Nonlinear Methods Workshop Lecture Notes

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

2024-07-01

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)
```

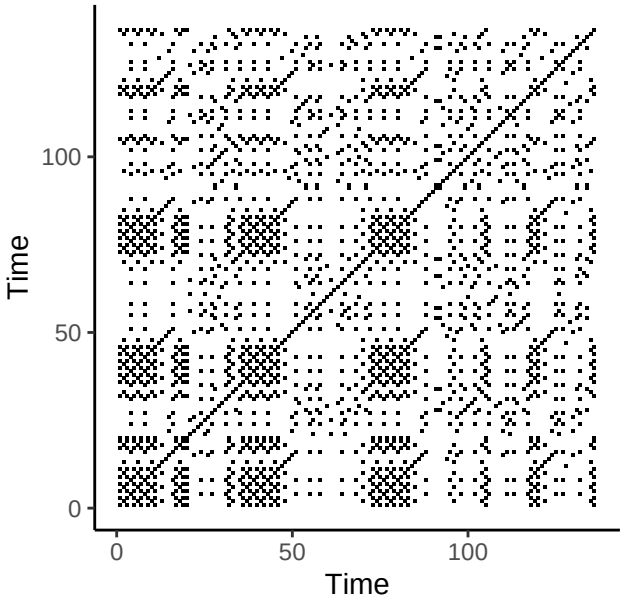


Figure 1.1: Recurrence plot of poem

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.

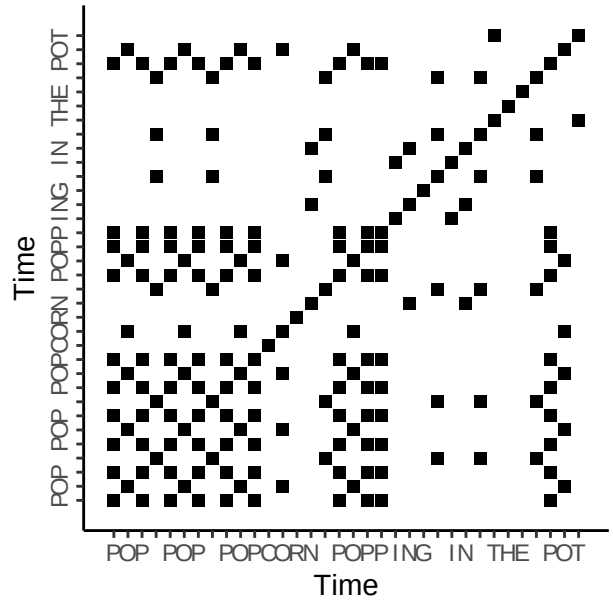


Figure 1.2: Recurrence plot of first two verses of the poem

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”.

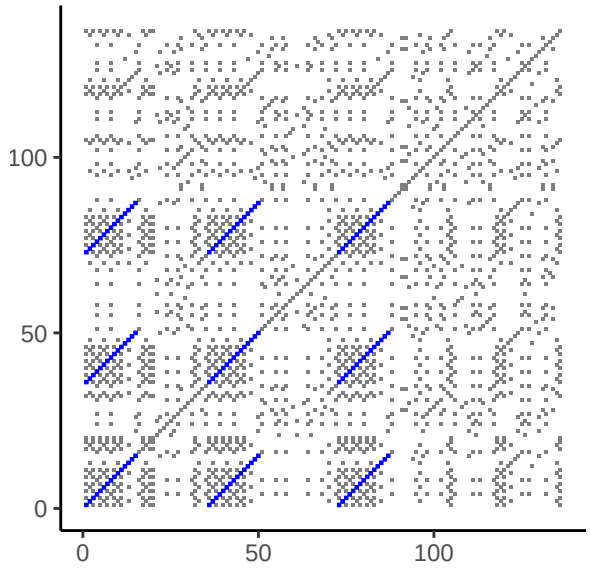


Figure 1.3: Recurring motifs in the poem

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

Convergent Cross Mapping (CCM) is a phase space-based method that aims to determine to what extent one time series, X, has a causal influence on another time series, Y, and vice versa.

CCM is based on the same phase space embedding as recurrence plot analyses, but it is not based on the concept of recurrence plots. Instead, CCM uses Takens' theorem and the non-decomposability of nonlinear systems to get a measure of how much one part of the system (X) influences another part of the system (Y).

Before looking at CCM, let us first take a look at one of the model systems we will use to understand and apply CCM.

8.1 The logistic map

The logistic map is defined by the difference equation

$$X_{n+1} = rX_n(1 - X_n)$$

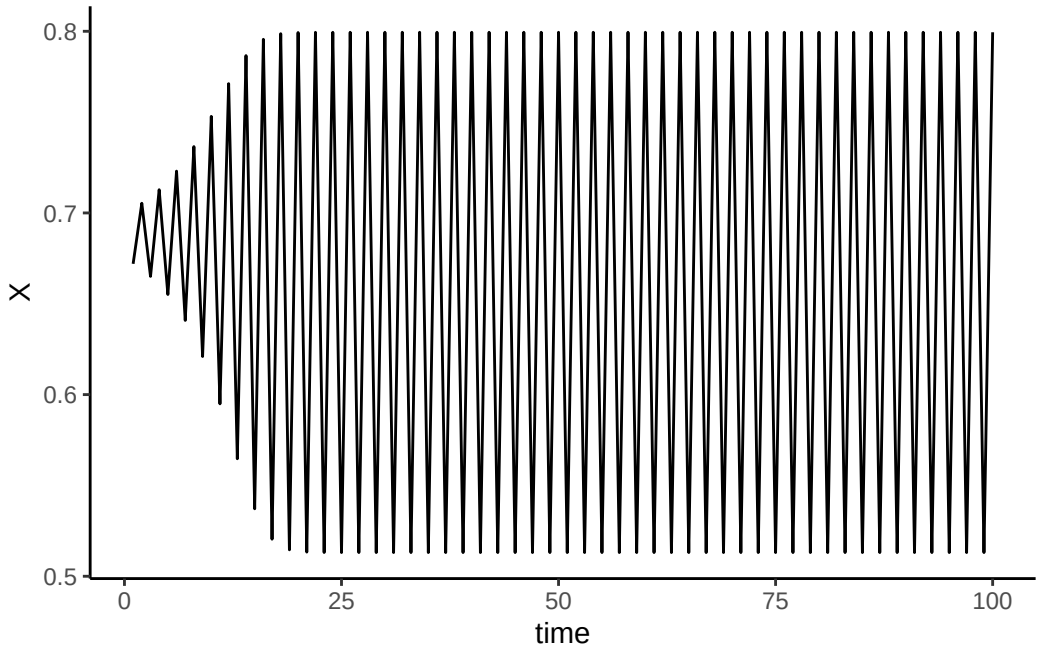
Let us start by generating a time series for a particular start value of X and a particular value of the growth parameter r. This is done by calling the function `logistic_map()` which is defined below.

```
library(ggplot2)
library(dplyr)

logistic_map <- function(x0 = 0.2,
                         r = 3.65,
                         N = 100,
                         N_skip = 0) {
  X <- rep(0, N)
  if (N_skip > 0) {
    #
    # Iterate for N_trans generations without collecting data (only
    # X[1] is updated, so the last data point will be
```

```
    # used as the new X[1]).
    #
    X0 <- x0
    for (t in 1:N_skip) {
      X[1] <-  X0 * (r - r * X0)
      X0 <-  X[1]
    }
  } else {
    X0 <- x0
    X[1] <-  X0 * (r - r * X0)
  }

