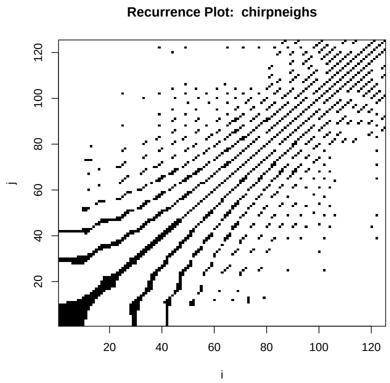
STATISTICAL DATA ANALYSIS:
RECURRENCE PLOT

GÜNTHER SAWITZKI



Contents

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: February 17, 2014 from cvs *Revision* : 1.7
`gs@statlab.uni-heidelberg.de` .

1. Setup

Input
```
save.RNGseed <- 87149 #.Random.seed
save.RNGkind <- RNGkind()
# save.RNGseed
save.RNGkind
```

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

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

Input
```
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)
```

We use

Input
```
library(nonlinearTseries)
```

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

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

Input
```
statepairs <- function(states, rank=FALSE){
        main <- paste("Takens states:",deparse(substitute(states)), "\n",
                "n=", dim(states)[1]," dim=", dim(states)[2])
        if (rank) {states <- apply(uniftakens,2,rank,ties.method="random")
        main <- paste(main," ranked")}
        pairs(states,
        main=main,
        col=rgb(0,0,0,0.2))
}
```

1.1. **Local Bottleneck.** To allow experimental implementations, functions from *nonlinearTseries* are aliased here.

Input
```
local.buildTakens <- buildTakens
```

Input
```
local.findAllNeighbours <- nonlinearTseries:::findAllNeighbours
```

minor cosmetics added to recurrencePlotAux

ToDo: propagate parameters from *buildTakens* and *findAllNeighbours* in a slot of the result,

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
              }
            }
          }

        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 `nonlinearTseries`

2. Test Cases

We set up a small series of test signals.

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.

Input
```
nsignal <- 128
system.time.start <- proc.time()
```

For signal representation, we use a common layout.

Input
```
plotsignal <- function (signal) {
par(mfrow=c(1,2))
plot(signal, col=rgb(0,0,1,0.4), pch=20, xlab="t" )

plot(signal, type="l",
        main=deparse(substitute(signal)), xlab="t", col=rgb(0,0,0,0.4))
points(signal, col=rgb(0,0,1,0.4), pch=20 )
}
```

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

See Figure 1.

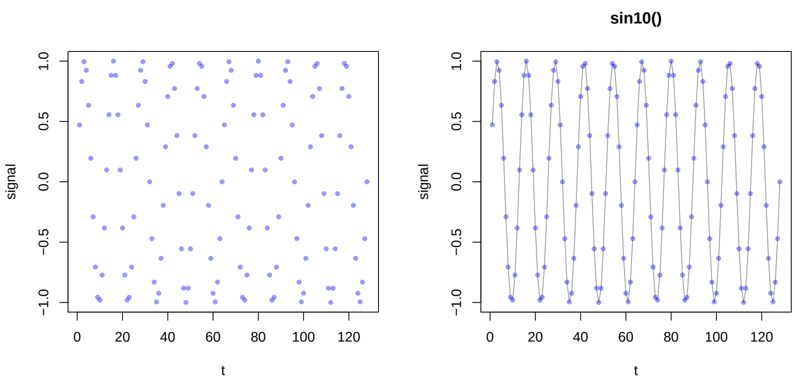


Figure 1. Test case: sin10. Signal and linear interpolation.

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

See Figure 2,

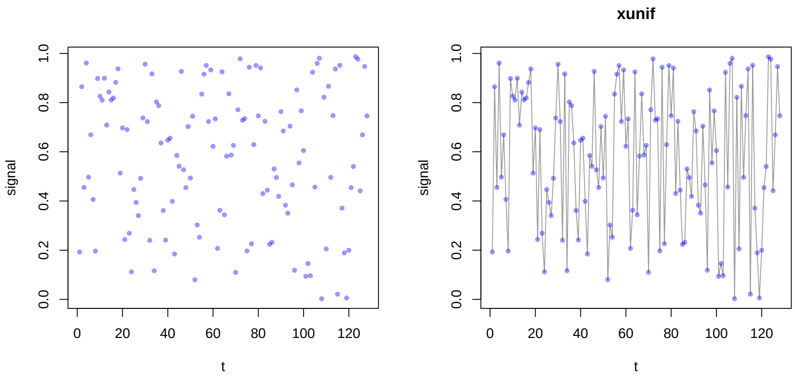


FIGURE 2. Test case: unif - uniform random numbers. Signal and linear interpolation.

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 on the following page,

ToDo: include doppler waveslim

3. TAKENS' RECURRENCE STATES

Recurrence plots have been introduced in an attempt to understand near periodic in hydrodynamics. On the one hand, and extended theory on dynamical systems was available,

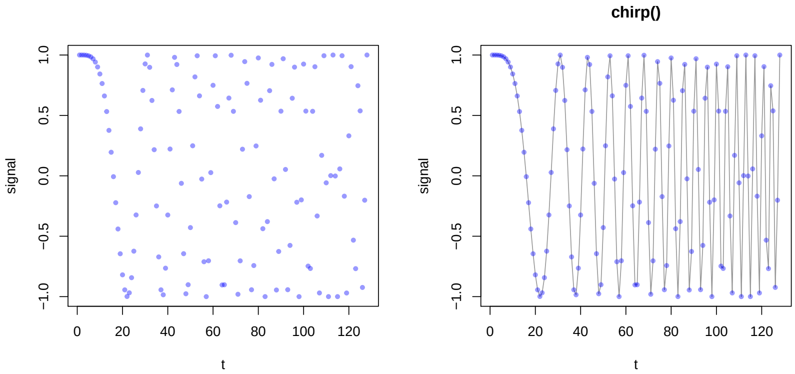


FIGURE 3. Test case: chirp signal. Signal and linear interpolation.

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.

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.

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

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

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

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.

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

See Figure 4.

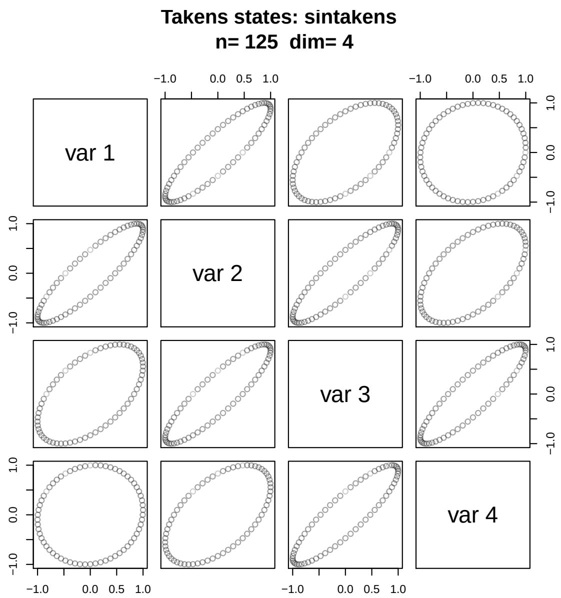


FIGURE 4. Test case: sinus. Note that marginal views of 1-dimensional circles in d space may appear as ellipses. Time used: 0.169 sec.

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

See Figure 5 on the following page.

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

See Figure 6 on the next page

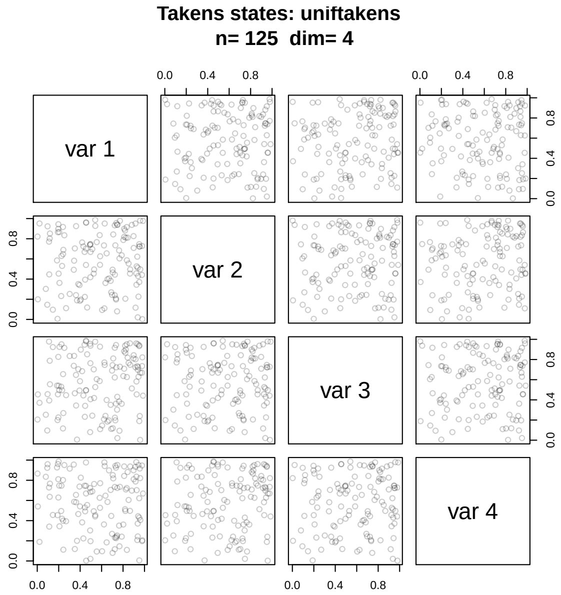


FIGURE 5. Test case: uniform random numbers. Time used: 0.187 sec.

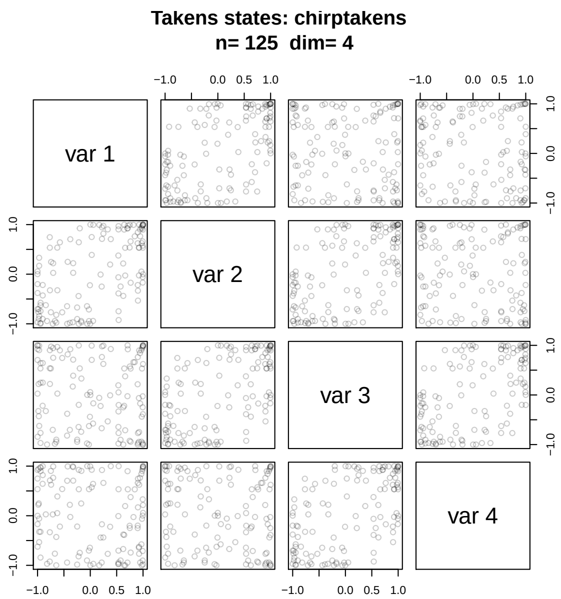


FIGURE 6. Test case: chirp signal. Time used: 0.178 sec.

4. Recurrence Plots

ToDo: consider dimension-adjusted radius

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>`.

5. 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.

This is a hack to report RQA information. $dim = NULL$ is added to align calling with other functions.

Input

```
showrqa <- function(takens, dim=NULL, radius)
{
        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]$REC, 3), "\n"))
        str(xxrqa[2:12])
        oldpar <- par(mfrow=c(2,1))
        barplot(xxrqa$diagonalHistogram,
                main=paste(deparse(substitute(takens)), "Diagonal",
                        "\n n=", dim(takens)[1], " Dim:", dim(takens)[2], " Radius: ",radius))
        barplot(xxrqa$recurrenceRate,
                main=paste(deparse(substitute(takens)),"Recurrence Rate",
                        "\n n=", dim(takens)[1], " Dim:", dim(takens)[2], " Radius: ",radius))
        par(oldpar)
        invisible(xxrqa)
}
```

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.

