Dynamical Models in Systems Biology

Instructor(s): Samuel Bernard bernard@math.univ-lyon1.fr

link to the latest version: https://moodle.univ-lyon1.fr/mod/resource/view.php?id=265039

1 What is a dynamical model?

A dynamical model is a mathematical or computer model where the variables, or quantities of interest, vary in time. They usually do so according to a causal mechanism, i.e. the values of the variables at a given time depend on their values in the past.

Perhaps the simplest dynamical models are **difference equations**

difference equations

$$\overbrace{x_{t+1}}^{\text{dynamical variable}} = \overbrace{f(x_t)}^{\text{update rule}} .$$

Time is discrete, and the causal mechanism is clear: x_{t+1} is a function of x_t. For example, the famous *logistic map* is

$$x_{t+1} = \overbrace{r}^{\text{growth rate}} x_t \overbrace{(1-x_t)}^{\text{limiting factor}} .$$

The variable x is the size of the population of a certain species, expressed as a fraction of a maximal size. The logistic equation was used to show how the size of a population can follow a complex dynamics in absence of environmental fluctuations (May, 1976). A small populations grows with a multiplication factor r (factor rx). When it gets larger, the growth is impeded by the factor $1-x$, which could represent competition for resources. We will see later how such a simple equation can lead to complex dynamics.

The most important class of dynamical models are **ordinary differential equations** (ODEs). The time t varies continuously, and the causal mechanism $f(x(t))$ describes how variables $x(t)$ should vary (this is the derivative on the left-hand side) as a function of their current state:

ordinary differential equations

$$\overbrace{\frac{dx}{dt}}^{\text{rate of change}} = \overbrace{f(x)}^{\text{rule of change}} .$$

The **state variable** x is a function of time, $x(t)$, but it is usual to drop the explicit dependence on time to highlight the fact that changes in the system depend on the state variable rather than time. Many physical laws can be formulated is such a way, and big parts of biology also have their laws, which we express as **motifs**. The *Lotka-Volterra model* was developed to study predator-prey interaction. The mechanism is reminiscent of the logistic equation. The prey density y grows at a constant rate by, and is harvested by the predator at a rate proportional to the predator density xy, while the predator density grows at the harvest rate and dies at a constant rate ax. The equation can be expressed as

state variable

motifs

$$\frac{dx}{dt} = \overbrace{yx}^{\text{harvest term}} - \overbrace{a}^{\text{death rate}} x, \quad x : \text{predator density},$$
$$\frac{dy}{dt} = \overbrace{b}^{\text{growth rate}} y - \overbrace{xy}^{\text{harvest term}}, \quad y : \text{prey density}.$$

The term dx/dt denotes the derivative of x with respect to time. The derivative is the rate or the speed at which the variable changes. Difference equations and ODEs may have analytic, or closed form solutions, but these are the exception, not the rule. These solutions, when we can find them, are not as useful as one might think; they are often opaque and do not provide additional insight into the behavior of the model.

Exercise 1 Units

What kind of physical units can x, y, a and b have? Are the equations consistent in terms of units?

In general, dynamical models include constant **model parameters**, that do not vary with time. Parameters are important because they can affect the behavior of the solution, and they often have biological significance. Estimating parameter values given a model and experimental data is the subject of the session Inferring model parameters.

Rather than looking for analytical solutions, we will use a combination of computer simulations and stability analysis to characterize the behavior of the dynamical models. R codes necessary to simulate the logistic equation and the Lotka-Volterra model are available on the [Systems Biology Class webpage].

Other dynamical modeling formalisms include **delay differential equations**, **stochastic processes**, **partial differential equations**, **individual-based models**. These are out of scope of this introduction to dynamical models.

delay differential equations

stochastic processes

partial differential equations

individual-based models

Why is it important in systems biology? Dynamic models provide mechanisms, and mechanisms provide understanding, which provide ground for validating results.

2 How to use dynamical models?

Here is a modelling-centric workflow for using dynamical models. Each step may, and will, fail; this is normal. Then go back to previous step and start over.

