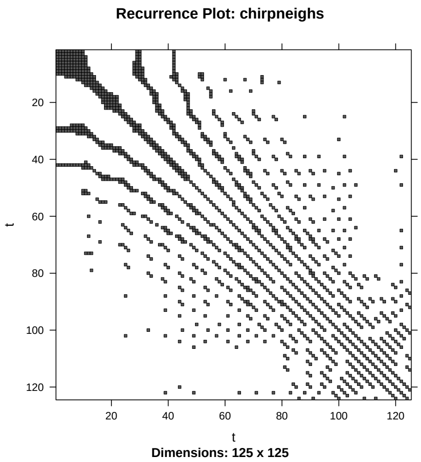
STATISTICAL DATA ANALYSIS: RECURRENCE PLOT

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 8, 2014 from cvs *Revision* : 1.2
`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)
```

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

Input
```
statepairs <- function(states, rank=FALSE){
        main <- paste("Takens states:",deparse(substitute(states)))
        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

Input
```
#local.recurrencePlotAux <- nonlinearTseries:::recurrencePlotAux
local.recurrencePlotAux=function(neighs){
  ntakens=length(neighs)
  neighs.matrix = nonlinearTseries:::neighbourListSparseNeighbourMatrix(neighs,ntakens)
  # need a print because it is a trellis object!!
  print(
          image(neighs.matrix,xlab="t", ylab="t",
                main=paste("Recurrence Plot:",
                        deparse(substitute(neighs))
                        )
                )
        )
}
```

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 representation, we use a common layout.

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

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

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)
```

See Figure 2 on the next page,

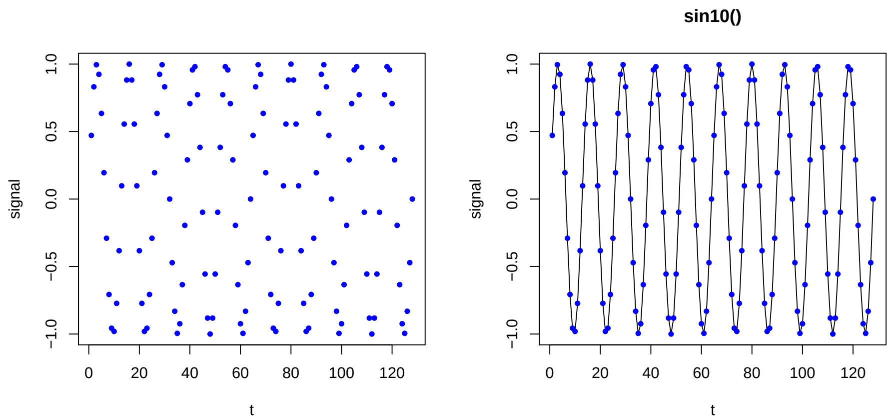


FIGURE 1. Test case: sin10. Signal and linear interpolation.

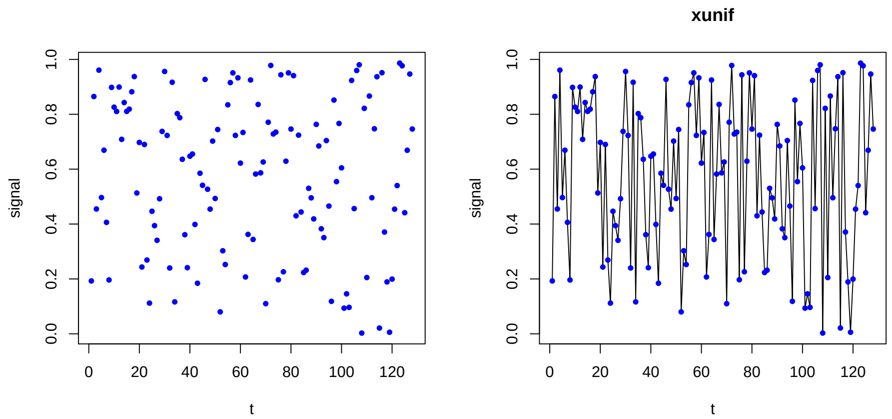


FIGURE 2. Test case: unif - uniform random numbers. Signal and linear interpolation.

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,

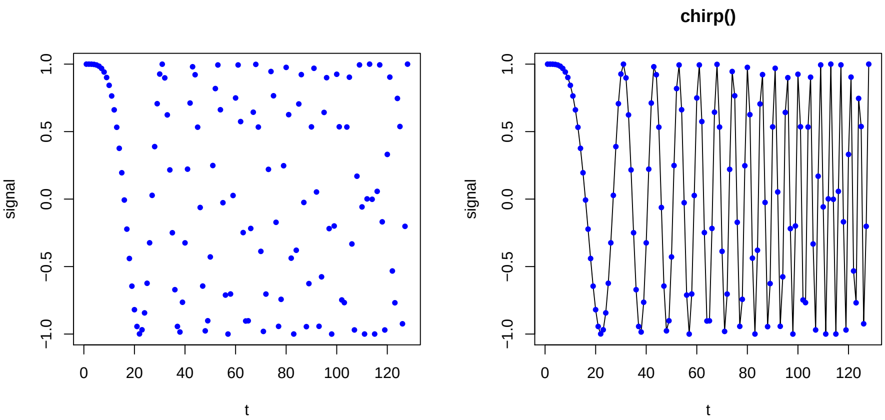


FIGURE 3. Test case: chirp signal. Signal and linear interpolation.

ToDo:
doppler wav

3. Recurrence States

Recurrence plots have been introduced in an attempt to understand near periodic in hydrodynamics. On the one hand, and etended 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 ist 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-anticpating) 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.

Recurrency 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 flattenened representation of t dimensional data. So you take

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

We ignore this detail here and take $m = 1$.

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 comles 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 staes 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 on the next page.

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

See Figure 5 on page 8.

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

See Figure 6 on page 9

4. 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 staes 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 evnironment 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.

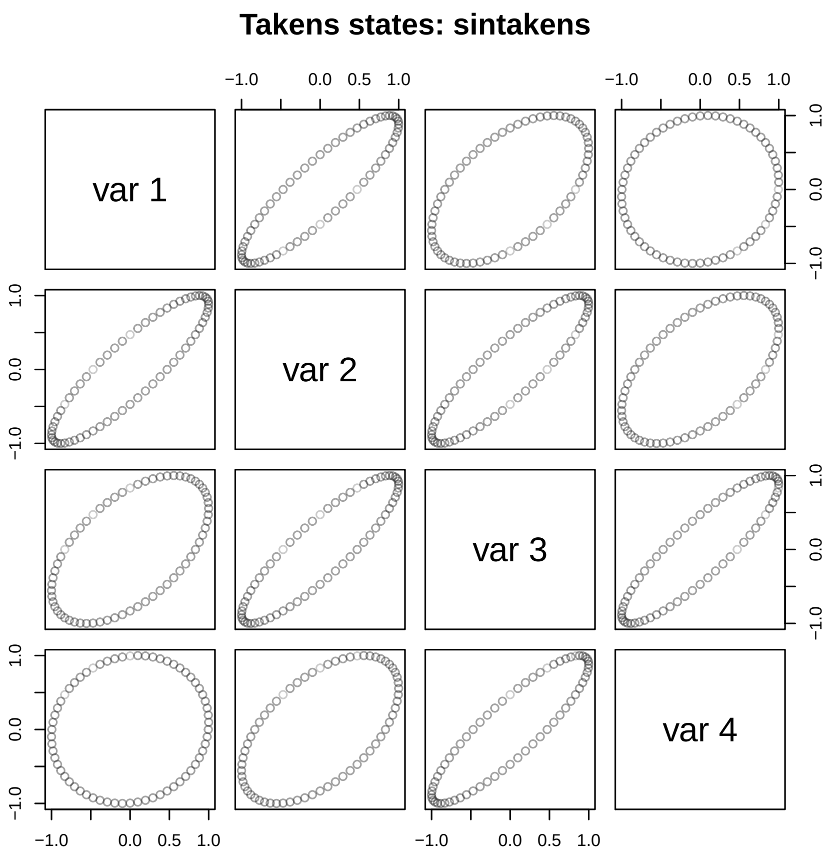


FIGURE 4. Test case: sinus. Note that marginal views of 1-dimensional circles in d space may appear as ellipses. Time used: 0.188 sec.

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.

4.1. Sinus.

Input

