Nonlinear Methods Workshop Lecture Notes

Dan Mønster, Alon Tomashin, Ralf Cox, Giuseppe Leonardi, Sebastian Wallot

2024-04-16

Table of contents

Preface

Online lecture notes for Nonlinear Methods Workshop held at Leuphana University in Lüneburg, 29 July – 2 August 2024.

There is also a PDF version of the lecture notes that can be downloaded.

1 Introduction to recurrence plots

This chapter introduces what a recurrence plot is and explains how it can be analyzed using the most common measures that quantify features of the recurrence plot. We will use some relatively simple examples, so that the basic concepts are not obscured by complications arising from more realistic data. We will deal with those issues in due time, but see Marwan et al. (2007) for a comprehensive introduction to recurrence plots with all the mathematical details.

Now, let us explore what a recurrence plot is.

1.1 A simple example

We start with the children's rhyme by Helen H. Moore, shown below, to explore what recurrence in a time series is — exemplified here by the text, where each letter is a datum and time is represented by the position of the letter in the text.

```
Pop, pop, popcorn
Popping in the pot!
Pop, pop, popcorn
Eat it while it's hot!
Pop, pop, popcorn
Butter on the top!
When I eat popcorn
I can't stop
```

The letters of the rhyme are encoded in the variable `popcorn` and we can the use the `crqa()` function from the package of the same name (Coco et al. 2021) to create the recurrence plot.

The first 30 elements of the `popcorn` vector are: P, O, P, ␣, P, O, P, ␣, P, O, P, C, O, R, N, ␣, P, O, P, P, I, N, G, ␣, I, N, ␣, T, H, E. Here the character "␣" represents a space between words. Line breaks and punctuation marks have been removed from the poem, and all letters are upper case, since we do not want to distinguish between lower case and upper case letters.

The R code shown below will compute and display the recurrence plot.

```
library(crqa)

rp <- crqa(popcorn, popcorn,
           delay = 0,
           embed = 0,
           radius = 0.1,
           method = "rqa",
           datatype = "categorical")

plot_rp(rp$RP)
```

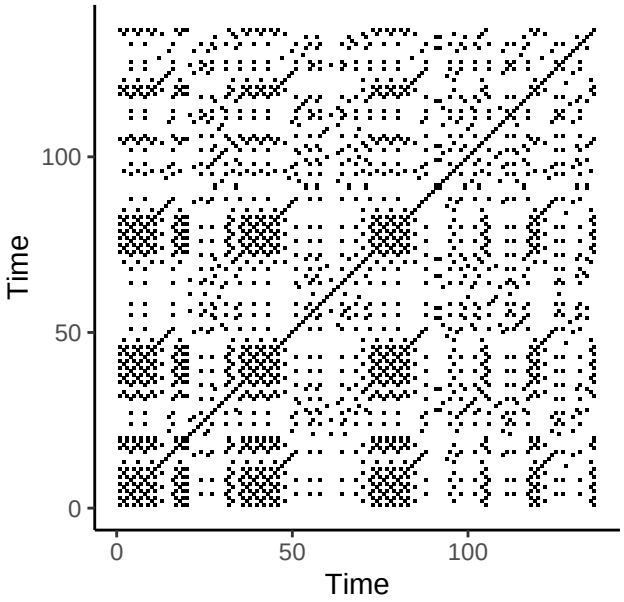

Figure 1.1: Recurrence plot of poem

We can also look at the first two verses of the poem, and put the letters on the axes, to make it a little easier to see how the underlying time series results in a recurrence plot.

