Dynamics of Complex Systems

Fred Hasselman & Maarten Wijnants

2016-11-23

Contents

Chapter 1

Course guide

This course discusses research methods and analysis techniques that allow for the study of human behaviour from a complex systems perspective. Complexity research transcends the boundaries the classical scientific disciplines in terms of explanatory goals (e.g. causal-mechanistic) and is a hot topic in physics, mathematics, biology, economy and psychology.

The main focus in the cognitive behavioural sciences is a description and explanation of behaviour based on interaction dominant dynamics: Many processes interact on many different (temporal and spatial) scales and observable behaviour emerges out of those interactions through a process of self-organization or soft-assembly. Contrary to what the term might suggest, complexity research is often about finding simple models that are able to simulate a wide range of complex behaviour.

This approach differs fundamentally from the more classical approaches where behaviour is caused by a system of many hidden (cognitive) components which interact in sequence as in a machine (component dominant dynamics). The most important difference is how 'change', and hence the time-evolution of a system, is studied.

The main focus of the course will be 'hands-on' data-analysis in `R`, or, in `Matlab` if student is already familiar with the scritping language.

Topics include: Analysis of fractal geometry (i.e. pink noise) in time series (Standardized Dispersion Analysis, Power Spectral Density Analysis, Detrended Fluctuation Analysis); Nonlinear and chaotic time series analysis (Phase Space Reconstruction, (Cross) Recurrence Quantification Analysis, Entropy Estimation); Growth Curve models; Potential Theory; and Catastrophe Theory (Cusp model), Complex Network Analysis.

1.1 Learning objectives

Students who followed this course will be able to critically evaluate whether their scientific inquiries can benefit from adopting theories, models, methods and analyses that were developed to study the dynamics of complex systems. The student will be able to understand in more detail the basics of formal theory evaluation, and be able to recognize, interpret and deduce theoretical accounts of human behaviour that are based on component-dominant versus interaction-dominant ontology.

Students will be able to use the basic mathematical models and analyses that allow study of complex interaction-dominant behaviour. Students who finish the course will be able to conduct analyses in `Excel`, SPSS, and `R` or `Matlab` and interpret the results from basic (non-)linear time series methods. At the end of this course, students have reached a level of understanding that allows them to find relevant scientific sources and understand and follow up on new developments in the complex systems approach to behavioural science.

Goals Summary

- Read and understand papers that use a complex systems approach to study human behaviour.
- Simulate the basic dynamical models.
- Perform the basic analyses.

1.2 Teaching methods

Each meeting starts with a *lecture session* addressing the mathematical background and practical application of a particular aspect of a model, or analysis technique. During the *assignment session*, students will get hands-on experience with applying the models or analysis techniques discussed during the lecture session by completing assignments provided on blackboard for each session.

Preparation

To prepare for each lecture students read a contemporary research paper or watch a videolecture (e.g., TED) featuring complexity theory and its application on a topic in behavioural science that will be discussed in the subsequent lecture. Students are required to formulate questions about each paper, and to initiate a discussion with their fellow-students on Blackboard.

Before each lecture, students should:

- Read (parts of) a scientific article, or watch a videolecture featuring a complex systems perspective and/or methodology.
- Ask (or answer) a question about what they have read / seen in the appropriate discussion forum on Blackboard.
  - The answers students provide will be discussed during the lecture.

1.3 Literature

The following is part of the literature for this course:

- Lecture slides.
- Articles and book chapters listed in the `Literature` folder on Blackboard for each session.
- In addition, at the secretariat of PWO (5th floor, Spinoza building) selected chapters from the book "Dynamical Psychology" by Jay Friedenberg are available. It is not necessary to own the book to complete this course, but if you can find a copy, it may help to structure all the information provided during the course.

Note: The literature for each session on Blackboard is provided for reference, to fascilitate looking up a topic when it is needed to complete the weekly assignments or the take-home exam.

Table **1.1**
Times and Places 2016-2017

Lecture	Topic	Week	Day	Date	Activity type	Location
1	Introduction to Complexity Science	46	Tue	2016-11-08	Lecture	SP A -1.59
1	Mathematics of Change I	46	Wed	2016-11-09	Lab course	SP A -1.55.A/B
2	Multivariate Systems	47	Tue	2016-11-15	Lecture	SP A -1.59
2	Mathematics of Change II	47	Wed	2016-11-16	Lab course	SP A -1.55.A/B
3	Additive vs. Multiplicative Interactions	48	Tue	2016-11-22	Lecture	SP A -1.59
3	Basic Timeseries Analysis	48	Wed	2016-11-23	Lab course	SP A -1.55.A/B
4	Fractal Geometry in Timeseries	49	Tue	2016-12-29	Lecture	SP A -1.59
4	Fluctuation and Disperion analyses I	49	Wed	2016-12-30	Lab course	SP A -1.55.A/B
5	Multi-Fractal Geometry in Timeseries	50	Tue	2016-12-06	Lecture	SP A -1.59
5	Fluctuation and Disperion analyses II	50	Wed	2016-12-07	Lab course	SP A -1.55.A/B
6	Quantifying Continuous Dynamics	51	Tue	2016-12-13	Lecture	SP A -1.59
6	Recurrence Quantification Analysis (RQA)	51	Wed	2016-12-14	Lab course	SP A -1.55.A/B
7	Quantifying Categorical and Interaction Dynamics	1	Tue	2016-01-20	Lecture	SP A -1.59
7	Categorical and Cross-Recurrence Quantification Analaysis (CRQA)	1	Wed	2016-01-21	Lab course	SP A -1.55.A/B
8	Potential Theory and Catastrophe Theory	2	Tue	2017-01-10	Lecture	SP A -1.59
8	The Cusp Model	2	Wed	2017-01-11	Lab course	SP A -1.55.A/B
9	Complex Networks	3	Tue	2017-01-17	Lecture	SP A -1.59
9	Q&A session	3	Wed	2017-01-21	Lab course	TvA 8.00.14

1.4 Schedule

The dates and locations can be found below. All lectures are on Tuesday from 10.45 to 12.30. The practical sessions take place on Wednesday from 15.45 to 17.30.

1.5 Examination

The evaluation of achievement of study goals takes two forms:

- **Participation** - The ability to formulate a question about an advanced topic is a first step towards understanding, answering a question that was posted by a peer is a second step. Questions and answers will not be graded, there will be a check to see if a student participated in all sessions.
- **Final Assignment** - This take-home assignment will be provided at the end of the course. It will consist of a series of practical assignments and at least one essay question to test theoretical knowledge. The submission deadline is two weeks after the last lecture.

Grading

The take home exam will be graded as a regular exam. A student passes the course if the exam grade is higher than 5.5 AND if the student participated in the discussion on Blackboard each session.

Submitting the assignment

The take-home exam must be submitted by sending them by email to both `f.hasselman@pwo.ru.nl` AND `m.wijnants@pwo.ru.nl` no later than **February 1st**.

1.6 We use R!

This text was transformed to HTML, PDF en ePUB using bookdown(Xie, 2016a) in **RStudio**, the graphical user interface of the statistical language **R** (R Core Team, 2016). bookdown makes use of the R version of markdown called Rmarkdown (Allaire et al., 2016), together with knitr (Xie, 2016c) and pandoc.

We'll use some web applications made in Shiny (Chang et al., 2016)

Other R packages used are: DT (Xie, 2016b), htmlTable (Gordon, 2016), plyr (Wickham, 2016a), dplyr (Wickham and Francois, 2016),tidyr (Wickham, 2016b), png (Urbanek, 2013), rio (hong Chan and Leeper, 2016).

Part I

Assignments

Chapter 2

Mathematics of Change I

In this assignment you will build two (relatively) simple one-dimensional maps. We start with the *Linear Map* and then proceed to the slightly more complicated *Logistic Map* (aka *Quadratic map*). If you are experienced in `R` or `Matlab` you can try to code the models using the hints in section 2.3, otherwise, continue below with your favourite spreadsheet software (e.g., Excel, Numbers, GoogleSheets).

2.1 The Linear Map

Equation (2.1) is the ordinary difference equation (ODE) discussed in the lecture (see lecture notes **??**) is called the *Linear Map*:

$$Y_{t+1} = Y_{t=0} + r * Y_t \tag{2.1}$$

In these excersises you will simulate *time series* produced by the change process in equation **??**(eq:linmap) for different parameter settings of the growth-rate parameter r (the *control parameter*) and the initial conditions Y_0. This is different from a statistical analysis in which parameters are estimated from a data set. The goal of the assignments is to get a feeling for what a dynamical model is, and how it is different from a linear statistical regression model like GLM.

2.1.1 The Linear Map in a Spreadsheet

Before you begin, be sure to check the following settings:

- Open a Microsoft Excel worksheet, a Google sheet, or other spreadsheet.
- Check whether the spreadsheet uses a 'decimal-comma' ($0,05$) or 'decimal-point' (0.05) notation.
  - The numbers given in the assignments of this course all use a 'decimal-point' notation.
- Check if the $ symbol fixes rows and columns when it used in a formula in your preferred spreadsheet program.
  - This is the default setting in Microsoft Excel and Google. If you use one of those programs you are all set, otherwise you will have to replace the $ used in the assignments with the one used by your software.
- Type `r` in cell `A5`. This is the *control parameter*. It receives the value 1.08 in cell`B5`.
- Type Y_0 in cell `A6`. This is the *initial value*. It receives the value 0.1 in cell `B6`.

- Use the output level (Y_t) of every step as the input to calculate the next level (Y_{t+1}).
  - Rows in the spreadsheet will represent the values of the process at different moments in time.
- Put the initial value (Y_0) in cell `A10`. This cell marks time $t = 0$.
  - To get it right, type: `=$B$6`. The = means that (in principle) there is a 'calculation' going on (a function is applied). The $ determines that column ($B) as well as the row ($6) keep the same value (i.e., constant) for each time step.
- Enter the **Linear Map** as a function in each cell. Type `=$B$5*A10` in cell `A11`.
  - This means that the value of cell `A11` (i.e. $Y_{t=1}$) will be calculated by multiplying the value of cell `B5` (parameter `r`) with the value of cell `A10` (previous value, here: $Y_{t=0}$). If everything is all right, cell `A11` now shows the value 0.108.
- Repeat this step for cell `A12`.
  - Remember what it is you are doing! You are calculating $Y_{t=2}$ now (i.e. the next step), which is determined by $Y_{t=1}$ (i.e., the previous step) and the parameter `r`.
- Now repeat this simple iterative step for 100 further steps. Instead of typing everything over and over again, copy-paste the whole thing.Most spreadsheet programs will automatically adjust the formula by advancing each row or column number that aren't fixed by $.
  - Copy cell `A12` all the way from `A13` to `A110` (keep the `SHIFT` button pressed to select all cells).

`You have just simulated a time series based on a theoretical change process!`

2.1.2 Visualizing the time series

1. Select cells `A10` to `A110` Create a line graph (`Insert`, 2D-line, Scatter). This will show you the graph. (There are other ways to do this, by the way, which work just as well.) You can play with the setting to make the best suitable view, like rescaling the axes.

2. If you change the values in cells `B5` and `B6` you will see an immediate change in the graph. To study the model???s behaviour, try the following growth parameters:

   - $a = -1.08$

- $a = 1,08$
- $a = 1$
- $a = -1$

3. Change the initial value Y_0 in cell B6 to 10. Subsequently give the growth parameter in cell B5 the following values 0.95 and $???0.95$.

2.2 The Logistic Map in a spreadsheet

The Logistic Map takes the following functional form:

$$Y_{t+1} = r * Y_t * (1 - Y_t) \tag{2.2}$$

To get started, copy the spreadsheet from the previous assignment to a new sheet. The parameters are the same as for the Linear Map, there has to be an initial value $Y_{t=0}$ (no longer explicit as a constant in equation (2.2)) and the control parameter r. What will have to change is

- Start with the following values for control parameter r:
  - $r = 1.9$
  - $Y_0 = 0.01$ (in A6).
- Take good notice of what is constant (parameter r), so for which the $ must be used, and what must change on every iterative step (variable Y_t).

2.2.1 Visualizing the time series and explore its behaviour

- Create the time series graphs as for the Linear Map.

To study the behavior of the Logistic Map you can start playing around with the values for the parameters and the initial values in cells B5 and B6.

- Be sure to try the following settings for r:
  - $r = 0.9$
  - $r = 1.9$
  - $r = 2.9$
  - $r = 3.3$
  - $r = 3.52$
  - $r = 3.9$
- Set r at 4.0:
  - Repeat the iterative process from A10 to A310 (300 steps)
  - Now copy A10:A310 to B9:B309 (i.e., move it one cell to the right, and one cell up)
  - Select both columns (A10 to B309!) and make a scatter-plot

2.2.2 The return plot

The plot you just produced is a so called **return plot**, in which you have plotted Y_{t+1} against Y_t.

- Can you explain the pattern you see (a 'parabola') by looking closely at the functional form of the Logistic Map? (hint: it's also called **Quadratic Map**)
  - Look at what happens in the return plot when you change the value of the parameter r (in A5).
  - What do you expect the return plot of the Linear Map will look like? Try it!

The meaning and use of this plot was discussed in the next session

2.3 Using R or Matlab to do the exercises.

The best (and easiest) way to simulate these simple models is to create a function which takes as input the parameters (Y_0, r) and a variable indicating the length of the time series.

For example for the Linear Map:

```
# In R
linearMap <- function(Y0 = 0, r = 1, N = 100){

    # Initialize Y as an NA vector of size N with as first entry Y0
    Y <- c(Y0, rep(NA,N-1))

    for(i in 1:N){

    Y[i+1] <- # Implement the function here

    }

    return(Y)
}


# In Matlab
function linearMap(Y0,r,N)
 # Implement the function here
end
```

Creating the time series graphs and the return plot should be easy if the function `linearMap` returns the time series. Both R and Matlab have a `plot()` function you can call.[1]

[1] Both R and Matlab have specialized objects to represent timeseries, and functions and packages for timeseries analysis. They are especially convenient for plotting time and date information on the X-axis. See Solutions: Mathematics of Change I

Chapter 3

Mathematics of Change II

In this assignment we will build a more sophisticated growth modeland look at its properties. The model will be the growth model by Van Geert (1991 etc.) as discussed in the book chapter you read. If your are experienced in R or Matlab you can try to code the models following the hints in section 2.3.

3.0.1 The growth model by Van Geert (1991)

The growth model by Van Geert has the following form:

$$L_{t+1} = L_t * (1 + r - r * \frac{L_t}{K}) \tag{3.1}$$

Note the similarities to Equation (2.2), the (stylized) logistic map.

3.1 The growth model in a spreadsheet

Before you begin, be sure to check the following settings (same as first asignment):

- Open a Microsoft Excel worksheet or a Google sheet
- Check whether the spreadsheet uses a 'decimal-comma' ($0,05$) or 'decimal-point' (0.05) notation.
  - The numbers given in the assignments of this course all use a 'decimal-point' notation.
- Check if the $ symbol fixes rows and columns when it used in a formula in your preferred spreadsheet program.
  - This is the default setting in Microsoft Excel and Google Sheets. If you use one of those programs you are all set.

To build it repeat some of the steps you performed in assignment 2 on a new worksheet.

- Define the appropriate constants (r in A5, L_0 in A6) and prepare the necessary things you need for building an iterative process.
- In particular, add the other parameter that appears in Van Geert???s model:
  - Type K in cell A7. This is the *carrying capacity*. It receives the value 1 in cell B7.
- Start with the following values:
  - $r = 1.2$
  - $L_0 = 0.01$

Take good notice of what is constant (parameters r and K), for which the $ must be used, and what must change on every iterative step (variable L_t). Take about 100 steps.

- Create the graphs
- You can start playing with the values for the parameters and the initial values in cells B5, B6 and B7. To study this model???s behavior, be sure to try the following growth parameters:
  - $r = 1.2$
  - $r = 2.2$
  - $r = 2.5$
  - $r = 2.7$
  - $r = 2.9$
- For the carrying capacity K (cell B7) you can try the following values:
  - $K = 1.5$
  - $K = 0.5$. (Leave $r = 2.9$. Mind the value on the vertical axis!)
- Changes in the values of K have an effect on the height of the graph. The pattern itself also changes a bit. Can you explain why this is so?

3.2 Conditional growth: Jumps and Stages

Auto-conditional jumps

Suppose we want to model that the growth rate r increases after a certain amount has been learned. In general, this is a very common phenomenon, for instance: when becoming proficient at a skill, growth (in proficiency) is at first slow, but then all of a sudden there can be a jump to the appropriate (and sustained) level of proficiency.

- Take the model you just built as a starting point with $r = 0.1$ (B5)
  - Type 0.5 in C5. This will be the new parameter value for r.
  - Build your new model in column B (leave the original in A).
- Suppose we want our parameter to change when a growth level of 0.2 is reached. We???ll need an IF statement which looks something like this: IF $L > 0.2$ then use the parameter value in C5, otherwise use the parameter value in B5.
  - Excel has a built in IF function (may be ALS in Dutch).
  - In the first cell in which calculations should start, press $=$ and then from the formula list choose the IF function, or just type it.
  - Try to figure out what you have to do. In the Logical_test box you should state something which expresses $L > 0.2$.
  - The other fields tell Excel what to do when this statement is TRUE (then use parameter value in C5) or when it is FALSE (then use paramter value in B5).
  - **Note:** the word *value* might be misleading; you can also input new statements.
- Make a graph in which the old and the new conditional models are represented by lines.
  - Try changing the value of r in C5 into: $1, 2, 2.8, 2.9, 3$.

Auto-conditional stages

Another conditional change we might want to explore is that when a certain growth level is reached the carrying capacity K increases, reflecting that new resources have become available to support further growth.

- Now we want K to change, so type 1 in B7, 2 in C7.
- Build your model in column C. Follow the same steps as above, but now make sure that when $L > 0.99$, K changes to the value in C7. Keep $r = 0.2$ (B5).
- If all goes well you should see two stages when you create a graph of the timeseries in column C. Change K in C7 to other values.
  - Try to also change the growth rate r after reaching $L > 0.99$ by referring to C5. Start with a value of 0.3 in C5. Set K in C7 to 2 again.
  - Also try $1, 2.1, 2.5, 2.6, 3$.

