.series=HRRV[1:nseries],
                     embedding.dim=12, time.lag=1)
hrv2RRVneighs12 <-
  local.findAllNeighbours(hrv2RRVtakens12[-(1:2),], radius=3/16)
save(hrv2RRVneighs12, file="hrv2RRVneighs12.Rdata")
```

---

Time used: 1.753 sec.

---

*Input*

```
showrqa(hrv2RRVtakens12[-(1:2),], radius=3/16, do.hist=FALSE)
```

---

*Output*

```
hrv2RRVtakens12[-(1:2), ] n= 1011 Dim: 12
Radius: 0.1875 Recurrence coverage REC: 0.306 log(REC)/log(R): 0.708
Determinism: 0.993 Laminarity: 0.6
DIV: 0.007
Trend: 0 Entropy: 3.417
Diagonal lines max: 141 Mean: 13.443 Mean off main: 13.4
Vertical lines max: 143 Mean: 4.312
```

---



---

*Input*

```
load(file="hrv2RRVneighs12.RData")
local.recurrencePlotAux(hrv2RRVneighs12, dim=12, radius=3/16)
```

---

Time used: 2.223 sec.

---

*Input*

```
hrv2RRVtakens16 <- local.buildTakens( time.series=HRRV[1:nseries],
                                         embedding.dim=16, time.lag=1)
hrv2RRVneighs16 <- local.findAllNeighbours(hrv2RRVtakens16[-(1:2),], radius=3/16)
save(hrv2RRVneighs16, file="hrv2RRVneighs16.Rdata")
```

---

Time used: 2.752 sec.

---

*Input*

```
showrqa(hrv2RRVtakens16[-(1:2),], radius=3/16, do.hist=FALSE)
```

---

*Output*

```
hrv2RRVtakens16[-(1:2), ] n= 1007 Dim: 16
Radius: 0.1875 Recurrence coverage REC: 0.221 log(REC)/log(R): 0.902
Determinism: 0.994 Laminarity: 0.555
DIV: 0.007
Trend: 0 Entropy: 3.419
Diagonal lines max: 137 Mean: 13.706 Mean off main: 13.644
Vertical lines max: 89 Mean: 4.104
```

---



---

*Input*