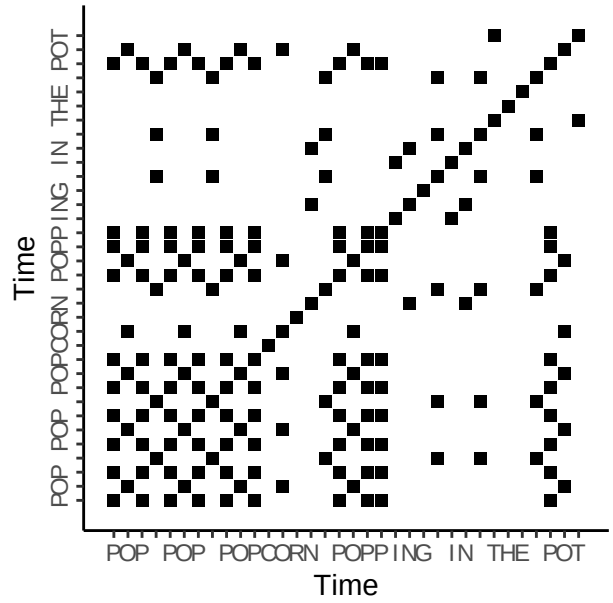

Figure 1.2: Recurrence plot of first two verses of the poem

1.2 Quantifying the recurrence plot

With some training you can learn how to spot various properties of a time series simply by looking at the corresponding recurrence plot. In the case of the poem, we can see, for instance, that certain motifs are repeated in the poem. This is evident from the blue diagonal lines in the figure below, which correspond to the recurring motif “POP POP POPCORN”.

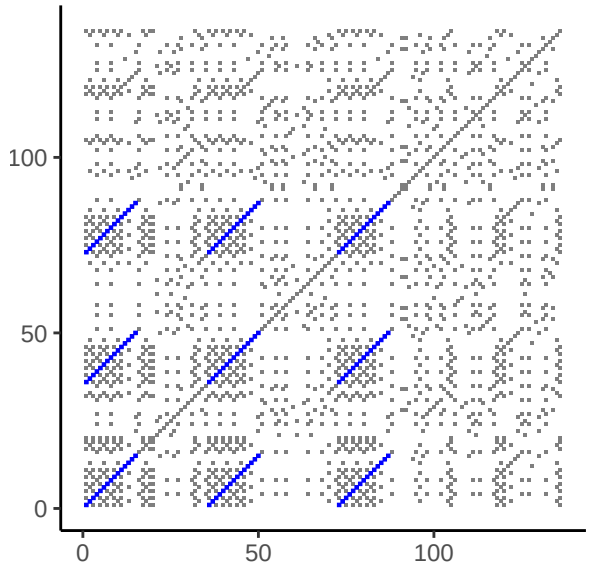

Figure 1.3: Recurring motifs in the poem

While we can gain some insights about the underlying time series from a visual inspection of the recurrence plot, we do not wish to rely on that. First, simply because we could overlook important features, and furthermore, it it not feasible for long time series or when comparing many time series. And, perhaps most importantly, we want *objective* measures, that do not depend on the skill of the person assessing the recurrence plot. With that said, it is still very useful to be able to visually inspect recurrence plots and infer something about the time series, so we definitely encourage you to work towards developing that skill as well.

But having objective quantitative measures that characterize features of a given recurrence plot is what has made recurrence plot analysis an effective tool, and that is why we will now look at *recurrence quantification measures*.

Some of the most common recurrence quantification measures are reproduced in table Table 1.1, reproduced from Coco et al. (2021).

Table 1.1: Common recurrence quantification measures

Measure	**Abbreviation**	**Definition**
Recurrence Rate	RR	$\frac{1}{N^2}\sum_{i,j=1}^{N} R_{ij}$
Determinism	DET	$\sum_{l=l_{\min}}^{N} lP(l) \Big/ \sum_{l=1}^{N} lP(l)$