Connected growers

You can now easly model coupled growth processes, in which the values in one series serve as the trigger for for parameter changes in the other process. Try to recreate the Figure of the connected growers printed in the chapter by Van Geert.

3.2.0.1 Demonstrations of dynamic modeling using spreadsheets

See the website by Paul Van Geert, scroll down to see models of:

- Learning and Teaching
- Behaviour Modification
- Connected Growers
- Interaction during Play

3.3 Iterating 2D Maps and Flows

In this assignment we will look at a 2D coupled dynamical system: **the Predator-Prey model** (aka Lotka-Volterra equations). If your are experienced in `R` or `Matlab` you can try to code the models following the instructions at the end of this assignment.

3.4 Predator-prey model

The dynamical system is given by the following set of first-order differential equations, one represents changes in a population of predators, (e.g., **F**oxes: $f_F(R_t, F_t)$), the other represents changes in a population of prey, (e.g., **R**abbits: $f_R(R_t, F_t)$).

$$\frac{dR}{dt} = (a - b * F) * R \tag{3.2}$$

(3.3)

$$\frac{dF}{dt} = (c * R - d) * F \tag{3.4}$$

This is not a *difference* equation but a *differential* equation, which means building this system is not as straightforward as was the case in the previous assignments. Simulation requires a numerical method to 'solve' this differential equation for time, which means we need a method to approach, or estimate continuous time in discrete time. Below you will receive a speed course in one of the simplest numerical procedures for integrating differential equations, the Euler method.

3.4.1 Euler Method

A general differential equation is given by:

$$\frac{dx}{dt} = f(x) \tag{3.5}$$

Read it as saying: "*a change in x over a change in time is a function of x itself*". This can be approximated by considering the change to be over some constant, small time step Δ:

$$\frac{(x_{t+1} = x_t)}{\Delta} = f(x_t) \tag{3.6}$$

After rearranging the terms a more familiar form reveals itself:

$$x_{t+1} = x_t \qquad = f(x_t) * \Delta \tag{3.7}$$

$$x_{t+1} = f(x_t) * \Delta + x_t \tag{3.8}$$

This looks like an ordinary iterative process, Δ the *time constant* determines the size of time step taken at every successive iteration. For a 2D system with variables **R** and **F** on would write:

$$R_{t+1} = f_R(R_t, Ft) * \Delta + R_t \tag{3.9}$$

$$F_{t+1} = f_F(R_t, F_t) * \Delta + F_t \tag{3.10}$$

3.4.2 Coupled System in a spreadsheet

Implement the model in a spreadsheet by substituting $f_R(R_t, Ft)$ and $f_F(R_t, F_t)$ by the differential equations for Foxes and Rabbits given above.

- Start with $a = d = 1$ and $b = c = 2$ and the initial conditions $R_0 = 0.1$ and $F_0 = 0.1$. Use a time constant of 0.01 and make at least 1000 iterations.
- Visualize the dynamics of the system by plotting:

  - F against R (i.e., the state space)
  - R and F against time (i.e., the timeseries) in one plot.
- Starting from the initial predator and prey population represented by the point $(R, F) = (0.1, 0.1)$, how do the populations evolve over time?
- Try to get a grip on the role of the time constant by increasing and decreasing it slightly (e.g. $\Delta = 0.015$) for fixed parameter values. (You might have to add some more iterations to complete the plot). What happens to the plot?
  - Hint: Remember that Δ is not a fundamental part of the dynamics, but that it is only introduced by the numerical integration (i.e., the approximation) of the differential equation. It should not change the dynamics of the system, but it has an effect nonetheless.

3.5 The Competetive Lottka-Volterra Equations

The coupled predator-prey dynamics in the previous assignment are not a very realistic model of an actual ecological system. Both equations are exponential growth functions, but **R**abbits for example, also have to eat! One way to increase realism is to consider coupled logistic growth by introducing a carrying capacity.

- Follow the link to the Wiki page and try to model the system!

> This is what *interaction dynamics* refers to, modeling mutual dependiencies using the `if ... then` conditional rules isn't really about interaction, or coupling between processes.

3.6 Predator-Prey (and other) dynamics in Agent Based Models

Agent-Based models are an expansion of the idea of “connected growers” that includes a spatial location of the things that is subject to change over time.

Have a look at some of the NETlogo demo’s:

- Rabbits Weeds Grass
- Wolf Sheep Grass

Chapter 4

Basic Timeseries Analysis

Most of the basic timeseries analyses can be performed in SPSS, because many of you will be familiar with the software we present the first assignments mainly as SPSS instructions, but you can go ahead an try your preferred environment for (statistical) computing (see comments about using R and Matlab)

4.1 Time series analysis in SPSS (17 and higher).

4.1.1 Nonlinear Growth curves in SPSS

- Open the file Growthregression.sav, it contains two variables: `Time` and `Y(t)`.

This is data from an iteration of the logistic growth differential equation you are familiar with by now, but let's pretend it's data from one subject measured on 100 occasions.

1. Plot Y(t) against Time Recognize the shape?
2. To get the growth parameter we'll try to fit the solution of the logistic flow with SPSS nonlinear regression
   - Select nonlinear… from the `Analysis` >> `Regression` menu.
   - Here we can build the solution equation. We need three parameters: a. **Yzero**, the initial condition. b. *K*, the carrying capacity. c. *r*, the growth rate.
   - Fill these in where it says `parameters` give all parameters a starting value of 0.01
3. Take a good look at the analytic solution of the (stilized) logistic flow:

$$Y(t) = \frac{K * Y_0}{Y_0 + (K - Y_0) * e^{(-K*r*t)}}$$

Tr to build this equation, the function fo e is called EXP in SPSS (`Function Group` >> `Arithmetic`) Group terms by using parentheses as shown in the equation.

5. If you think you have built the model correctly, click on `Save` choose `predicted values`. Then paste your syntax and run it!
   - Check the estimated parameter values.
   - Check R^2!!!
6. Plot a line graph of both the original data and the predicted values. (Smile)
7. A polynomial fishing expedition:
   - Create time-varying covariates of $Y(t)$:

     ```
     COMPUTE T1=Yt * Time.
     COMPUTE T2=Yt * (Time ** 2).
     COMPUTE T3=Yt * (Time ** 3).
     COMPUTE T4=Yt * (Time ** 4).
     EXECUTE.
     ```

   - Use these variables as predictors of $Y(t)$ in a regular linear regression analysis. This is called a *polynomial regression*: Fitting combinations of curves of different shapes on the data.
   - Before you run the analysis: Click `Save` Choose `Predicted Values: Unstandardized`
8. Look at R^2. This is also almost 1. Which model is better? Think about this: Based o the results o the linear regression what can yo tell about the *growth rate*, the *carrying capacity* or the *initial condition*?
9. Create a line graph of $Y(t)$, plot the predicted values of the nonlinear regression and the unstandardized predicted values of the linear polynomial regression against `time` in one figure.
10. Now you can see that the shape is approximated by the polynomials, but it is not quite the same. Is this really a model of a growth process as we could encounter it in nature?

4.1.2 Correlation functions and AR-MA models

1. Download the file `series.sav` from blackboard. It contains three time series `TS_1`, `TS_2` and `TS_3`. As a first step look at the mean and the standard deviation (`Analyze` >> `Descriptives`). Suppose these were time series from

three subjects in an experiment, what would you conclude based on the means and SD's?

2. Let's visualize these data. Go to `Forecasting` >> `Time Series` >> `Sequence Charts`. Check the box One chart per variable and move all the variables to Variables. Are they really the same?

3. Let's look at the ACF and PCF

   - Go to `Analyze` >> `Forecasting` >> `Autocorrelations`.
   - Enter all the variables and make sure both *Autocorrelations* (ACF) and *Partial autocorrelations* (PACF) boxes are checked. Click `Options`, and change the `Maximum Number of Lags` to 30.
   - Use the table to characterize the time series:

SHAPE	INDICATED MODEL
Exponential, decaying to zero	Autoregressive model. Use the partial autocorrelation plot to identify the order of the autoregressive model
Alternating positive and negative, decaying to zero	Autoregressive model. Use the partial autocorrelation plot to help identify the order.
One or more spikes, rest are essentially zero	Moving average model, order identified by where plot becomes zero.
Decay, starting after a few lags	Mixed autoregressive and moving average model.
All zero or close to zero	Data is essentially random.
High values at fixed intervals	Include seasonal autoregressive term.
No decay to zero	Series is not stationary.

4. You should have identified just one time series with autocorrelations: `TS_2`. Try to fit an `ARIMA(p,0,q)` model on this time series.

   - Go to `Analyze` >> `Forecasting` >> `Create Model`, and at `Method` (Expert modeler) choose `ARIMA`.
   - Look back at the PACF to identify which order (p) you need (last lag value at which the correlation is still significant). This lag value should go in the Autocorrelation p box.
   - Start with a Moving Average q of one. The time series variable `TS_2` is the `Dependent`.
   - You can check the statistical significance of the parameters in the output under `Statistics`, by checking the box `Parameter Estimates`.
   - This value for p is probably too high, because not all AR parameters are significant.
   - Run ARIMA again and decrease the number of AR parameters by leaving out the non-significant ones.

5. By default SPSS saves the predicted values and 95% confidence limits (check the data file). We can now check how well the prediction is: Go to `Graphs` >> `Legacy Dialogs` >> `Line`. Select `Multiple` and `Summaries of Separate Variables`. Now enter `TS_2`, `Fit_X`, `LCL_X` and `UCL_X` in `Lines Represent`. X should be the number of the last (best) model you fitted, probably 2. Enter `TIME` as the `Category Axis`.

6. In the simulation part of this course we have learned a very simple way to explore the dynamics of a system: The return plot. The time series is plotted against itself shifted by 1 step in time.

   - Create return plots (use a Scatterplot) for the three time series. Tip: You can easily create a `t+1` version of the time series by using the LAG function in a `COMPUTE` statement. For instance:

     ```
     COMPUTE TS_1_lag1 = LAG(TS_1)
     ```

   - Are your conclusions about the time series the same as in 3. after interpreting these return plots?

4.2 Notes on TSA in R and Matlab

If you use R the command below will install all the packages we will use during the entire course on you private computer. This might take too long on a university PC, just install the packages you need for an assignment each session.

```
install.packages(c("devtools", "rio", "plyr", "dplyr", "tidyr", "Matrix",
                   "ggplot2", "lattice", "latticeExtra", "grid", "gridExtra", "rgl",
                   "fractal",  "nonlinearTseries",  "crqa",
                   "signal", "sapa", "ifultools", "pracma",
                   "nlme", "lme4", "lmerTest",
                   "igrpah","qgrap","graphicalVAR","bootGraph","IsingSampler","IsingFit"),
                 dependencies = TRUE)
```

There is also a function library you need to source, the most recent version is on Github, use the devtools library to source the latest online version, or just follow the link, save as an .R file from your browser and open it in R and source it.

```
library(devtools)
source_url("https://raw.githubusercontent.com/FredHasselman/DCS/master/functionLib/nlRtsa_SOURCE.R")
```

4.2.1 Importing data in R

If you have package rio installed in R, you can load the data directly into the local environment.

```
series <- import("https://github.com/FredHasselman/DCS/raw/master/assignmentData/BasicTSA_arma/series.s
```

You can use the function arima(), acf() and pacf() in R (Matlab has functions that go by slightly different names, check the Matlab Help pages).

There are many extensions to these linear models, check the CRAN Task View on Time Series Analysis to learn more (e.g. about package zoo and forecast).

4.3 Heartbeat dynamics

Download three different time series of heartbeat intervals (HBI) here. If you use R and have package `rio` installed you can run this code and the load the data into a `data.frame` directly from `Github`.

```
library(rio)
TS1 <- rio::import("https://github.com/FredHasselman/DCS/raw/master/assignmentData/RelativeRoughness/TS
TS2 <- rio::import("https://github.com/FredHasselman/DCS/raw/master/assignmentData/RelativeRoughness/TS
TS3 <- rio::import("https://github.com/FredHasselman/DCS/raw/master/assignmentData/RelativeRoughness/TS
```

The Excel files did not have any column names, so let's create them in the `data.frame`

4.3.1 The recordings

These HBI's were constructed from the R-R intervals in electrocardiogram (ECG) recordings, as defined in Figure **4.1**.

- One HBI series is a sample from a male adult, 62 years old (called *Jimmy*). Approximately two years before the recording, the subject has had a coronary artery bypass, as advised by his physician following a diagnosis of congestive heart failure. *Jimmy* used antiarrhythmic medicines at the time of measurement.
- Another HBI series is a sample from a healthy male adult, 21 years old (called *Tommy*). This subject never reported any cardiac complaint. Tommy was playing the piano during the recording.
- A third supposed HBI series is fictitious, and was never recorded from a human subject (let's call this counterfeit *Dummy*). Your challenge

The assignment is to scrutinise the data and find out which time series belongs to *Jimmy*, *Tommy*, and *Dummy* respectively. [1]

4.3.2 First inspection

The chances that you are an experienced cardiologist are slim. We therefore suggest you precede your detective work as follows:

- Construct a graphical representation of the time series, and inspect their dynamics visually (use the code examples provided in the solutions to previous sessions to plot your time series).
- Write down your first guesses about which time series belongs to which subject. Take your time for this visual inspection (i.e., which one looks more like a line than a plane, which one looks more 'smooth' than 'rough').
- Next, explore some measures of central tendency and dispersion, etc.
- Third, compute the Relative Roughness for each time series, use Equation (4.1)

$$RR = 2\left[1\frac{\gamma_1(x_i)}{Var(x_i)}\right] \tag{4.1}$$

The numerator in the formula stands for the `lag 1` autocovariance of the HBI time series x_i. The denominator stands for the (global) variance of x_i. Most statistics packages can calculate these variances, `R` and `Matlab` have built in functions. Alternatively, you can create the formula yourself.

- Compare your (intuitive) visual inspection with these preliminary dynamic quantifications, and find out where each of the HIB series are positions on the 'colorful spectrum of noises' (i.e., line them up with Figure **4.2**).

[1] The HBI intervals were truncated (not rounded) to a multiple of 10 ms (e.g., an interval of 0.457s is represented as 0.450s), and to 750 data points each. The means and standard deviations among the HBI series are approximately equidistant, which might complicate your challenge.

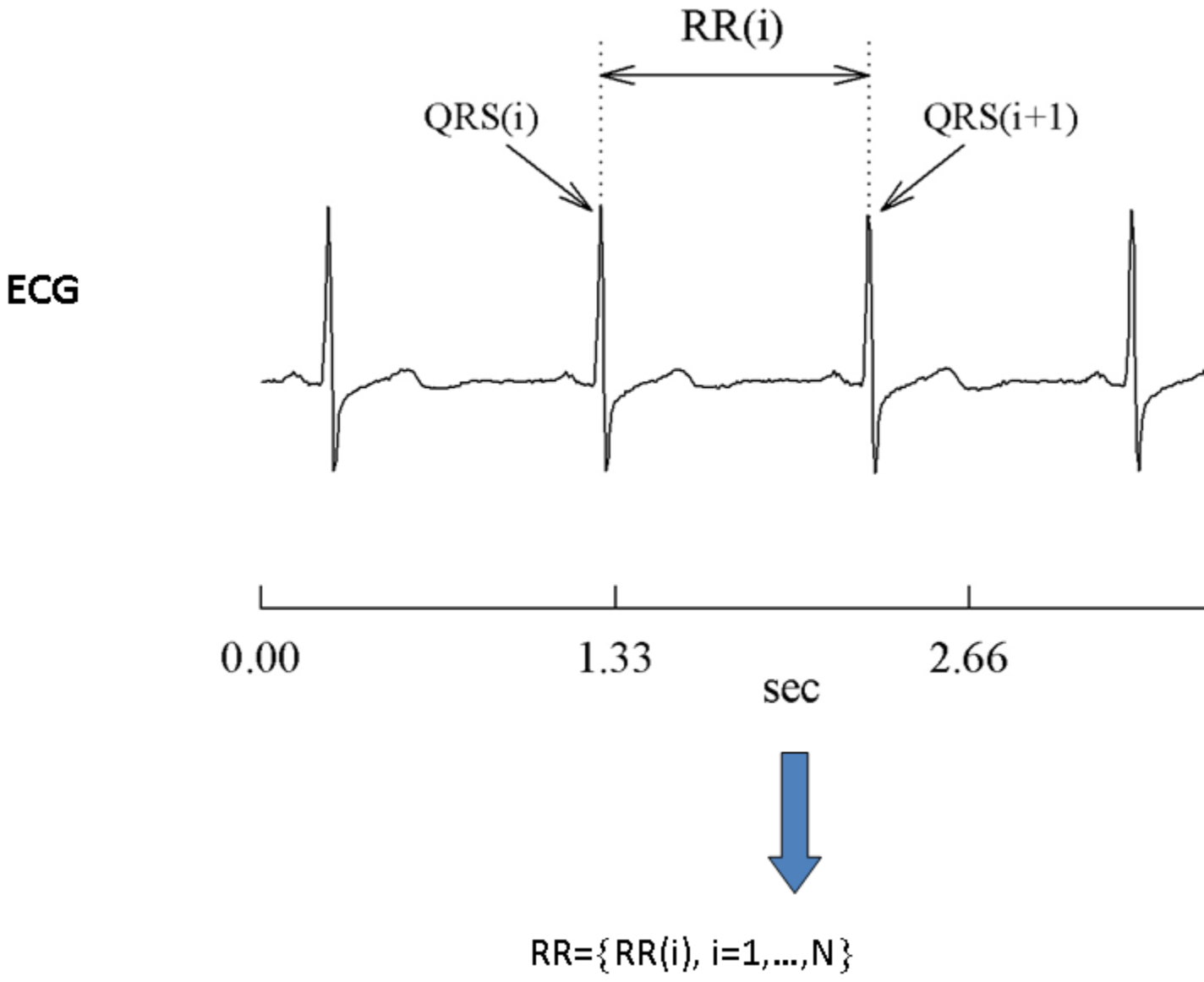


Figure 4.1 – Definition of Heart Beat Periods.