  # Iterate the coupled maps for N generations
  for (t in 2:N) {
    X[t] <-  r * X[t - 1] * (1 - X[t - 1])
  }

  return(
    data.frame(
      time = 1:N,
      X = X
    )
  )
}
```

We can call this function with an initial value X_0 and some value for the growth parameter or control parameter. If we do not set the number of points to generate and the number of points to skip (the transient phase) these parameters will have the default values defined in the function above.

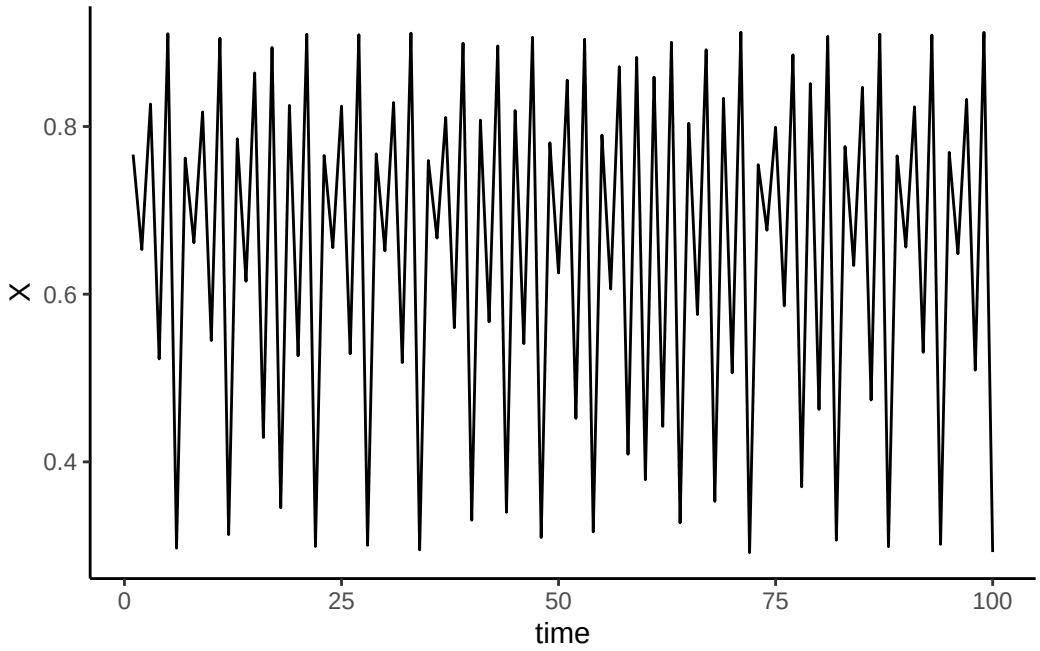
```
time_series <- logistic_map(x0 = 0.3, r = 3.2)

ggplot(time_series,
       aes(x = time, y = X)) +
  geom_line() +
  theme_classic()
```



Here, we see that the system is periodic with period 2, after a short initial period of transient behaviour.

We can choose a different value for the growth (control) parameter, r.

```
time_series <- logistic_map(x0 = 0.3, r = 3.65)

ggplot(time_series,
       aes(x = time, y = X)) +
  geom_line() +
  theme_classic()
```



Let us try to construct the bifurcation diagram, i.e., a plot of the asymptotic states as a function of the growth rate, r.

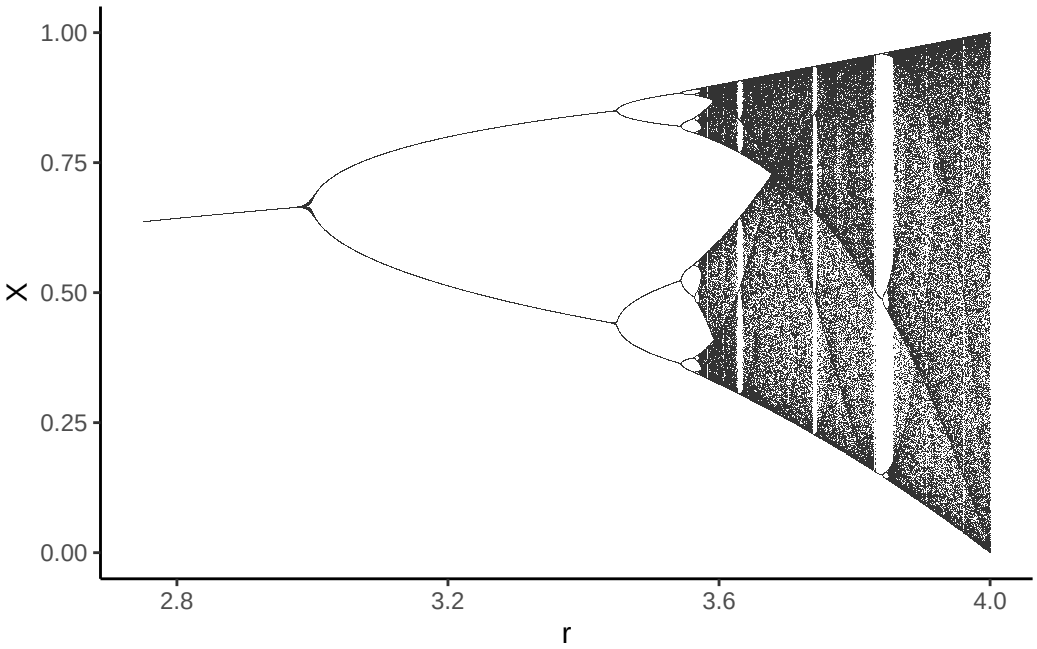
The function `get_stable_points()` collects a number, `N_plot`, points that are approximations to the asymptotic values. The first 200 points are skipped, and then `N_plot` points are saved. The minimum (`r_min`) and maximum, `r_max`, r-values can be set, as well as the number of r-values to sample in the interval [`r_min`, `r_max`].

```
get_stable_points <- function(r_min = 2.75,
                              r_max = 4,
                              r_steps = 1200,
                              N_plot = 100) {
  #  number(N_r)  and step size (dr) in r
  dr <- (r_max - r_min) / r_steps

  stable_points <- data.frame()

  r_values <- seq(r_min, r_max, by = dr)
  for (r in r_values) {
    time_series <- logistic_map(x0 = 0.2, r = r, N = N_plot, N_skip = 200)
    stable_points <- bind_rows(
      stable_points,
      data.frame(r = rep(r, N_plot),
                 X = time_series$X)
    )
```

```
  }
  return(stable_points)
}
```

Now we can use this function to generate the stable points and plot them. This may take a while, but should not take several minutes.

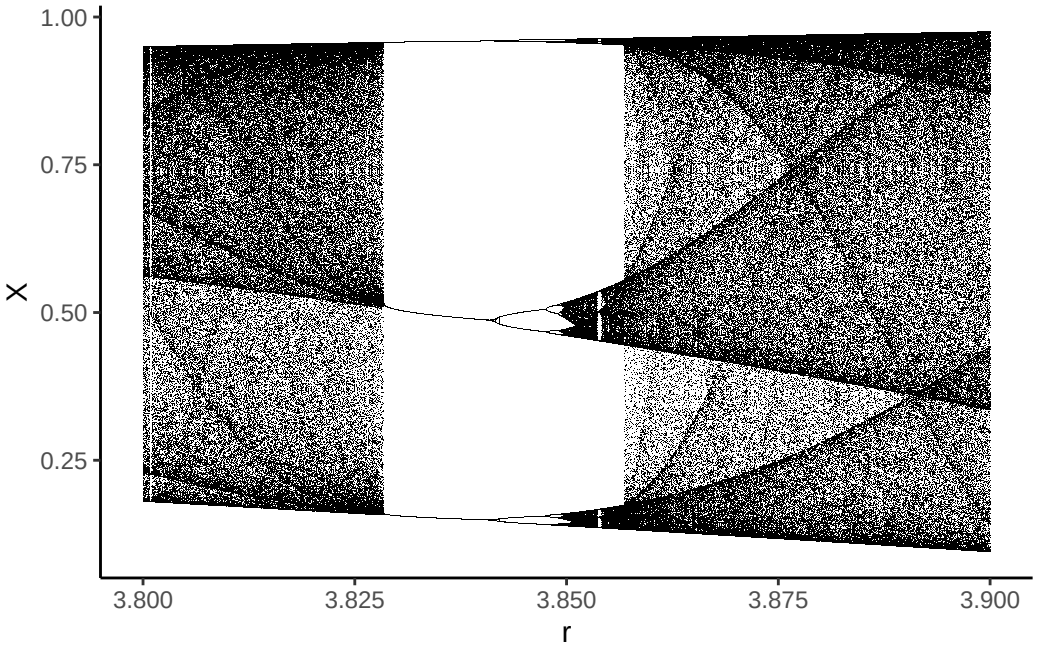
```
stable_points <- get_stable_points(r_steps = 2500, N_plot = 200)

# Retrieve the step size in r
r_values <- sort(unique(stable_points$r))
dr <- r_values[2] - r_values[1]

ggplot(stable_points,
       aes(x = r, y = X)) +
  geom_tile(width = dr, height = dr) +
  theme_classic()
```



Let us zoom in on the interval from $r = 3.8$ to $r = 3.9$. To do this, we generate a new set of stable points in this interval of r values, so a little patience is required again.