Measure	**Abbreviation**	**Definition**
Average Diagonal Line Length	L	$\sum_{l=l_{\min}}^{N} lP(l) \Big/ \sum_{l=l_{\min}}^{N} P(l)$
Maximum Diagonal Line Length	maxL	$\max(\{l_i\}_{i=1}^{N_l}), \quad N_l = \sum_{l \geq l_{\min}} P(l)$
Diagonal Line Entropy	ENTR	$-\sum_{l=l_{\min}}^{N} p(l) \log p(l)$
Laminarity	LAM	$\sum_{v=v_{\min}}^{N} vP(v) \Big/ \sum_{v=1}^{N} vP(v)$
Trapping Time	TT	$\sum_{v=v_{\min}}^{N} vP(v) \Big/ \sum_{v=v_{\min}}^{N} P(v)$
Categorical Area-based Entropy	catH	$-\sum_{a>1}^{N_a} p(a) \log p(a)$

2 Recurrence Quantification Analysis

Under construction

3 Cross Recurrence Quantification Analysis

Under construction

4 Multidimensional Recurrence Quantification Analysis

Here you will find 6 exercises to learn different aspects of MdRQA and some questions to discuss. Copy the codes to RStudio, set the working directory to 06_MdRQA (setwd), and run the codes to look for the solutions. (Make sure you have 'crqa' and 'ggplot2' installed)

1. Performing MdRQA on the Lorenz system. How do the results differ from those found using RQA? For what reasons?

```
# set path...
setwd("...")

# load package crqa
library(crqa)
library(ggplot2)

# Run MdRQA on the 3d-Lorenz system

# load data
Lorenz <- read.csv("Lorenz.csv")

# run MdRQA
res <- crqa(ts1=Lorenz, ts2=Lorenz, delay=1, embed=1, radius=1.5,
            normalize=0, tw=1, method="mdcrqa")

# check out recurrence measures
head(res)

# plot RP
RP <- res$RP
plot_rp(RP, xlabel = "Time", ylabel = "Time", geom = "void") + geom_point(size = 0.5)
```

2. The differences between a Theiler Window (tw) of 1 and 0.

   Adjust the code above to check.

What measures were more affected by the change?

Would you choose the same Theiler Window for CRQA?

3. The effects of noise on MdRQA outcomes.

   How did the RP and the recurrence measures change?

   What could you do to receive more informative MdRQA outcomes?

```
# load package crqa
library(crqa)

# load data
Lorenz <- read.csv("Lorenz.csv")
# load noisy data
noisy_Lorenz <- read.csv("R.csv")

# add R to Lorenz data.frame
Lorenz$r <- noisy_Lorenz$R

# run MdRQA
res <- crqa(ts1=Lorenz, ts2=Lorenz, delay=1, embed=1, radius=1.5,
            normalize=0, tw=1, method="mdcrqa")

# check out recurrence measures
head(res)

# plot RP
RP <- res$RP
plot_rp(RP, xlabel = "Time", ylabel = "Time", geom = "void") + geom_point(size = 0.5)
```

4. The effect of data normalization in MdRQA.

Load the file R.csv and add it to the Lorenz-data.frame as fourth variable. However, now use the scale() function to z-score the data from the R.csv-file before adding it as fourth variable to the Lorenz data frame.

How do the results change?

```
# load package crqa
library(crqa)

# load data
Lorenz <- read.csv("Lorenz.csv")
```

```r
# load noisy data
noisy_Lorenz <- read.csv("R.csv")

# z-score R
noisy_Lorenz <- scale(noisy_Lorenz$R)

# add R to Lorenz data.frame
Lorenz$r <- noisy_Lorenz

# run MdRQA
res <- crqa(ts1=Lorenz, ts2=Lorenz, delay=1, embed=1, radius=1.5,
            normalize=0, tw=1, method="mdcrqa")

# check out recurrence measures
head(res)

# plot RP
RP <- res$RP
plot_rp(RP, xlabel = "Time", ylabel = "Time", geom = "void") + geom_point(size = 0.5)
```