6. Applied Recurrence Plots

6.1. Sinus.

Input
```
system.time(sin10neighs<-local.findAllNeighbours(sintakens, radius=0.2))[3]
```

Output
```
elapsed
  0.006
```

Input
```
save(sin10neighs, file="sin10neighs.Rdata")
```

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

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

Output
```
sintakens  n= 125  Dim: 4
Radius: 0.2  Recurrence coverage REC: 0.079
List of 11
 $ RATIO            : num 12.4
 $ DET              : num 0.987
 $ DIV              : num 0.00893
```

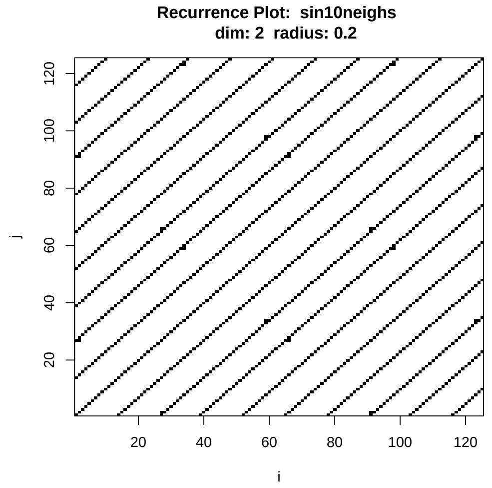


FIGURE 7. Recurrence Plot. Test case: sinus curves. Time used: 0.106 sec.

```
$ Lmax             : int 112
$ Lmean            : num 64.4
$ LmeanWithoutMain: num 61
$ ENTR             : num 2.29
$ TREND            : num 0.000113
$ LAM              : num 0.0258
$ Vmax             : int 2
$ Vmean            : num 2
```

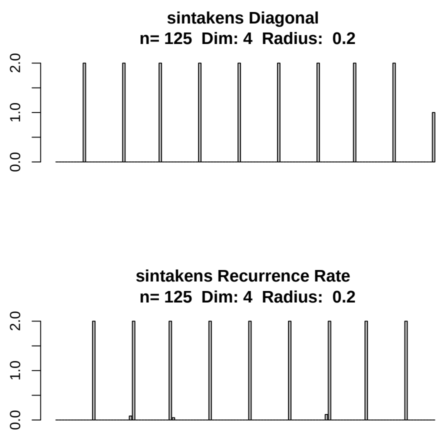


FIGURE 8. Recurrence Plot RQA. Test case: sinus curves. Time used: 0.104 sec.

6.2. Uniform random.

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

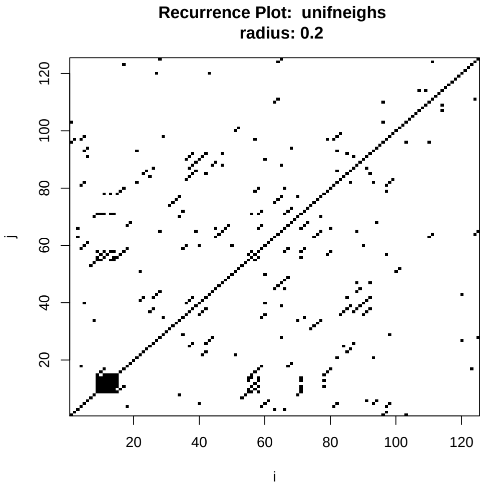


FIGURE 9. Recurrence Plot. Test case: uniform random numbers. Time used: 0.106 sec.

6.3. 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)
```

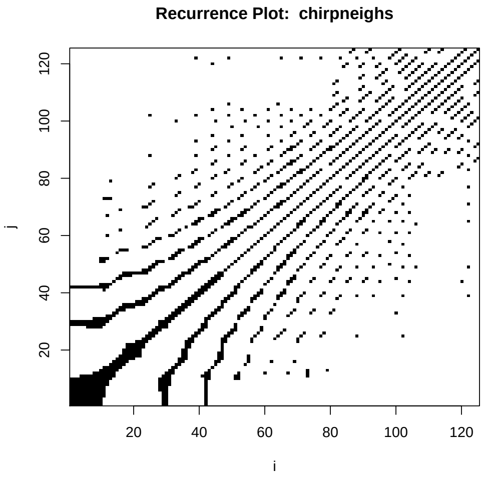


FIGURE 10. Recurrence Plot. Test case: chirp signal. Time used: 0.078 sec.

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

```
                                   Output
chirptakens  n= 125  Dim: 4
Radius: 0.6  Recurrence coverage REC: 0.092
List of 11
 $ RATIO           : num 9.75
 $ DET             : num 0.9
 $ DIV             : num 0.0217
 $ Lmax            : int 46
 $ Lmean           : num 5.18
 $ LmeanWithoutMain: num 4.7
 $ ENTR            : num 1.98
 $ TREND           : num -0.00167
 $ LAM             : num 0.459
 $ Vmax            : int 13
 $ Vmean           : num 3.23
```

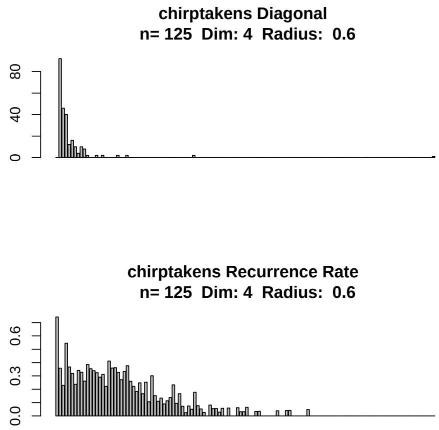


FIGURE 11. Recurrence Plot RQA. Test case: chirp signal. Time used: 0.238 sec.

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

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

7. Case Study: Geyser data

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

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

7.1. Geyser Eruptions.

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

See Figure 12,

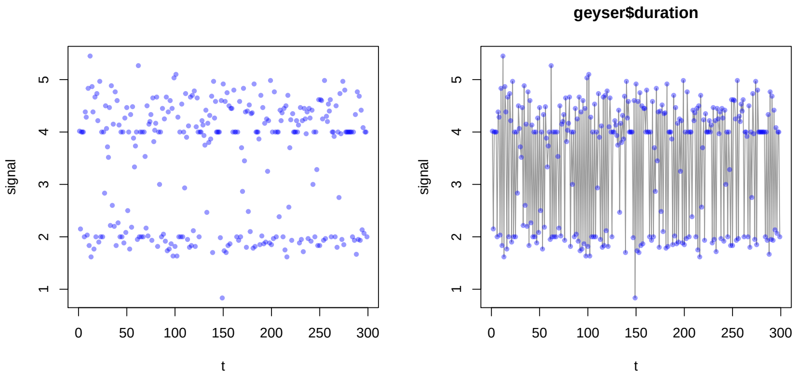


Figure 12. Example case: Old Faithful Geyser eruptions. Signal and linear interpolation.

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

See Figure 13 on the facing page

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

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

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

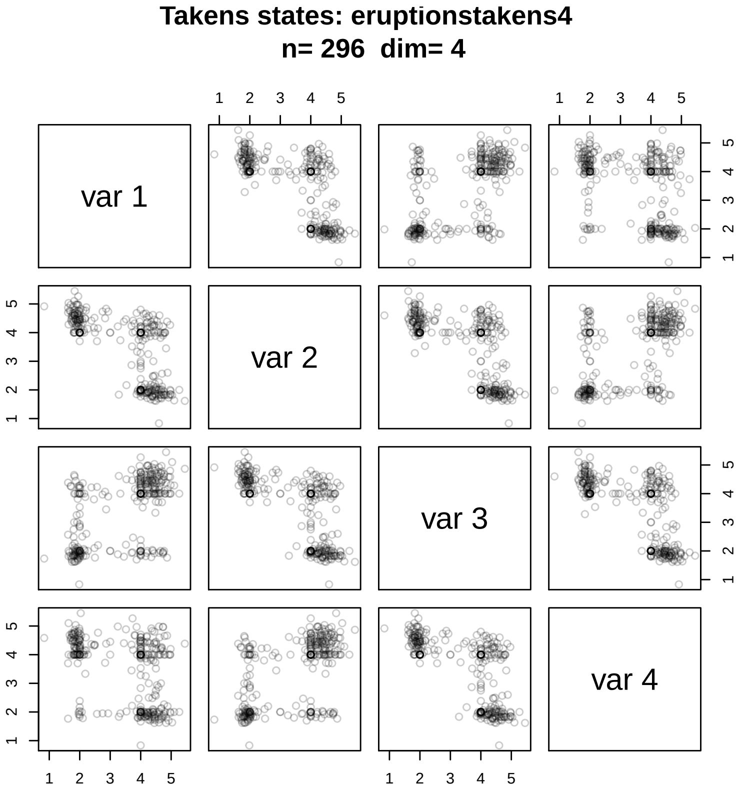


FIGURE 13. Example case: Old Faithful Geyser eruptions. Time used: 0.265 sec.

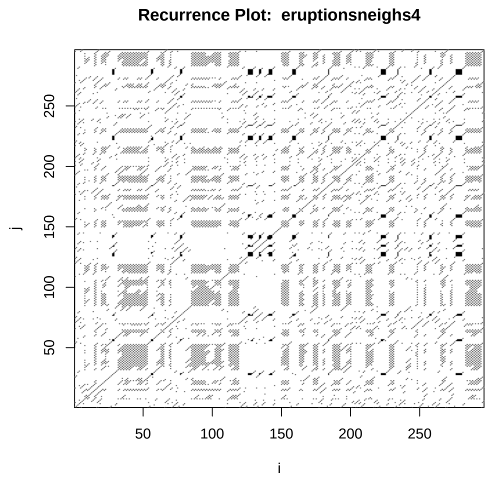


FIGURE 14. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.161 sec.

Output

```
eruptionstakens4  n= 296  Dim: 4
Radius: 0.8  Recurrence coverage REC: 0.112
List of 11
 $ RATIO           : num 8.08
```