```
stable_points <- get_stable_points(r_min = 3.8,
                                   r_max = 3.9,
```

```
                                    r_steps = 2500,
                                    N_plot = 200)

# Retrieve the step size in r to set the size of tiles in plot
r_values <- sort(unique(stable_points$r))
dr <- r_values[2] - r_values[1]

ggplot(stable_points,
       aes(x = r, y = X)) +
  # geom_point(size = 0.1, stroke = 0, shape = ".") +
  # Smallest points are too large, so we use tiles instead
  geom_tile(width = dr, height = 10 * dr, fill = "black") +
  theme_classic()
```



Here we see periodic behaviour with a period of three emerge around $r \approx 3.83$ after being in a chaotic regime.

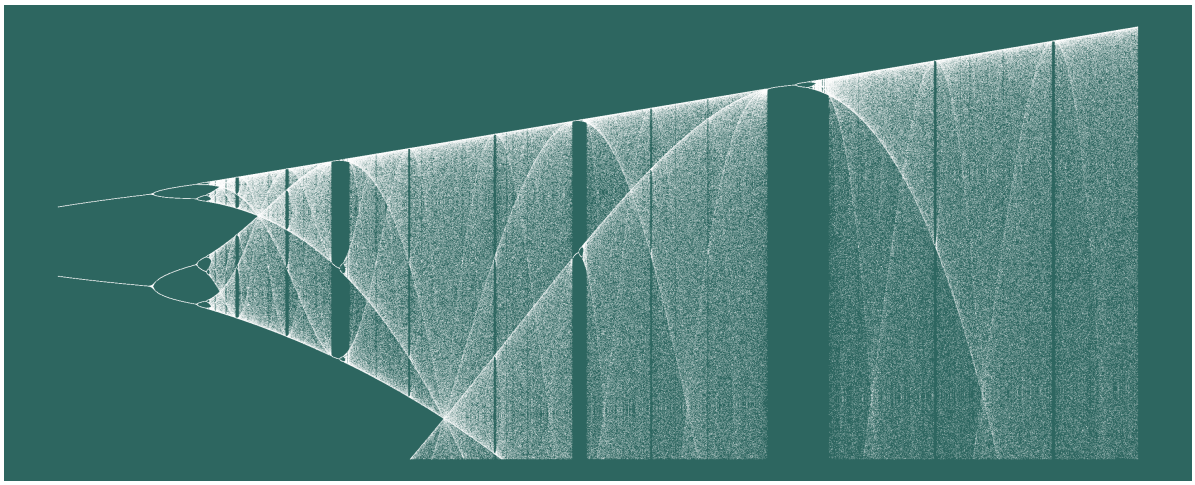
> **Question**
>
> Can you see examples of self-similarity?

8.1.1 NLM logo

If we want to reproduce the workshop logo, we have to change colors and remove axes, etc.

```
stable_points <- get_stable_points(r_min = "Set value here",
                                   r_max = "set value here",
                                   r_steps = "Set value here",
                                   N_plot = "Set value here")

# Retrieve the step size in r to set the size of tiles in plot
r_values <- sort(unique(stable_points$r))
dr <- r_values[2] - r_values[1]

bg_colour <- "#2D6660"
fg_colour <- "white"

ggplot(stable_points |>
         filter(X > "Set value here"),
       aes(x = r, y = X)) +
  geom_tile(width = dr, height = dr, fill = fg_colour) +
  theme_void() +
  theme(plot.background = element_rect(fill = bg_colour, colour = bg_colour))

# You can save the plot with this command
ggsave("NLM_logo.png", width = 10, height = 4)
```

If you set the right values, you should get something similar to the plot below.



8.2 Model example: The coupled logistic map

A simple example of a nonlinear system with two variables is the coupled logistic map. The two variables X and Y have internal dynamics depending on the growth rates r_X and r_Y. In addition there is interdependent dynamics modeled as a causal effect of X on Y depending on the coupling constant β_{YX} and a coupling in the inverse direction depending on β_{XY}.

The model is expressed by these two equations:

$$X_{n+1} = X_n(r_X - r_X X_n - \beta_{XY} Y_n)$$

$$Y_{n+1} = Y_n(r_Y - r_Y Y_n - \beta_{YX} X_n)$$

As for the simple logistic map, we will use a function to generate values by iterating the equations above that define the coupled logistic map.

```
coupled_logistic_map <- function(x0 = 0.2,
                                 y0 = 0.6,
                                 rx = 3.65,
                                 ry = 3.8,
                                 bxy = 0,
                                 byx = 0.4,
                                 N = 100,
                                 N_skip = 0) {
  X <- rep(0, N)
  Y <- rep(0, N)
  if (N_skip > 0) {
    #
    # Iterate for N_trans generations without collecting data (only
    # X[1] and Y[1] are updated, so the last data point will be
    # used as the new X[1] and Y[1]).
    #
    X0 <- x0
    Y0 <- y0
    for (t in 1:N_skip) {
      X[1] <-  X0 * (rx - rx * X0 - bxy * Y0)
      Y[1] <-  Y0 * (ry - ry * Y0 - byx * X0)
      X0 <-  X[1]
      Y0 <-  Y[1]
    }
  } else {
    X0 <- x0
```

```
    Y0 <- y0
    X[1] <-  X0 * (rx - rx * X0 - bxy * Y0)
    Y[1] <-  Y0 * (ry - ry * Y0 - byx * X0)
  }

  # Iterate the coupled maps for N generations
  for (t in 2:N) {
    X[t] <-  X[t - 1] * (rx - rx * X[t - 1] - bxy * Y[t - 1])
    Y[t] <-  Y[t - 1] * (ry - ry * Y[t - 1] - byx * X[t - 1])
  }

  return(
    data.frame(
      time = 1:N,
      X = X,
      Y = Y
    )
  )
}
```

8.2.1 Time series (hands on exercise 1)

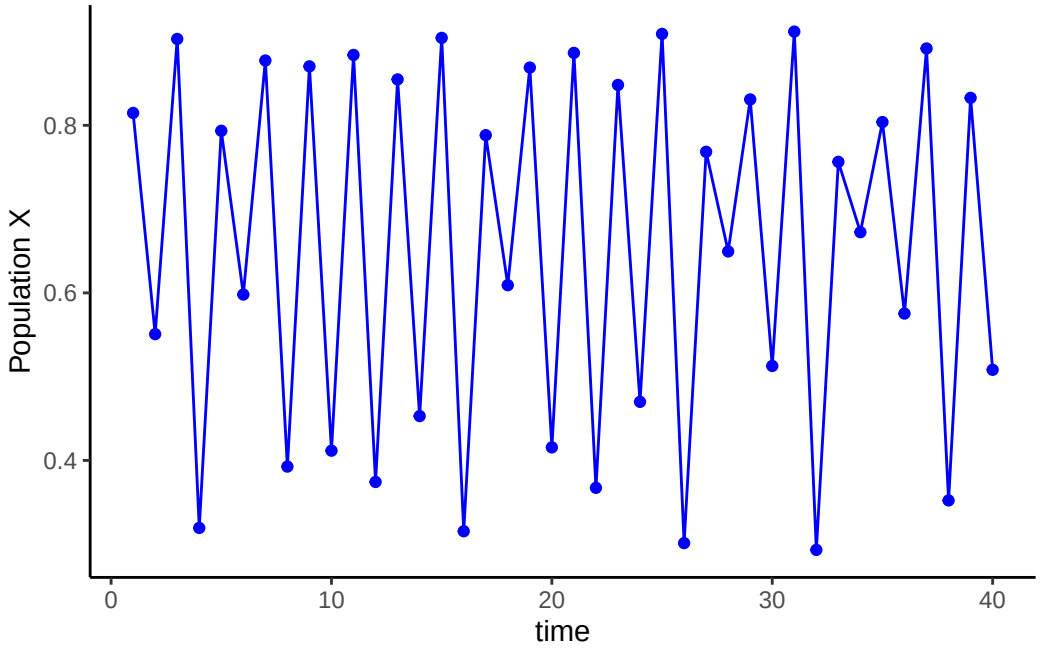
The model is implemented in the function `coupled_logistic_map` which can generate time series of length `N` given initial values `x0`and `y0` of the two variables and values for all the parameters in the model. To avoid transient and possible idiosyncratic dynamics in the beginning, an optional parameter `N_skip` can be set that will skip the first `N_skip` data points and then run the model for an additional `N` generations.