5. Rerun MdRQA on the Lorenz system with z-scored data (normalize = 2). Find a radius that yields %REC=5-10%.

```r
# load package crqa
library(crqa)
library(ggplot2)

# Run MdRQA on the 3d-Lorenz system

# load data
Lorenz <- read.csv("Lorenz.csv")

# run MdRQA
res <- crqa(ts1=Lorenz, ts2=Lorenz, delay=1, embed=1, radius=1.2,
            normalize=2, tw=1, method="mdcrqa")

# check out recurrence measures
head(res)

# plot RP (it takes a minute here)
RP <- res$RP
plot_rp(RP, xlabel = "Time", ylabel = "Time", geom = "void") + geom_point(size = 0.5)
```

6. Use Lagged MdRQA to find the lags in the Lorenz system, inspect the Lorenz time series. Do the lags seem reasonable? Use the lagged MdRQA wrapper. Here we resample the Lorenz system for a quicker computation.

First run the Lagged MdRQA function: (no need to dive in) Then use the next code for the task.

```
laggedMdrqa <- function(maxlag, ts1, ts2, delay, embed, rescale,
                        radius, normalize, mindiagline, minvertline, tw, method) {
  # Note:
  # This function wraps the crqa()-function from 'crqa' package.
  # In order to run the function, the package 'crqa',
  # as well its dependencies, need to be installed and loaded.
  # The authors give no warranty for the correct functioning of
  # the software and cannot be held legally accountable.

# load libraries
library(crqa)

# infer parameters to create empty matrix
noTs <- dim(ts1)[2]
lagList <- matrix(0, ncol = noTs+1, nrow = (maxlag+1)^noTs)

# compute list of lag combinations
for(i in 1:noTs) {
    lagList[,i] <-rep(0:maxlag, each = (maxlag+1)^(i-1), (maxlag+1)^(noTs-i))
}

# equate lags on lag0
for(i in 1:(maxlag+1)^noTs) {
  lagList[i,1:noTs] <- lagList[i,1:noTs]-min(lagList[i,1:noTs])
}

# compute lag identifier
for(i in 1:noTs) {
  lagList[,noTs+1] <- lagList[,noTs+1] + lagList[,i]*100^(noTs-i)
}

# discard all lags that are not unique
lagList <- subset(lagList, duplicated(lagList[,noTs+1]) == FALSE)

# create results matrix and add lag parameters
results <- matrix(nrow = dim(lagList)[1], ncol = (9+dim(lagList)[2]-1))
```

```
results[,10:(10+(dim(lagList)[2]-2))] <- lagList[,1:(dim(lagList)[2]-1)]

# run lagged mdrqa
for(i in 1:dim(lagList)[1]) {

  # create temporary data matrix
  temp_ts1 <- matrix(ncol = dim(ts1)[2], nrow = length((1+lagList[i,1]):(dim(ts1)[1]-maxlag+l
  temp_ts2 <- matrix(ncol = dim(ts2)[2], nrow = length((1+lagList[i,1]):(dim(ts2)[1]-maxlag+l

  # construct lagged time series
  for(j in 1:dim(ts1)[2]) {
    temp_ts1[,j] <- ts1[(1+lagList[i,j]):(dim(ts1)[1]-maxlag+lagList[i,j]),j]
  }
  for(j in 1:dim(ts2)[2]) {
    temp_ts2[,j] <- ts2[(1+lagList[j,1]):(dim(ts2)[1]-maxlag+lagList[j,1]),j]
  }

  # rund mdrqa
        temp_res <- crqa(ts1=temp_ts1, ts2=temp_ts2, delay = delay, embed = embed,
                         rescale = rescale, radius = radius, normalize = normalize,
                         mindiagline = mindiagline, minvertline = minvertline,
                         tw = tw, method = "mdcrqa")

        # store recurrence measures on each iteration
        results[i,1:9] <- unlist(temp_res[1:9])
}

# convert results to data frame
results <- as.data.frame(results)

# generate and add labels
newLabels <- c("RR","DER","NRLINE","maxL","L","ENTR","rENTR","LAM","TT")
j <- 0
for(i in 10:(10+(dim(lagList)[2]-2))) {
  j <- j+1
  newLabels[i] <- paste("ts",as.character(j),sep="")
}
colnames(results) <- newLabels

# sort data frame by RR
results <- results[order(results$RR, decreasing = TRUE),]
```

```
# return results
return(results)
}
```