```
load(file="sin10neighs.RData")
local.recurrencePlotAux(sin10neighs)
```

4.2. Uniform random.

Input

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

4.3. Chirp Signal.

Input

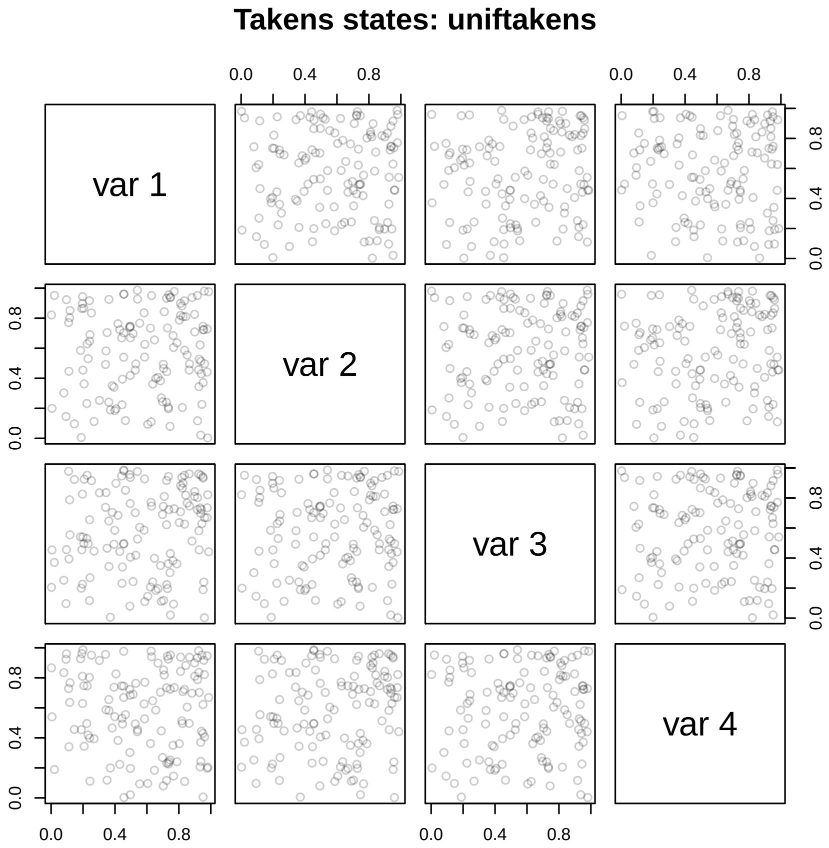


FIGURE 5. Test case: uniform random numbers. Time used: 0.209 sec.

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

Input

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

5. Case Study: Geyser data

ToDo: extend to two-dimensional data

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

Input