```
# Note the parameters here are not the exact ones used in the slides
time_series <- coupled_logistic_map(
  x0 = 0.2,
  y0 = 0.6,
  rx = 3.65,
  ry = 3.8,
  bxy = 0,
  byx = 0.4,
  N = 1000,
  N_skip = 300
)
```

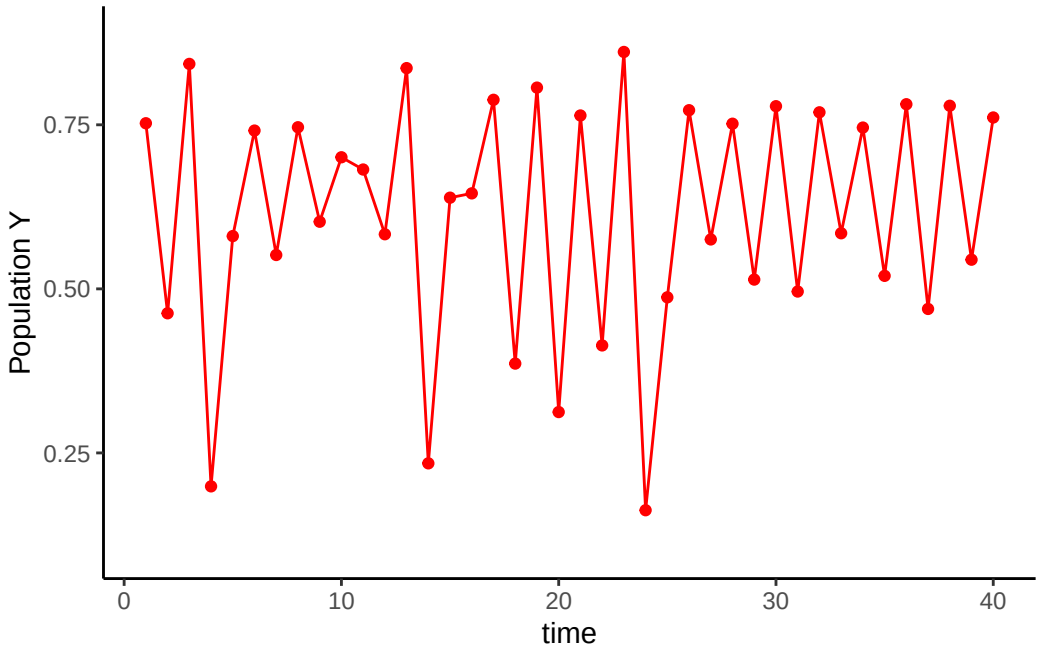
Here are plots of the first 40 values of X and Y.

> **Note**
>
> These plots are slightly different from those in the slides, because the parameters are not exactly the same.

```
library(ggplot2)

ggplot(time_series, aes(x = time, y = X)) +
  geom_point(colour = "blue") +
  geom_line(colour = "blue") +
  xlim(c(1, 40)) +
  ylab("Population X") +
  theme_classic()
```




```
ggplot(time_series, aes(x = time, y = Y)) +
  geom_point(colour = "red") +
  geom_line(colour = "red") +
  xlim(c(1, 40)) +
  ylab("Population Y") +
  theme_classic()
```



8.2.2 Attractor reconstruction

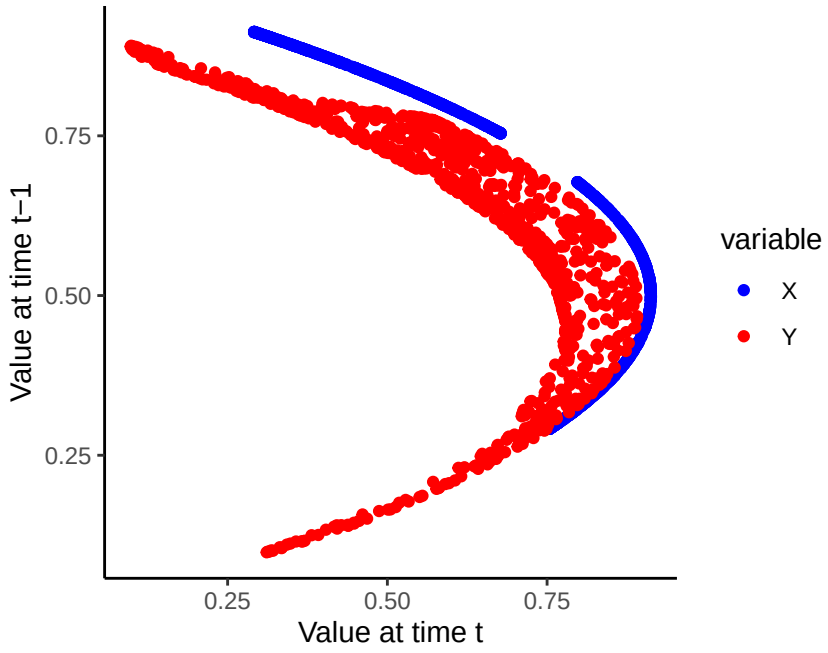
Reconstruct attractors from X and Y. Here you need to set the embedding dimension.

```
library(tseriesChaos)
library(dplyr)

# Construct attractors using tseriesChaos::embedd()
# Note: I use a negative sign for delay to match the plot in the slides.
# What happens if you have a positive sign instead?

# Set the embedding dimension here. What should it be?
dimension <- NA
delay <- 1

MX <- embedd(time_series$X, m = dimension, d = -delay)
MY <- embedd(time_series$Y, m = dimension, d = -delay)

# Change the column names
colnames(MX) <- c("t", "t_minus_delay")
colnames(MY) <- c("t", "t_minus_delay")
```

We can then add a variable name and bind the two time series into a data frame, so that we can plot the result.

```
attractors <- rbind(
  as.data.frame(MX) |> mutate(variable = "X"),
  as.data.frame(MY) |> mutate(variable = "Y")
)

ggplot(attractors, aes(x = t, y = t_minus_delay, colour = variable)) +
  geom_point() +
  scale_color_manual(values = c("X" = "blue", "Y" = "red")) +
  xlab("Value at time t") +
  ylab("Value at time t-1") +
  coord_fixed() +
  theme_classic()
```

You should get something similar to the plots below.



What did you discover?

What was the correct embedding dimension?
What happens if $\beta_{XY} > 0$?

8.2.3 Convergent cross mapping (hands-on exercise 2)

Use the `CCM()`function to calculate cross map skill.

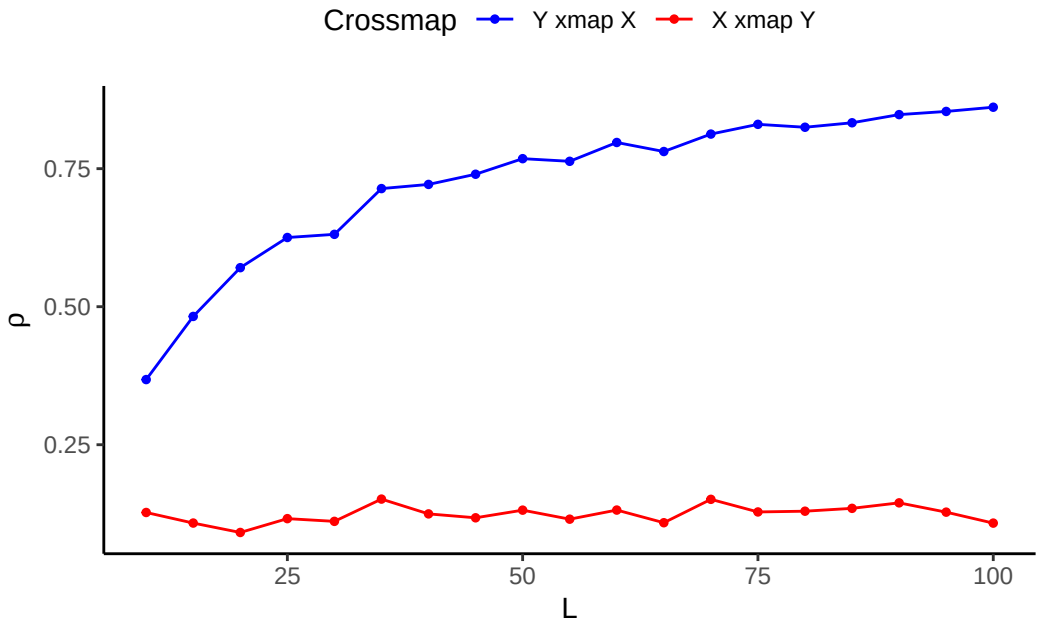
> **Note**
>
> This function takes a while to run. If you wish to experiment, use a smaller library size, and/or smaller sample. To decrease noise, increase the sample size.