- **What biological question do I want to model?**
  Two favorite questions of mine are: Can we reproduce these strange looking data with a very simple model (sufficient mechanism), and what are the conditions for my treatment to work (necessary mechanisms). These questions are best studied with dynamical models because they relate to mechanisms. When the mechanism is well understood, we can try to estimate model parameters.
- **What would be the appropriate model for it?**
  Choose the variables
  The choice mostly depend on the availability of experimental data and the question asked.
  Select the mechanisms
  Here the devil is in the details. Which of the known mechanism should we include? Here there is no fixed method, but everybody would agree that given two models with similar solution, the simplest model should be favored. This is the Occam's razor, or parsimony principle. By simple, we mean few parameters, dynamical variables and nonlinear terms (in that order).
- **What type of data do I need?**
  Are data available, or should I collect new data?
- Find an existing model, or develop a new one
  Translating the mechanisms into equations has many pitfalls. Ambiguous language or imprecise wording makes it difficult to define equations uniquely. Once interpreted, the mechanism must satisfy physical and biological constraints, which may be easily overlooked.
- Get an intuition of the behavior of the model
  Very important step. How should the solution look like? This step is useful to detect any error in the equations or in the numerical implementation of the model (they are not the same! See below).
- Implement the model and run simulations to confirm intuition
  The numerical implementation of the model is not the same as the equations. Furthermore the equations might not represent well the mechanisms.
- **Compare to experimental data**
  Before any attempt to fit the model to experimental data, it is important to look whether the model reproduces the important features of the data. Only then fitting the model makes sense.
- Perform analysis of the model
  Models can reproduce experimental data very well for some sets of parameters, but may be fragile in the sense that small change of parameters can lead to vastly different dynamical behavior. This can be a weakness when the system is expected to be robust, but may also provide testable predictions: can the different behaviors be observed experimentally?
- Answer the question Once the model is validated and the parameters are known,

do not forget to answer the initial question!
- Find a new question

3 Modeling with motifs

Unlike chemical and physical systems, biological systems are not easy to reduce to simple parts. Whether we look at the gene expression, protein interaction, cell fate, metabolism, tissue or organ physiology, *in vivo* systems are complex and interrelated. This does not mean that it is impossible to isolate single mechanisms, but that there is no fundamental rules on how to express them. For example, how gene expression is affected by a transcription factor depends on the availability of the binding site, which depends on the DNA conformation, which depends on histone acetylation, and so on. **Motifs** are small blocks of regulation that can be used to distill all the complexity of biology into simple parametric term.

In the following list, the variable x denote the species of interest. This can be gene expression level, mRNA or protein concentration, cell density, drug concentration.

- Loss/death/degradation rate - Linear. The species dies or is removed at a rate proportional to its level, with a constant k: kx.
  Example: a protein with initial concentration x_0 is degraded at a rate $k = 0.1$ per hour, and is not synthesized. The equation for the concentration of x is $dx/dt = -kx$, $x(0) = x_0$. This ODE has an explicit solution $x(t) = x_0 e^{-kt}$, the concentration decreases exponentially.
  - Saturated. The loss rate is linear with constant k_0/K when x is small, but saturate to a fixed value k_0 when x is large. The simplest model for saturated kinetics is the Michaelis-Menten model: $k_0 x/(K + x)$.
- Constant production rate. Production refers to a supply of the species that does not depend directly on its concentration: b.
- Proliferation/reproduction/synthesis rate - Linear: rx, - Logistic (competition) $rx(1-x/K)$. - Negative feedback: $r_0/(K^h+x^h)$. - Positive feedback: $r_0 x^h/(K^h+x^h)$.

The parameter h is a cooperativity coefficient, called Hill coefficient. It defines the strength of the feedback. High Hill coefficient will make the feedback quite sensitive to small variations of x. This can lead to complex dynamics such as **oscillations** and **bistability**.

oscillations

bistability

4 Examples

Examples are implemented in `R`, with the package `deSolve`. All major programming languages offer some numerical solvers: `matlab`, `python`. Here we use `R` because it offers many functionalities to deal with complex datasets as found in systems biology. `Python` and `matlab` also offer similar functionalities but user-friendliness may vary.

Example 1 Birth/death model

Here is the simplest ODE model we can think of, the linear birth-death ODE model,

$$\frac{dx}{dt} = \overbrace{b}^{\text{immigration}} + \overbrace{rx}^{\text{proliferation}} - \overbrace{kx}^{\text{loss}}.$$

The species has a constant production rate b (immigration), a linear growth rate r (proliferation) and a linear death or loss rate k.

```
birthDeath <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {

    # ---------------------------------------------
    # Define the equations here
    # ---------------------------------------------
    deathRate         <- k * x
    productionRate    <- b
    proliferationRate <- r * x

    dxdt     <- productionRate + proliferationRate - deathRate

    return(list(c(dxdt)))
    # ---------------------------------------------
  })
}

pars  <- c(k  = 0.5,     # per day, death rate
           b  = 0.2,     # individuals per day, production from
           ↪  outside source
           r  = 0.1 )    # per day proliferation (or reproduction)
           ↪  rate

y0  <- c(x = 1.0)
timespan <- seq(0, 20, by = 0.1)
birthDeath.sol <- ode(y0, timespan, birthDeath, pars)
summary(birthDeath.sol)
```

To plot the result

```
plot(birthDeath.sol)
```

Exercise 2 Exercises on the birth/death model

(a) With the parameters given above, the solution $x(t)$ converges to a constant value, which one?