Choose some value for the „maxlag“ parameter between 10-30. Investigate the lag structure compared to the Lorenz system plot.

```
# load packages
library(crqa)
library(ggplot2)

#  Load Lorenz data
Lorenz <- read.csv("Lorenz.csv")
Lorenz$r <- 1:nrow(Lorenz)

# down-sample Lorenz data
down_Lorenz <- Lorenz[seq(1,2500,by=10),]

# plot down-sampled Lorenz data, dimension x

ggplot(down_Lorenz, aes(x = r)) +
  geom_line(aes(y = x, color = "x")) +
  geom_line(aes(y = y, color = "y")) +
  geom_line(aes(y = z, color = "z")) +
  labs(title = "Down-Sampled Lorenz") +
  ylab('') + xlab('') +
  theme_minimal() +
  scale_color_manual(values = c("x" = "red", "y" = "green", "z" = "blue"),
                     name = "Variables") +
  theme(legend.title = element_blank())


# run lagged Mdrqa - chose a value for maxlag between 10 and 30
res <- laggedMdrqa(maxlag = ...,
                   ts1 = down_Lorenz[,1:3],
                   ts2 = down_Lorenz[,1:3],
                   delay = 1,
                   embed = 1,
                   rescale = 0,
                   radius = 0.5,
                   normalize = 2,
                   mindiagline = 2,
```

```
                    minvertline = 2,
                    tw = 1,
                    method = "mdcrqa")

# plot RR-function
plot(res$RR, type = "l")

# investigate the first couple of lags
head(res)
```