```
library(rEDM)
# Perform the cross mapping. This can take some time.

#
# Exercise: supply the libSizes argument to the CCM() function
#

# Hint: it has the form "min max step", can be generated like this:
lib_min <- NA # Insert value of minimum library size here
lib_max <- NA # Insert value of maximum library size here
lib_step <- NA # Insert value of step here. Not too small!
lib_sizes <- paste(lib_min, lib_max, lib_step)

model_ccm <- CCM(dataFrame = time_series,
                 E = dimension, tau = -delay, Tp = 0,
                 columns = "Y", target = "X",
                 libSizes = lib_sizes, sample = 20,
                 showPlot = TRUE)

# Make sure the data frame has valid names (no colons)
colnames(model_ccm) <- make.names(colnames(model_ccm))
```

Since the `CCM()` function returns that data, we can also make our own plot.

```
library(tidyr)
library(latex2exp)

# Construct a long form version of the data
model_ccm_long <- model_ccm |>
  rename(Y.MX = X.Y, X.MY = Y.X, L = LibSize) %>%
  pivot_longer(cols = c("Y.MX", "X.MY"),
               names_to = "Crossmap",
               values_to = "Rho")
```

```
ggplot(model_ccm_long, aes(x = L, y = Rho, color = Crossmap)) +
  geom_point(size = 1) +
  geom_line() +
  scale_color_manual(values = c("X.MY" = "blue", "Y.MX" = "red"),
                     aesthetics = c("colour", "fill"),
                     breaks = c("X.MY", "Y.MX"),
                     labels =  c("Y xmap X", "X xmap Y")) +
  ylab(TeX("$\\rho$")) +
  theme_classic() +
  theme(legend.position = "top")
```



i What did you discover?

What an you conclude based on the plot?
Describe your observations and reasoning.

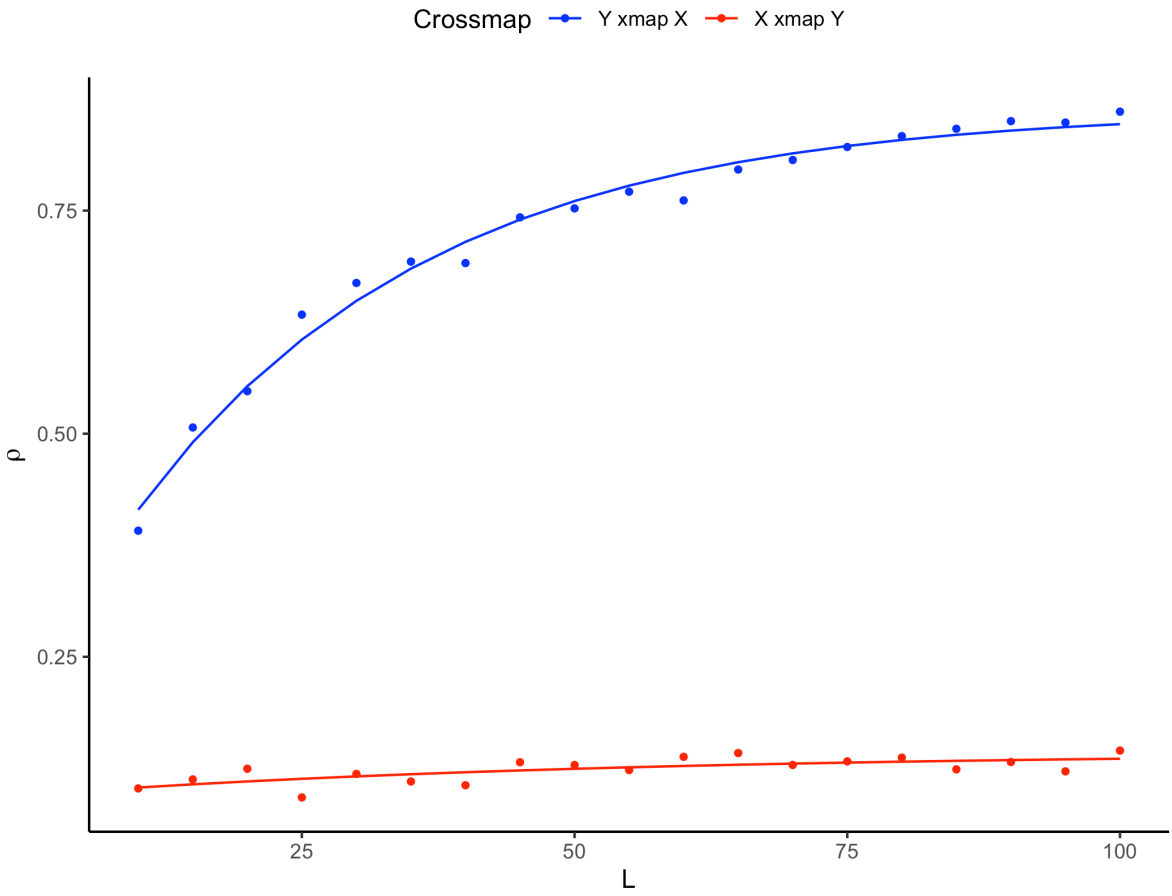
8.2.4 Fitting the convergence (hands-on exercise 3)

The first step is to fit the cross-mapping data to the function using nonlinear least squares regression. Unlike lineaer regression, there is no closed-form solution, so the regression equations are solved using a numerical iterative approach that is *not* guaranteed to converge on a solution.

You may therefore need to provide an initial guess that is not too far from the least squares solution.

```
ccm_fit_X <- nls(Rho ~ a * exp(-g * L) + r,
                 data = model_ccm_long |> filter(Crossmap == "X.MY"),
                 start = list(a = -0.5, g = 0.05, r = 0))

# summary(ccm_fit_X)

# Note: this model might not converge.
# If it fails, you can comment this out.
ccm_fit_Y <- nls(Rho ~ a * exp(-g * L) + r,
                 data = model_ccm_long |>  filter(Crossmap == "Y.MX"),
                 start = list(a = -0.1, g = 0.05, r = 0))

# summary(ccm_fit_Y)
```

```
# Extract the predictions. If a fit failed, you have to put a comment the line
model_ccm_long$fit <- NA
model_ccm_long$fit[model_ccm_long$Crossmap == "X.MY"] <- predict(ccm_fit_X)
model_ccm_long$fit[model_ccm_long$Crossmap == "Y.MX"] <- predict(ccm_fit_Y)