(b) The solution does not always converge to a constant. Find conditions on the parameters so that the solution always converge to a positive value given a positive initial condition.

(c) How can the equation be modified to ensure that the solution will always remain bounded given positive initial conditions? Write down the modified birth/death model and try to guess to which value the solution will converge.

(d) Implement the modified birth/death model in R and run simulations to verify your intuition.

Example 2 Lotka-Volterra

We have seen above the equations for the Lotka-Volterra model

$$\frac{dx}{dt} = \overbrace{yx}^{\text{harvest term}} - \overbrace{a}^{\text{death rate}} x, \quad \text{predator},$$
$$\frac{dy}{dt} = \overbrace{b}^{\text{growth rate}} y - \overbrace{xy}^{\text{harvest term}}, \quad \text{prey}.$$

The R code for the Lotka-Volterra equations

```
LotkaVolterra <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {

    # ------------------------------------------------
    # Define the equations here
    # ------------------------------------------------
    killingRate       <- Prey * Predator
    preyGrowthRate    <- rGrowth * Prey
    predatorDeathRate <- rDeath * Predator

    dPreydt     <- preyGrowthRate - killingRate
    dPredatordt <- killingRate - predatorDeathRate

    return(list(c(dPreydt, dPredatordt)))
    # ------------------------------------------------
  })
}

pars  <- c(rGrowth  = 0.5,    # per day, growth rate of prey
           rDeath   = 0.2 )   # per day, death rate of predator

y0  <- c(Prey = 10, Predator = 2)
timespan <- seq(0, 200, by = 1)
LotkaVolterra.sol <- ode(y0, timespan, LotkaVolterra, pars)
summary(LotkaVolterra.sol)
```

The solution can be plotted with

```
plot(LotkaVolterra.sol)
```

Exercise 3 Exercises on the Lotka-Volterra model

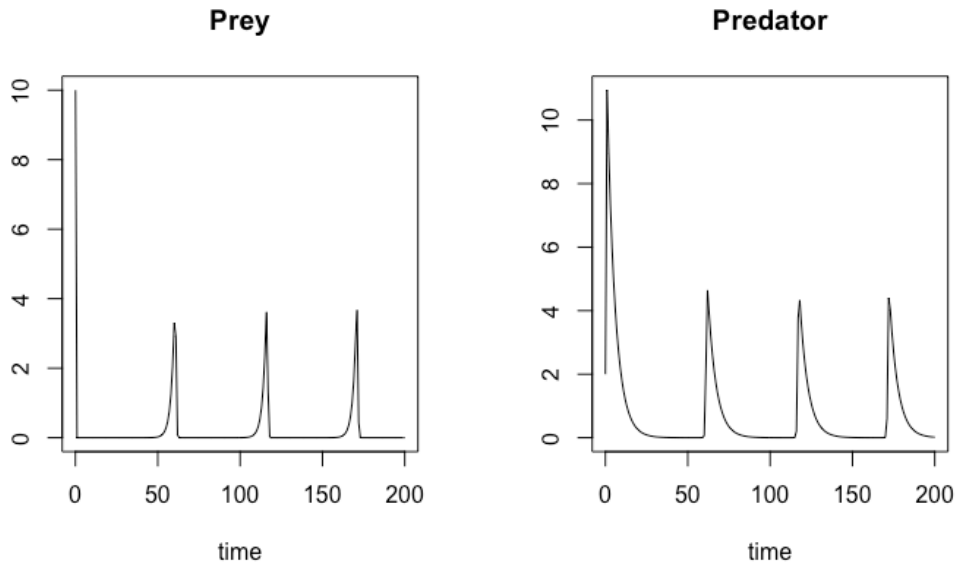

Figure 1. Lotka-Volterra dynamics

(a) Run the simulations with different initials conditions. What do you observe?

(b) Modify the code above so that the growth rate of the prey also include competition between the preys for resources.