```
library(MASS)
data(faithful)
```

5.1. Geyser Eruptions.

Input

```
plotsignal(faithful$eruptions)
```

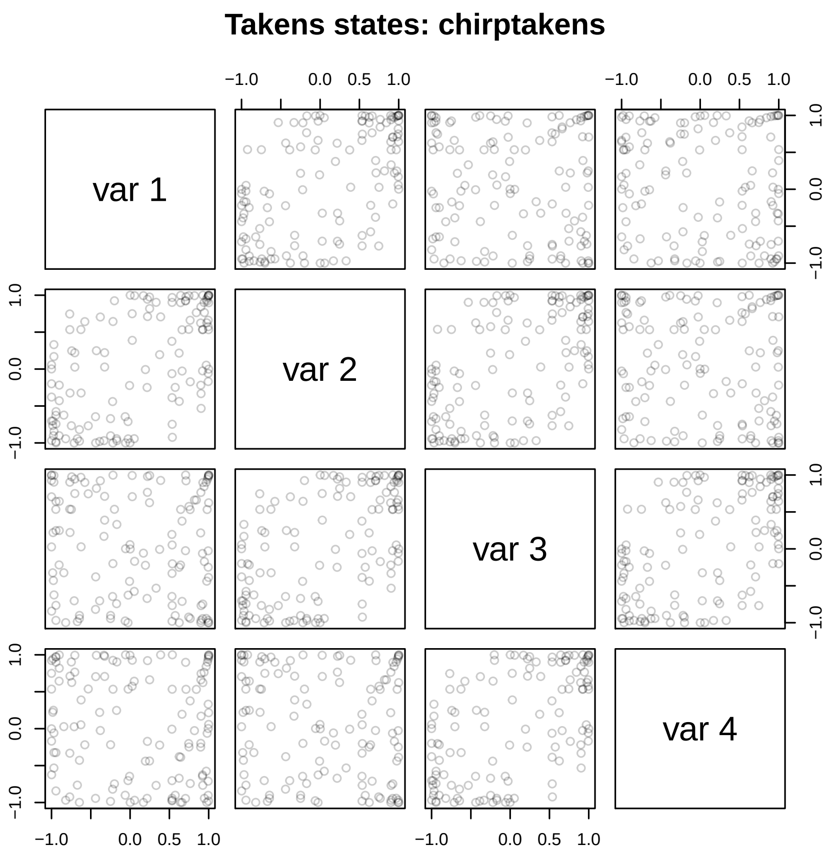


FIGURE 6. Test case: chirp signal. Time used: 0.171 sec.

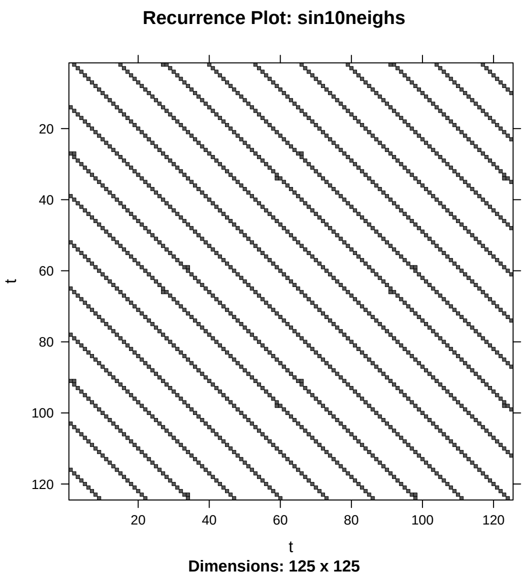


FIGURE 7. Recurrence Plot. Test case: sinus curves. Time used: 4.684 sec.

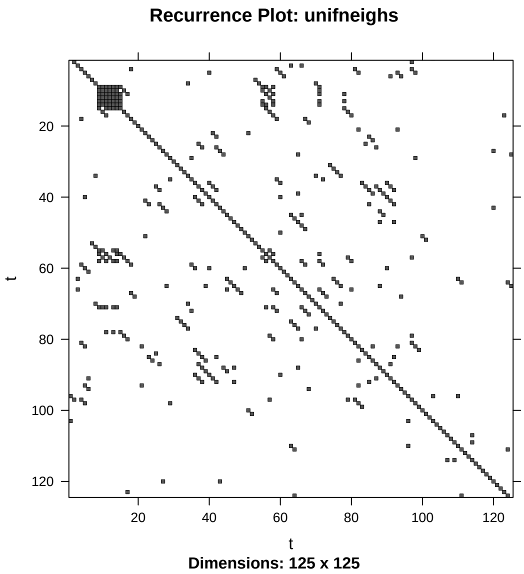


FIGURE 8. Recurrence Plot. Test case: uniform random numbers. Time used: 1.41 sec.

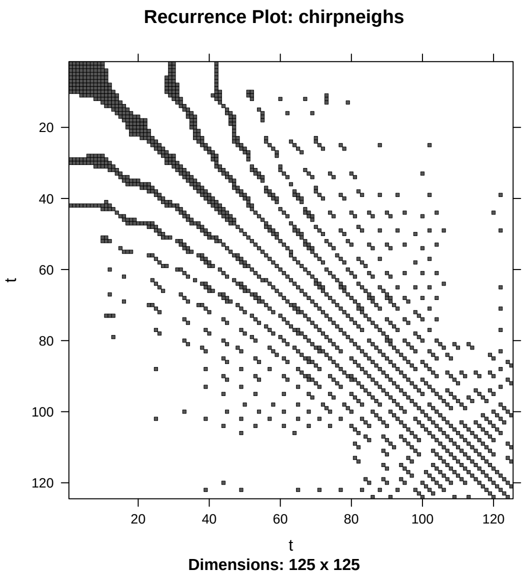


FIGURE 9. Recurrence Plot. Test case: chirp signal. Time used: 4.818 sec.

See Figure 10 on the facing page,

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

See Figure 11 on the next page

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

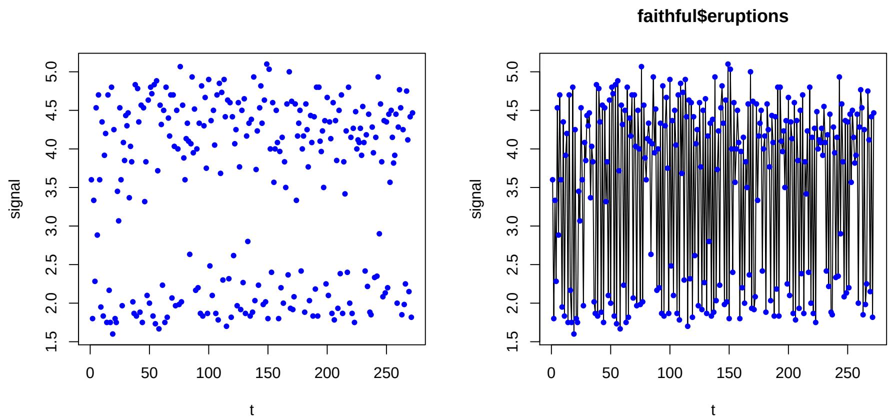


FIGURE 10. Example case: Old Faithful Geyser eruptions. Signal and linear interpolation.

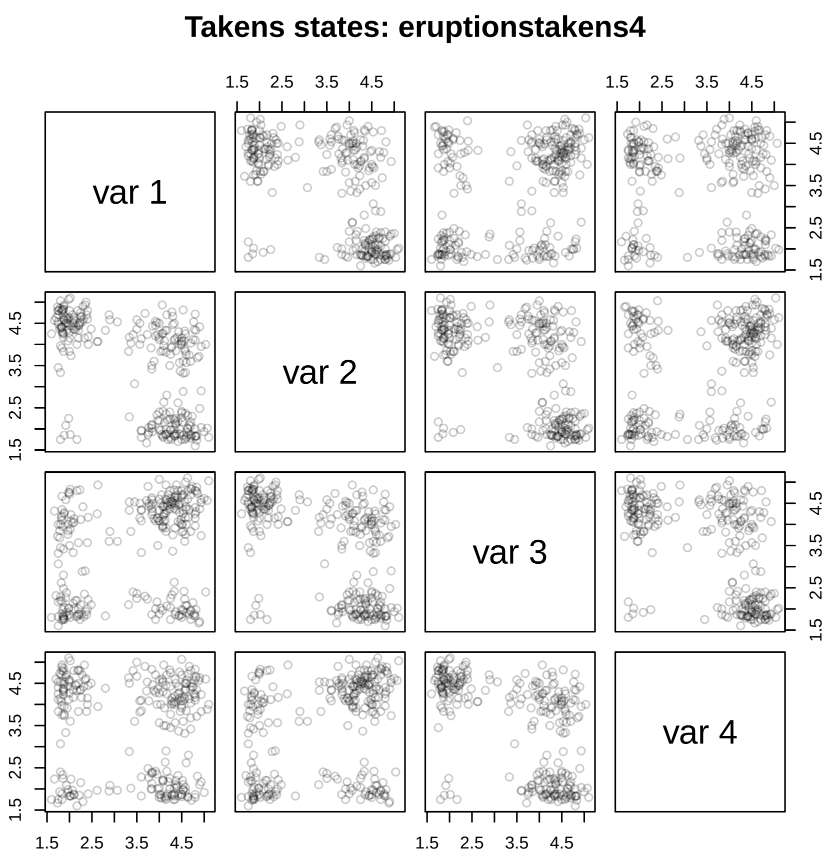


FIGURE 11. Example case: Old Faithful Geyser eruptions. Time used: 0.272 sec.

Input

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

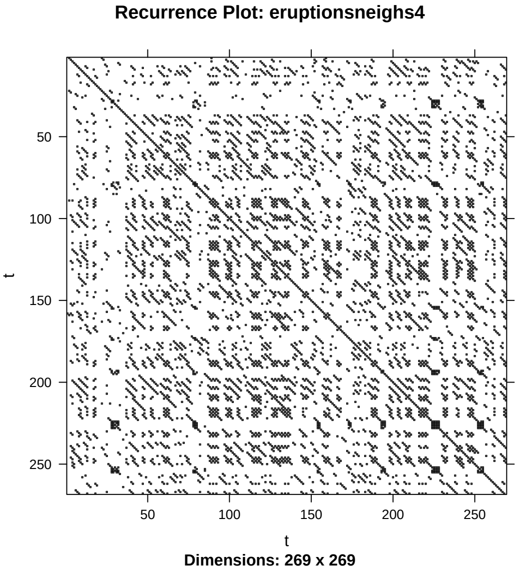


FIGURE 12. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=4. Time used: 21.445 sec.

5.1.1. *Dim=2.*

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

See Figure 13 on the facing page

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

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

5.1.2. *Dim=6.*

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

See Figure 15 on page 14

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

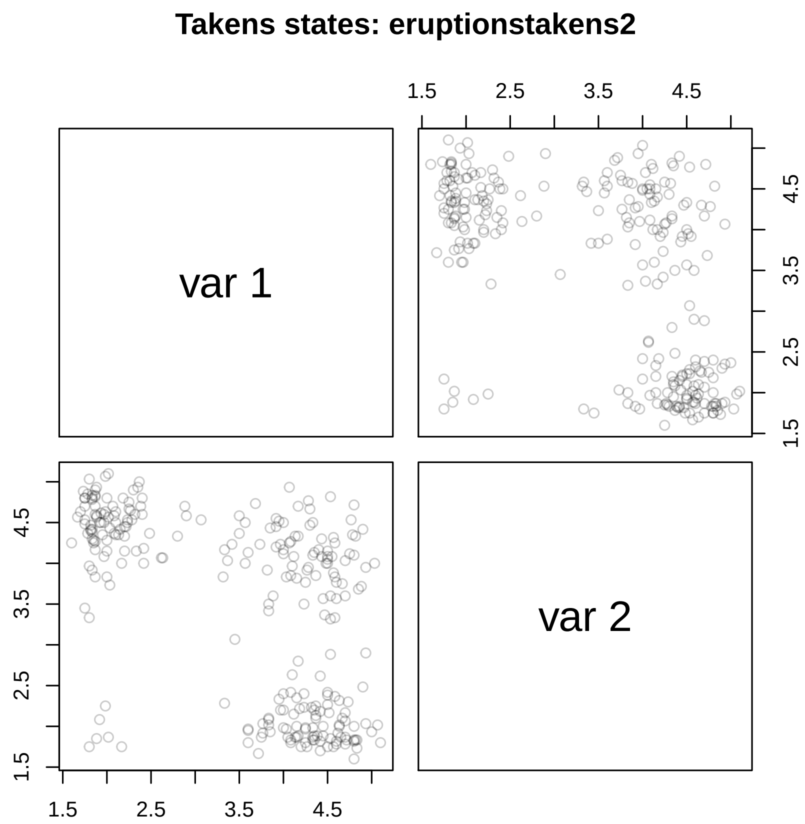


FIGURE 13. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 0.119 sec.

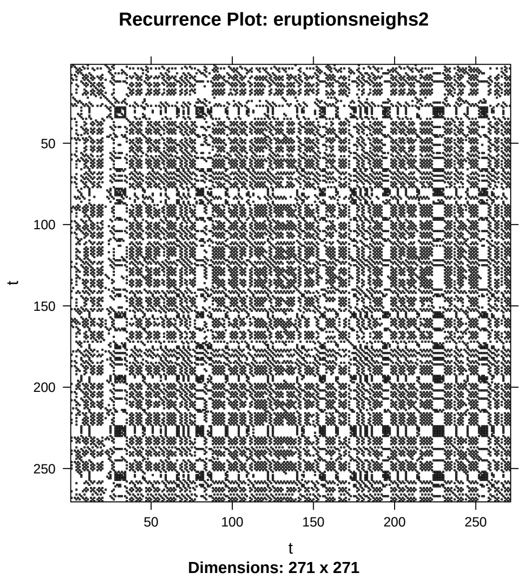


FIGURE 14. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=2. Time used: 67.178 sec.