ggplot(model_ccm_long, aes(x = L, y = Rho, color = Crossmap)) +
  geom_point(size = 1) +
  geom_line(aes(x = L, y = fit)) +
  scale_color_manual(values = c("X.MY" = "blue", "Y.MX" = "red"),
                     aesthetics = c("colour", "fill"),
                     breaks = c("X.MY", "Y.MX"),
                     labels =  c("Y xmap X", "X xmap Y")) +
  ylab(TeX("$\\rho$")) +
  theme_classic() +
  theme(legend.position = "top")
```



8.2.4.1 Causal network

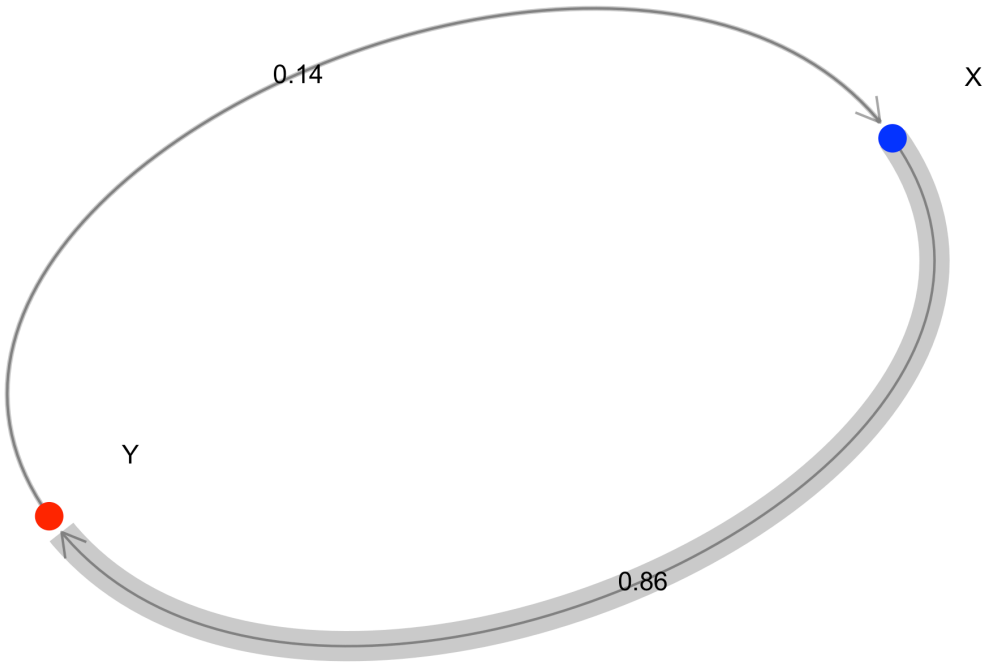
In order to construct the causal network, we start by extracting the ρ-values from the fitted models. The `broom` package makes this a lot easier.

```
library(broom)

rho_X_to_Y <- broom::tidy(ccm_fit_X) |>
  filter(term == "r") |>
  select(estimate) |>
  as.numeric()

# If the ccm_fit_Y model did not converge, replace this by zero
# rho_Y_to_X <- 0
rho_Y_to_X <- broom::tidy(ccm_fit_Y) |>
```

```
  filter(term == "r") |>
  select(estimate) |>
  as.numeric()

# Construct a data frame that defines the network
rho_links <- data.frame(from = c("X", "Y"),
                        to = c("Y", "X"),
                        rho = c(rho_X_to_Y, rho_Y_to_X))
```

Now we can use the extracted parameter estimates and use them to create a network plot of the assessed causal couplings.

```
library(igraph)
library(ggraph)

rho_graph <- rho_links %>%
  graph_from_data_frame()

ggraph(rho_graph, layout = "fr") +
  geom_edge_arc(aes(label = round(rho, 2)),
                color = "darkgrey",
                arrow = arrow(length = unit(4, 'mm')),
                end_cap = circle(3, 'mm')) +
  geom_edge_arc(aes(width = rho),
                alpha = .25,
                end_cap = circle(3, 'mm')) +
  geom_node_point(aes(color = name), size = 5) +
  geom_node_text(aes(label = name),  repel = TRUE,
                 nudge_x = 0.05, nudge_y = 0.05) +
  scale_color_manual(values = c("X" = "blue", "Y" = "red"),
                     aesthetics = c("colour", "fill")) +
  theme_graph() +
  theme(legend.position = "none")
```



8.3 Example using empirical data (hands-on exercise 4)

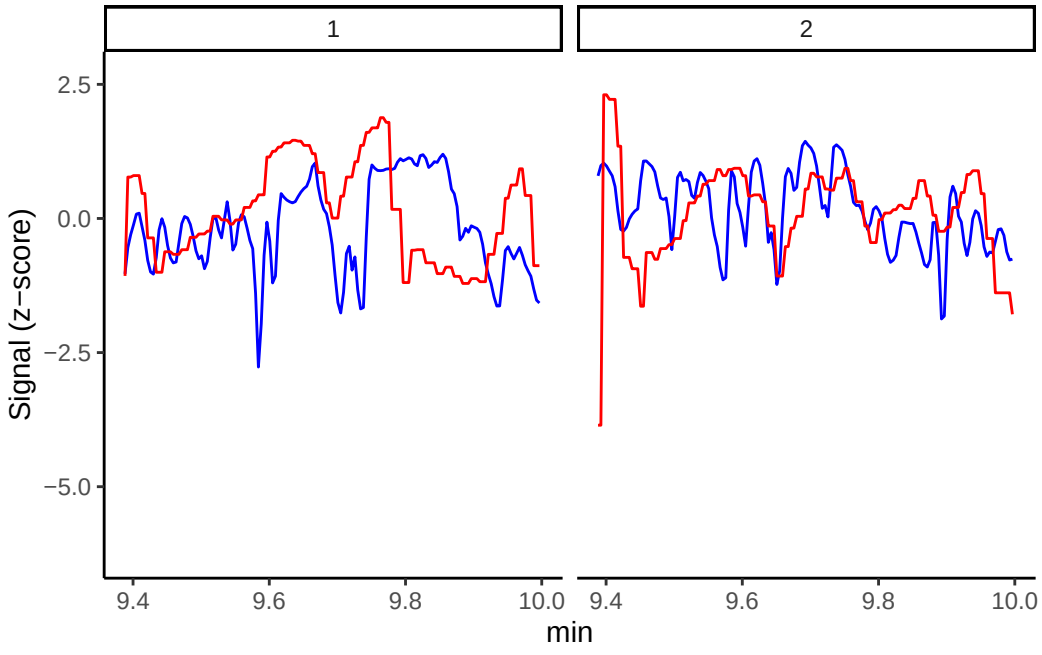
This section uses partial and down sampled data from a pilot experiment, where pairs of participants performed various tasks including speaking and moving together. The data are from an unpublished study by Fusaroli, Tylén & Mønster.

```
library(readr)
phys <- read_csv("https://tildeweb.au.dk/~au78495/physiology.csv",
                 show_col_types = FALSE)

# Make z-scores
phys$Resp1 <- scale(phys$Resp1)
phys$HR1 <- scale(phys$HR1)
phys$Resp2 <- scale(phys$Resp2)
phys$HR2 <- scale(phys$HR2)
```

```
# Produce a long form of the data with a Subject variable
phys_long <- phys %>%
  pivot_longer(cols = c(Resp1, Resp2),
               names_to = "Subject",
               names_pattern = "Resp([0-9]+)",
               values_to = "Resp") %>%
  pivot_longer(cols = c(HR1, HR2),
               names_to = "S2",
               names_pattern = "HR([0-9]+)",
               values_to = "HR") %>%
  filter(Subject == S2) %>%
  select(-S2)
```

Here is a plot of part of the data.

```
ggplot(phys_long, aes(x = min, y = Resp)) +
  geom_line(colour = "blue", na.rm = TRUE) +
  geom_line(aes(x = min, y = HR), colour = "red", na.rm = TRUE) +
  xlim(c(min(phys$min), 10)) +
  ylab("Signal (z-score)") +
  theme_classic() +
  facet_wrap(.~ Subject)
```



8.3.1 Estimate embedding parameters

Try to find a single value of the time delay and embedding dimension that can be used for all four time series, i.e., respiration and heart rate for subject 1 and subject 2.