```
$ DET               : num 0.903
$ DIV               : num 0.05
$ Lmax              : int 20
$ Lmean             : num 3.92
$ LmeanWithoutMain: num 3.79
$ ENTR              : num 1.78
$ TREND             : num 1.39e-05
$ LAM               : num 0.0764
$ Vmax              : int 5
$ Vmean             : num 3.04
```

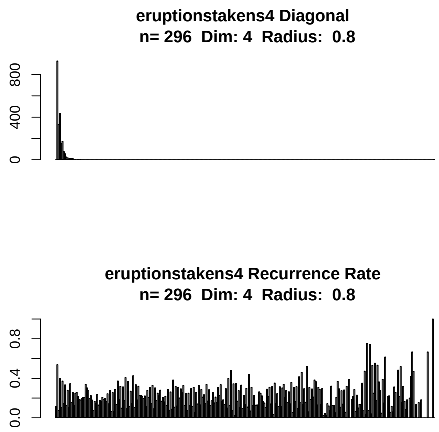


FIGURE 15. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.334 sec.

7.1.1. *Geyser eruptions. Dim=2.*

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

See Figure 16 on the next page

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

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

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

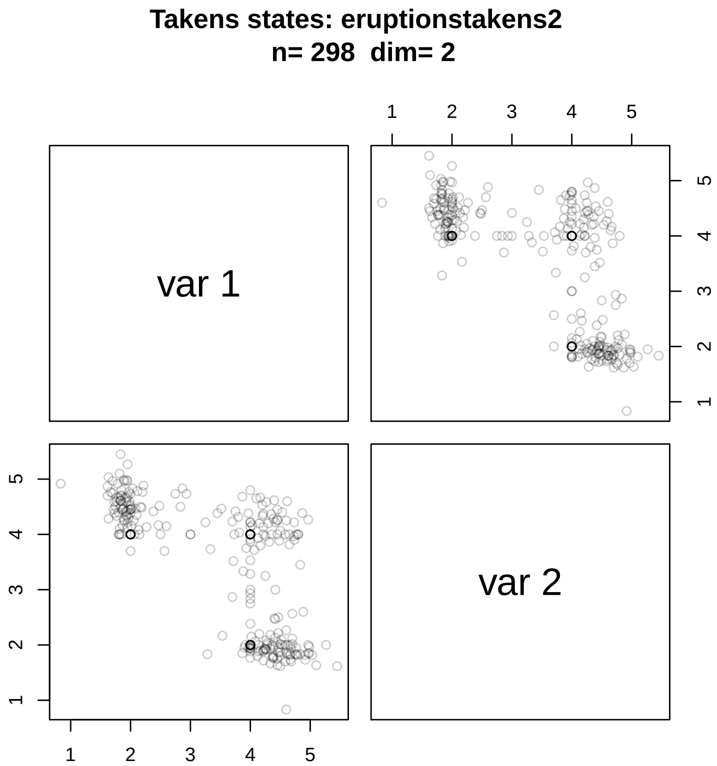


FIGURE 16. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.118 sec.

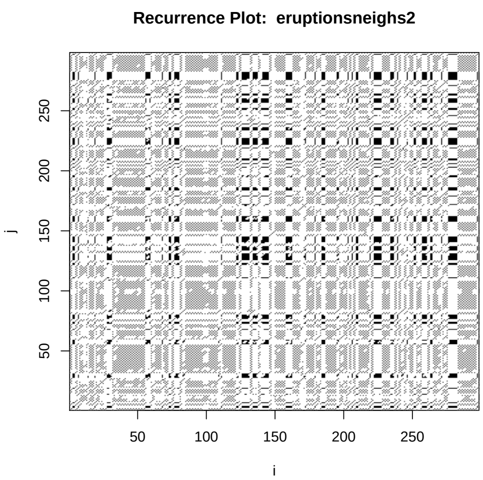


FIGURE 17. Recurrence Plot. Example case: Old Faithful Geyser Geyser eruptions. Dim=2. Time used: 0.272 sec.

Output
```
eruptionstakens2  n= 298  Dim: 2
Radius: 0.8  Recurrence coverage REC: 0.274
List of 11
 $ RATIO           : num 3.26
```

```
$ DET              : num 0.892
$ DIV              : num 0.0455
$ Lmax             : int 22
$ Lmean            : num 3.64
$ LmeanWithoutMain: num 3.59
$ ENTR             : num 1.69
$ TREND            : num 2.28e-05
$ LAM              : num 0.205
$ Vmax             : int 7
$ Vmean            : num 3.46
```

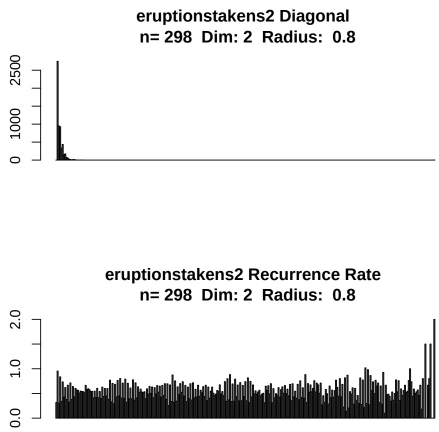


FIGURE 18. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.586 sec.

7.1.2. *Geyser eruptions. Dim=6.*

Input
```
eruptionstakens6 <- local.buildTakens( time.series=geyser$duration,embedding.dim=6,time.lag=1)
statepairs(eruptionstakens6)
```

See Figure 19 on the facing page

Input
```
eruptionsneighs6<-local.findAllNeighbours(eruptionstakens6, radius=0.8)
save(eruptionsneighs6, file="eruptionsneighs6.RData")
```

Input
```
load(file="eruptionsneighs6.RData")
local.recurrencePlotAux(eruptionsneighs6)
```

Input
```
showrqa(eruptionstakens6, radius=0.8)
```

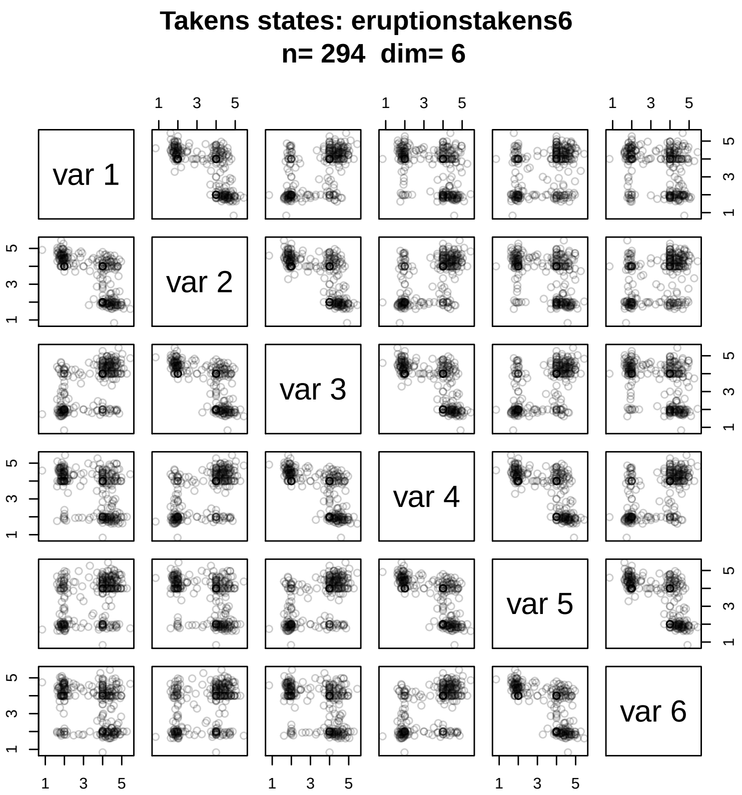


FIGURE 19. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.472 sec.

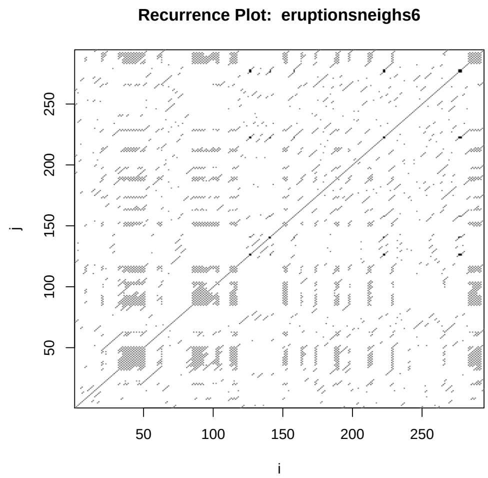


FIGURE 20. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.136 sec.

Output

```
eruptionstakens6  n= 294  Dim: 6
Radius: 0.8  Recurrence coverage REC: 0.05
List of 11
 $ RATIO           : num 18.4
```

```
 $ DET              : num 0.923
 $ DIV              : num 0.0556
 $ Lmax             : int 18
 $ Lmean            : num 4.02
 $ LmeanWithoutMain: num 3.73
 $ ENTR             : num 1.75
 $ TREND            : num 1.16e-06
 $ LAM              : num 0.0164
 $ Vmax             : int 3
 $ Vmean            : num 2.29
```

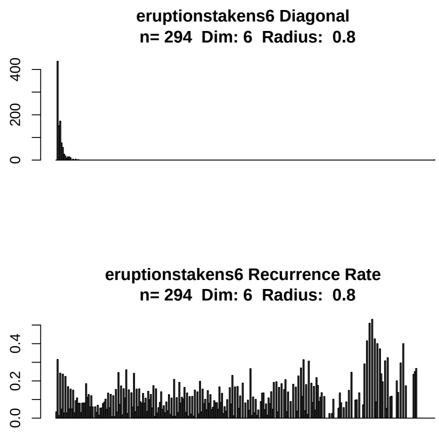


FIGURE 21. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.344 sec.

7.1.3. *Geyser eruptions. Dim=8.*

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

See Figure 22 on the next page

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

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

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

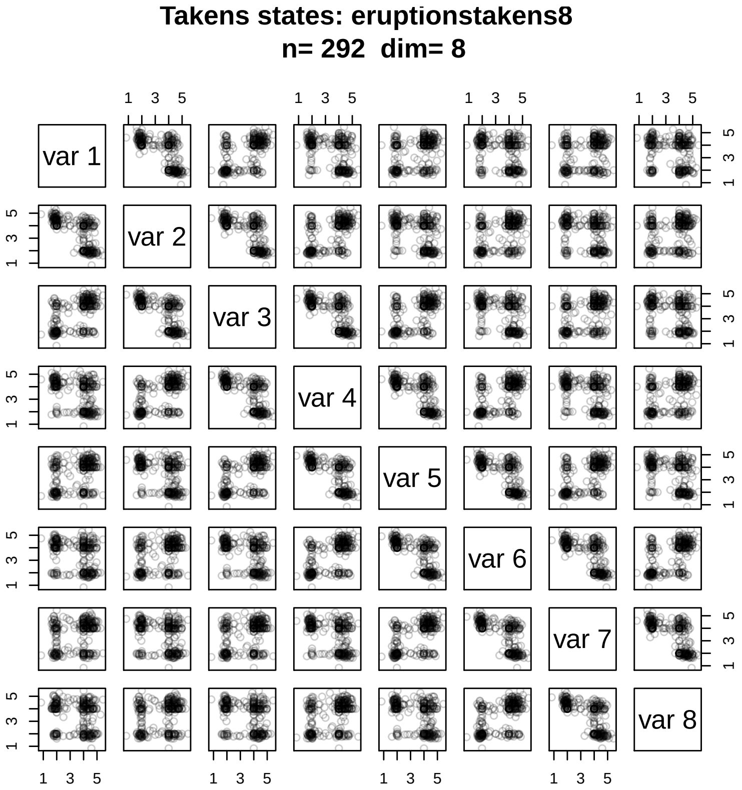


FIGURE 22. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.83 sec.

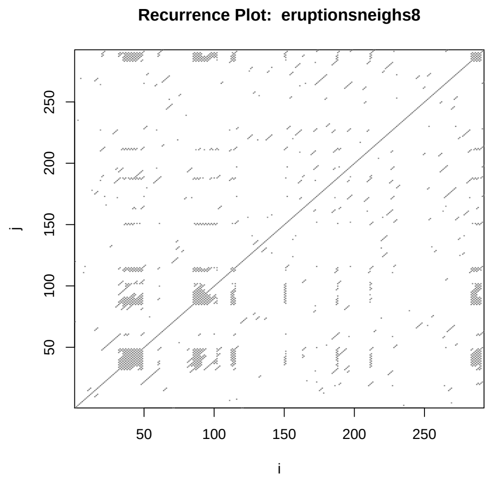


FIGURE 23. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.121 sec.

Output
```
eruptionstakens8  n= 292  Dim: 8
Radius: 0.8  Recurrence coverage REC: 0.024
List of 11
 $ RATIO           : num 39.2
```