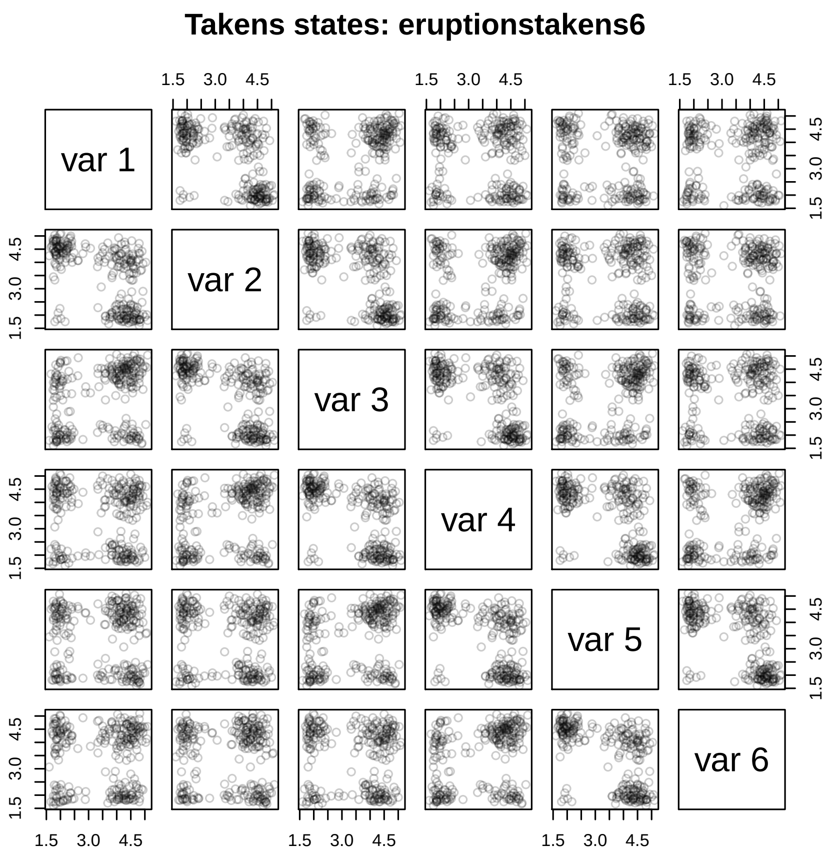


FIGURE 15. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 0.531 sec.

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

5.1.3. *Dim=8.*

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

See Figure 17 on the facing page

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

Input

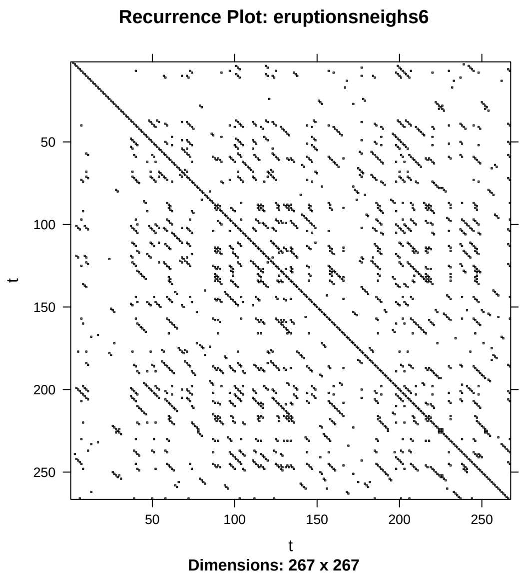


FIGURE 16. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=6. Time used: 7.054 sec.

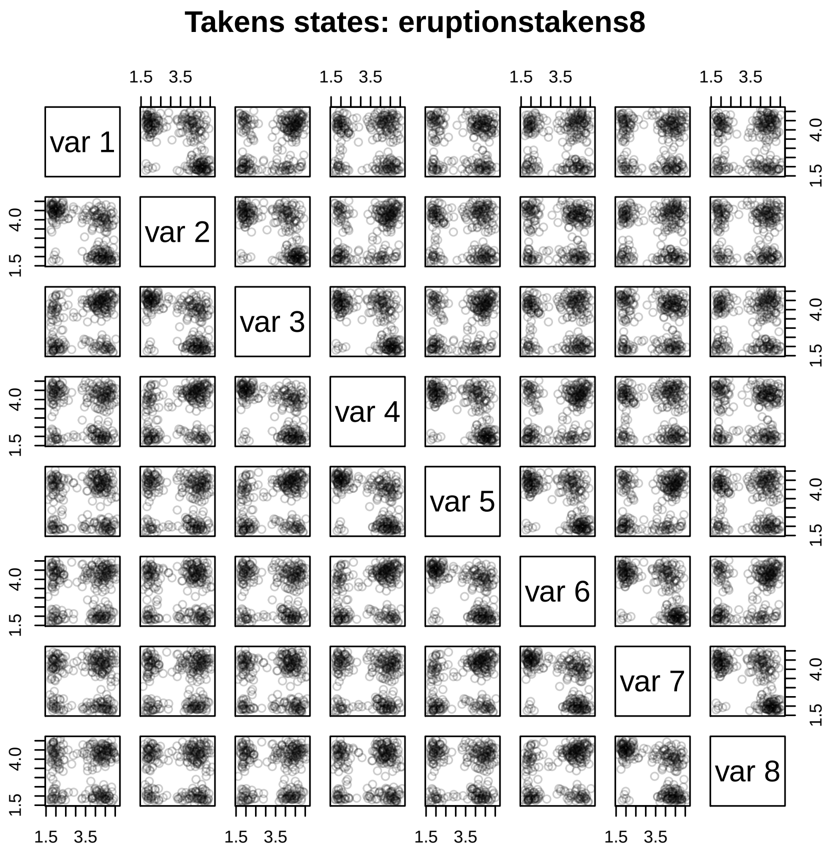


FIGURE 17. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 0.838 sec.

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

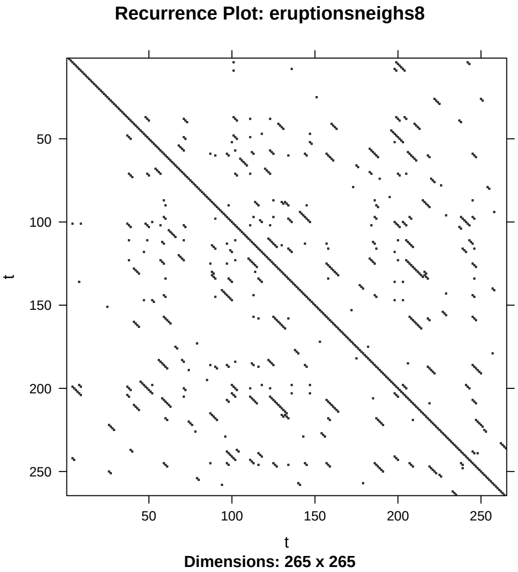


FIGURE 18. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=8. Time used: 2.514 sec.

5.2. **Geyser Eruptions: Comparison by Dimension.** For comparison, recurrence plots for the Geyser data with varying dimension are in Figure 19 on the next page

5.3. **Geyser Waiting.**

Input
```
plotsignal(faithful$waiting)
```

See Figure 20 on the facing page,

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

See Figure 21 on page 18

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

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

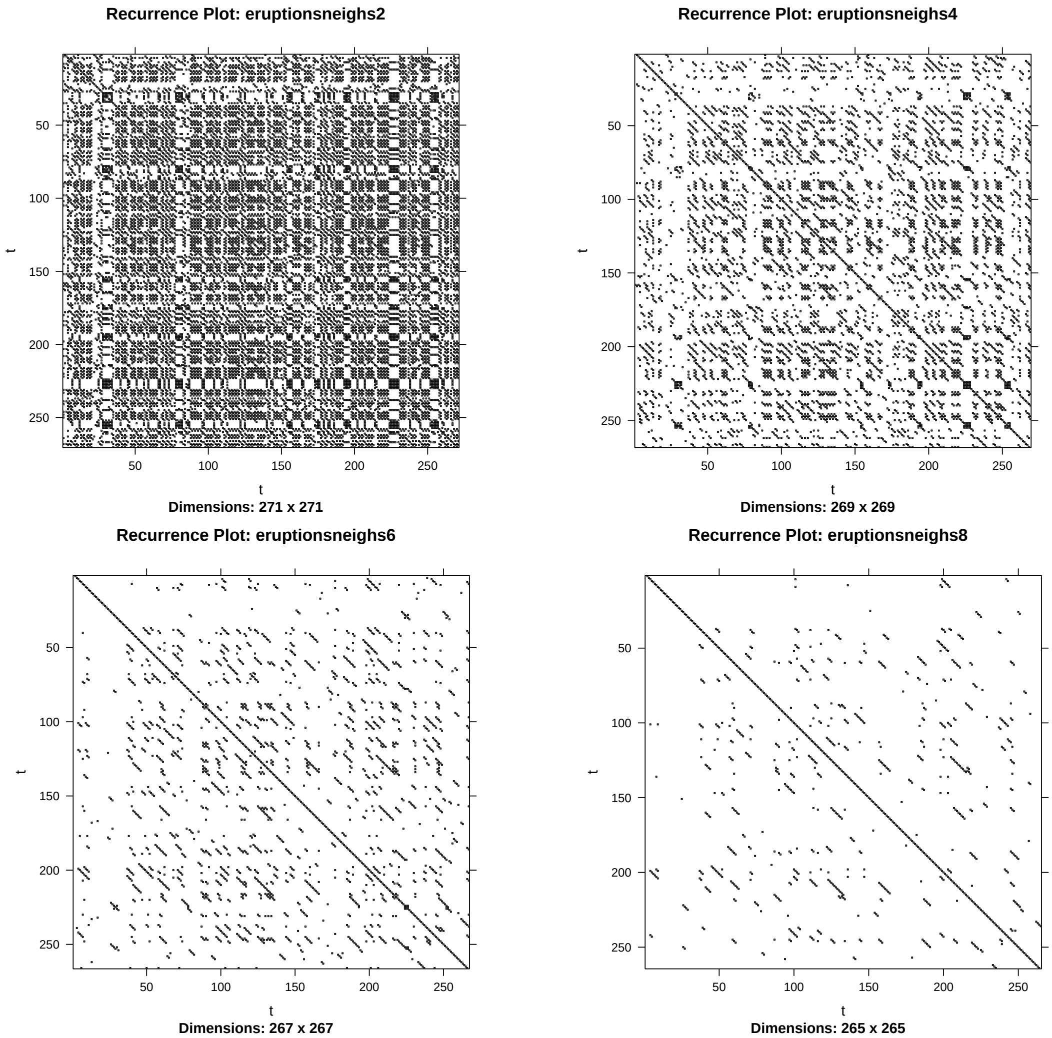


FIGURE 19. Recurrence Plot. Example case: Old Faithful Geyser eruptions. Dim=2, 4, 6, 8.

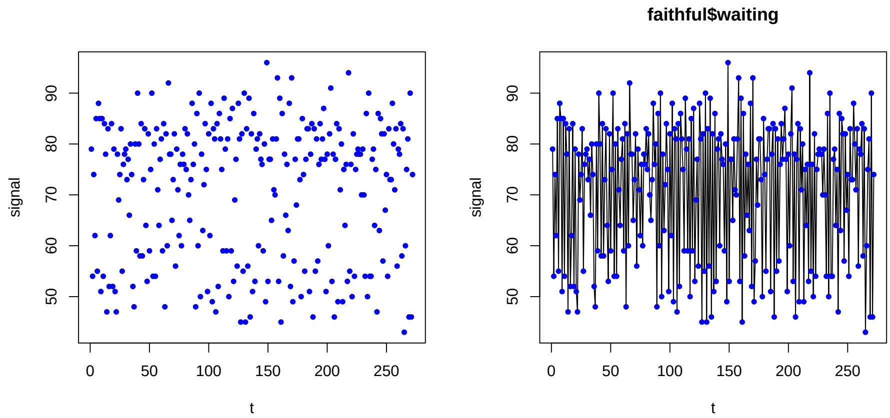


FIGURE 20. Example case: Old Faithful Geyser waiting. Signal and linear interpolation. Time used: 2.715 sec.

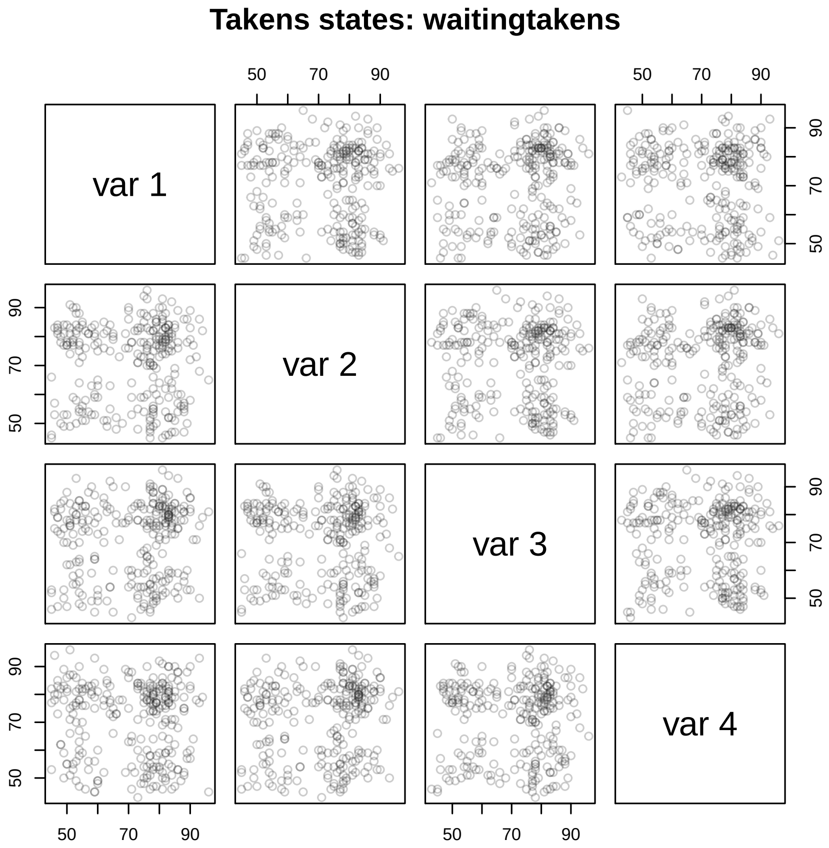


FIGURE 21. Example case: Old Faithful Geyser waiting. Time used: 0.254 sec.

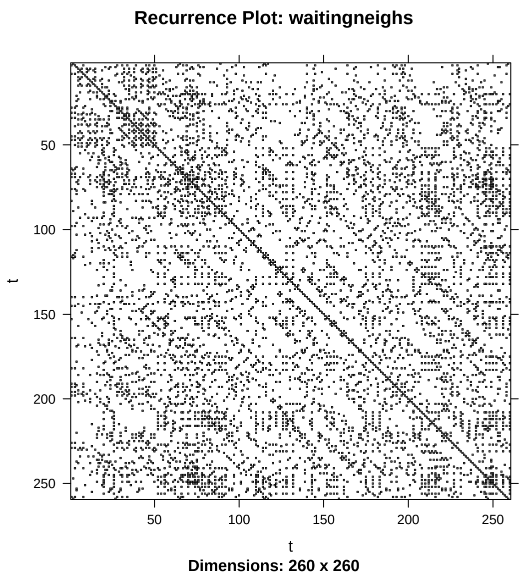


FIGURE 22. Recurrence Plot. Example case: Old Faithful Geyser waiting. Time used: 22.701 sec.

6. Case Study: HRV data