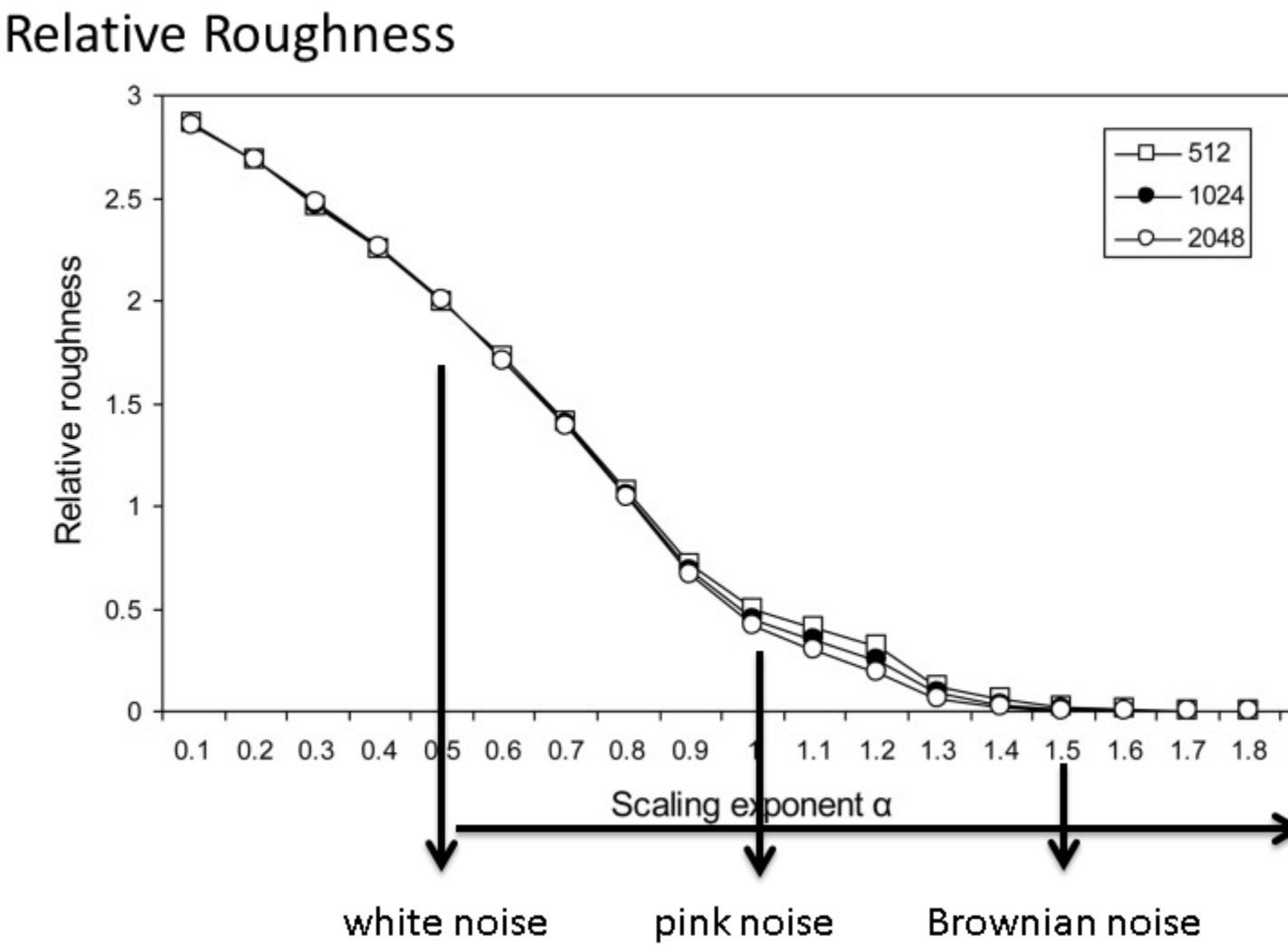


Figure 4.2 – Coloured Noise versus Relative Roughness

4.3.3 What do we know now, that we didn't knew before?

Any updates on Jimmy's, Tommy's and Dummy's health? You may start to wonder about the 'meaning' of these dynamics, and not find immediate answers.

Don't worry; we'll cover the interpretation over the next two weeks in further depth. Let's focus the dynamics just a little further for now. It might give you some clues.

- Use the `randperm` function (in `Matlab` or in package `pracma` in `R`) to randomize the temporal ordering of the HBI series.
- Visualize the resulting time series to check whether they were randomized successfully
- Next estimate the Relative Roughness of the randomized series. Did the estimates change compared to your previous outcomes (if so, why)?
- Now suppose you would repeat what you did the previous, but instead of using shuffle you would integrate the fictitious HBI series (i.e., normalize, then use `x=cumsum(x)`). You can look up `cumsum` in `R` or `Matlab`'s Help documentation). Would you get an estimate of Relative Roughness that is approximately comparable with what you got in another HBI series? If so, why?

4.4 EXTRA: Creating fractals from random processes

Below are examples of so-called Iterated Function Systems, copy the code and run it in R (Matlab scripts are here)

Try to understand what is going on in the two examples below. - How does the structure come about? We are drawing random numbers! - What's the difference between the Siepinsky Gasket and the Fern?

4.4.1 A Triangle

```
# Sierpinski Gasket using Iterated Function Systems
#
# RM-course Advanced Data Analysis
# Module Dynamical and Nonlinear Data analysis and Modeling
#
# May 2008
# Fred Hasselman & Ralf Cox

require(dplyr)

x = 0                   # Starting points
y = 0

# Emppty plot
plot(x,y, xlim=c(0,2), ylim=c(0,1))

for(i in 1:20000){      # This takes some time: 20.000 iterations

    coor=runif(1)       # coor becomes a random number between 0 and 1 drawn from the uniform distribut

    # Equal chances (33%) to perform one of these 3 transformations of x and y
    if(coor<=0.33){
        x=0.5*x
        y=0.5*y
        points(x,y,pch=".", col="green") #plot these points in green
    }

    if(between(coor,0.33,0.66)){
        x=0.5*x+0.5
        y=0.5*y+0.5
        points(x,y, pch=".", col="blue") # plot these points in blue
    }

    if(coor>0.66){
        x=0.5*x+1
        y=0.5*y
        points(x,y, pch=".", col="red") #plot these points in red
    }
} # for ...
```

4.4.2 A Fern

```
# Barnsley's Fern using Iterated Function Systems
#
# RM-course Advanced Data Analysis
# Module Dynamical and Nonlinear Data analysis and Modeling
#
# May 2008
# Fred Hasselman & Ralf Cox

require(dplyr)

x = 0                   # Starting points
y = 0

# Emppty plot
plot(x,y, pch=".", xlim=c(-3,3), ylim=c(0,12))

for(i in 1:50000){      # This takes some time: 20.000 iterations

    coor=runif(1)       # coor becomes a random number between 0 and 1 drawn from the uniform distribut

    if(coor<=0.01){                  #This transformation 1% of the time
        x=0
        y=0.16*y
        points(x,y, pch=".", col="darkgreen")
    }

    if(between(coor,0.01,0.08)){   #This transformation 7% of the time
        x=0.2*x-0.26*y
        y=0.23*x+0.22*y+1.6
        points(x,y, pch=".", col="darkolivegreen")
    }

    if(between(coor,0.08,0.15)){   #This transformation 7% of the time
        x=-0.15*x+0.28*y
        y=0.26*x+0.24*y+0.44
       points(x,y, pch=".", col="palegreen")
    }

    if(coor>0.15){      #This transformation 85% of the time
        x=-0.85*x+0.04*y
        y=-0.04*x+0.85*y+1.6
        points(x,y, pch=".", col="springgreen")
    }

} # for ...
```

4.4.3 The fractal / chaos game

These Iterated Function Systems also go by the name of ‘the fractal game’ and are used in computer science, the gaming industry, graphic design, etc.

This Wikipedia page on Barnsley's fern has some good info on the topic. At the end they display *Mutant varieties*. Try to implement them!

You can by now probably guess that the these simple rules can be described as constraints on the degrees of freedom of the system. Like with the models of growth we simulated, the rules of the fractal game can be made dependent on other processes or events. A great example are the so-called fractal flames implemented in a screen-saver called *electric sheep*, which combines genetic algorithms, distributed computind and user input ("likes") to create intruiging visual patterns on your computer screen.[2]

[2] Use at your own risk! You will find yourself silently staring at the screen for longer periods of time.

Chapter 5

Fluctuation and Disperion analyses I

Chapter 6

Fluctuation and Disperion analyses II

Chapter 7

Phase Space Reconstruction and RQA

Chapter 8

Categorical and Cross-RQA (CRQA)

Chapter 9

Potential and Catasrophe Models

Chapter 10

Complex Networks

Part II

Lecture Notes

Lecture 1

Modeling change processes in 1D

The simplest non-trivial *iterative change process* can be described by the following *difference equation*:

$$Y_{t+1} = Y_{t=0} + a * Y_t \tag{10.1}$$

Equation (10.1) describes the way in which the value of Y changes between two adjacent, discrete moments in time (hence the term difference equation, or recurrence relation). There are two parameters resembling an intercept and a slope:

1. The starting value Y_0 at $t = 0$, also called the *starting value*, or the *initial conditions*.
2. A rule for incrementing time, here the change in Y takes place over a discrete time step of 1: $t + 1$.

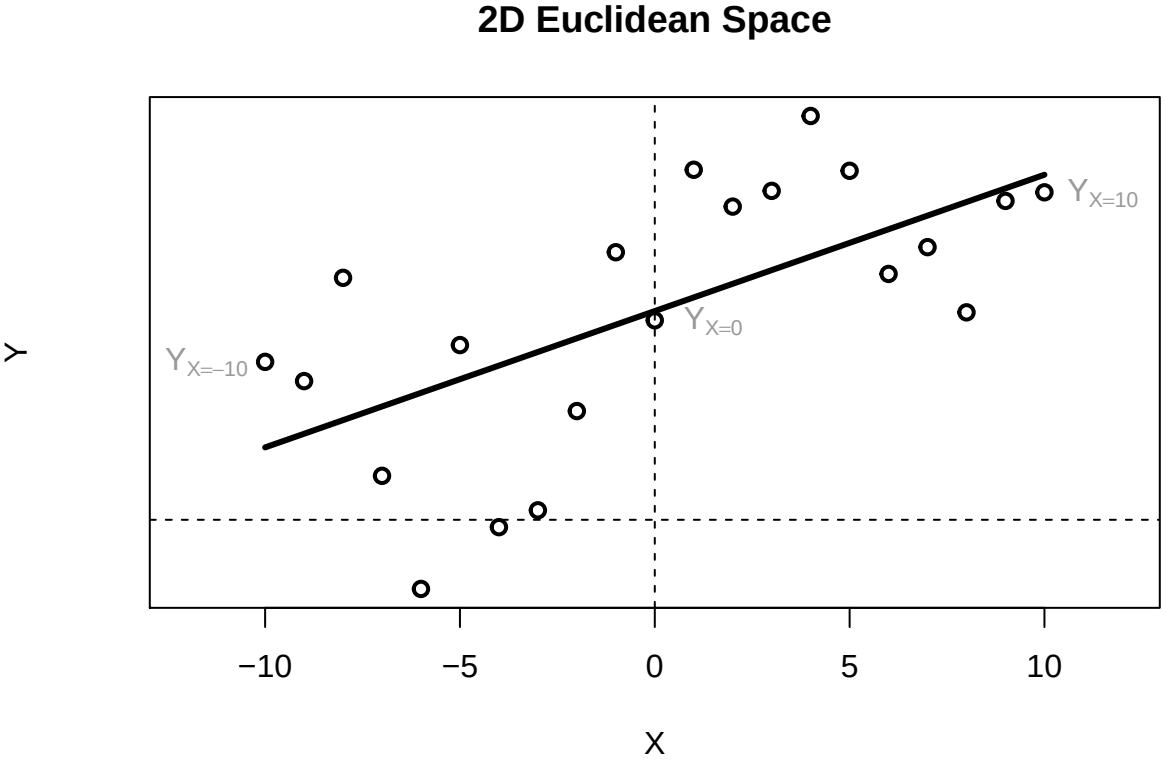
The values taken on by variable Y are considered to represent the states quantifiable observable leAlternative ways to describe the change of states :

- A dynamical rule describing the propagation of the states of a system observable measured by the values of variable Y through discrete time.
- A dynamic law describing the time-evolution of the states of a system observable measured by the variable Y.

These descriptions all refer to the change processes that govern system observables (properties of dynamical systems that can be observed through measurement).

It's a line! It's a plane!

The formula resembles the equation of a line. There is a constant value Y_0 which is added to a proportion of the value of Y at time t, given by parameter a. This is equivalent to the slope of a line. However, in a (X, Y) plane there are two 'spatial' (metric) dimensions representing the values two variables X and Y can take on (see figure).



The best fitting straight line would be called a statistical model of the linear relationship between the observed values of X and Y. It can be obtained by fitting a General Linear Model (GLM) to the data. If X were to represent repeated measurements the

multivariate GLM for repeated measures would have to be fitted to the data. This can be very problematic, because statistical models rely on Ergodic theory:

> "… it is the study of the long term average behavior of systems evolving in time." [1]

need to assume independence of measurements within and between subjects. These assumptions can be translated to certain conditions that must hold for the model to be valid, known as *Compound Symmetry* and *Sphericity*:

> The compound symmetry assumption requires that the variances (pooled within-group) and covariances (across subjects) of the different repeated measures are homogeneous (identical). This is a sufficient condition for the univariate F test for repeated measures to be valid (i.e., for the reported F values to actually follow the F distribution). However, it is not a necessary condition. The sphericity assumption is a necessary and sufficient condition for the F test to be valid; it states that the within-subject "model" consists of independent (orthogonal) components. The nature of these assumptions, and the effects of violations are usually not well-described in ANOVA textbooks; [2]

As you can read in the quoted text above, these conditions must hold in order to be able to identify unique independent components as the sources of variation of Y over time within a subject. This is the a clear example of:

> It is the theory that decides what we may observe [3]

If you choose to use GLM repeated measures to model change over time, you will only be able to infer independent components that are responsible for the time-evolution of Y. As is hinted in the last sentence of the quote, the validity of such inferences is not a common topic of discussion statistics textbooks.

No! … It's a time series!

The important difference between a regular 2-dimensional Euclidean plane and the space in which we model change processes is that the X-axis represents the physical dimension **time**. In the case of the Linear Map we have a 1D space with one 'spatial' dimension Y and a time dimension t. This is called time series if Y is sampled as a continuous process, or a trial series if the time between subsequent observations is not relevant, just the fact that there was a temporal order (for example, a series of response latencies to trials in a psychological experiment in the order in which they were presented to the subject).

Time behaves different from a spatial dimension in that it is directional (time cannot be reversed), it cannot take on negative values, and, unless one is dealing with a truly random process, there will be a temporal correlation across one or more values of Y seperated by an amount of time. In the linear difference equation this occurs because each value one step in the future is calculated based on the current value. If the values of Y represent an observable of a dynamical system, the system can be said to have a history, or a memory. Ergodic systems do not have a history or a memory that extends across more than one time step. This is very convenient, because one can calculate the expected value of a system observable given infinite time, by making use of of the laws of probabilities of random events (or random fields). This means: The average of an observable of an Ergodic system measured across infinite time (its entire history, the **time-average**), will be the be the same value as the average of this observable measured at one instance in time, but in an infinite amount of systems of the same kind (the population, the **spatial average**) [4].

The simple linear difference equation will have a form of *perfect memory' across the smallest time scale (i.e., the increment of 1, $t+1$). This 'memory' concerns a correlation of 1 between values at adjacent time points (a short range temporal correlation, SRC), because the change from Y_t to Y_{t+1} is exactly equal to $a * Y_t$ at each iteration step. This is the meaning of deterministic, not that each value of Y is the same, but that the value of Y now can be perfectly explained form the value of Y one moment in the past.

[1] See Dajani & Dirksin (2008, p. 5, "A simple introduction to Ergodic Theory")

[2] Retreived from www.statsoft.com

[3] Einstein as quoted by Heisenberg.

[4] In other words: If you throw 1 die 100 times in a row, the average of the 100 numbers is the **time-average** of one of the observables of die-throwing systems. If this system is ergodic, then its **time-average** is expected to be similar to the average of the numbers that turn up if you throw 100 dice all at the same instance of time. The dice layed out on the table represent a spatial sample, a snapshot frozen in time, of the possible states the system can be in. Taking the average would be the **spatial average** this observable of die-throwing systems. This ergodic condiciotn is often implicitly assumed in Behavioural Science when studies claim to study change by taking different samples of individuals (snapshots of system states) and comparing if they are the same.

Summarising, the most profound difference is not the fact that the equation of linear change is a deterministic model and the GLM is a probabilistic model with parameters fitted from data, this is something we can (and will) do for a as well. The profound difference between the models is the role given to the passage of time:

- The linear difference equation represents changes in Y as a function of the physical dimension *time* and Y itself.
- The GLM represents changes in Y as a function of a linear predictor composed of additive components that can be regarded as independent sources of variation that sum up to the observed values of Y.

Lecture 2

Numerical integration

In order to 'solve' a differential equation for continuous time using a method of numerical integration, one could code it like in the spreadsheet assignment below. For R and Matlab there are so-called *solvers* available, functions that will do the integration for you. For R look at the Examples in package deSolve.

Euler's method and more...

The result of applying a method of numerical integration is called a **numerical solution** of the differential equation. The **analytical solution** is the equation which will give you a value of Y for any point in time, given an initial value Y_0. Systems which have an analytical solution can be used to test the accuracy of **numerical solutions**.