```
 $ DET              : num 0.924
 $ DIV              : num 0.0625
 $ Lmax             : int 16
 $ Lmean            : num 4.58
 $ LmeanWithoutMain: num 3.87
 $ ENTR             : num 1.8
 $ TREND            : num 5.57e-06
 $ LAM              : num 0
 $ Vmax             : num 0
 $ Vmean            : num 0
```

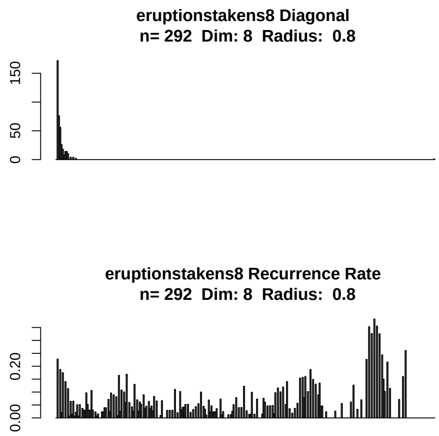


FIGURE 24. Recurrence Plot RQA. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.285 sec.

7.2. **Geyser Eruptions: Comparison by Dimension.** For comparison, recurrence plots for the Geyser data with varying dimension are in Figure 25 on the facing page

7.3. **Geyser Waiting.**

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

See Figure 26 on the next page,

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

See Figure 27 on page 24

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

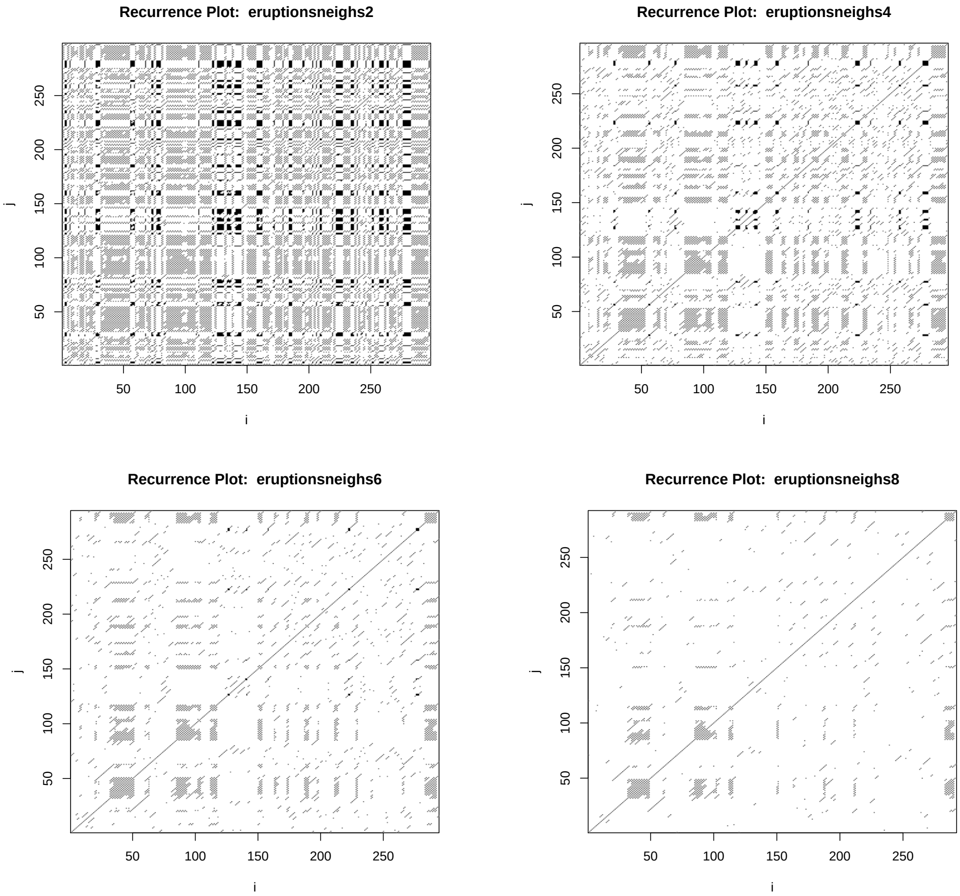


FIGURE 25. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=2, 4, 6, 8.

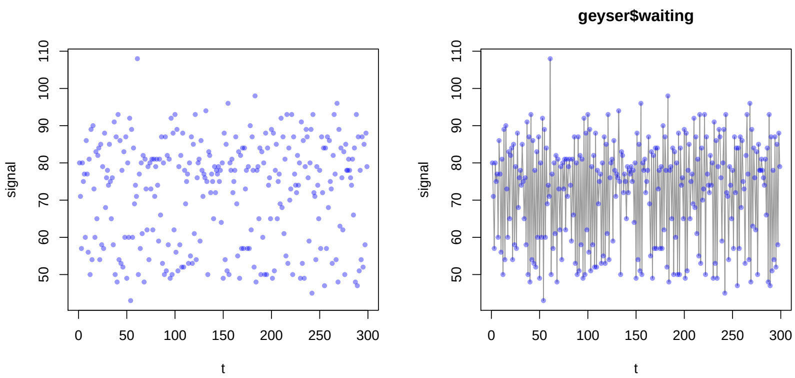


FIGURE 26. Example case: Old Faithful Geyser waiting. Signal and linear interpolation. Time used: 0.447 sec.

Input

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

Output

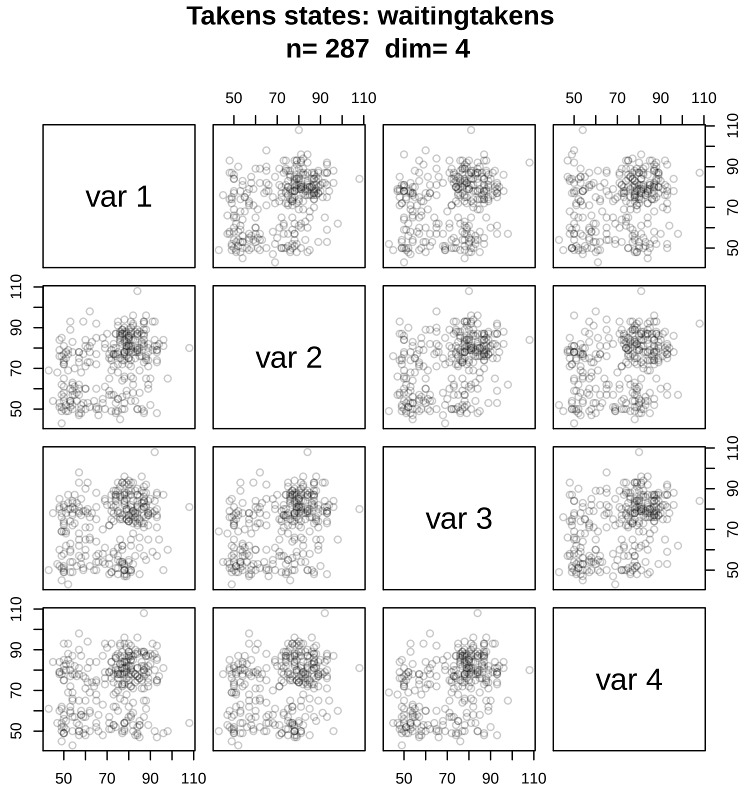


FIGURE 27. Example case: Old Faithful Geyser waiting. Time used: 0.264 sec.