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 23 on the following page,

ToDo: We l lies at appro 2*RR. Coul an artefact processing, out too m pulses?

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

See Figure 24 on the next page

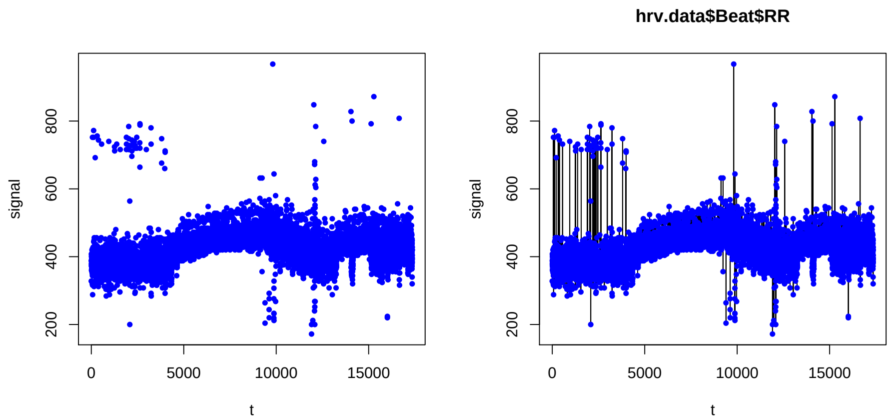


FIGURE 23. Example case: RHRV tutorial. Signal and linear interpolation.

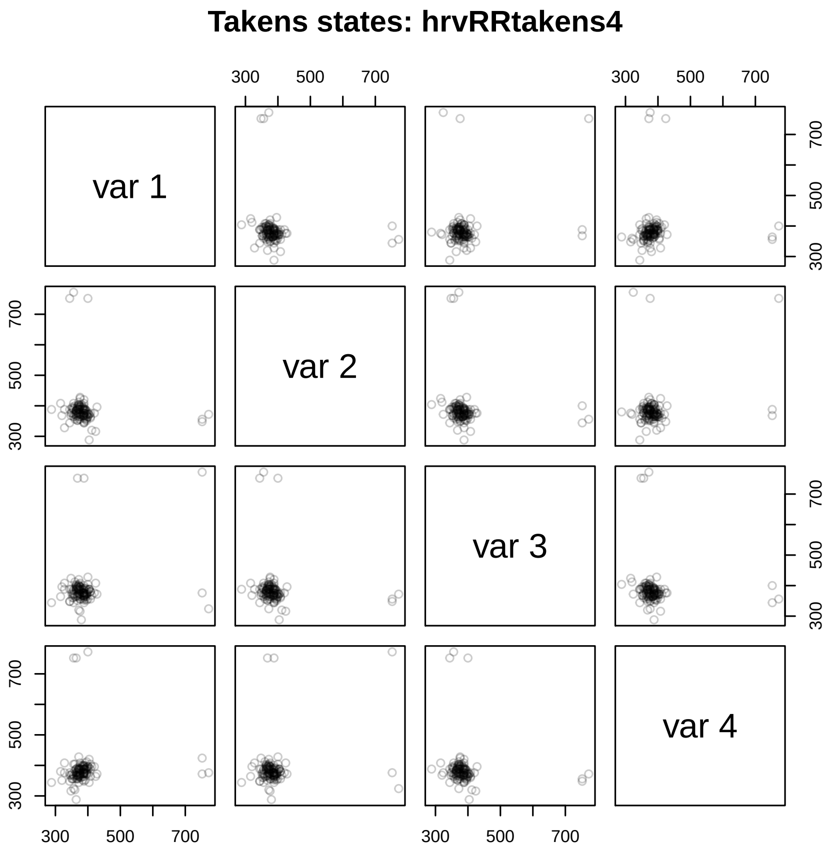


FIGURE 24. Example case: RHRV tutorial. Time used: 0.182 sec.

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

See Figure 25 on the facing page

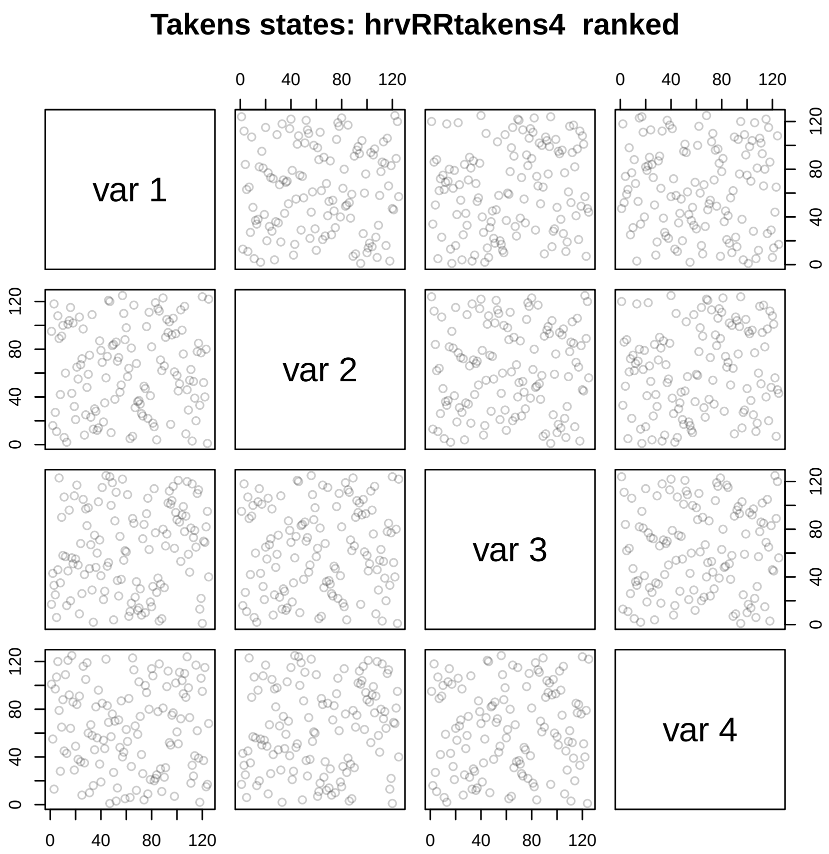


FIGURE 25. Example case: RHRV tutorial. Ranked data. Time used: 0.398 sec.

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

Time used: 0.02 sec.

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

We should
the breathin
so a time la
order of 10
expected.

6.1. 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: 0.039 sec.

Input

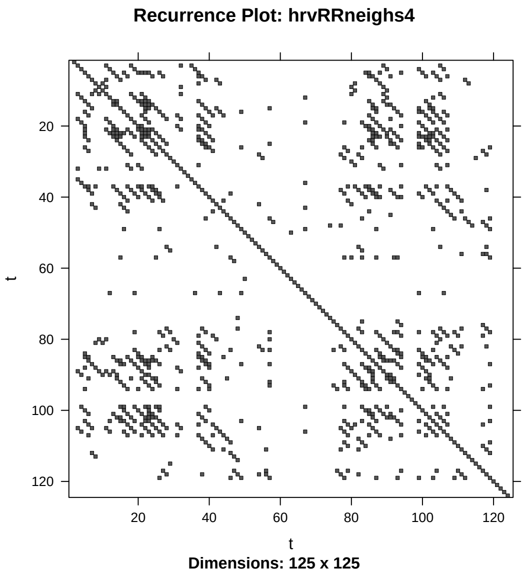


FIGURE 26. Recurrence Plot. Example case: RHRV tutorial. Dim=4. Time used: 3.286 sec.

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

Time used: 10.51 sec.

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

Time used: 0.04 sec.

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

Dim=6. Time used: 1.042 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: 0.044 sec.

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

Dim=8. Time used: 8.009 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: 8.056 sec.

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

Time used: 10.769 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: 10.812 sec.

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

Time used: 1.83 sec.

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

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");
        }
```