Example 3 A negative feedback loop (Goodwin model)

Negative feedback loops occur everywhere were the product inhibits its own production. This can be through limited food or space, or through homeostatic regulation, to control body temperature or blood pressure for instance. In most cases negative feedback loops have the effect of making steady state more stable, i.e. after external perturbation the system quickly returns to its natural state. This useful for instance for rapid red blood cell mobilisation after blood loss. However too much of a good thing can have unintended effects, and negative loop can destabilize an otherwise stable steady state. This occurs in the Goodwin model below.

The Goodwin model is the prototype of the negative feedback loop that occurs in many gene regulation networks. To work properly, we need three species. Here we take mRNA concentration X, a protein product concentration Y and a modified protein complex concentration Z. All species have linear degradation rates. The protein is produced at a rate proportional to the mRNA concentration, and the protein complex is produced at a rate proportional to the protein concentration. For simplicity, we set the degradation and production rates of the protein and the complex to the value β, and the mRNA degradation rate to α. To construct the negative feedback loop, we will assume that the protein complex binds to the gene promoter to repress mRNA synthesis is a concentration-dependent manner. Moreover we will assume that in absence of the repressor, mRNA is transcribed (produced) at a constant

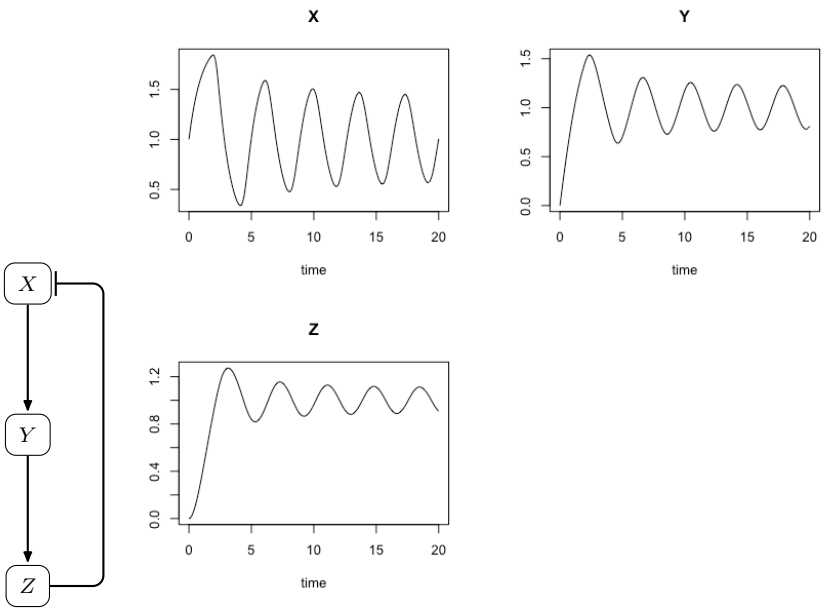

Figure 2. (Left) Diagram of the Goodwin network. Arrow heads denote positive effect and "tee" heads denote negative effects. (Right) Solution of the Goodwin model. Solutions are oscillatory.

rate.

Using the negative feedback motif, we can write down the mRNA synthesis rate as

$$f(Z) = k_0 \frac{K^h}{K^h + Z^h}.$$

When there is no repressor (Z=0), the synthesis rate is k_0, and when the repressor expression is $Z = K$, the synthesis rate is reduced by half $k_0 K^h/(K^h + K^h) = k_0/2$. We obtain a set of three ODEs

$$\begin{aligned} \frac{dX}{dt} &= f(Z) - \alpha X, \\ \frac{dY}{dt} &= \beta(X - Y), \\ \frac{dZ}{dt} &= \beta(Y - Z). \end{aligned}$$

The R code to implement the Goodwin model

```
Goodwin <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {

    # ---------------------------------------------
    # Define the equations here
    # ---------------------------------------------
    mRNAproductionRate <- k0*K^h/(K^h + Z^h)
```

```
    mRNAdegradationRate    <- alpha * X

    dXdt <- mRNAproductionRate - mRNAdegradationRate
    dYdt <- beta * ( X - Y )
    dZdt <- beta * ( Y - Z )

    return(list(c(dXdt, dYdt, dZdt)))
    # ------------------------------------------------
  })
}

pars  <- c(k0      = 2,    # transcripts per hour, max mRNA
↪  synthesis rate
           alpha   = 1.0,  # per hour, mRNA degradation rate
           beta    = 1.0,  # per hour, kinetic rate
           K       = 1,    # mmol, half-repression concentration
           h       = 20 )  # no unit, Hill coefficient

y0  <- c(X = 1, Y = 0, Z = 0)
timespan <- seq(0, 20, by = 0.1)
Goodwin.sol <- ode(y0, timespan, Goodwin, pars)
summary(Goodwin.sol)
```

Note There is a Goodwin model in the field of economics as well, and to make thing confusing, the economic Goodwin model is mathematically equivalent to the Lotka-Volterra model. Thus although both economic and biological Goodwin model can display oscillations, they are completely unrelated to each other.