```
waitingtakens  n= 287  Dim: 4
Radius: 16  Recurrence coverage REC: 0.137
List of 11
 $ RATIO             : num 2.8
 $ DET               : num 0.382
 $ DIV               : num 0.0526
 $ Lmax              : int 19
 $ Lmean             : num 2.88
 $ LmeanWithoutMain: num 2.69
 $ ENTR              : num 1
 $ TREND             : num -0.000128
 $ LAM               : num 0.0529
 $ Vmax              : int 3
 $ Vmean             : num 2.32
```

Input

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

7.4. **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.

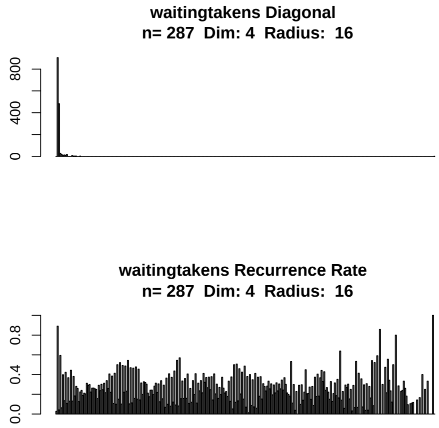


FIGURE 28. Recurrence Plot RQA. Example case: Old Faithful Geyser waiting. Time used: 0.168 sec.

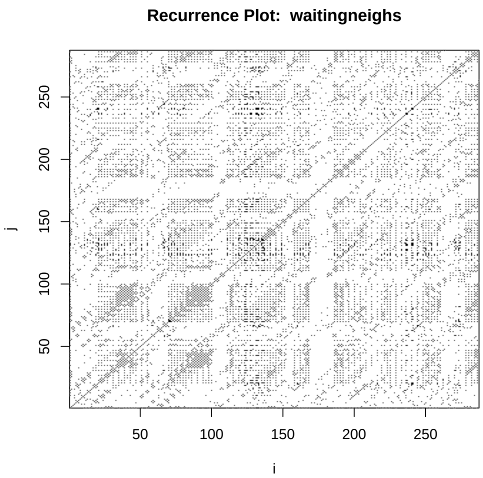


FIGURE 29. Recurrence Plot. Example case: Old Faithful Geyser waiting. Time used: 0.356 sec.

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

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.

7.5. Geyser Eruptions linearized.

Input
```
plotsignal(geyserlin)
```

See Figure 30,

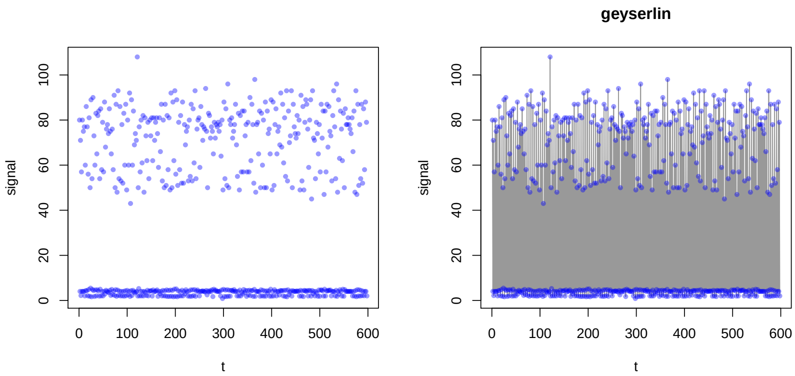


FIGURE 30. Example case: Old Faithful Geyser eruptions. Signal and linear interpolation.

Input
```
gleruptionstakens4 <-
        local.buildTakens( time.series=geyserlin, embedding.dim=4, time.lag=2)
statepairs(gleruptionstakens4)
```

See Figure 31 on the facing page

Input
```
eruptionsneighs4<-local.findAllNeighbours(gleruptionstakens4, radius=0.8)
save(eruptionsneighs4, file="eruptionsneighs4.RData")
```

Input
```
showrqa(gleruptionstakens4, radius=0.8)
```

Output
```
gleruptionstakens4  n= 592  Dim: 4
Radius: 0.8  Recurrence coverage REC: 0.029
List of 11
 $ RATIO            : num 2.04
 $ DET              : num 0.0587
 $ DIV              : num Inf
 $ Lmax             : num 0
 $ Lmean            : num 592
 $ LmeanWithoutMain: num NaN
 $ ENTR             : num 0
 $ TREND            : num 1.26e-06
 $ LAM              : num 0
 $ Vmax             : num 0
 $ Vmean            : num 0
```

See Figure 32 on the next page.

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

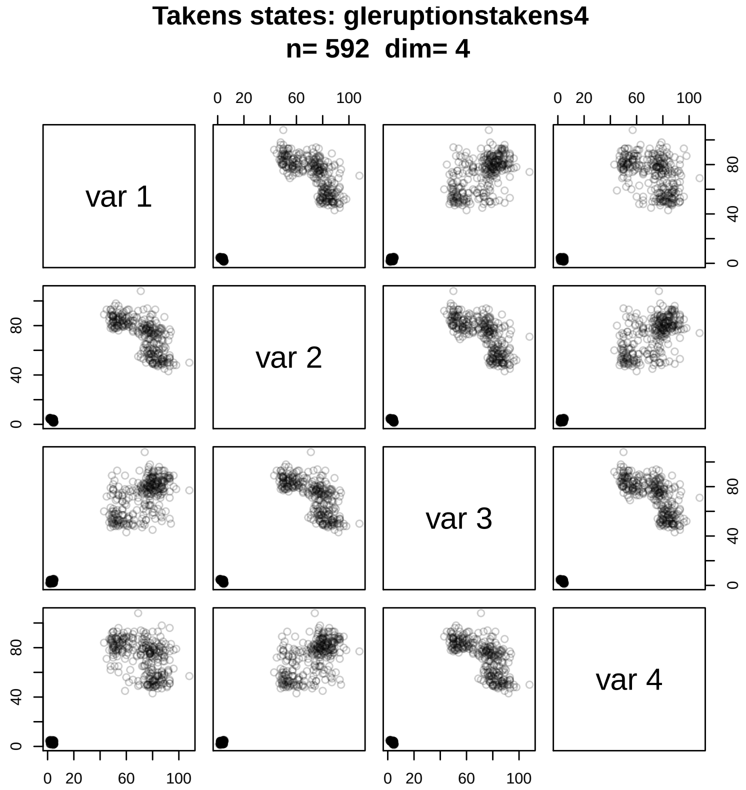


FIGURE 31. Example case: Old Faithful Geyser eruptions. Time used: 0.413 sec.

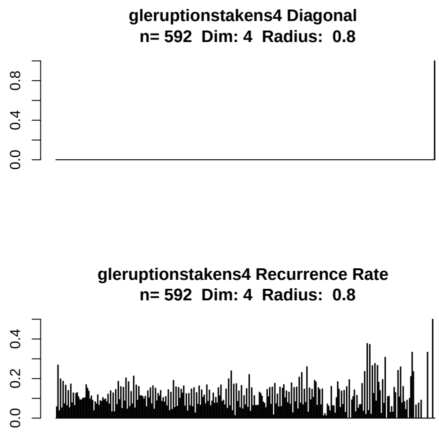


FIGURE 32. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.244 sec.

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

Input

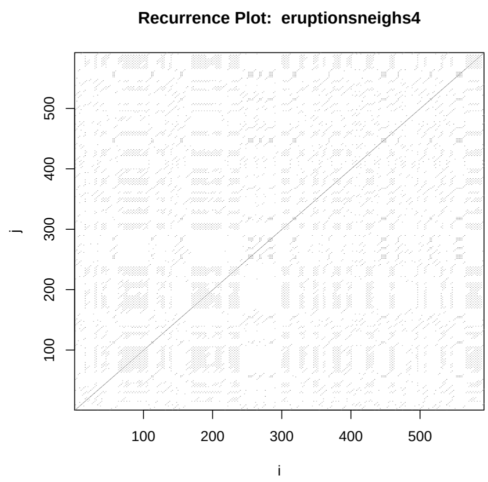


FIGURE 33. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 0.59 sec.

```
gleruptionstakens2 <-
        local.buildTakens(time.series=geyserlin, embedding.dim=2, time.lag=2)
statepairs(gleruptionstakens2)
```

See Figure 34 on the facing page

Input
```
eruptionsneighs2<-local.findAllNeighbours(gleruptionstakens2, radius=0.8)
save(eruptionsneighs2, file="eruptionsneighs2.RData")
```

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

7.5.2. *Geyser eruptions - linearized. Dim=6.*

Input
```
gleruptionstakens6 <- local.buildTakens( time.series=geyserlin,embedding.dim=6,time.lag=2)
statepairs(gleruptionstakens6)
```

See Figure 36 on page 30

Input
```
eruptionsneighs6<-local.findAllNeighbours(gleruptionstakens6, radius=0.8)
save(eruptionsneighs6, file="eruptionsneighs6.RData")
```

Input
```
load(file="eruptionsneighs6.RData")
local.recurrencePlotAux(eruptionsneighs6)
```

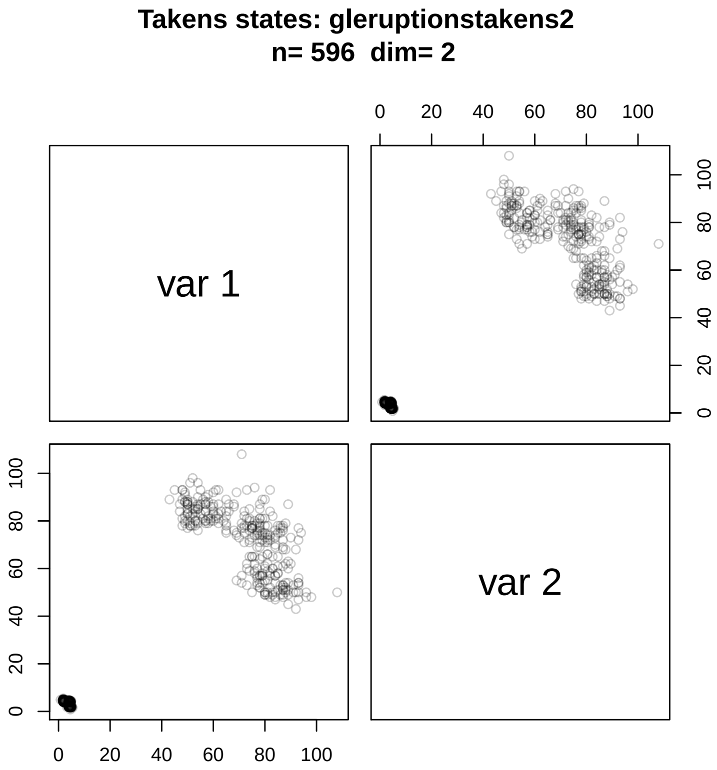


FIGURE 34. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.136 sec.

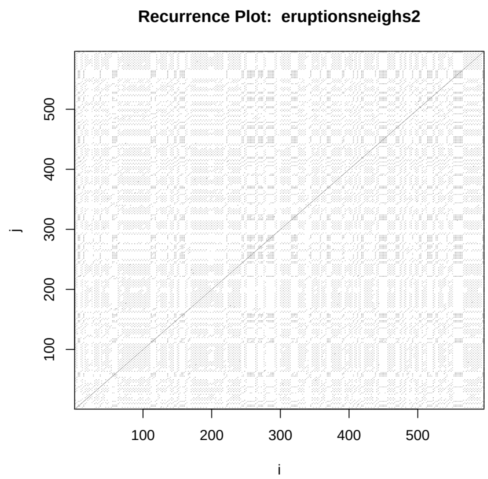


FIGURE 35. Recurrence Plot. Example case: Old Faithful Geyser eruptions linearized. Dim=2. Time used: 0.378 sec.

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

Input

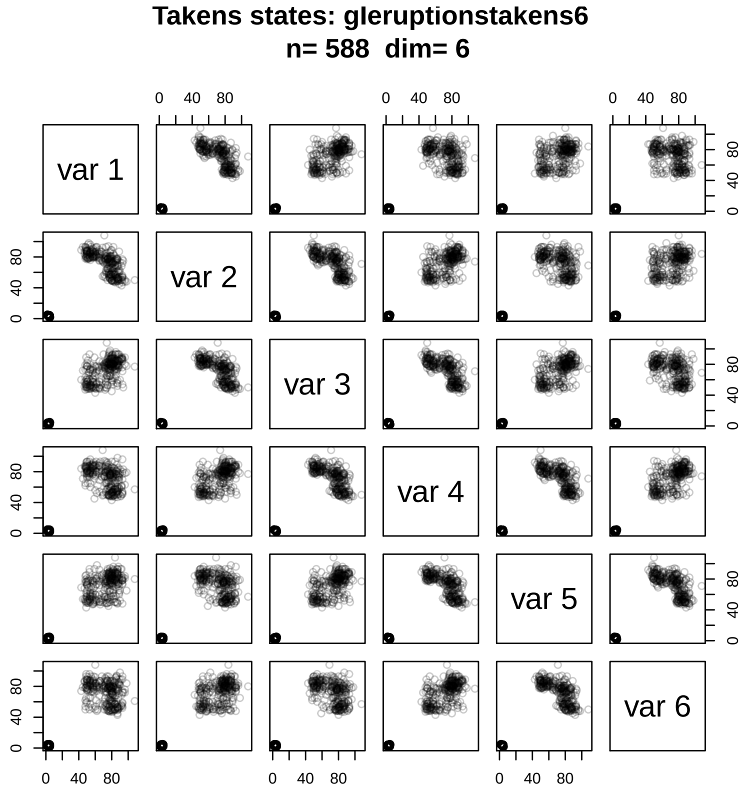


FIGURE 36. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.837 sec.

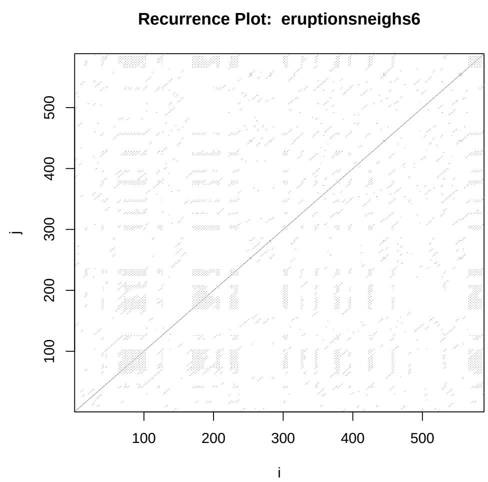


FIGURE 37. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.35 sec.