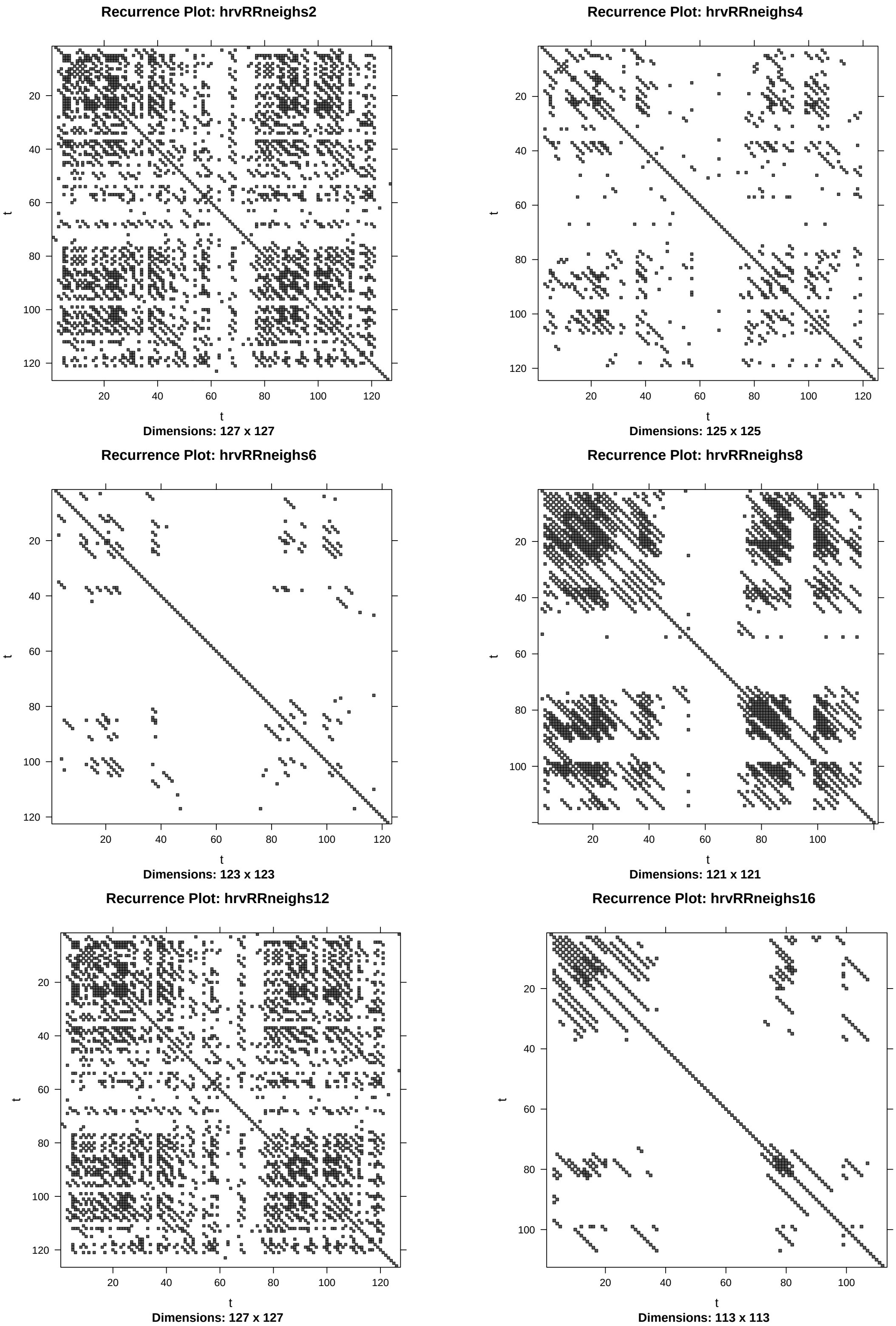


FIGURE 27. Recurrence Plot. Example case: RHRV tutorial. Dim=2, 4, 6, 8, 12, 16. Time used: 1.831 sec.

```
    #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

         return(HRVData)
 }
 as
```