5 Parameter Estimation for RQA

Under construction

6 Parameter Sensitivity Analysis for RQA

Assuming we are dealing with continuous time series rather than categorical time series, we estimate embedding parameters (delay and embedding dimension) as well as an appropriate radius parameter used to construct recurrence plots.

```
# You may need to set the path here
setwd("")

# load packages
library(crqa)
library(dplyr)
library(ggplot2)

# Load data files
# You may need to change the path, depending on your working directory and where you have the
load("../07_sample_analyses/dataSampleAnalysis_new.Rdata")

# Set the experimental condition
# (o)ral reading = 0, (s)ilent reading = 1
CON <- factor(c(rep(0,6),rep(1,6)))

# Enter the estimated parameters from the previous exercise
opt_delay <-  1
opt_embed <-  4
opt_radius <-  0.65

# Now assign the part of parameter space to be explored
delay_values <- seq(1, ...) # Set values here
embed_values <-             # Set values here
radius_values <-            # Set values here

# Start with an empty data frame for the RQA results and parameters
exploration <- data.frame()

for (d in delay_values) {
  for (e in embed_values) {
```

```
    for (r in radius_values) {
      for(i in 1:12) {
        # Note: we exclude the last data point
        temp <- crqa(ts1 = data[1:1098, i],
                     ts2 = data[1:1098, i],
                     rescale = 0,
                     delay = d,
                     embed = e,
                     radius = r,
                     normalize = 2,
                     tw = 1,
                     method = "rqa")
        result <- data.frame(
          REC = temp$RR,
          CON = CON[i],
          delay = d,
          embed = e,
          radius = r
        )
        exploration <- bind_rows(exploration, result)
      }
    }
  }
}

# Compute exp_test which holds the results of the t-test for each set of
# parameters.
exp_test <- data.frame()

for (d in delay_values) {
  for (e in embed_values) {
    for (r in radius_values) {
      # Add t-test here
      TT <- t.test(REC ~ CON,
                   data = exploration |>
                     filter(delay == d, embed == e, radius == r))
      exp_test <- bind_rows(
        exp_test,
        data.frame(
          delay = d,
          embed = e,
          radius = r,
```

```
          p_value = TT$p.value,
          difference = TT$estimate[1] - TT$estimate[2],
          CI_lo = TT$conf.int[1],
          CI_hi = TT$conf.int[2]
        )
      )
    }
  }
}

# Add a variable for whether the t-test is significant
exp_test <- exp_test |>
  mutate(significant = (p_value < 0.05))


# Plot the result for the optimal parameters
ggplot(exploration |>
         filter(delay == opt_delay,
                embed == opt_embed,
                radius == opt_radius),
       aes(x = CON, y = REC)) +
  geom_boxplot() +
  geom_point(aes(y = REC), position = position_jitter()) +
  labs(title = "Difference in conditions with chosen parameters") +
  theme_classic()



# Same plot but as a violin plot
ggplot(exploration |>
         filter(delay == opt_delay,
                embed == opt_embed,
                radius == opt_radius),
       aes(x = CON, y = REC)) +
  geom_violin() +
  geom_point(aes(y = REC), position = position_jitter()) +
  labs(title = "Difference in conditions with chosen parameters") +
  theme_classic()


# Look at result of t-test for different values of embedding dimension
ggplot(exp_test |>
```

```
         filter(delay == opt_delay, radius == opt_radius),
       aes(x = embed, y = p_value)) +
  geom_hline(yintercept = 0.05, linetype = "dashed") +
  geom_line() +
  geom_point(data = exp_test |>
               filter(embed == opt_embed,
                      radius == opt_radius,
                      delay == opt_delay),
             size = 4, colour = "blue") +
  xlab("Embedding dimension") +
  ylab("p-value") +
  theme_bw()


ggplot(exp_test |>
         filter(delay == opt_delay, radius == opt_radius),
       aes(x = embed, y = difference)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_errorbar(aes(ymin = CI_lo, ymax = CI_hi)) +
  geom_point() +
  geom_point(data = exp_test |>
               filter(embed == opt_embed,
                      radius == opt_radius,
                      delay == opt_delay),
             aes(y = difference),
             size = 4, colour = "blue") +
  xlab("Embedding dimension") +
  ylab("95% CI") +
  theme_bw()



# Look at result of t-test for different values of delay
ggplot(exp_test |>
         filter(embed == opt_embed, radius == opt_radius),
       aes(x = delay, y = p_value)) +
  geom_hline(yintercept = 0.05, linetype = "dashed") +
  geom_line() +
  geom_point(data = exp_test |>
               filter(embed == opt_embed,
                      radius == opt_radius,
                      delay == opt_delay),
```

```