```
gleruptionstakens8 <- local.buildTakens( time.series=geyserlin,embedding.dim=8,time.lag=2)
statepairs(gleruptionstakens8)
```

See Figure 38 on the facing page

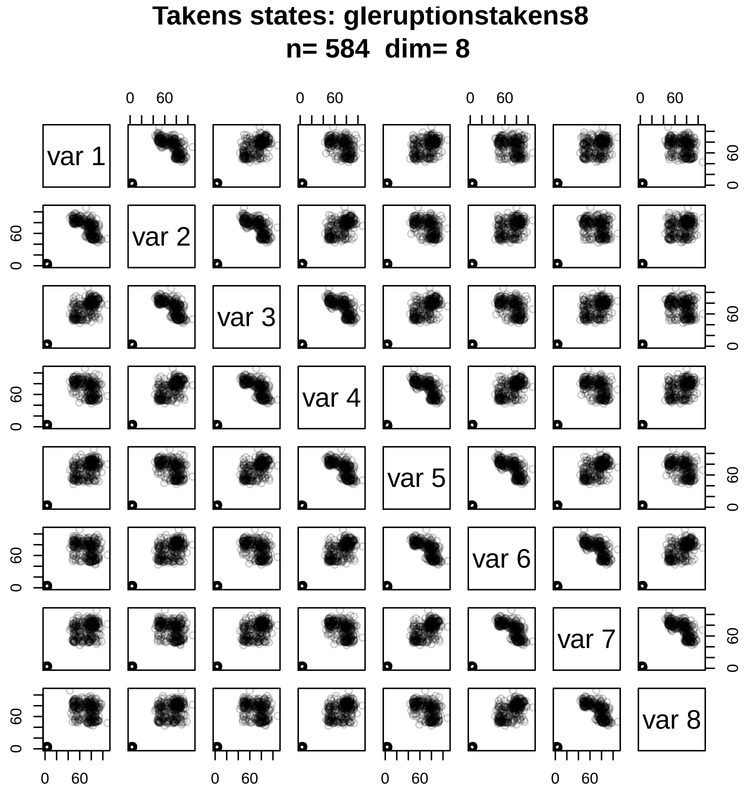


FIGURE 38. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 1.427 sec.

Input
```
eruptionsneighs8<-local.findAllNeighbours(gleruptionstakens8, radius=0.8)
save(eruptionsneighs8, file="eruptionsneighs8.RData")
```

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

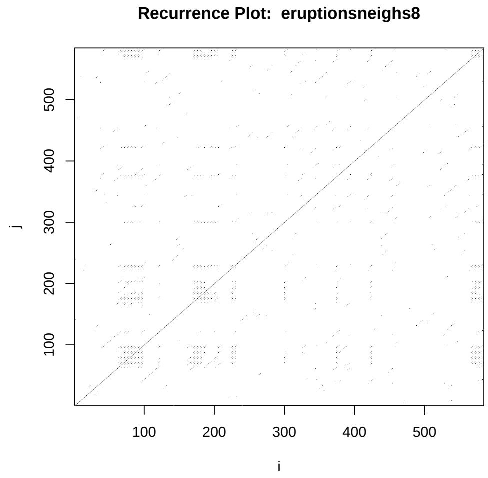


Figure 39. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.33 sec.

8. Case Study: HRV data

Only 128 data points used in this section

Input
```
library(RHRV)
load("/data/pulse/rhrv/pkg/data/HRVData.rda")
load("/data/pulse/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 = "/data/pulse/rhrv/tutorial/beatsFolder")
```

Output
```
** Loading beats positions for record: example.beats **
   Path: /data/pulse/rhrv/tutorial/beatsFolder
   Scale: 1
   Date: 01/01/1900
   Time: 00:00:00
   Number of beats: 17360
```

Input
```
#       RecordPath = "beatsFolder")


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

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

Input

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

See Figure 40 on the next page,

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

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

See Figure 41 on the following page

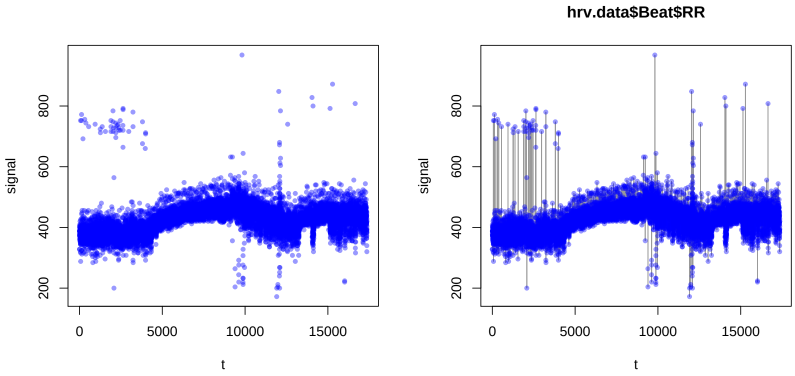


FIGURE 40. RHRV tutorial example.beats. Signal and linear interpolation.

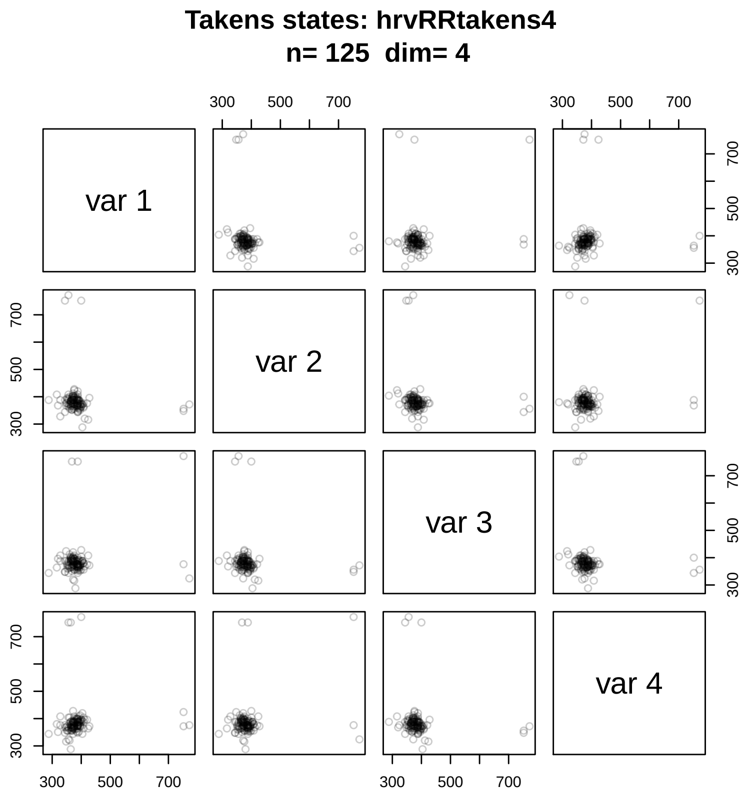


FIGURE 41. RHRV tutorial example.beats. Time used: 0.185 sec.

Input
```
statepairs(hrvRRtakens4, rank=TRUE)
```

See Figure 42 on the next page.