Analytical solution

Remember that the analytical solution for the logistic equation is:

$$Y(t) = \frac{K}{1 + \left(\frac{K}{Y_0 - 1}\right) * e^{-r*t}}$$

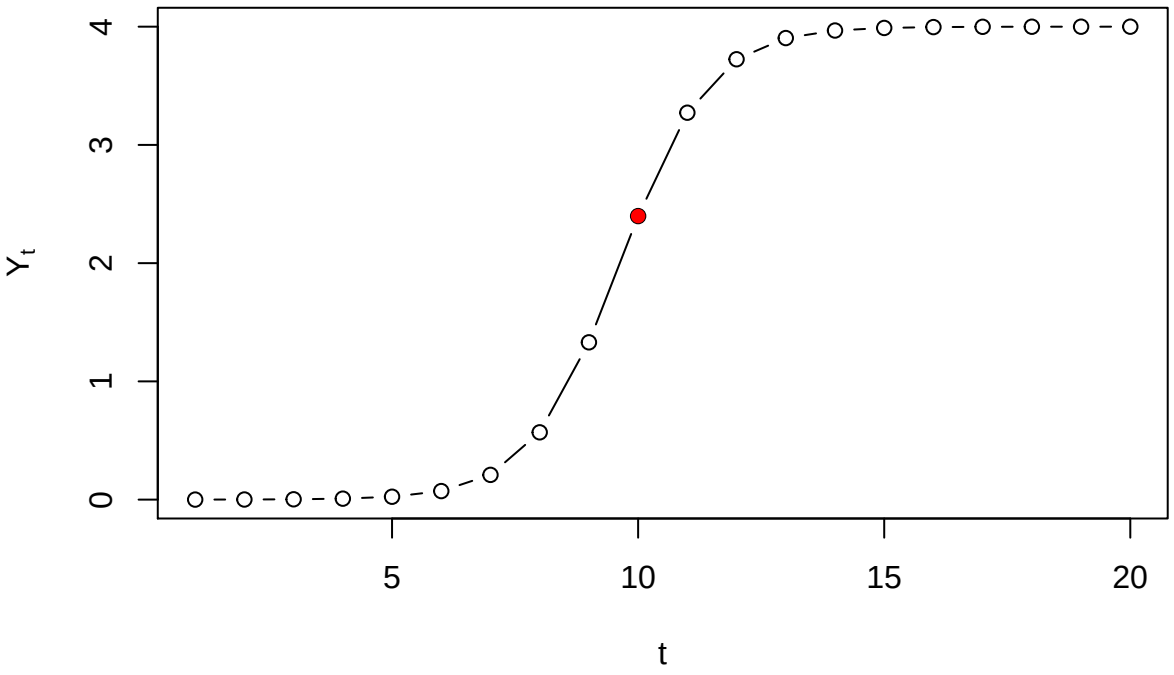
If we want to know the growth level Y_t at $t = 10$, with $Y_0 = .0001$, $r = 1.1$ and $K = 4$, we can just `fill it in`:

```
# Define a function for the solution
logSol <- function(Y0, r, K, t){K/(1+(K/Y0-1)*exp(-r*t))}

# Call the function
logSol(Y0=.0001, r=1.1, K=4, t=10)
```

```
## [1] 2.398008
```

We can pas a vector of timepoints to create the exact solution, the same we would get if we were to iterate the differential/difference equation.

```
# Plot from t=1 to t=100
plot(logSol(Y0=.0001, r=1.1, K=4, t=seq(1,20)), type = "b",
     ylab = expression(Y[t]), xlab = "t")
# Plot t=10 in red
points(10,logSol(Y0=.0001, r=1.1, K=4, t=10), col="red", pch=16)
```



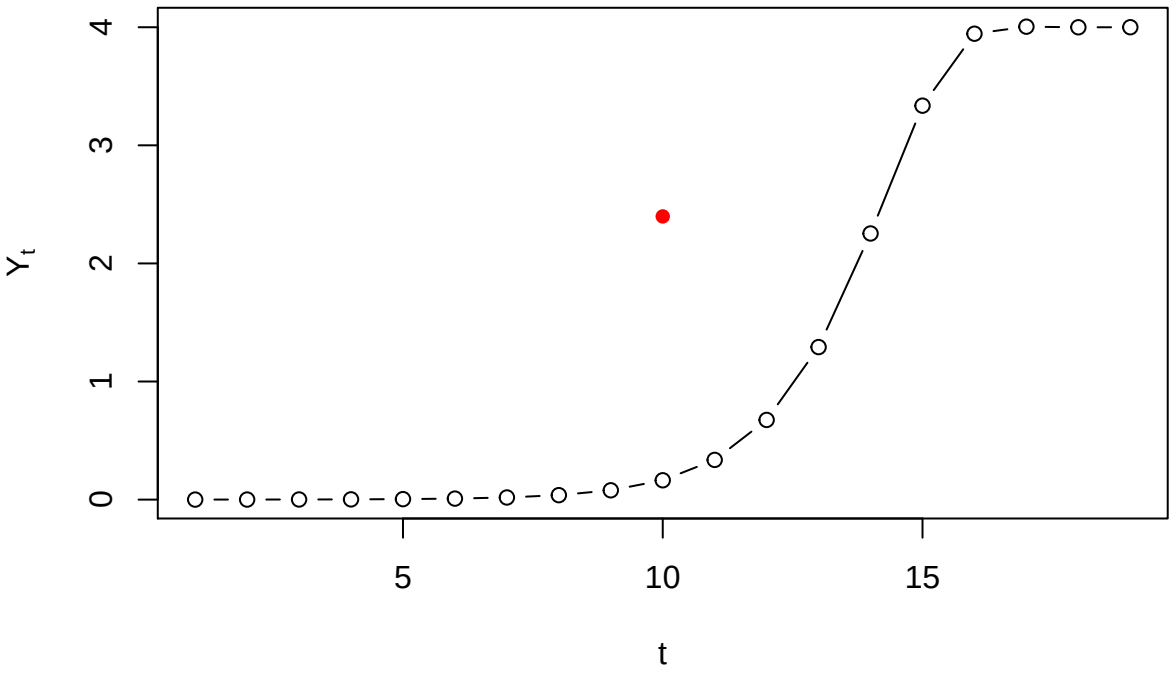
Numerical solution (discrete)

If we would iterate the differential equation …

$$\frac{dY}{dt} = Y_t * (1 + r - r * \frac{Y_t}{K})$$

… as if it were a difference equation, that is, *not* simulating continuous time.

```
logIter <-  function(Y0,r,K,t){
  N <- length(t)
  Y <- as.numeric(c(Y0, rep(NA,N-2)))
  sapply(seq_along(Y), function(t){ Y[[t+1]] <<- Y[t] * (1 + r - r * Y[t] / K)})
  }

# Plot from t=1 to t=100
plot(logIter(Y0=.0001, r=1.1, K=4, t=seq(1,20)), type = "b",
     ylab = expression(Y[t]), xlab = "t")
# Plot t=10 in red
points(10,logSol(Y0=.0001, r=1.1, K=4, t=10), col="red", pch=16)
```


Y_t
0
1
2
3
4
5
10
15
t

Lecture 3

Lecture 4

Lecture 5

Lecture 6

Lecture 7

Lecture 8

Lecture 9

Appendix A

Mathematics of change I

Solutions to assignments in section 2.

- Linear and logistic growth
- Deterministic Chaos

A.1 Linear and logistic growth

Solutions in a spreadsheet

The solutions to iterating the Linear Map and theLogistic Map in a spreadsheet can be found in this GoogleSheet.

Solutions in R

Coding the difference equations in Matlab and R is always easier than using a spreadsheet. One obvious way to do it is to use a counter variable representing the iterations of time in a `for ... next` loop. The iterations should run over a vector (which is the same concept as a row or a column in a spreadsheet: An indexed array of numbers or characters). The first entry should be the starting value, so the vector index 1 represents Y_0.

The loop can be implemented a number of ways, for example as a function which can be called from a script or the command / console window. In R working with functions is easy, and very much recommended, because it will speed up calculations considerably, and it will reduce the amount of code you need to write. You need to gain some experience with coding in R before you'll get it right. In order to get it lean and clean (and possibly even mean as well) you'll need a lot of experience with coding in R,therefore, we will (eventually) provide you the functions you'll need to complete the assignments. All you have to do is figure out how to use, or modify them to suit your specific needs.

To model the autocatalytic growth equations we provide a function `growth.ac()`, which is able to simulate all of the processes discussed in the lectures. Using just a few lines of code, each of the 4 difference equations used in the assignments can be simulated. Basically the code block below contains the solutions to the Linear Map, the stylized Logisitc Map and the Van Geert model for cognitive growth.

```
growth.ac <- function(Y0 = 0.01, r = 1, k = 1, N = 100, type = c("driving", "damping", "logistic", "van
    # Create a vector Y of length N, which has value Y0 at Y[1]
    if(N>1){
    Y <- as.numeric(c(Y0, rep(NA,N-2)))
    # Conditional on the value of type ...
    switch(type,
           # Iterate N steps of the difference function with values passed for Y0, k and r.
           driving  = sapply(seq_along(Y), function(t) Y[[t+1]] <<- r * Y[t] ),
           damping  = k + sapply(seq_along(Y), function(t) Y[[t+1]] <<- - r * Y[t]^2 / k),
           logistic = sapply(seq_along(Y), function(t) Y[[t+1]] <<- r * Y[t] * ((k - Y[t]) / k)),
           vanGeert = sapply(seq_along(Y), function(t) Y[[t+1]] <<- Y[t] * (1 + r - r * Y[t] / k))
    )}
    return(ts(Y))
}

# Call the function with default settings and r = 1.1
Y <- growth.ac(r = 1.1)
```

Some notes about this function:

- To select which growth process to simulate, the argument `type` is defined which takes the values `driving` (default), `damping`, `logistic` and `vanGeert`.
  - The statement `switch(type, ...)` will iterate an equation based on the value of `type`.
- A `time series` object is returned due to the function `ts()`. This is a convenient way to represent time series data, it can also store the sample rate of the signal and start and end times.
  - Most of the basic functions, like `plot()` and `summary()` will recognise a time series object when it is passed as an argument and use settings appropriate for time series data.
- The `sapply()` function iterates t from 1 to the number of elements in Y (`seq_along(Y)`) and then applies the function.

- The double headed arrow `<<-` is necessary because we want to update vector Y, which is defined outside the `sapply()` environment.

The `time series` object

The time series object is expected to have a time-dimension on the x-axis. This is very convenient, because R will generate the time axis for you by looking at the *t*ime *s*eries *p*roperties attribute of the object. Even though we are not working with measurement ourcomes, consider a value at a time-index in a time series object a **sample**:

- `Start` - The value of time at the first sample in the series (e.g., 0, or 1905)
- `End` - The value of time at the last sample in the series (e.g., 100, or 2005)
- `Frequency` - The amount of time that passed between two samples, or, the sample rate (e.g., 0.5, or 10)

Examples of using the time series object.

```
# Get sample rate info
tsp(Y)
```

```
## [1]   1 100   1
```

```
# Extract the time vector
time(Y)
```

```
## Time Series:
## Start = 1
## End = 100
## Frequency = 1
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

For now, these values are in principle all arbitrary units (`a.u.`). These settings only make sense if they represent the parameters of an actual measurement procedure.

It is easy to adjust the time vector, by assigning new values using `tsp()` (values have to be possible given the timeseries length). For example, suppose the sampling frequency was 0.1 instead of 1 and the Start time was 10 and End time was 1000.

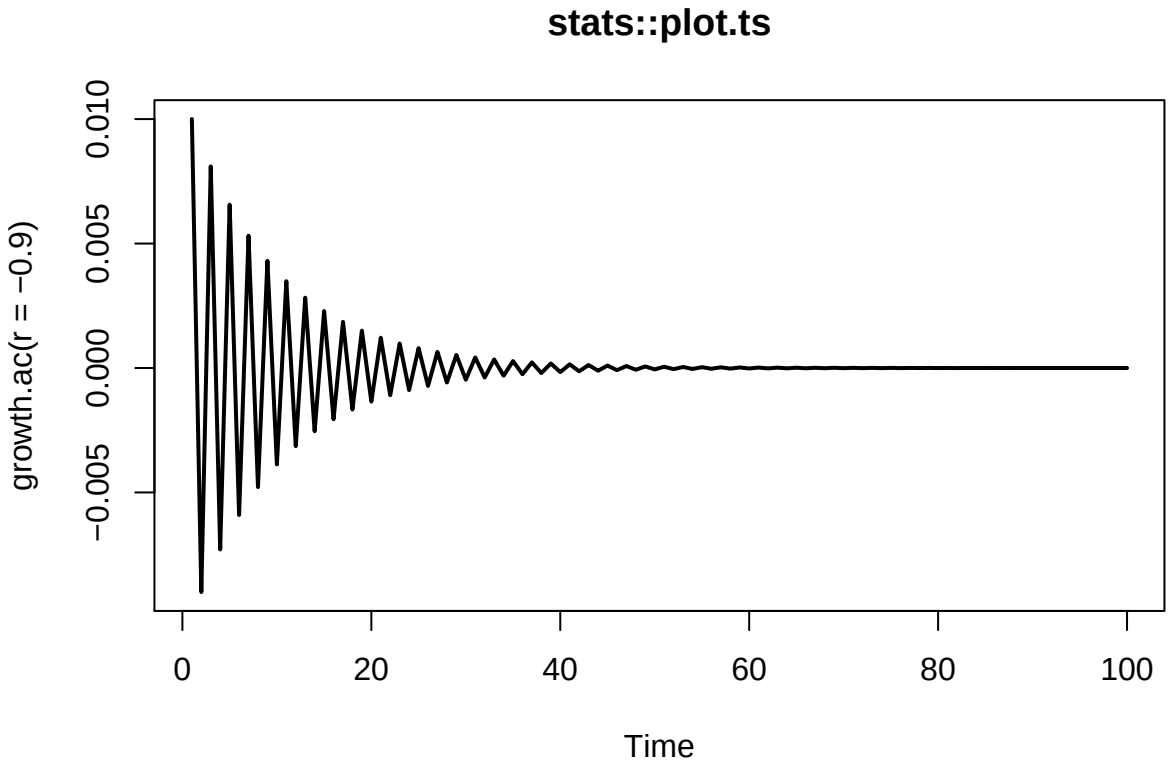
```
# Assign new values
tsp(Y) <- c(10, 1000, .1)
# Time axis is automatically adjusted
time(Y)
```

```
## Time Series:
## Start = 10
## End = 1000
## Frequency = 0.1
##   [1]   10   20   30   40   50   60   70   80   90  100  110  120  130  140
##  [15]  150  160  170  180  190  200  210  220  230  240  250  260  270  280
##  [29]  290  300  310  320  330  340  350  360  370  380  390  400  410  420
##  [43]  430  440  450  460  470  480  490  500  510  520  530  540  550  560
##  [57]  570  580  590  600  610  620  630  640  650  660  670  680  690  700
##  [71]  710  720  730  740  750  760  770  780  790  800  810  820  830  840
##  [85]  850  860  870  880  890  900  910  920  930  940  950  960  970  980
##  [99]  990 1000
```

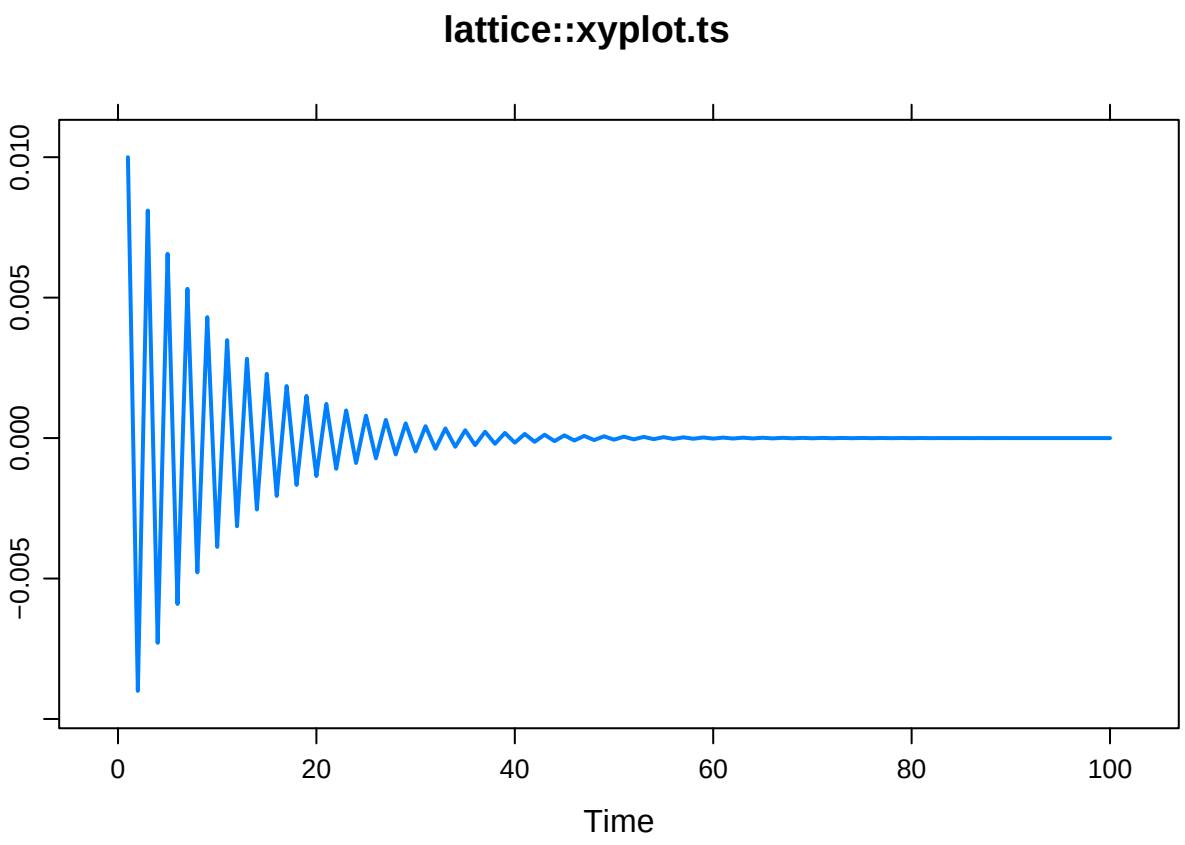
Plotting a `ts` object as a time series

Depending on which packages you use, there will be different settings applied to time series objects created by `ts()`. Below are some examples of differences between plotting routines.