```
load(file="hrv2RRVneighs16.RData")
local.recurrencePlotAux(hrv2RRVneighs16, dim=16, radius=3/16)
```

---

Time used: 1.717 sec.

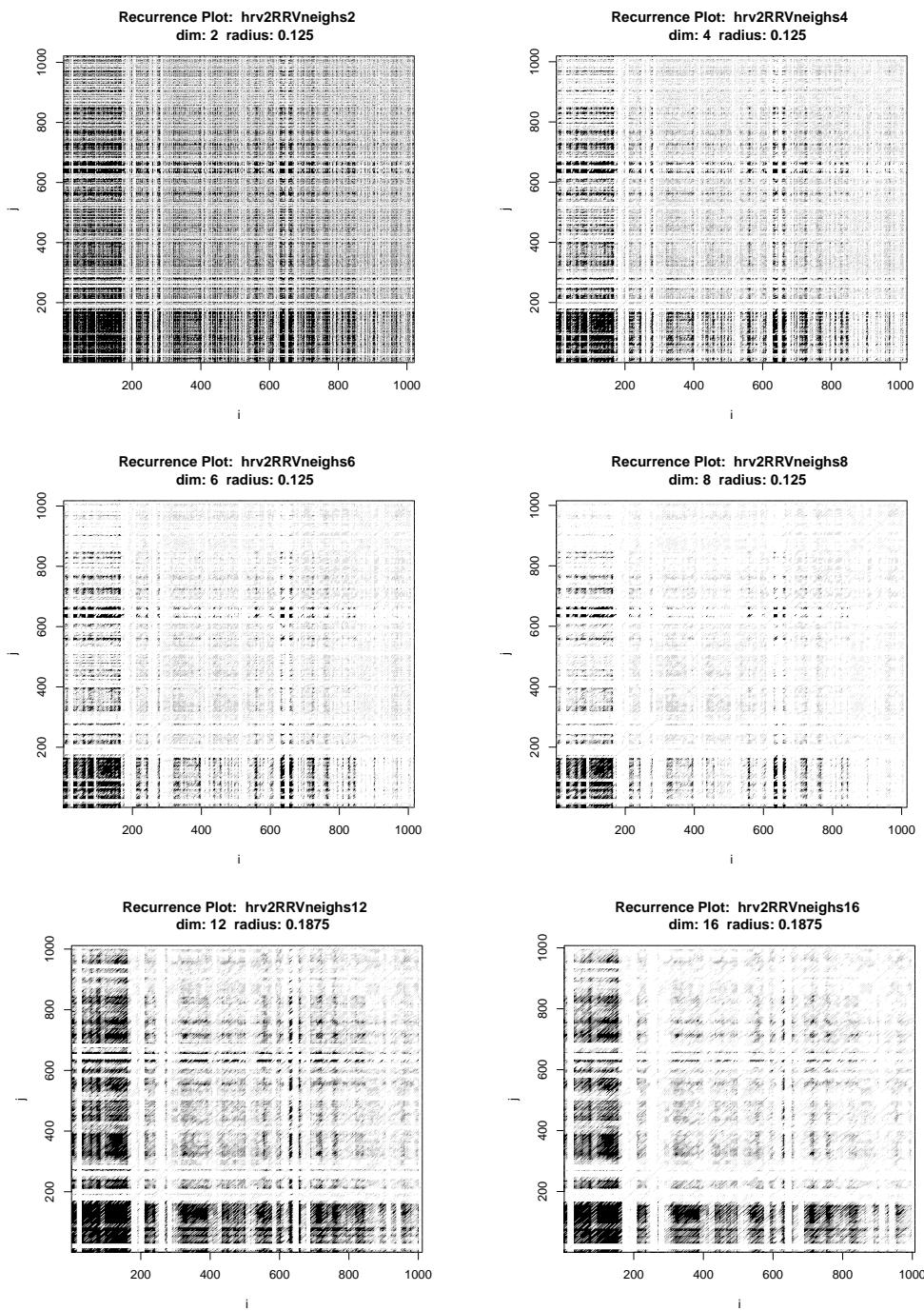


FIGURE 86. Recurrence Plot. Example case: RHRV tutorial example2.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 1.717 sec.

## 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 takens 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*, 7
- 2: improve to a full *show* method, 7
- 2: propagate parameters from *buildTakens* and *findAllNeighbours* in a slot of the result, instead of using explicit parameters in *recurrencePlotAux.*, 6
- 6: Geyser: extended to two-dimensional data in *geyserlin*. Experimental only. Check., 33
- 6: double check: *MASS:::geyser* should be used, not *faithful*, 33
- 7: This is an experimental proposal, 61
- 7: We have outliers at approximately 2\*RR.  
Could this be an artefact of preprocessing, filtering out too many impulses?, 54
- 7: check. There seem to be strange artefacts., 62
- 7: *findAllNeighbours* does not handle NAs, 62
- 7: fix default setting for radius. Eckmann uses nearest neighbours with NN=10, 62
- 8: Consider using differences, 75
- 8: We have outliers at approximately 2\*RR.  
Could this be an artefact of preprocessing, filtering out too many impulses?, 68
- 8: check. There seem to be strange artefacts., 76
- 8: *findAllNeighbours* does not handle NAs, 76
- 8: fix default setting for radius. Eckmann uses nearest neighbours with NN=10, 76

delay embedding, 2

Geyser, 33

heart rate, 54, 68

heart rate variation, 62, 75

hrv, 54, 68

recurrence plot, 2

recurrence quantification analysis, 3

RQA, 3

takens plot

hrv2, 69

Takens state, 2

R session info:

Total Sweave time used: 97.362 sec. at Wed Aug 13 21:39:11 2014.

- R version 3.1.1 (2014-07-10), x86\_64-apple-darwin13.1.0
- Locale: en\_GB.UTF-8/en\_GB.UTF-8/en\_GB.UTF-8/C/en\_GB.UTF-8/en\_GB.UTF-8
- Base packages: base, datasets, graphics, grDevices, methods, stats, tcltk, utils
- Other packages: leaps 2.9, locfit 1.5-9.1, MASS 7.3-33, Matrix 1.1-4, mgcv 1.8-1, nlme 3.1-117, nonlinearTseries 0.2.1, rgl 0.93.1098, RHRV 4.0, sintro 0.1-3, tkrplot 0.0-23, TSA 1.01, tseries 0.10-32, waveslim 1.7.3
- Loaded via a namespace (and not attached): grid 3.1.1, lattice 0.20-29, quadprog 1.5-5, tools 3.1.1, zoo 1.7-11

L<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: 159 $
>Date: 2014-08-13 21:35:17 +0200 (Wed, 13 Aug 2014) *
$Name: 
$Author: gsawitzki $
```

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

GÜNTHER SAWITZKI  
STATLAB HEIDELBERG  
IM NEUENHEIMER FELD 294  
D 69120 HEIDELBERG