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

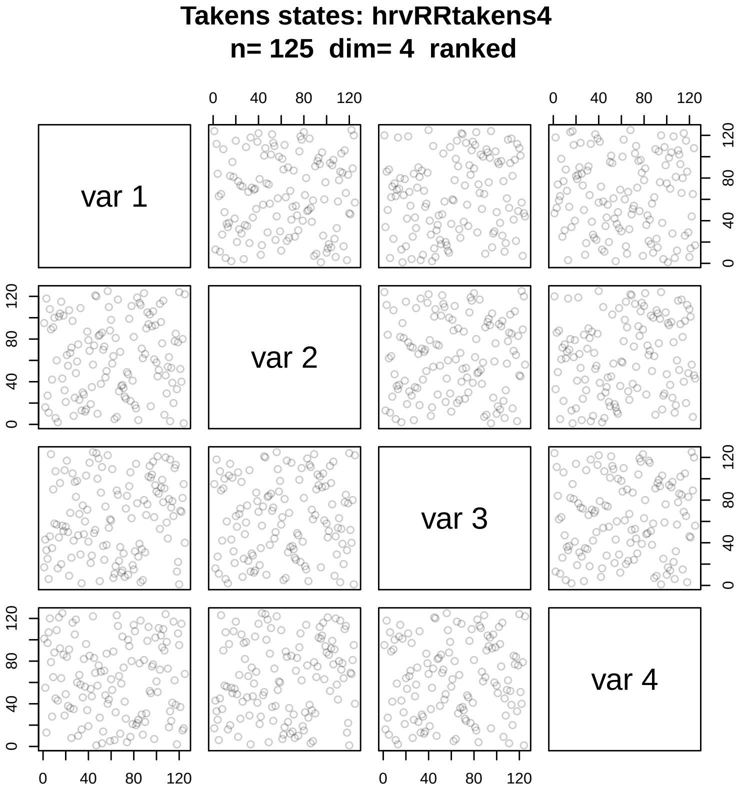


FIGURE 42. RHRV tutorial example.beats. Ranked data. Time used: 0.399 sec.

Time used: 0.023 sec.

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

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

8.1. RHRV: 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: 0.041 sec.

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

Time used: 0.104 sec.

Input

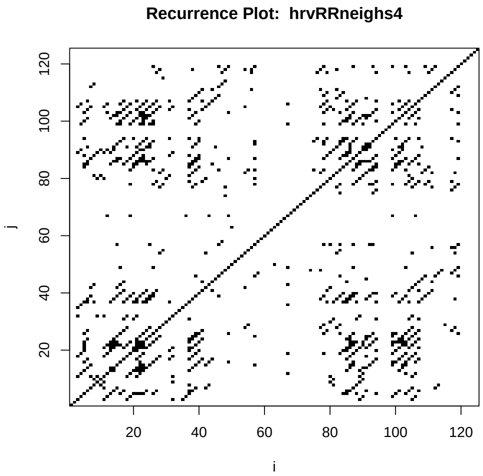


FIGURE 43. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=4. Time used: 0.225 sec.

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

Time used: 0.039 sec.

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

Dim=6. Time used: 0.079 sec.

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

Time used: 0.041 sec.

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

Dim=8. Time used: 0.103 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: 0.148 sec.

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

Time used: 0.125 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: 0.168 sec.

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

Time used: 0.083 sec.

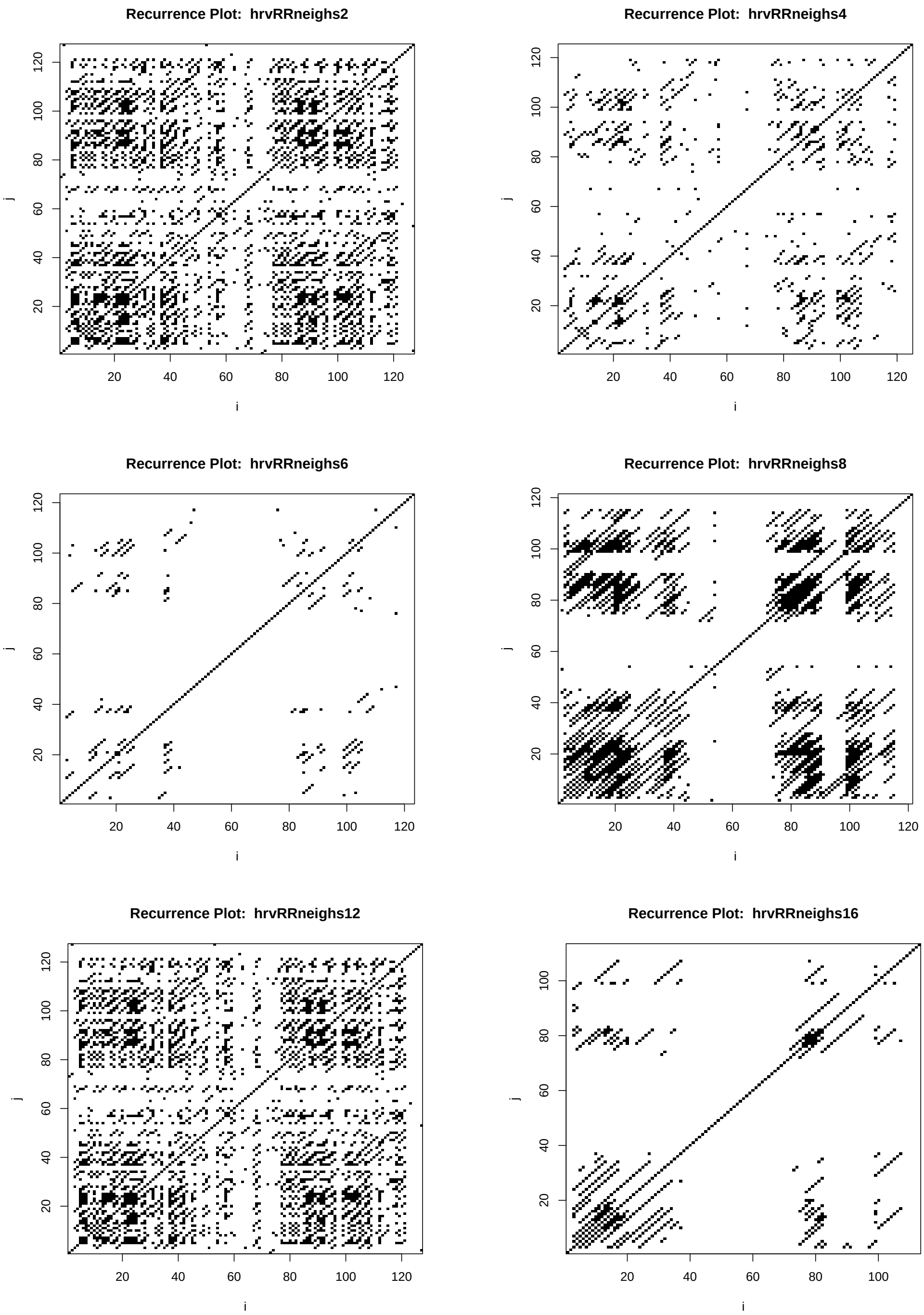


FIGURE 44. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 0.083 sec.

ToDo: Consider using differences

8.2. **Hart Rate Variation.** Since we are not interested in heart rate (or pulse), but in heart rate variation, a proposal is to use scaled differences.

Input
```
# source('/data/pulse/rhrv/pkg/R/BuildNIHR2.R', chdir = TRUE)
BuildNIDHR <-
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 hel
                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!! -
                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 <- BuildNIDHR(hrv.data)
```

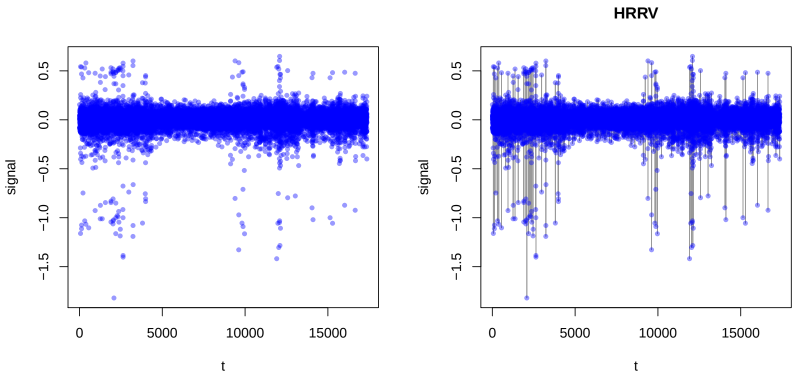
Output
```
** Calculating non-interpolated heart rate differences **
   Number of beats: 17360
```

Input
```
HRRV <- hrv.data$Beat$HRRV
```

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

Input
```
plotsignal(HRRV)
```

See Figure 45,



Only 128 data points used in this section

FIGURE 45. RHRV tutorial example.beats. HRRV Signal and linear interpolation.

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

See Figure 46 on the next page

Input
```
statepairs(hrvRRVtakens4, rank=TRUE)
```

ToDo: findAllNeighbours does not handle NAs

See Figure 47 on page 42

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

Time used: 0.023 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. Eckmann uses nearest neighbours with NN=10

8.3. **RHRV Variation: Comparison by Dimension.**

Input
```
hrvRRVtakens2 <- local.buildTakens( time.series=HRRV[1:nsignal], embedding.dim=2, time.lag=1)
hrvRRVneighs2 <-local.findAllNeighbours(hrvRRVtakens2[-(1:2),], radius=0.125)
save(hrvRRVneighs2, file="hrvRRVneighs2.Rdata")
```

Time used: 0.04 sec.

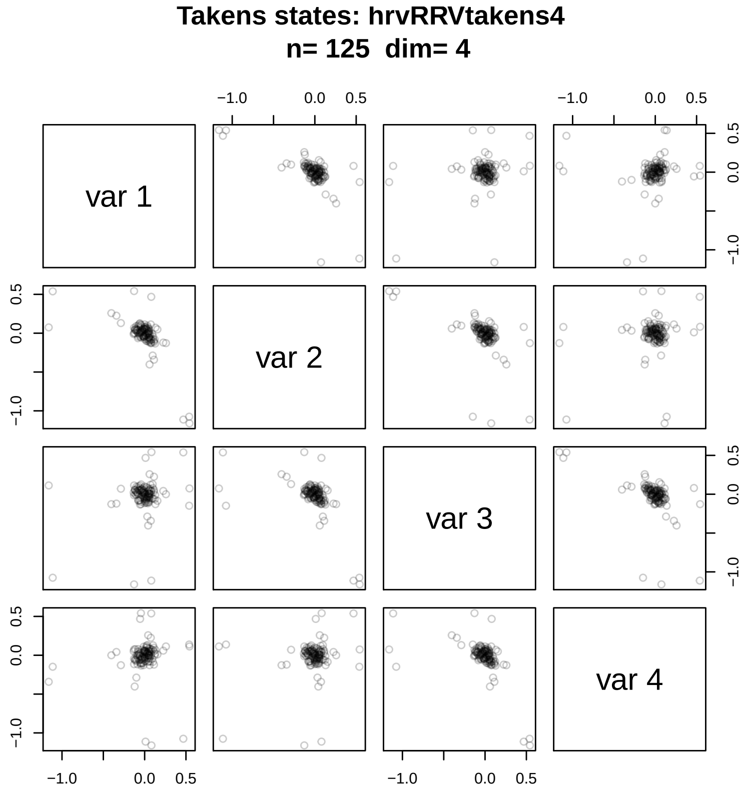


FIGURE 46. RHRV tutorial example.beats. HRRV Time used: 0.172 sec.