```
require(lattice)       # Needed for plotting
require(latticeExtra)  # Needed for plotting

# stats::plot.ts
plot(growth.ac(r = -.9), lwd = 2, main = "stats::plot.ts")
```



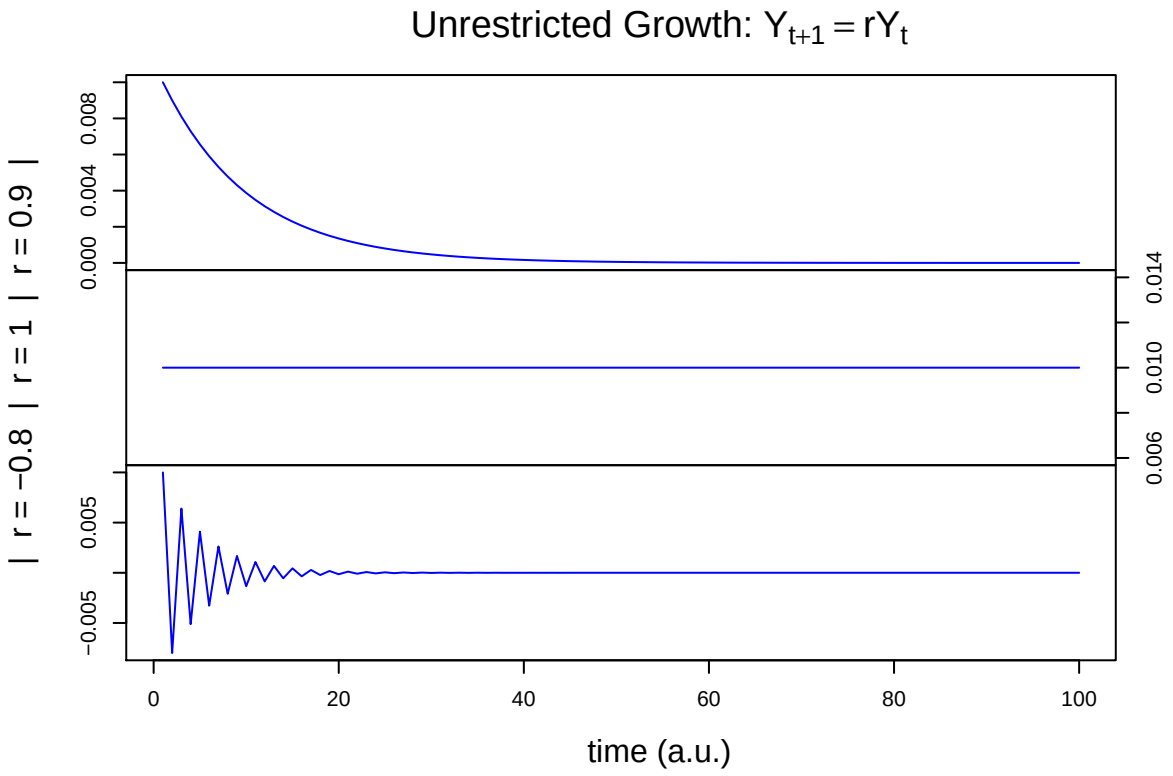

```
# lattice::xyplot.ts
xyplot(growth.ac(r = -.9), lwd = 2, main = "lattice::xyplot.ts")
```



Plotting multiple time series in one figure

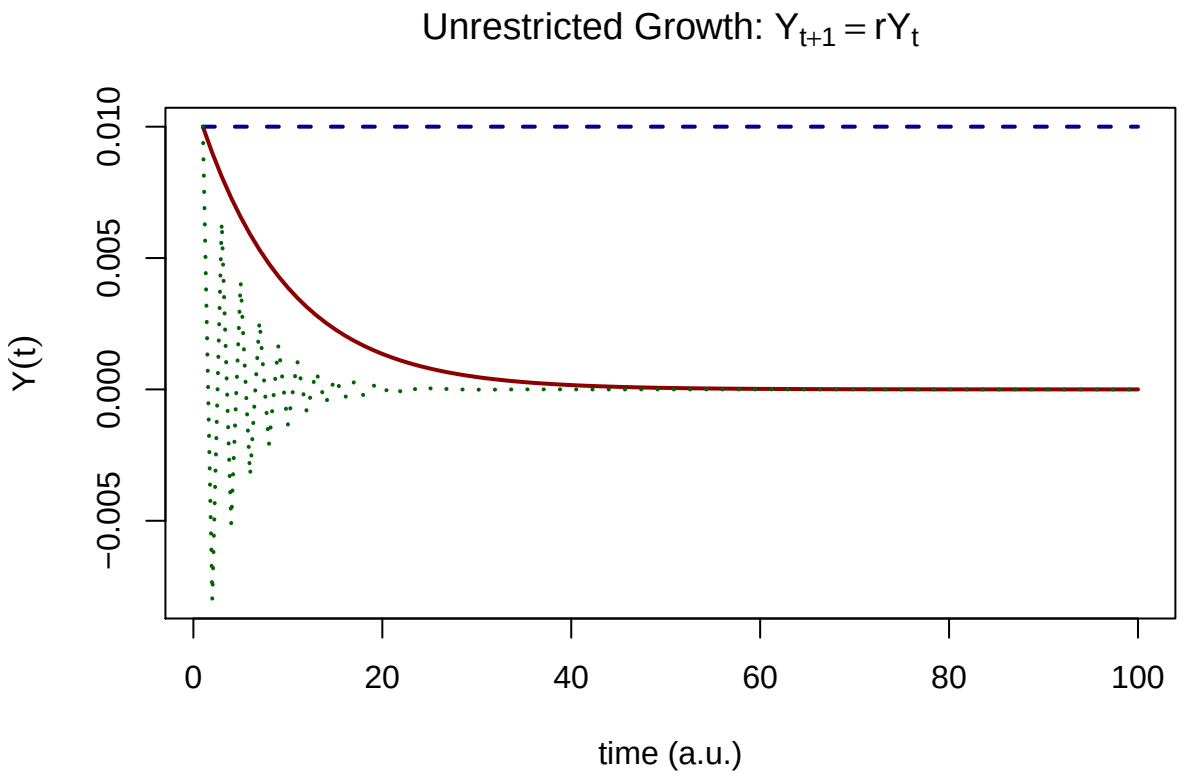
Plot multiple timeseries in frames with `plot.ts()` in `package::stats`. This function takes a matrix as input, here we use `cbind( ... )`.

```
# stats::plot.ts
plot(cbind(growth.ac(r =  0.9),
           growth.ac(r =  1.0),
           growth.ac(r = -0.8)
           ),
     yax.flip = TRUE, ann = FALSE, col = "blue", frame.plot = TRUE)
title(main = expression(paste("Unrestricted Growth: ",Y[t+1]==r*Y[t])),
      ylab = "|  r = -0.8  |  r = 1  |  r = 0.9  |",
      xlab = "time (a.u.)")
```



Plot multiple timeseries in one graph with `ts.plot()` in `package::graphics`. This function can handle multiple `ts` objects as arguments.

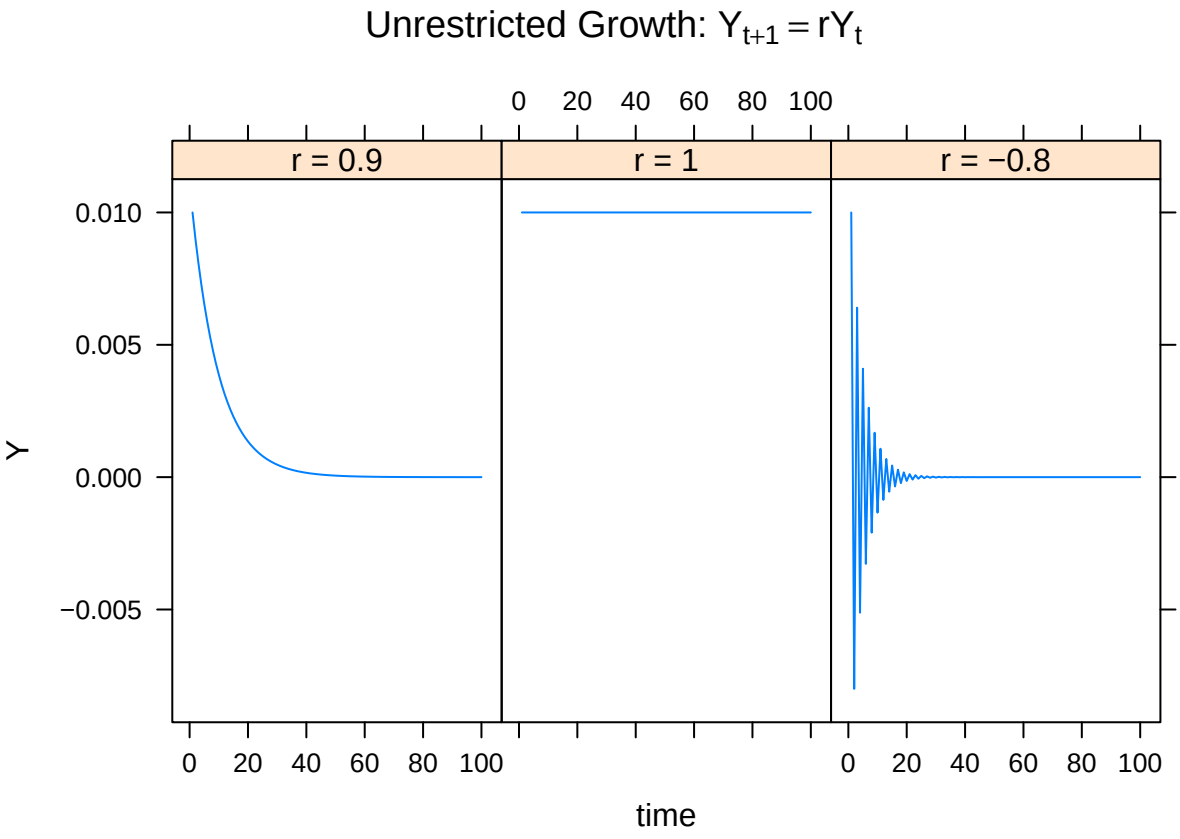
```
# graphics::ts.plot
ts.plot(growth.ac(r = 0.9),
        growth.ac(r = 1),
        growth.ac(r = -.8),
        gpars = list(xlab = "time (a.u.)",
                     ylab = expression(Y(t)),
                     main = expression(paste("Unrestricted Growth: ",Y[t+1]==r*Y[t])),
                     lwd = rep(2,3),
                     lty = c(1:3),
                     col = c("darkred","darkblue","darkgreen")
                     )
        )
legend(70, -0.015, c("r = 0.9","r = 1.0", "r = -0.8"), lwd = rep(2,3), lty = c(1:3), col = c("darkred",
```



Use `xyplot()` in `package::lattice` to create a plot with panels. The easiest way to do this is to create a dataset in so-called "long" format. This means the variable to plot is in 1 column and other variables indicate different levels, or conditions under which the variable was observed or simulated.

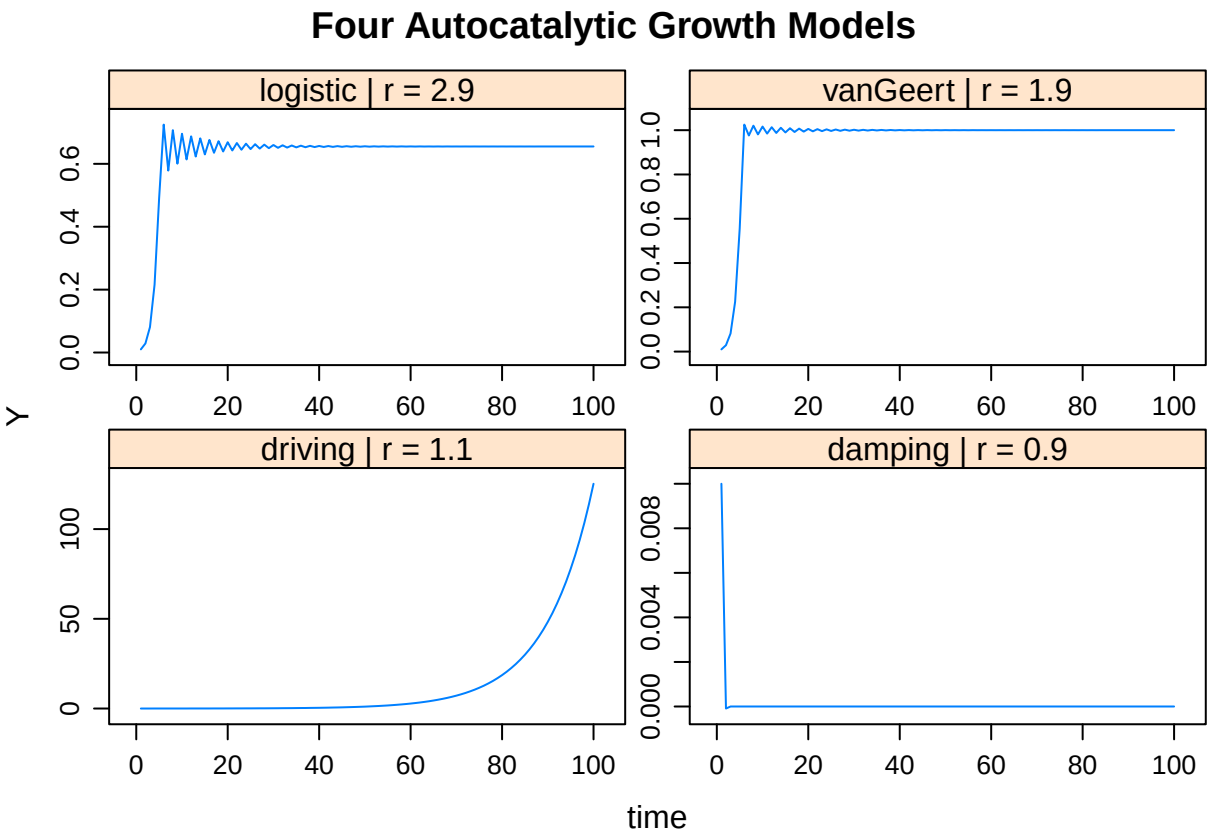
Function `ldply()` is used to generate Y for three different settings of r. The values of r are passed as a **l**ist and after a function is applied the result is returned as a **d**ataframe.

```
require(plyr)          # Needed for function ldply()

# Create a long format dataframe for various values for `r`
data <- ldply(c(0.9,1,-0.8), function(r) cbind.data.frame(Y    = as.numeric(growth.ac(r = r)),
                                                          time = as.numeric(time(growth.ac(r = r))),
                                                          r    = paste0("r = ", r)))
# Plot using the formula interface
xyplot(Y ~ time | r, data = data, type = "l", main = expression(paste("Unrestricted Growth: ",Y[t+1]==r
```



One can also have different panels represent different growth functions.

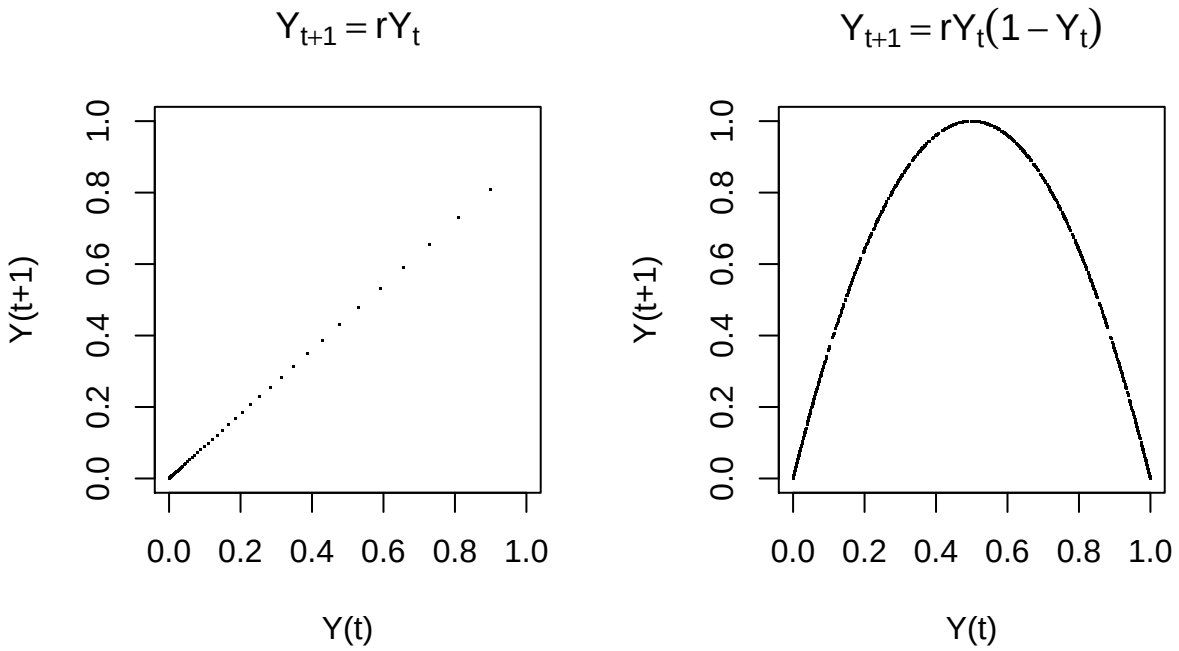
```
# Create a long format dataframe for combinations of `type` and `r`
param <- list(driving  = 1.1,
              damping  = 0.9,
              logistic = 2.9,
              vanGeert = 1.9)
# Use the `names()` function to pass the `type` string as an argument.
data <- ldply(seq_along(param), function(p){
    cbind.data.frame(Y    = as.numeric(growth.ac(r = param[[p]], type = names(param[p]))),
                     time = as.numeric(time(growth.ac(r = param[[p]], type = names(param[p])))),
                     type = paste0(names(param[p]), " | r = ", param[p]))
    })
# Plot using the formula interface
xyplot(Y ~ time | factor(type), data = data, type = "l", scales = c(relation = "free"),
       main = "Four Autocatalytic Growth Models")
```



The return plot

To create a return plot the values of Y have to be shifted by a certain lag. The functions `lead()` and `lag()` in `package::dplyr` are excellent for this purpose (note that `dplyr::lag()` behaves different from `stats::lag()`).

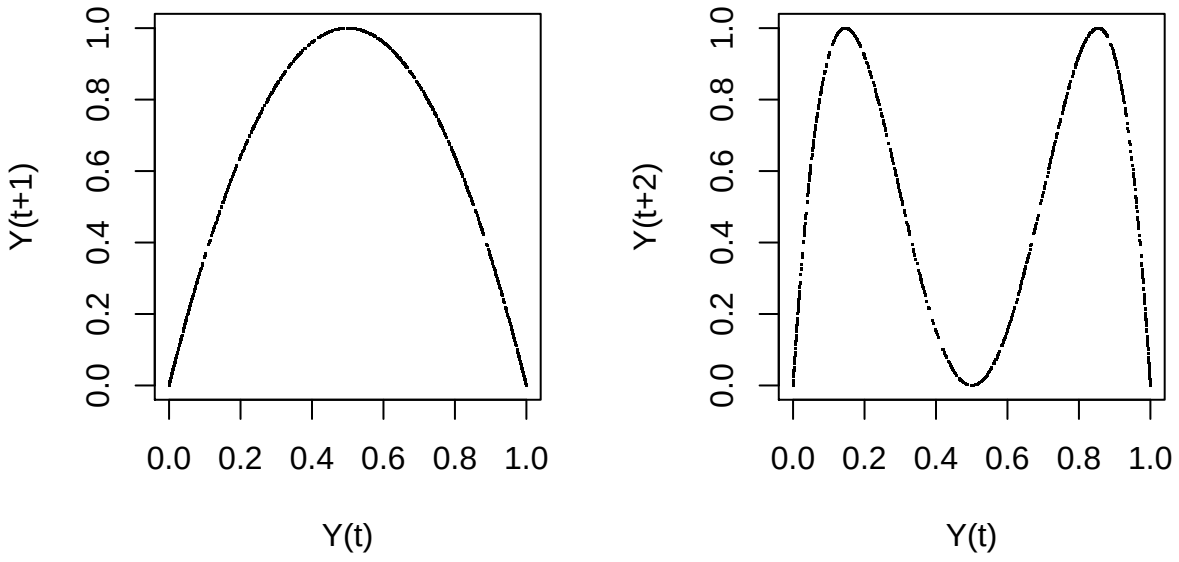
```
# Function lag() and lead()
require(dplyr)