Output

```
function (object, Class, strict = TRUE, ext = possibleExtends(thisClass,
    Class))
{
    if (.identC(Class, "double"))
        Class <- "numeric"
    thisClass <- .class1(object)
    if (.identC(thisClass, Class) || .identC(Class, "ANY"))
        return(object)
    where <- .classEnv(thisClass, mustFind = FALSE)
    coerceFun <- getGeneric("coerce", where = where)
    coerceMethods <- .getMethodsTable(coerceFun, environment(coerceFun),
        inherited = TRUE)
    asMethod <- .quickCoerceSelect(thisClass, Class, coerceFun,
        coerceMethods, where)
    if (is.null(asMethod)) {
        sig <- c(from = thisClass, to = Class)
        asMethod <- selectMethod("coerce", sig, optional = TRUE,
            useInherited = FALSE, fdef = coerceFun, mlist = getMethodsForDispatch(coerceFun))
        if (is.null(asMethod)) {
            canCache <- TRUE
            inherited <- FALSE
            if (is(object, Class)) {
                ClassDef <- getClassDef(Class, where)
                if (identical(ext, FALSE))
                  stop(sprintf("internal problem in as(): %s is(object, \"%s\") is TRUE, but the metadata
                    dQuote(thisClass), Class), domain = NA)
                else if (identical(ext, TRUE))
                  asMethod <- .makeAsMethod(quote(from), TRUE,
                    Class, ClassDef, where)
                else {
                  test <- ext@test
                  asMethod <- .makeAsMethod(ext@coerce, ext@simple,
                    Class, ClassDef, where)
                  canCache <- (!is(test, "function")) || identical(body(test),
                    TRUE)
                }
            }
            if (is.null(asMethod) && extends(Class, thisClass)) {
                ClassDef <- getClassDef(Class, where)
                asMethod <- .asFromReplace(thisClass, Class,
                  ClassDef, where)
            }
            if (is.null(asMethod)) {
                asMethod <- selectMethod("coerce", sig, optional = TRUE,
                  c(from = TRUE, to = FALSE), fdef = coerceFun,
                  mlist = coerceMethods)
```

```
                inherited <- TRUE
            }
            else if (canCache)
                asMethod <- .asCoerceMethod(asMethod, thisClass,
                  ClassDef, FALSE, where)
            if (is.null(asMethod))
                stop(gettextf("no method or default for coercing %s to %s",
                  dQuote(thisClass), dQuote(Class)), domain = NA)
            else if (canCache) {
                cacheMethod("coerce", sig, asMethod, fdef = coerceFun,
                  inherited = inherited)
            }
        }
    }
    if (strict)
        asMethod(object)
    else asMethod(object, strict = FALSE)
}
<bytecode: 0x10b6ae588>
<environment: namespace:methods>
```

Input

Input

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

Output

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

Input

```
HRRV <- hrv.data$Beat$dRR/hrv.data$Beat$avRR
hrv.data$Beat$HRRV <- HRRV
```

These are the displays we used before, now for HRV:

Input

```
plotsignal(HRRV)
```

See Figure 28 on the next page,

Input

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

See Figure 29 on the facing page

Input

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

ToDo: findAllNeighbours does not handle NAs

See Figure 30 on page 28

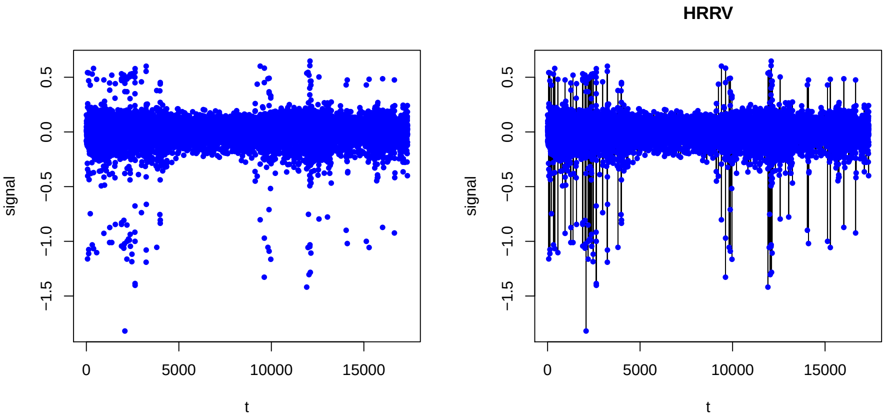


FIGURE 28. Example case: RHRV tutorial. Signal and linear interpolation.

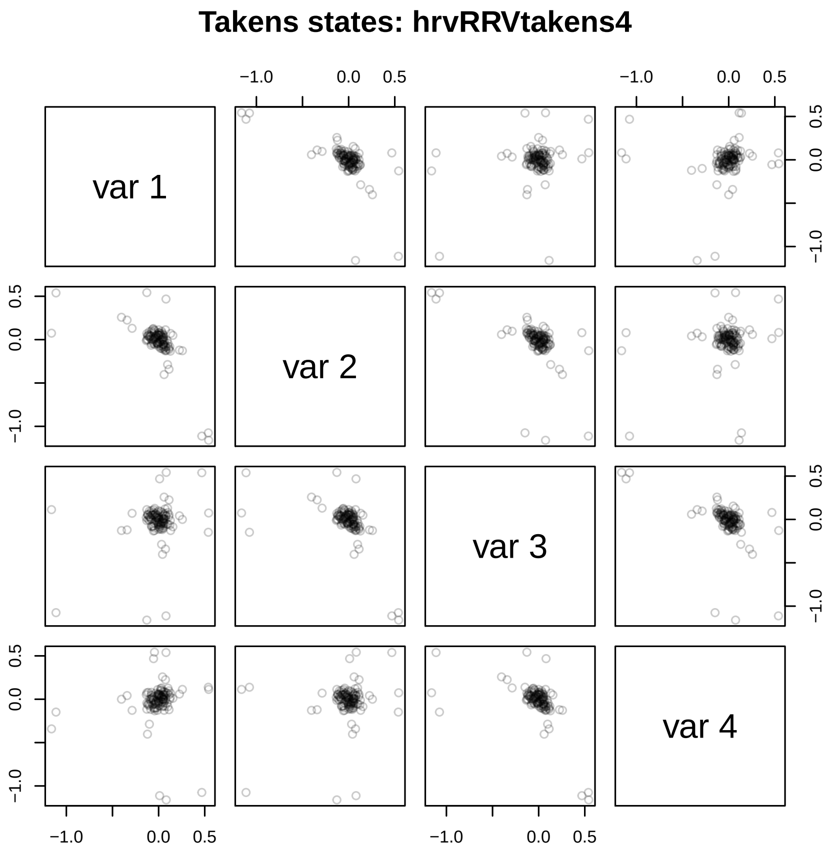


FIGURE 29. Example case: RHRV tutorial. Time used: 0.19 sec.

Input
```
hrvRRVneighs4 <-local.findAllNeighbours(hrvRRVtakens4[-(1:2),],, radius=0.25)
save(hrvRRVneighs4, file="hrvRRVneighs4.Rdata")
```

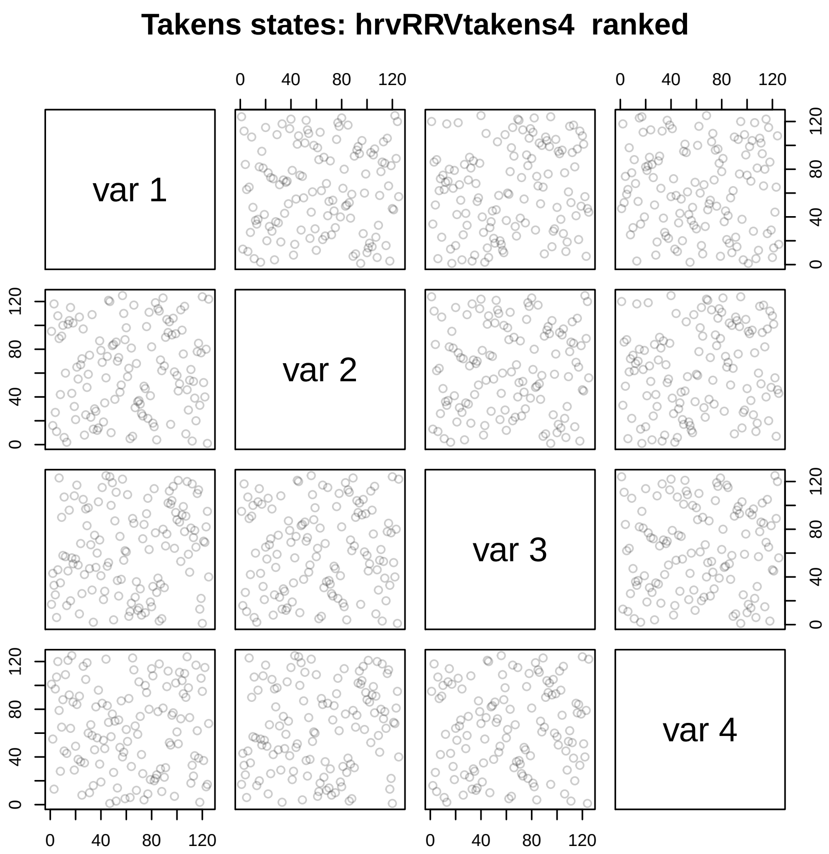


FIGURE 30. Example case: RHRV tutorial. Ranked data. Time used: 0.418 sec.