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

Time used: 0.127 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: 0.039 sec.

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

Dim=6. Time used: 0.121 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")
```

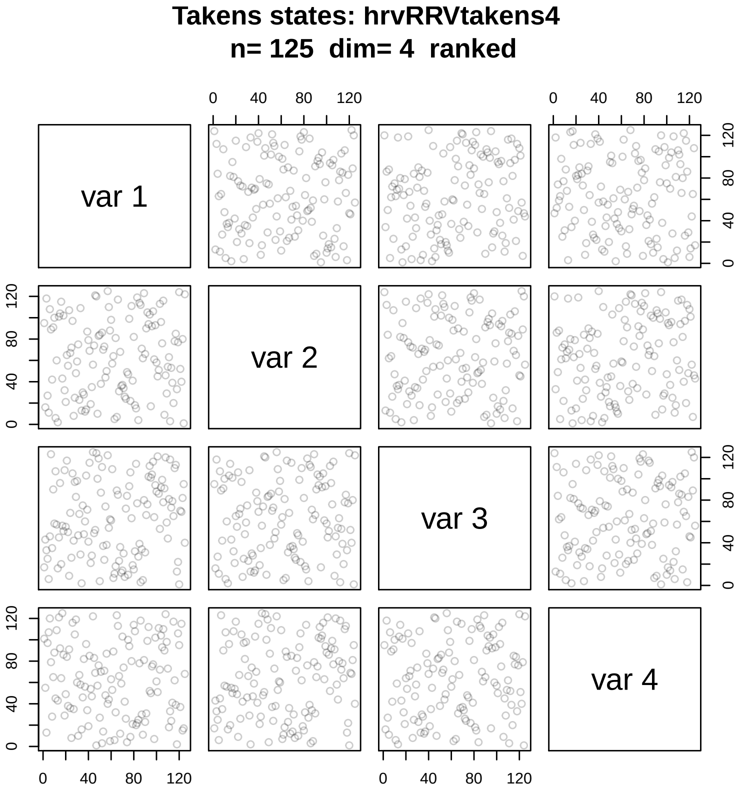


FIGURE 47. RHRV tutorial example.beats. Ranked HRRV data. Time used: 0.393 sec.

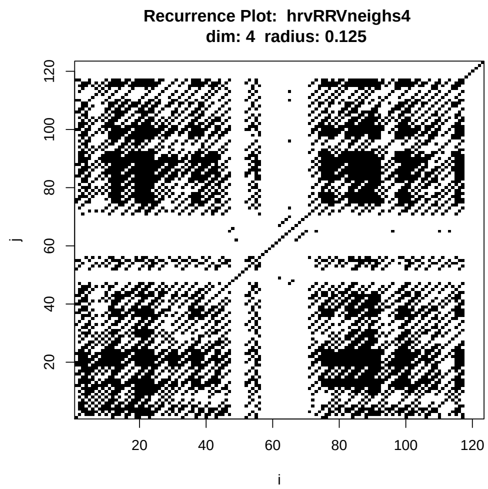


FIGURE 48. Recurrence Plot. Example case: RHRV tutorial example.beats. HRRV Dim=4. Time used: 0.134 sec.

Time used: 0.036 sec.

Input

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

Dim=8. Time used: 0.097 sec.

Input
```
hrvRRVtakens12 <-
        local.buildTakens( time.series=HRRV[1:nsignal],embedding.dim=12,time.lag=1)
hrvRRVneighs12 <-
        local.findAllNeighbours(hrvRRVtakens12[-(1:2),], radius=0.125)
save(hrvRRVneighs12, file="hrvRRVneighs12.Rdata")
```

Time used: 0.146 sec.

Input
```
load(file="hrvRRVneighs12.RData")
local.recurrencePlotAux(hrvRRVneighs12, dim=12,  radius=0.125)
```

Time used: 0.089 sec.

Input
```
hrvRRVtakens16 <- local.buildTakens( time.series=HRRV[1:nsignal],embedding.dim=16,time.lag=1)
hrvRRVneighs16 <-local.findAllNeighbours(hrvRRVtakens16[-(1:2),], radius=0.125)
save(hrvRRVneighs16, file="hrvRRVneighs16.Rdata")
```

Time used: 0.136 sec.

Input
```
load(file="hrvRRVneighs16.RData")
local.recurrencePlotAux(hrvRRVneighs16, dim=16, radius=0.125)
```

Time used: 0.076 sec.

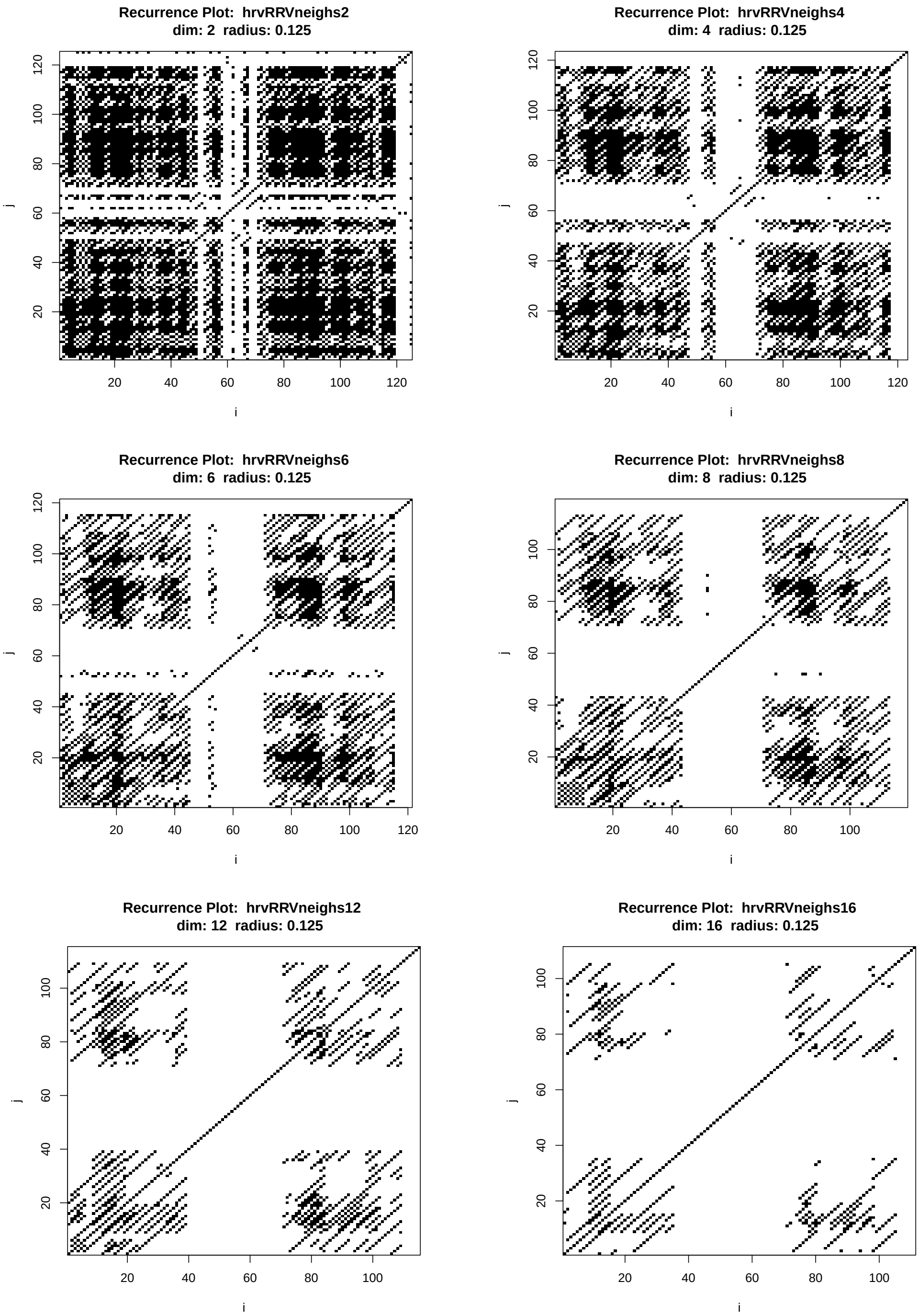


FIGURE 49. Recurrence Plot. Example case: RHRV tutorial example.beats. Dim=2, 4, 6, 8, 12, 16. Time used: 0.077 sec.

References

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

Webber Jr CL, Zbilut JP (2005). "Recurrence quantification analysis of nonlinear dynamical systems." *Tutorials in contemporary nonlinear methods for the behavioral sciences*, pp. 26–94.

Zbilut JP, Webber CL (2006). "Recurrence quantification analysis." *Wiley encyclopedia of biomedical engineering*. URL `http://onlinelibrary.wiley.com/doi/10.1002/9780471740360.ebs1355/full`.

Index

R session info:

Total Sweave time used: 18.929 sec. at Mon Feb 17 23:01:20 2014.

- R version 3.0.2 (2013-09-25), x86_64-apple-darwin10.8.0
- 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 7.3-29, Matrix 1.1-2, mgcv 1.7-28, nlme 3.1-113, nonlinearTseries 0.2, rgl 0.93.996, 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 3.0.2, lattice 0.20-25, quadprog 1.5-5, tools 3.0.2, zoo 1.7-11

LaTeX information:

textwidth: 5.37607in linewidth:5.37607in

textheight: 9.21922in

Bibliography style: jss

CVS/Svn repository information:

```
$Source: /u/math/j40/cvsroot/lectures/src/dataanalysis/Rnw/recurrence.Rnw,v $
$HeadURL: svnssh://gsawitzki@scm.r-forge.r-project.org/svnroot/rhrv/gs/recurrence.Rnw +
$Revision: 1.7 $
$Date: 2014/02/17 22:00:31 $
$name:  $
$Author: j40 $
```

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

Günther Sawitzki
StatLab Heidelberg
Im Neuenheimer Feld 294
D 69120 Heidelberg