# Get exponential growth
Y1 <- growth.ac(Y0 = .9, r = .9, N = 1000, type = "driving")
# Get logistic growth in the chaotic regime
Y2 <- growth.ac(r = 4, N = 1000, type = "logistic")
# Use the `lag` function from package `dplyr`
op <- par(mfrow = c(1,2), pty = "s")
plot(lag(Y1), Y1, xy.labels = FALSE, pch = ".", xlim = c(0,1), ylim = c(0,1), xlab = "Y(t)", ylab = "Y(
     main = expression(paste(Y[t+1]==r*Y[t])))
plot(lag(Y2), Y2, xy.labels = FALSE, pch = ".", xlim = c(0,1), ylim = c(0,1), xlab = "Y(t)", ylab = "Y(
     main = expression(paste(Y[t+1]==r*Y[t]*(1-Y[t]))))
```

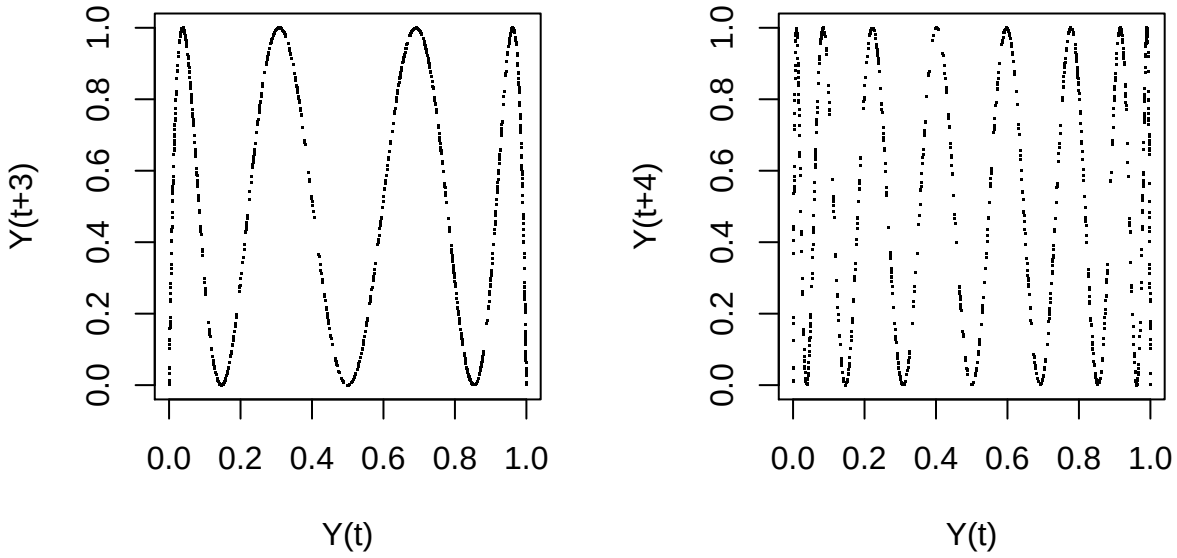



```
par(op)
```

Use `l_ply()` from `package::plyr` to create return plots with different lags. The **l_** before **ply** means the function will take a **l**ist as input to a function, but it will not expect any data to be returned, for example in the case of a function that is used to plot something.

```
# Explore different lags
op <- par(mfrow = c(1,2), pty = "s")
l_ply(1:4, function(l) plot(lag(Y2, n = l), Y2, xy.labels = FALSE, pch = ".", xlim = c(0,1), ylim = c(0
```






```
par(op)
```

A.1.1 Solutions in Matlab

For Matlab we provide an example of a simple `for ... next` loop, which should be easy to translate to R if you want to.

Linear Map

Logistic Map

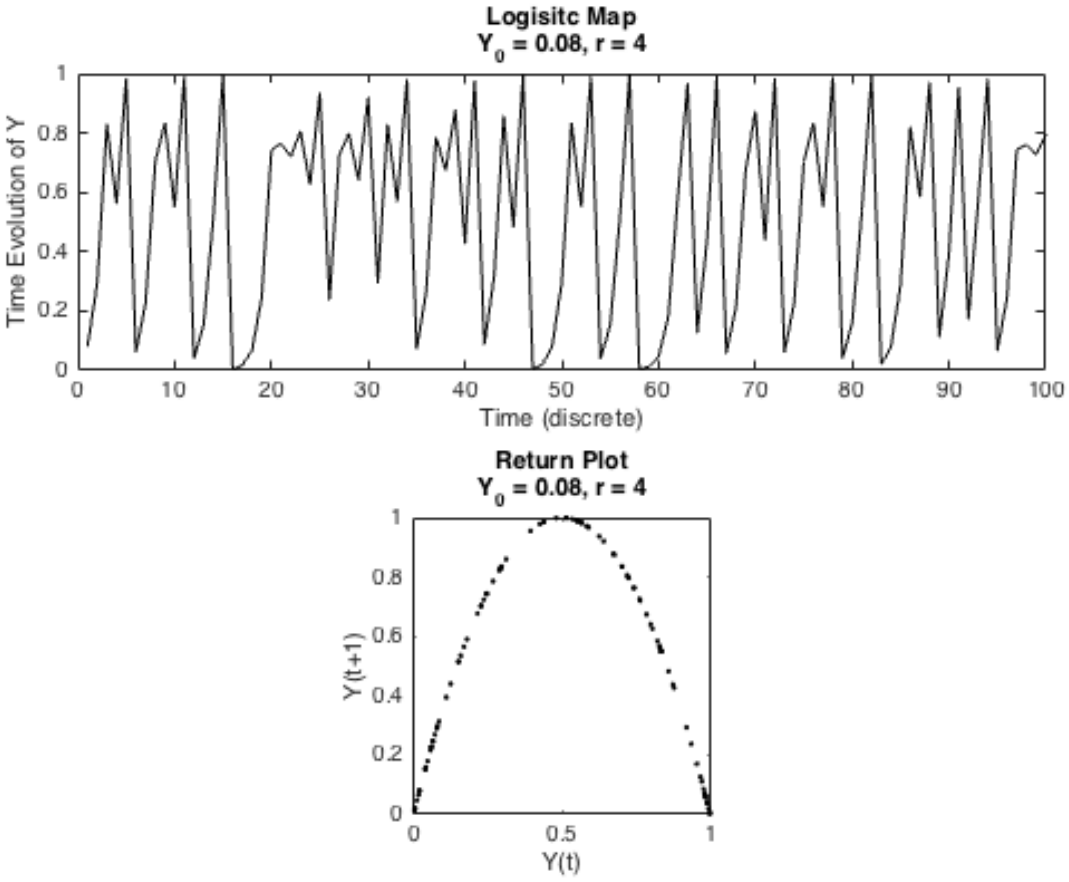


Figure 1.1 – Solution Logistic Map - Matlab

Appendix B

Mathematics of Change II

Solutions to assignments in section 3

- Time-varying parameters
- Predator-prey dynamics

B.1 Time-varying parameters

Solutions in a spreadsheet

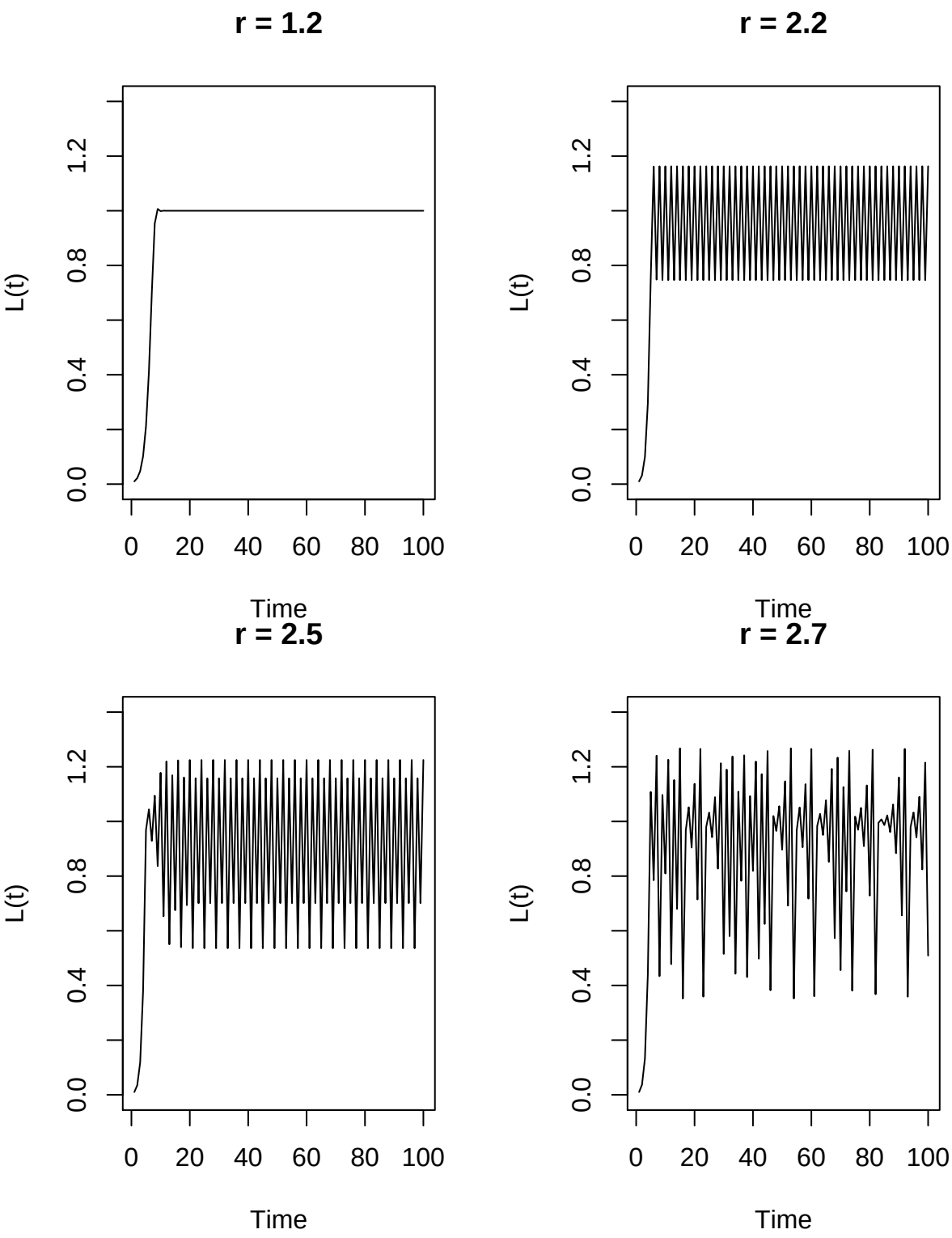
- Van Geert, including jumps and stages.

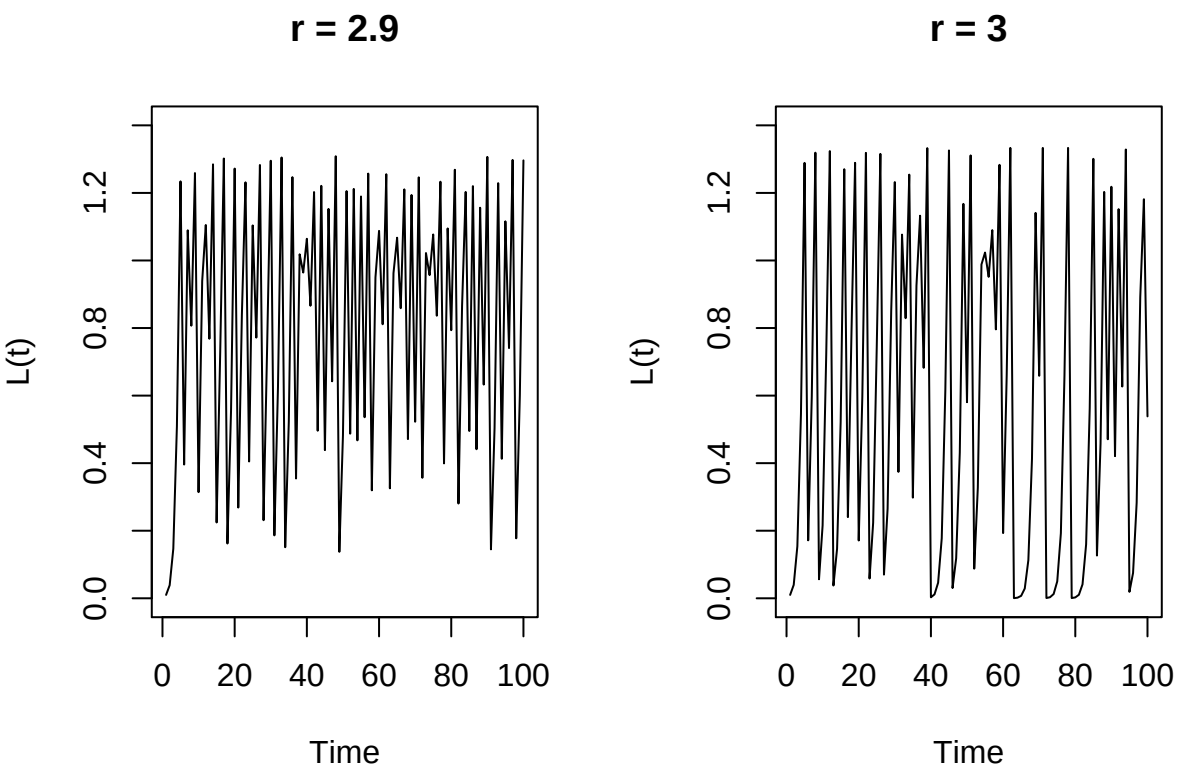
B.1.1 Solutions in R

The growth model by Van Geert (1991)

Different values for `r`:

```
library(plyr)
# Parameters
rs <- c(1.2, 2.2, 2.5, 2.7, 2.9, 3)
# Plot
op <- par(mfrow=c(1,2))
l_ply(rs,function(r){plot(growth.ac(r = r,  Y0 = 0.01, type = "vanGeert"),
                          ylim = c(0,1.4), ylab = "L(t)", main = paste("r =",r))})
```

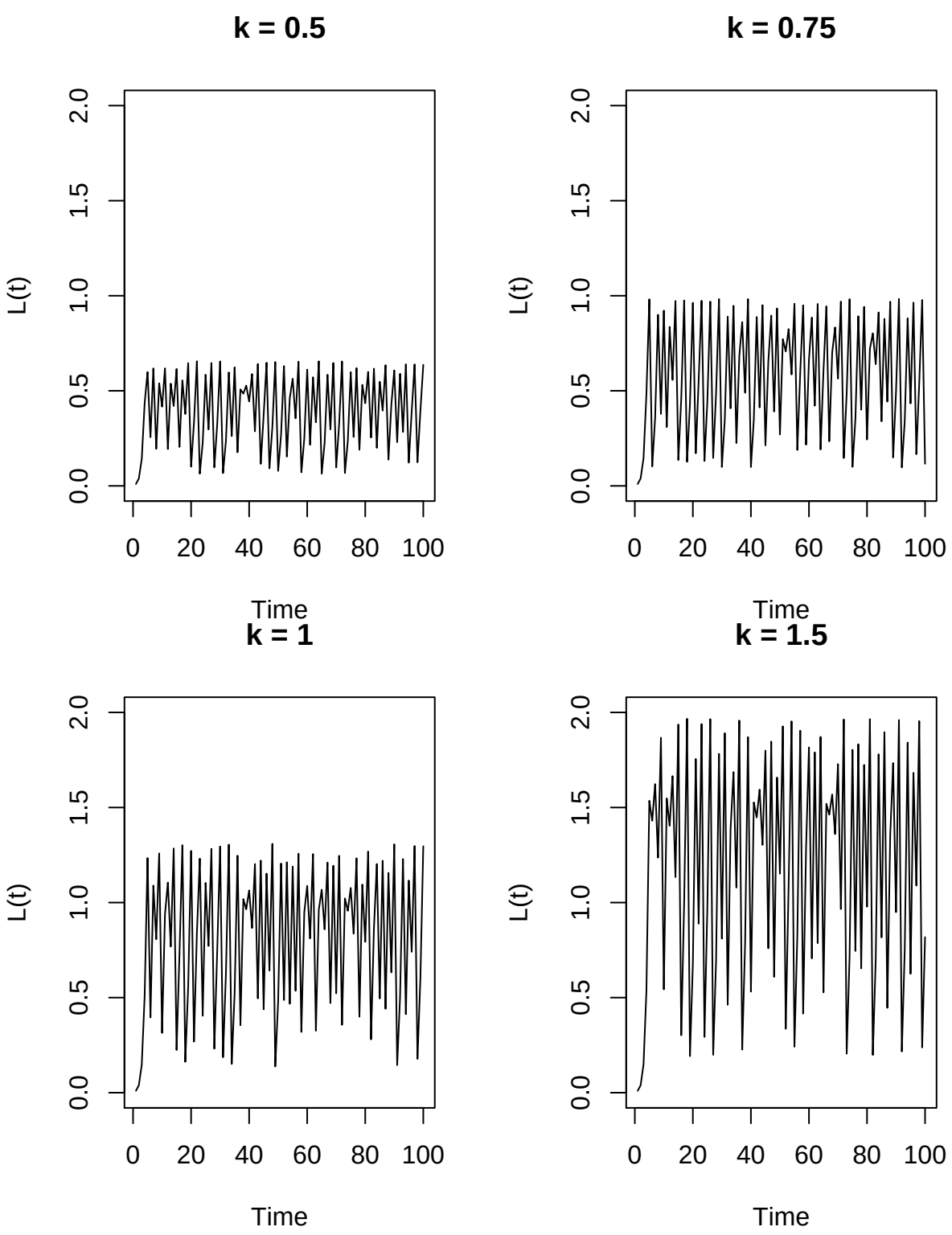

r = 1.2
r = 2.2
r = 2.5
r = 2.7
L(t)
Time
0.0
0.4
0.8
1.2
0
20
40
60
80
100




```
par(op)
```

Different values for k reveal that the dispersion of values (variance) increases if the carrying capacity increases. This occurs because we are dealing with nonlinear changes to the values of Y and if larger values of Y are allowed by a hihger k, these values will be amplified once they occur.

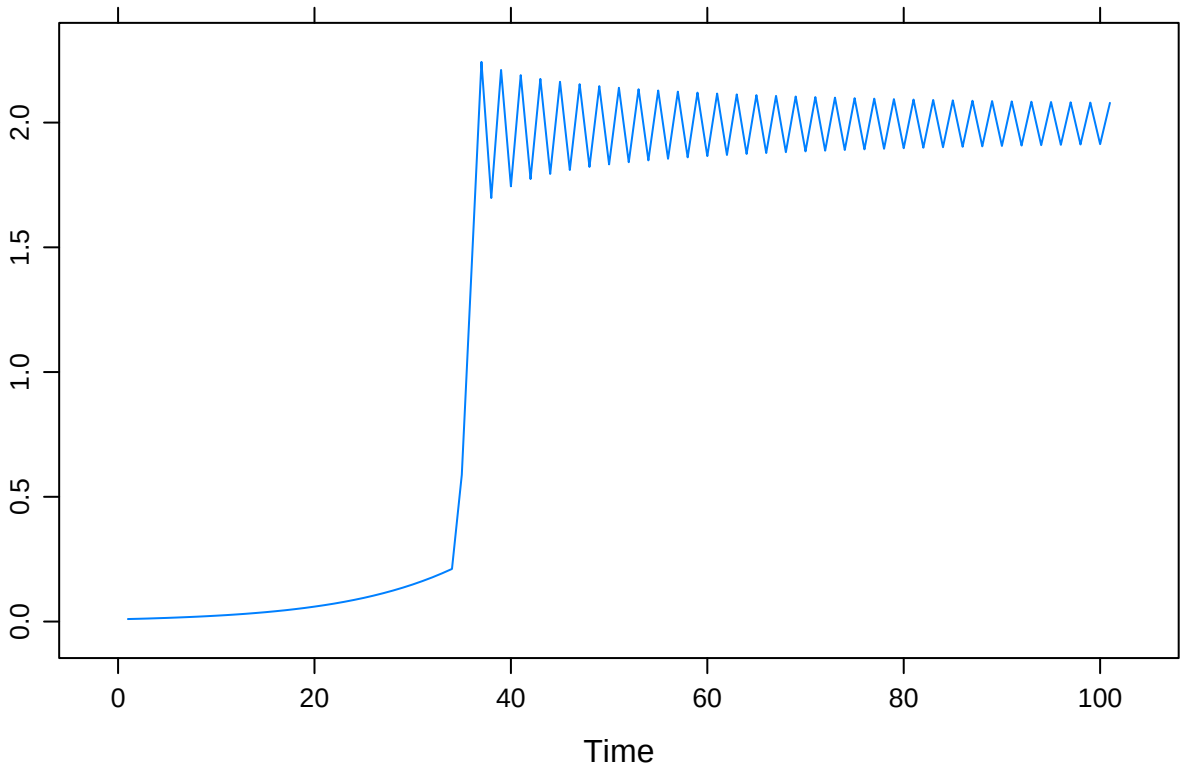
```
# Parameters
ks <- c(0.5, 0.75, 1, 1.5)
# Plot
op <- par(mfrow=c(1,2))
l_ply(ks,function(k){plot(growth.ac(r = 2.9, k = k, Y0 = 0.01, type = "vanGeert"),
                          ylim = c(0, 2), ylab = "L(t)", main = paste("k =",k))})
```




```
par(op)
```

Stages and Jumps

```
growth.ac.cond <- function(Y0 = 0.01, r = 0.1, k = 2, cond = cbind.data.frame(Y = 0.2, par = "r", val =
    # Create a vector Y of length N, which has value Y0 at Y[1]
    Y <- c(Y0, rep(NA, N-1))
    # Iterate N steps of the difference equation with values passed for Y0, k and r.
    cnt <- 1
    for(t in seq_along(Y)){
        # Check if the current value of Y is greater than the threshold for the current conditional rul
        if(Y[t] > cond$Y[cnt]){
            # If the threshold is surpassed, change the parameter settings by evaluating: cond$par = co
            eval(parse(text = paste(cond$par[cnt], "=", cond$val[cnt])))
            # Update the counter if there is another conditional rule in cond
            if(cnt < nrow(cond)){cnt <- cnt + 1}
        }
        # Van Geert growth model
        Y[[t+1]] <- Y[t] * (1 + r - r * Y[t] / k)
    }
    return(ts(Y))
}

# Plot with the default settings (same as first step in the assignment)
xyplot(growth.ac.cond())
```



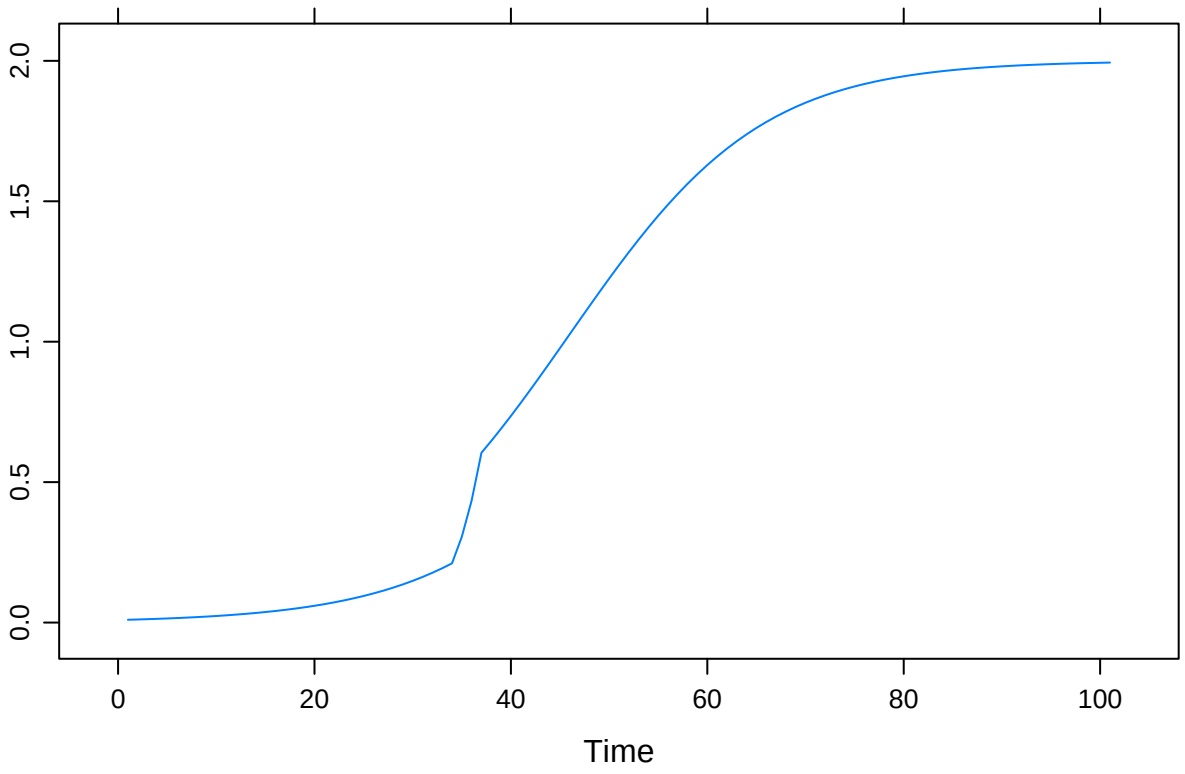
The 'trick' used here is to define the function such that it can take a set of conditional rules and apply them sequentially during the iterations. The conditiona rule is passed as a `data.frame`, but one could also use a `list` object.

```
(cond <- cbind.data.frame(Y = c(0.2, 0.6), par = c("r", "r"), val = c(0.5, 0.1)))
```

```
##     Y par val
```

```
## 1 0.2   r 0.5
## 2 0.6   r 0.1
```

```
xyplot(growth.ac.cond(cond=cond))
```

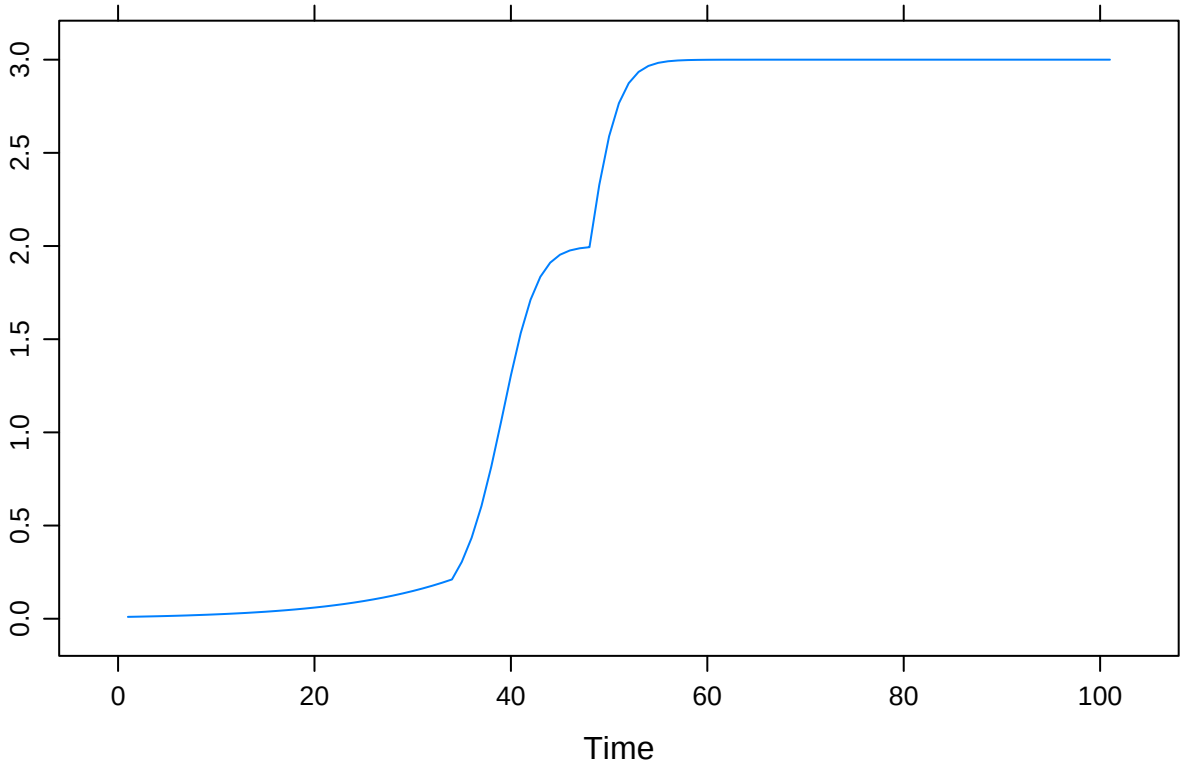


Or, combine a change of `r` and a change of `k`

```
(cond <- cbind.data.frame(Y = c(0.2, 1.99), par = c("r", "k"), val = c(0.5, 3)))
```

```
##      Y par val
## 1 0.20   r 0.5
## 2 1.99   k 3.0
```

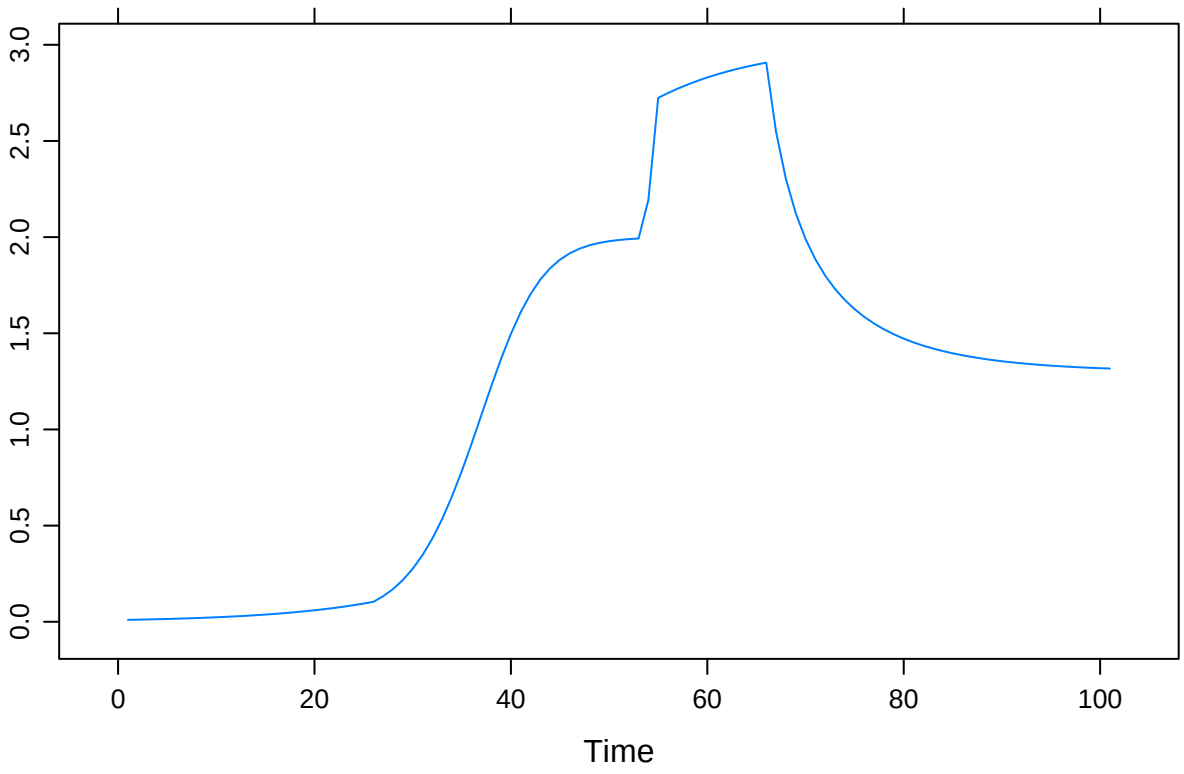
```
xyplot(growth.ac.cond(cond=cond))
```




```
# A fantasy growth process
(cond <- cbind.data.frame(Y = c(0.1, 1.99, 1.999, 2.5, 2.9), par = c("r", "k", "r", "r","k"), val = c(0
```

```
##       Y par val
## 1 0.100   r 0.3
## 2 1.990   k 3.0
## 3 1.999   r 0.9
## 4 2.500   r 0.1
## 5 2.900   k 1.3
```

```
xyplot(growth.ac.cond(cond=cond))
```



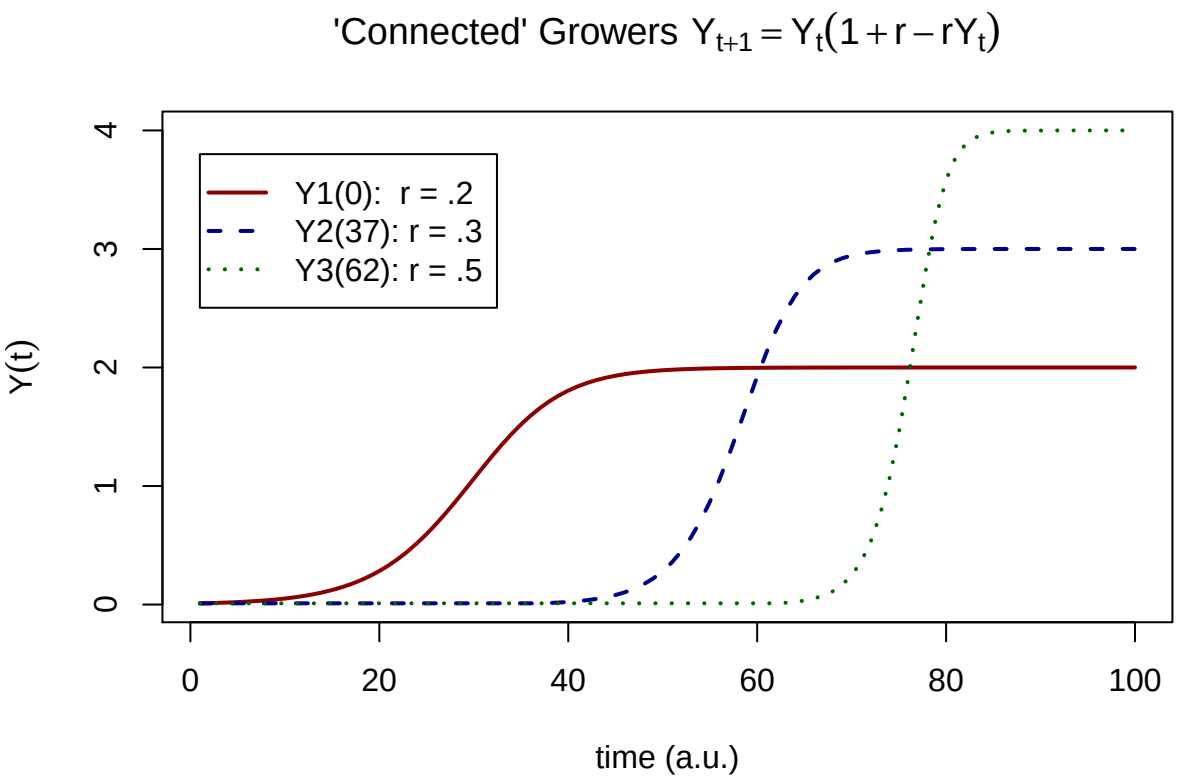
Connected Growers

Somewhat more realstic would be to model a change of `r` as dependent on the values of another process. The proper 'dynamical' way to do this would be to define a coupled system of difference or differential equations in which the interaction dynamics regulate growth. An example is the predator-prey system discussed in the next assignment.

Using the 'conditional' rules on a number of seperate processes will 'work' as a model, but it isn't exactly what is meant by *interaction dynamics*, or *multiplicative interactions*. Basically, these processes will be independent and non-interacting. The conditional rules that change the parameters are 'given'.