```
# library() # You may need to load some packages here

# Option 1: Use optimizeParam from the crqa package
param <- list(method = "crqa", metric = "euclidean",
              maxlag = 20, radiusspan = 100, normalize = 0, rescale = 0,
              mindiagline = 2, minvertline = 2, tw = 0, whiteline = FALSE,
              recpt = FALSE , side = "both", datatype = "continuous",
              fnnpercent = 10, typeami = "mindip")

# You need to put in some time series here

# embed_param <- crqa::optimizeParam(TS1, TS2,
#                       par = param)

# Option 2: Use mutualInformation and estimateEmbeddingDim from nonlinearTseries

# nonlinearTseries::mutualInformation() # Fill in the blanks
# nonlinearTseries::estimateEmbeddingDim() # Fill in the blanks

# Option 3: Use mutual and false.nearest from tseriesChaos

# Get the variable names, but drop the time variables.
var_names <- colnames(phys)[3:6]

# Calculate AMI for each variable and collect the results in a data frame
AMI <- data.frame()
for (v in var_names) {
  v_ami <- tseriesChaos::mutual(phys[, v], lag.max = 20, plot = FALSE)
  AMI <- rbind(AMI,
               as.vector(v_ami) %>%
                 as.data.frame() %>%
                 mutate(var = v))
}

# Set column names and add time delay
colnames(AMI) <- c("ami", "var")
AMI <- AMI %>%
  group_by(var) %>%
```

```
  mutate(delay = row_number()) %>%
  ungroup()

# Plot the results, so the delay can be estimated
ggplot(AMI, aes(x = delay, y = ami)) +
  geom_line() +
  geom_point() +
  ylab("AMI") +
  theme_classic() +
  facet_wrap(.~ var, ncol = 2)
ggsave("Plots/physiology_ami.pdf",
       width = 16, height = 10, units = "cm")

#
# Set the delay here
#
delay <- NA # Enter value

# Calculate FNN for each variable and collect the results in a data frame
FNN <- data.frame()
for (v in var_names) {
  v_fnn <- tseriesChaos::false.nearest(series = phys[, v],
                   m = 15, d = delay, t = 1, eps = 1)
  FNN <- rbind(FNN,
               as.vector(v_fnn["total", ]) %>%
                 as.data.frame() %>%
                 mutate(var = v))
}

# Set column names and add time delay
colnames(FNN) <- c("fnn", "var")
FNN <- FNN %>%
  group_by(var) %>%
  mutate(m = row_number()) %>%
  ungroup()

# Plot the results, so the embedding dimension can be estimated
ggplot(FNN, aes(x = m, y = fnn)) +
  geom_line() +
  geom_point() +
  ylab("FNN") +
  theme_classic() +
```

```
  facet_wrap(.~ var, ncol = 2)
ggsave("Plots/physiology_fnn.pdf",
       width = 16, height = 10, units = "cm")

#
# Set embedding dimension
#
dimen <- NA # Enter value
```

8.3.2 Convergent cross mapping

Look at all pairwise cross mappings

```
# Generate all pairwise combinations of variables
variables <- c("Resp1", "HR1", "Resp2", "HR2")
combinations <- combn(variables, 2)
# Get the row numbers and add them as the first column as required by CCM
idx <- as.numeric(rownames(phys))
ccm_data <- cbind(time = idx, phys)

# Create an empty list to hold results
ccm_output_list <- list()
for (pair in 1:ncol(combinations)) {
  ccm_out <- CCM(dataFrame = ccm_data, E = dimen, tau = -delay, Tp = 0,
                 columns = combinations[1, pair],
                 target = combinations[2, pair],
                 libSizes = "20 450 10", sample = 30,
                 showPlot = FALSE)

  # Rename columns to valid R names
  colnames(ccm_out) <- make.names(colnames(ccm_out))
  ccm_output_list[[pair]] <- ccm_out
}

# Merge all the data frames in the list to a single data frame
ccm_all_pairs <- Reduce(function(x,y) merge(x = x, y = y, by = "LibSize"),
       ccm_output_list)
```

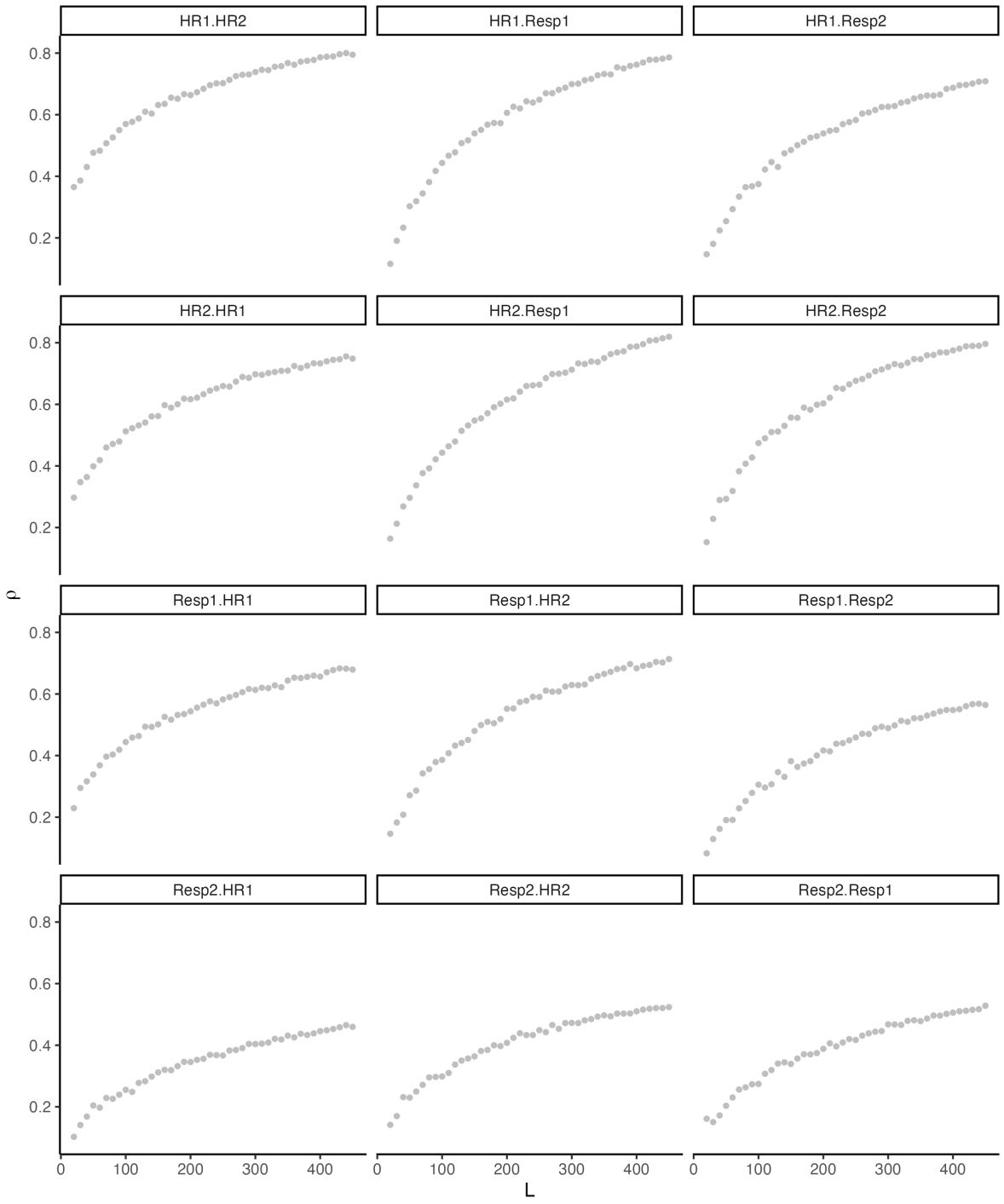
Create a long form version of the data for easier plotting.

```
ccm_results <- ccm_all_pairs %>%
  rename(L = LibSize) %>%
  pivot_longer(cols = -L,
               names_to = "Crossmap",
               values_to = "Rho")
```

Make our own plot.