            size = 4, colour = "blue") +
  xlab("Delay") +
  ylab("p-value") +
  theme_bw()


ggplot(exp_test |>
         filter(embed == opt_embed, radius == opt_radius),
       aes(x = delay, y = difference)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_errorbar(aes(ymin = CI_lo, ymax = CI_hi)) +
  geom_point() +
  geom_point(data = exp_test |>
               filter(embed == opt_embed,
                      radius == opt_radius,
                      delay == opt_delay),
             aes(y = difference),
             size = 4, colour = "blue") +
  xlab("Delay") +
  ylab("95% CI") +
  theme_bw()



# Look at result of t-test for different values of radius
ggplot(exp_test |>
         filter(embed == opt_embed, delay == opt_delay),
       aes(x = radius, y = p_value)) +
  geom_hline(yintercept = 0.05, linetype = "dashed") +
  geom_line() +
  geom_point(data = exp_test |>
               filter(embed == opt_embed,
                      radius == opt_radius,
                      delay == opt_delay),
             size = 4, colour = "blue") +
  xlab("Radius") +
  ylab("p-value") +
  theme_bw()


ggplot(exp_test |>
         filter(embed == opt_embed, delay == opt_delay),
```

```
       aes(x = radius, y = difference)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_errorbar(aes(ymin = CI_lo, ymax = CI_hi)) +
  geom_point() +
  geom_point(data = exp_test |>
               filter(embed == opt_embed,
                      radius == opt_radius,
                      delay == opt_delay),
             aes(y = difference),
             size = 4, colour = "blue") +
  xlab("Radius") +
  ylab("95% CI") +
  theme_bw()


#
# Keep radius fixed. Vary the other parameters.
#
ggplot(exploration |> filter(radius == opt_radius),
       aes(x = CON, y = REC)) +
  geom_rect(data = exp_test |> filter(radius == opt_radius),
            aes(fill = significant), inherit.aes = FALSE,
            xmin = -Inf, xmax = Inf,
            ymin = -Inf, ymax = Inf, alpha = 0.3) +
  geom_rect(data = exp_test |>
              filter(embed == opt_embed,
                     radius == opt_radius,
                     delay == opt_delay),
            colour = "blue",
            linewidth = 1.5,
            fill = NA,
            inherit.aes = FALSE,
            xmin = -Inf, xmax = Inf,
            ymin = -Inf, ymax = Inf, alpha = 0.3) +
  geom_violin() +
  geom_point(position = position_jitter(),
             size = 1, shape = "o") +
  theme_classic() +
  facet_grid(delay ~ embed) +
  theme(legend.position = "top")
```

```
# Same but with radius as facet instead of delay
ggplot(exploration |> filter(delay == opt_delay),
       aes(x = CON, y = REC)) +
  geom_rect(data = exp_test |> filter(delay == opt_delay),
            aes(fill = significant), inherit.aes = FALSE,
            xmin = -Inf, xmax = Inf,
            ymin = -Inf, ymax = Inf, alpha = 0.3) +
  geom_rect(data = exp_test |>
              filter(embed == opt_embed,
                     radius == opt_radius,
                     delay == opt_delay),
            colour = "blue",
            linewidth = 1.5,
            fill = NA,
            inherit.aes = FALSE,
            xmin = -Inf, xmax = Inf,
            ymin = -Inf, ymax = Inf, alpha = 0.3) +
  # geom_boxplot() +
  geom_violin() +
  geom_point(position = position_jitter(),
             size = 1, shape = "o") +
  theme_classic() +
  facet_grid(radius ~ embed) +
  theme(legend.position = "top")



# Same but with embedding fixed
ggplot(exploration |> filter(embed == opt_embed),
       aes(x = CON, y = REC)) +
  geom_rect(data = exp_test |> filter(embed == opt_embed),
            aes(fill = significant), inherit.aes = FALSE,
            xmin = -Inf, xmax = Inf,
            ymin = -Inf, ymax = Inf, alpha = 0.3) +
  geom_rect(data = exp_test |>
              filter(embed == opt_embed,
                     radius == opt_radius,
                     delay == opt_delay),
            colour = "blue",
            linewidth = 1.5,
            fill = NA,
            inherit.aes = FALSE,
```

```
           xmin = -Inf, xmax = Inf,
           ymin = -Inf, ymax = Inf, alpha = 0.3) +
# geom_boxplot() +
geom_violin() +
geom_point(position = position_jitter(),
           size = 1, shape = "o") +
theme_classic() +
facet_grid(delay ~ radius) +
theme(legend.position = "top")
```

7 Fractal Analysis

Under construction

8 Convergent Cross Mapping

Under construction

References

Coco, Moreno I., Dan Mønster, Giuseppe Leonardi, Rick Dale, and Sebastian Wallot. 2021. "Unidimensional and Multidimensional Methods for Recurrence Quantification Analysis with Crqa." *The R Journal* 13 (1): 145–63. https://doi.org/10.32614/RJ-2021-062.

Marwan, Norbert, M. Carmen Romano, Marco Thiel, and Jürgen Kurths. 2007. "Recurrence Plots for the Analysis of Complex Systems." *Physics Reports* 438 (5–6): 237–329. https://doi.org/10.1016/j.physrep.2006.11.001.