Time used: 0.029 sec.

Input
```
load(file="hrvRRVneighs4.RData")
local.recurrencePlotAux(hrvRRVneighs4)
```

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 NN=10

6.3. RHRV: 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=0.25)
save(hrvRRVneighs2, file="hrvRRVneighs2.Rdata")
```

Time used: 0.05 sec.

Input
```
load(file="hrvRRVneighs2.RData")
local.recurrencePlotAux(hrvRRVneighs2)
```

Time used: 42.76 sec.

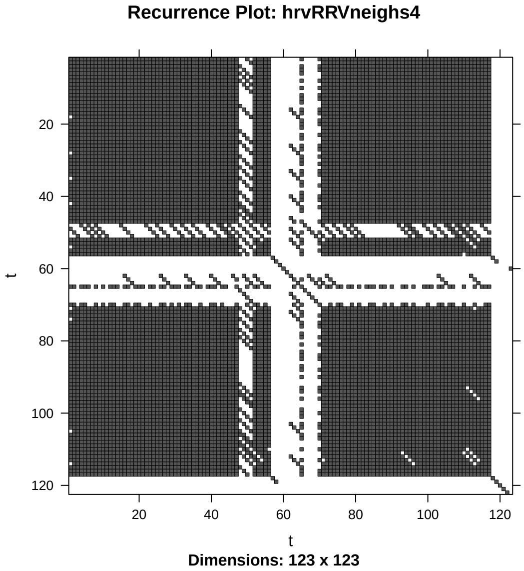


FIGURE 31. Recurrence Plot. Example case: RHRV tutorial. Dim=4. Time used: 35.55 sec.

Input
```
hrvRRVtakens6 <- local.buildTakens( time.series=HRRV[1:nsignal],embedding.dim=6,time.lag=1)
hrvRRVneighs6 <-local.findAllNeighbours(hrvRRVtakens6[-(1:2),], radius=0.25)
save(hrvRRVneighs6, file="hrvRRVneighs6.Rdata")
```

Time used: 0.048 sec.

Input
```
load(file="hrvRRVneighs6.RData")
local.recurrencePlotAux(hrvRRVneighs6)
```

Dim=6. Time used: 31.136 sec.

Input
```
hrvRRVtakens8 <- local.buildTakens( time.series=HRRV[1:nsignal],embedding.dim=8,time.lag=1)
hrvRRVneighs8 <-local.findAllNeighbours(hrvRRVtakens8[-(1:2),], radius=0.25)
save(hrvRRVneighs8, file="hrvRRVneighs8.Rdata")
```

Time used: 0.045 sec.

Input
```
load(file="hrvRRVneighs8.RData")
local.recurrencePlotAux(hrvRRVneighs8)
```

Dim=8. Time used: 27.614 sec.

Input
```
hrvRRVtakens12 <- local.buildTakens( time.series=HRRV[1:nsignal],embedding.dim=2,time.lag=1)
hrvRRVneighs12 <-local.findAllNeighbours(hrvRRVtakens12[-(1:2),], radius=0.25)
save(hrvRRVneighs12, file="hrvRRVneighs12.Rdata")
```

Time used: 27.667 sec.

Input
```
load(file="hrvRRVneighs12.RData")
local.recurrencePlotAux(hrvRRVneighs12)
```

Time used: 42.592 sec.

Input
```
hrvRRVtakens16 <- local.buildTakens( time.series=HRRV[1:nsignal],embedding.dim=16,time.lag=1)
hrvRRVneighs16 <-local.findAllNeighbours(hrvRRVtakens16[-(1:2),], radius=0.25)
save(hrvRRVneighs16, file="hrvRRVneighs16.Rdata")
```

Time used: 42.639 sec.

Input
```
load(file="hrvRRVneighs16.RData")
local.recurrencePlotAux(hrvRRVneighs16)
```

Time used: 18.82 sec.

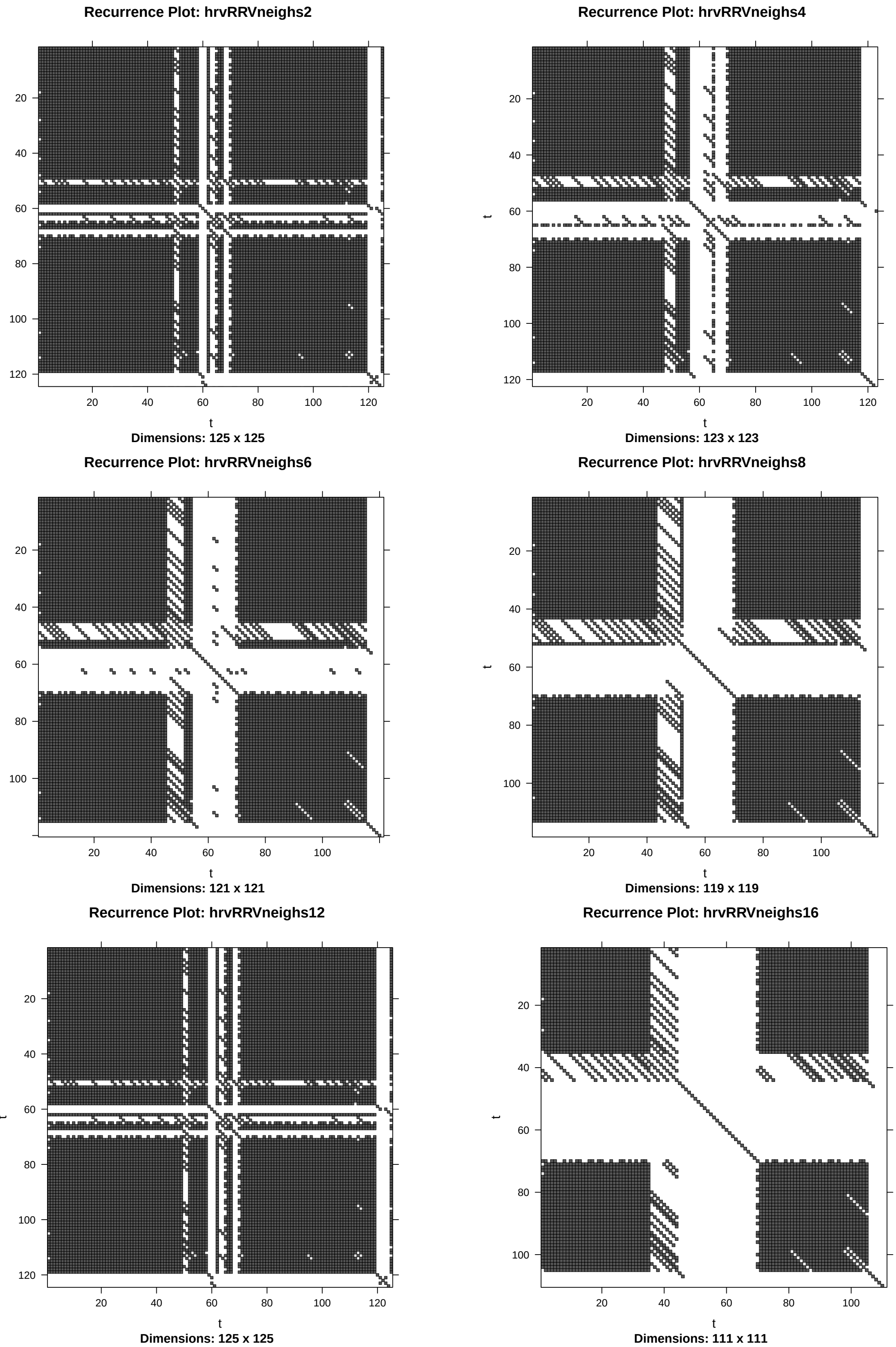


FIGURE 32. Recurrence Plot. Example case: RHRV tutorial. Dim=2, 4, 6, 8, 12, 16. Time used: 18.821 sec.

References

Eckmann, Jean-Pierre, Kamphorst, S Oliffson, & Ruelle, David. 1987. Recurrence plots of dynamical systems. *Europhys. Lett*, **4**(9), 973–977.

Index

R session info:

Total Sweave time used: 374.387 sec.

- 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 1.7-27, 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: 6.00612in linewidth:6.00612in

textheight: 9.21922in

CVS/Svn repository information:

$Source: /u/math/j40/cvsroot/lectures/src/dataanalysis/Rnw/recurrence.Rnw,v $

$HeadURL: /u/math/j40/cvsroot/lectures/src/dataanalysis/Rnw/recurrence.Rnw,v $

$Revision: 1.2 $

$Date: 2014/02/05 20:05:07 $

$name: $

$Author: j40 $

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

Günther Sawitzki
StatLab Heidelberg
Im Neuenheimer Feld 294
D 69120 Heidelberg