```
ggplot(ccm_results, aes(x = L, y = Rho)) +
  geom_point(size = 1, colour = "grey") +
  ylab(TeX("$\\rho$")) +
  theme_classic() +
  theme(legend.position = "top") +
  facet_wrap(.~ Crossmap, ncol = 3)
```

It should produce something similar to the plot below.



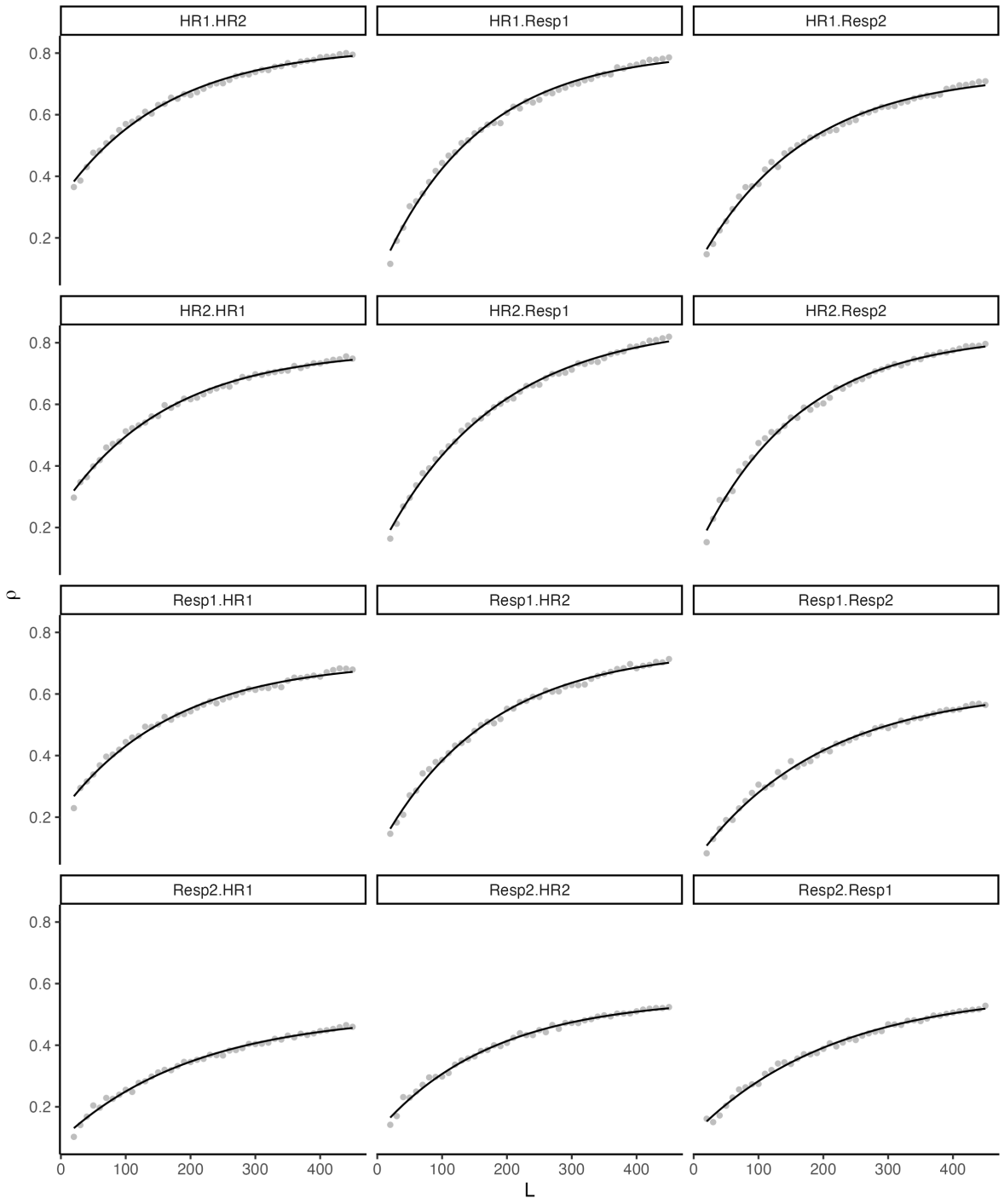
Fit all cross-mapped values to $\rho(L)$ and gather the r-values (ρ_∞) in a data frame.

```
ccm_results$fit <- NA

# To catch errors, put nls() call inside try()
model_list <- list()
for (xmap in unique(ccm_results$Crossmap)) {
  model_fit <- try(
    nls(Rho ~ a * exp(-g * L) + r,
                    data = ccm_results %>% filter(Crossmap == xmap),
                    start = list(a = -0.55, g = 0.05, r = 0.5)),
    TRUE
  )
  if(class(model_fit) == "try-error") {
    next
  } else {
    model_list[xmap] <- model_fit
  }
}

# Now calculate model predictions and add them to ccm_results
for (xmap in names(model_list)) {
  ccm_results$fit[ccm_results$Crossmap == xmap] <- model_list[[xmap]]$predict()
}
```

Plot the fitted results

```
ggplot(ccm_results, aes(x = L, y = Rho)) +
  geom_point(size = 1, colour = "grey") +
  geom_line(aes(x = L, y = fit)) +
  ylab(TeX("$\\rho$")) +
  theme_classic() +
  theme(legend.position = "top") +
  facet_wrap(.~ Crossmap, ncol = 3)
```

If all went well, you should get a plot like the one shown below.

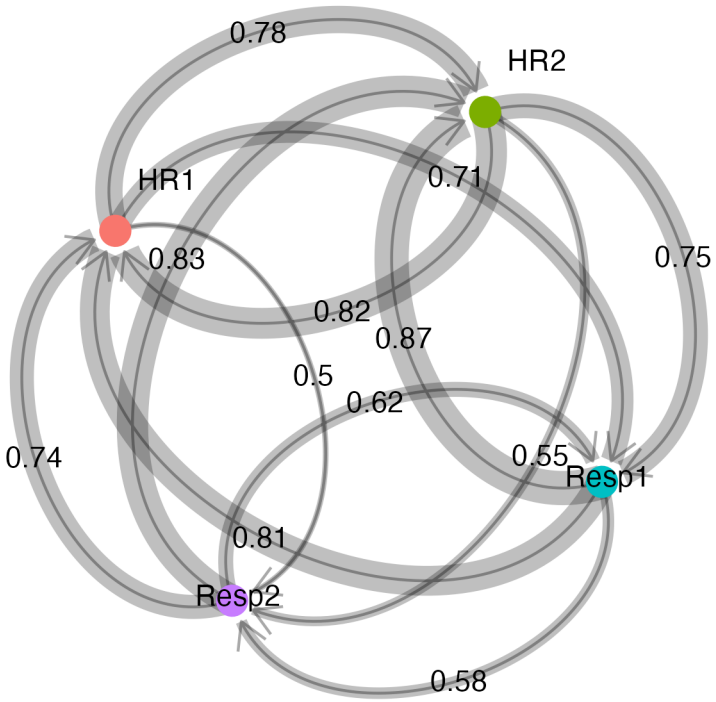


Extract the fitted rho values and construct a data frame that defines the causal network.

```
# Construct a data frame that defines the network
rho_links <- data.frame(from = character(),
                        to = character(),
                        rho = numeric())

# Extract the variables involved in the cross mapping
# and the fitted rho at infinite library size.
# Note: causality goes opposite to cross map direction
# i.e., high rho X:Y means causal link from Y to X.

counter <- 0
row_list <- list()
for (xmap in names(model_list)) {
  counter <- counter + 1
  from_to <- strsplit(xmap, "\\.")[[1]]
  from_value <- from_to[2]
  to_value <- from_to[1]
  rho_value <- as.numeric(model_list[[xmap]]$getPars()["r"])
  row_list[[counter]] <-  data.frame(from = from_value,
                                     to = to_value,
                                     rho = rho_value)
}

rho_links <- do.call(rbind, row_list)
```

```
rho_graph <- rho_links |>
  graph_from_data_frame()

ggraph(rho_graph, layout = "fr") +
  geom_edge_arc(aes(label = round(rho, 2)),
                color = "darkgrey",
                arrow = arrow(length = unit(4, 'mm')),
                end_cap = circle(3, 'mm')) +
  geom_edge_arc(aes(width = rho),
                alpha = .25,
                end_cap = circle(3, 'mm')) +
  geom_node_point(aes(color = name), size = 5) +
  geom_node_text(aes(label = name),  repel = TRUE,
                 nudge_x = 0.05, nudge_y = 0.05) +
  theme_graph() +
  theme(legend.position = "none")
```

The resulting network graph might look something like this, but note that the layout can change a lot from time to time, so yours may look quite different, even with the same values.



Note that some variables, such as speaking and movement of the two subjects have been left out of the data for simplicity. Could these variables explain some of the observed high couplings between the two subjects?

Further reading and useful links

R package: rEDM

MATLAB code: xmap

G. Sugihara, R. May, H. Ye, C.-h. Hsieh, E. Deyle, M. Fogarty, S. Munch. (2012). Detecting causality in complex ecosystems *Science*, 338 (6106), 496-500, DOI:10.1126/science.1227079

Mønster, D., Fusaroli, R., Tylén, K., Roepstorff, A., & Sherson, J. F. (2017). Causal inference from noisy time-series data—Testing the Convergent Cross-Mapping algorithm in the presence of noise and external influence. *Future Generation Computer Systems*, 73, 52-62. DOI: 10.1016/j.future.2016.12.009

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.