```
# Generate 3 timeseries
Y1 <- growth.ac(k = 2, r =.2, type = "vanGeert")
# Y2 and Y3 start at r = 0.001
Y3 <- Y2 <- growth.ac(k = 2, r = 0.001, type = "vanGeert")

# Y2 and Y3 start when k is approached
c1 <- 1.6
c2 <- 2.2
Y2[Y1 > c1] <- growth.ac(r = .3, k = 3, type = "vanGeert", N = sum(Y1 > c1))
Y3[Y2 > c2] <- growth.ac(r = .5, k = 4, type = "vanGeert", N = sum(Y2 > c2))

# Make a nice plot
ts.plot(Y1, Y2, Y3,
        gpars = list(xlab = "time (a.u.)",
                     ylab = expression(Y(t)),
                     main = expression(paste("'Connected' Growers ",Y[t+1]==Y[t]*(1 + r - r*Y[t]))),
                     lwd = rep(2,3),
                     lty = c(1:3),
                     col = c("darkred","darkblue","darkgreen")
```

```
                    )
        )
legend(1, 3.8, c("Y1(0):  r = .2",
                 paste0("Y2(",which(Y1 > c1)[1],"): r = .3"),
                 paste0("Y3(",which(Y2 > c2)[1],"): r = .5")),
       lwd = rep(2,3), lty = c(1:3), col = c("darkred","darkblue","darkgreen"), merge = TRUE)
```



B.2 Predator-prey dynamics

Iterating 2D Maps and Flows

In order to 'solve' a differential equation for time using a method of numerical integration, one could code it like in the spreadsheet assignment. For R and Matlab there are so-called *solvers* available, functions that will do the integration for you. Look at the Examples in package deSolve.

Solutions in a spreadsheet

- Predator-Prey Dynamics

B.2.1 Solutions in R

Euler's method and more...

The result of applying a method of numerical integration is called a **numerical solution** of the differential equation. The **analytical solution** is the equation which will give you a value of Y for any point in time, given an initial value Y_0. Systems which have an analytical solution can be used to test the accuracy of **numerical solutions**.

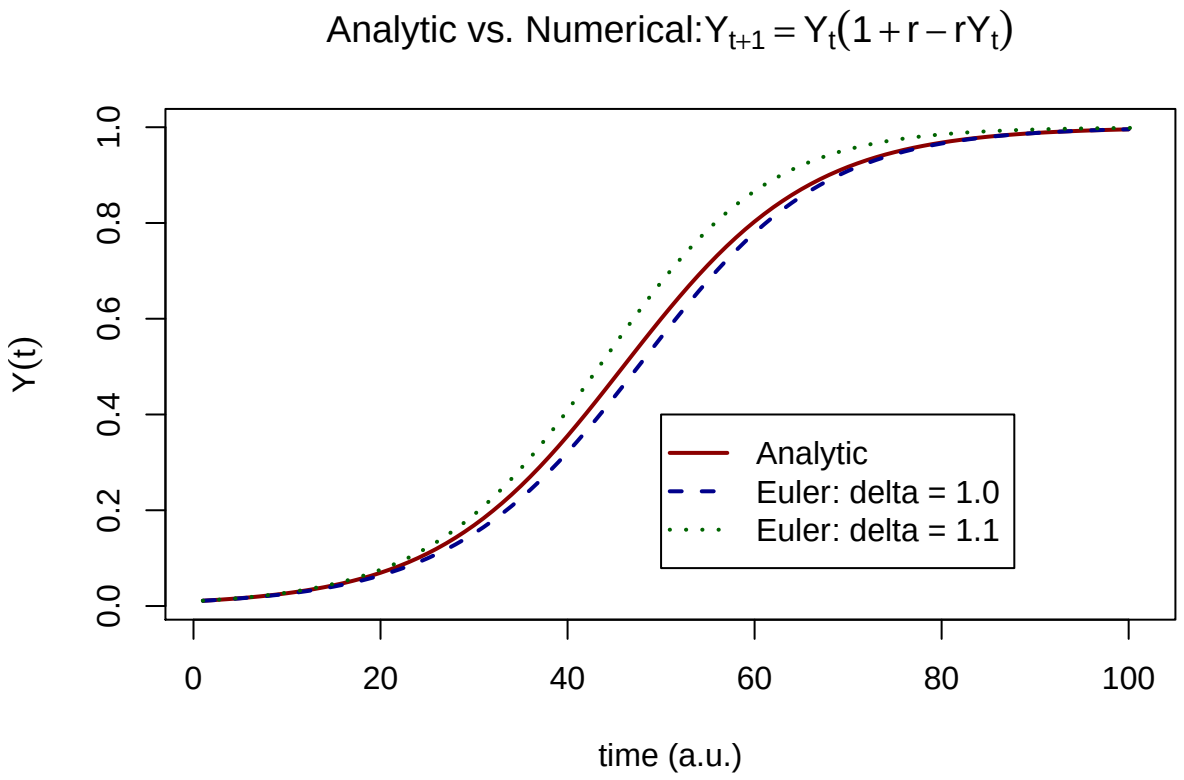
Remember that the analytical solution for the logistic equation is:

$$\frac{K}{1 + \left(\frac{K}{Y_0 - 1}\right) * e^{-r*t}} \tag{B.1}$$

We have the function `growth.ac()` and could easily adapt all the functions to use Euler's method.
Below is a comparison of the analytic solution with Euler's method.

```
# Parameter settings
d <- 1
N <- 100
r <- .1
k <- 1
Y0 <- 0.01

Y <- as.numeric(c(Y0, rep(NA,N-1)))

# Numerical integration of the logistic differential equation
Y.euler1 <- ts( sapply(seq_along(Y), function(t) Y[[t+1]] <<- (r * Y[t] * (k - Y[t])) * d + Y[t] ))
Y.euler2 <- ts( sapply(seq_along(Y), function(t) Y[[t+1]] <<- (r * Y[t] * (k - Y[t])) * (d+.1) + Y[t] )

## analytical solution
Y.analytic <- ts( k / (1 + (k / Y0 - 1) * exp(-r*(time(Y.euler1)))) )

ts.plot(Y.analytic, Y.euler1, Y.euler2,
        gpars = list(xlab = "time (a.u.)",
                     ylab = expression(Y(t)),
                     main = expression(paste("Analytic vs. Numerical:",Y[t+1]==Y[t]*(1 + r - r*Y[t]))),
                     lwd = rep(2,3),
                     lty = c(1:3),
                     col = c("darkred","darkblue","darkgreen")
                     )
```

```
        )
legend(50, 0.4, c("Analytic",
                  "Euler: delta = 1.0",
                  "Euler: delta = 1.1"),
       lwd = rep(2,3), lty = c(1:3), col = c("darkred","darkblue","darkgreen"), merge = TRUE)
```



Numerical integration

The Euler setup:

$$R_{t+1} = f_R(R_t, Ft) * \Delta + R_t \tag{B.2}$$
$$F_{t+1} = f_F(R_t, F_t) * \Delta + F_t \tag{B.3}$$

With the equations:

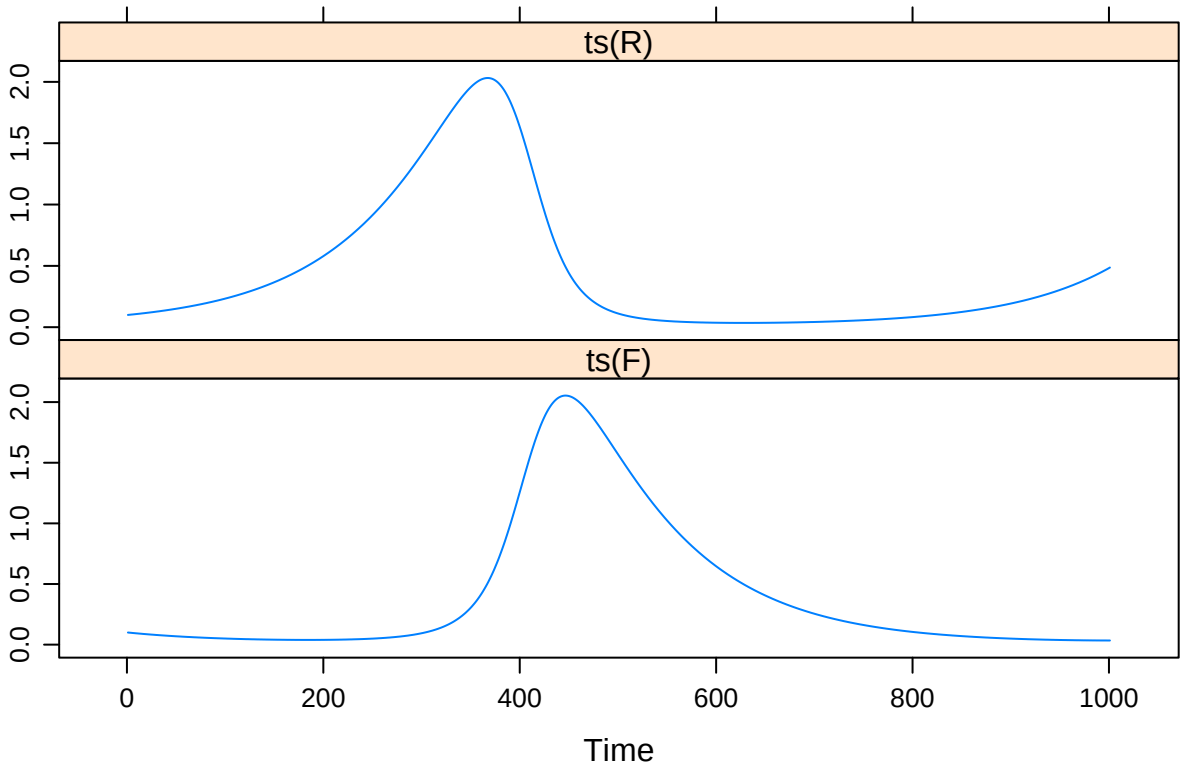
$$R_{t+1} = (a - b * F_t) * R_t * \Delta + R_t \tag{B.4}$$

(B.5)

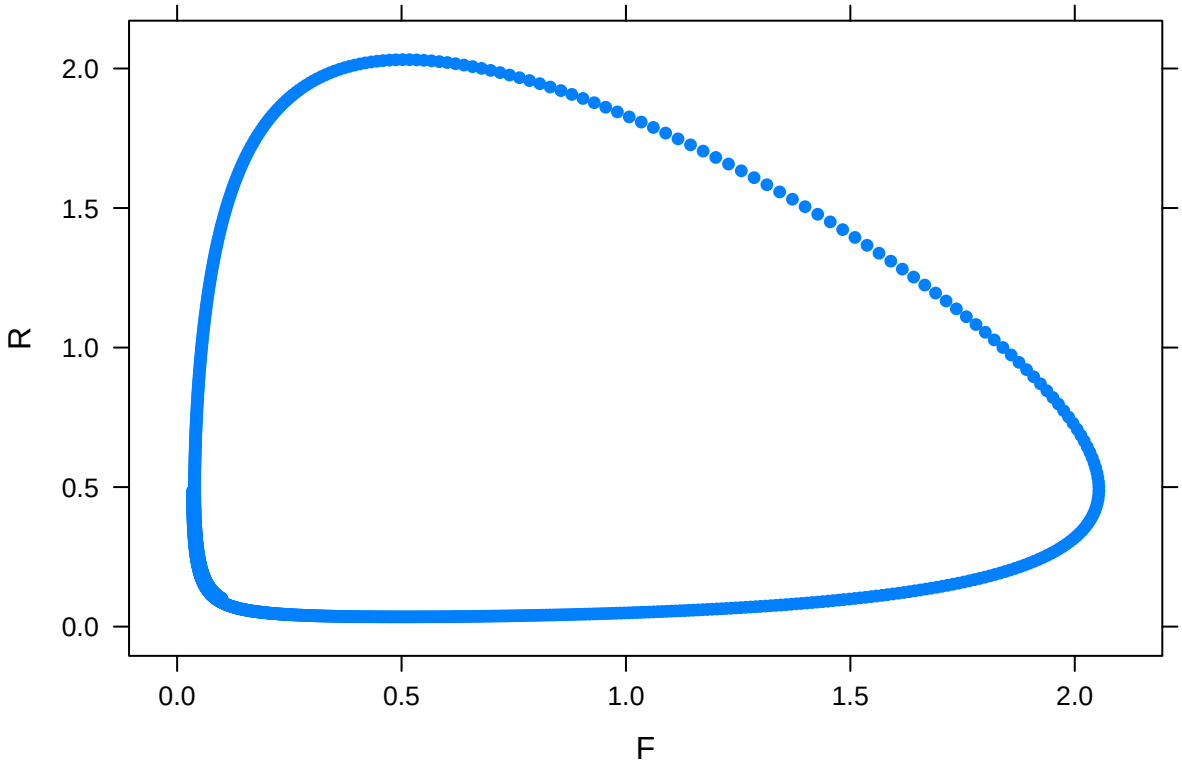
$$F_{t+1} = (c * R_t - d) * F_t * \Delta + F_t \tag{B.6}$$

```
# Parameters
N  <- 1000
a  <- d <- 1
b  <- c <- 2
R0 <- F0 <- 0.1
```

```
R  <- as.numeric(c(R0, rep(NA,N-1)))
F  <- as.numeric(c(F0, rep(NA,N-1)))

# Time constant
delta <- 0.01

# Numerical integration of the logistic differential equation
l_ply(seq_along(R), function(t){
    R[[t+1]] <<- (a - b * F[t]) * R[t] * delta + R[t]
    F[[t+1]] <<- (c * R[t] - d) * F[t] * delta + F[t]
    })

# Note different behaviour when ts() is applied
xyplot(cbind(ts(R),ts(F)))
```




```
xyplot(R ~ F, pch = 16)
```


2.0
1.5
1.0
0.5
0.0
R
0.0
0.5
1.0
1.5
2.0
F

Appendix C

Basic Timeseries Analysis

Appendix D

Fluctuation and Disperion analyses I

Appendix E

Fluctuation and Disperion analyses II

Appendix F

Phase Space Reconstruction and Recurrence Quantification Analysis (RQA)

Appendix G

Categorical and Cross-RQA (CRQA)

Appendix H

The Cusp Catasrophe Model & Warning Signs

Appendix I

Complex Networks

Bibliography

Bibliography

Allaire, J., Cheng, J., Xie, Y., McPherson, J., Chang, W., Allen, J., Wickham, H., Atkins, A., and Hyndman, R. (2016). *rmarkdown: Dynamic Documents for R*. R package version 1.1.

Chang, W., Cheng, J., Allaire, J., Xie, Y., and McPherson, J. (2016). *shiny: Web Application Framework for R*. R package version 0.14.2.

Gordon, M. (2016). *htmlTable: Advanced Tables for Markdown/HTML*. R package version 1.7.

hong Chan, C. and Leeper, T. J. (2016). *rio: A Swiss-Army Knife for Data I/O*. R package version 0.4.16.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Urbanek, S. (2013). *png: Read and write PNG images*. R package version 0.1-7.

Wickham, H. (2016a). *plyr: Tools for Splitting, Applying and Combining Data*. R package version 1.8.4.

Wickham, H. (2016b). *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*. R package version 0.6.0.

Wickham, H. and Francois, R. (2016). *dplyr: A Grammar of Data Manipulation*. R package version 0.5.0.

Xie, Y. (2016a). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.2.

Xie, Y. (2016b). *DT: A Wrapper of the JavaScript Library 'DataTables'*. R package version 0.2.

Xie, Y. (2016c). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.15.