Exercise 4 Goodwin model

(a) There is a unique positive steady state. Using the parameters values in the R code, find it (set all derivative to zero in the ODE system and solve the three equations for X, Y and Z).

(c) Vary the Hill coefficient h until the steady state becomes stable. What is the value of h? At this value, we say that the Goodwin model undergoes a bifurcation, i.e. a change in the qualitative behavior of the system. Many diseases are associated to qualitative changes in physiology, and dynamical models are used to devise therapeutic strategies to reverse bifurcations. The most successful ones are for heart arrhythmia, such as calcium channel blockers or pacemakers.

Example 4 A positive feedback loop

Positive feedback loop are inherently unstable. They do occur in irreversible events such as mitosis, birth giving, differentiation and lineage choice, etc.

The positive feedback loop rests on the positive loop motif for the production of the species with concentration X.

$$g(X) = k_0 \frac{X^h}{K^h + X^h},$$

where the Hill coefficient $h > 1$. The production depends strongly on X. For low concentrations, the production is low. However for high concentrations, the production is much higher. This nonlinear production curve leads to possible low and high concentrations stable steady states. The ODE reads

$$\frac{dX}{dt} = g(X) - aX.$$

When they exist, the two stable steady states are always separated by a third steady state, which is unstable. They can be found by setting $dX/dt = 0$. This leads to a fixed point equation on X: $aX = g(X)$.

The R code to obtain bistability is

```
Bistability <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {

    # ---------------------------------------------
    # Define the equations here
    # ---------------------------------------------
    mRNAproductionRate <- k0*X^h/(K^h + X^h)
    mRNAdegradationRate    <- a * X

    dXdt <- mRNAproductionRate - mRNAdegradationRate

    return(list(c(dXdt)))
    # ---------------------------------------------
  })
}

pars  <- c(k0      = 2,    # transcripts per hour, max mRNA
↪  synthesis rate
           a       = 1.0,  # per hour, mRNA degradation rate
           K       = 1,    # mmol, half-repression concentration
           h       = 20 )  # no unit, Hill coefficient

y0  <- c(X = 1.1)
timespan <- seq(0, 20, by = 0.1)
Bistability.sol1 <- ode( c(X = 0.9), timespan, Bistability, pars)
Bistability.sol2 <- ode( c(X = 1.1), timespan, Bistability, pars)
Bistability.sol3 <- ode( c(X = 1.0), timespan, Bistability, pars)
```

To plot all three solutions on one graph

```
plot(Bistability.sol1, Bistability.sol2, Bistability.sol3)
```

Exercise 5 Bistable model

(a) With the parameters given in the code above, find (approximately) all three steady states. Which ones are stable, unstable?

Example 5 Logistic map

The logistic map is the difference equation

$$x_{t+1} = rx_t(1 - x_t),$$

for $t = 0, 1, ...$, with the initial condition x_0 given. The R codes to solve the logistic map follow the same lines as ODE models, except that the we use the `iteration` method.

```
LogisticMap <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {

    # ---------------------------------------------
    # Define the equations here
    # ---------------------------------------------
    xiter     <- r * x * ( 1 - x )

    return(list(c(xiter)))
    # ---------------------------------------------
  })
}

pars  <- c(r  = 3.76 )     # basal growth parameter

y0  <- c(x = 0.2)
timespan <- seq(0, 50, by = 1)
LogisticMap.sol <- ode(y0, timespan, LogisticMap, pars, method =
↪  "iteration")
summary(LogisticMap.sol)
```

The logistic map is one of the simplest dynamical model displaying chaos, oscillatory solutions but very irregular and sensitive to initial conditions. Chaos arises as the parameter r increases from 1.0 to 4.0. For small values of r, the logistic map has one stable steady state. As the parameter is increased, the steady state becomes unstable and is replaced by a periodic solution with period 2. This is followed by a series of period doubling bifurcations, ultimately ending up in chaos.

Exercise 6 Logistic map

(a) Explore the solutions of the logistic map. Try to find solutions with periods 2, 4, 8… Can you find a solution with period 3? With period 5?

5 Delay Differential Equations

Models of self-regulating systems often include discrete delays in the feedback loop to account for the finite time required to perform essential steps before the loop is closed. Such mathematical simplifications are especially welcome in biological applications, where knowledge about the loop steps is usually sparse. This includes

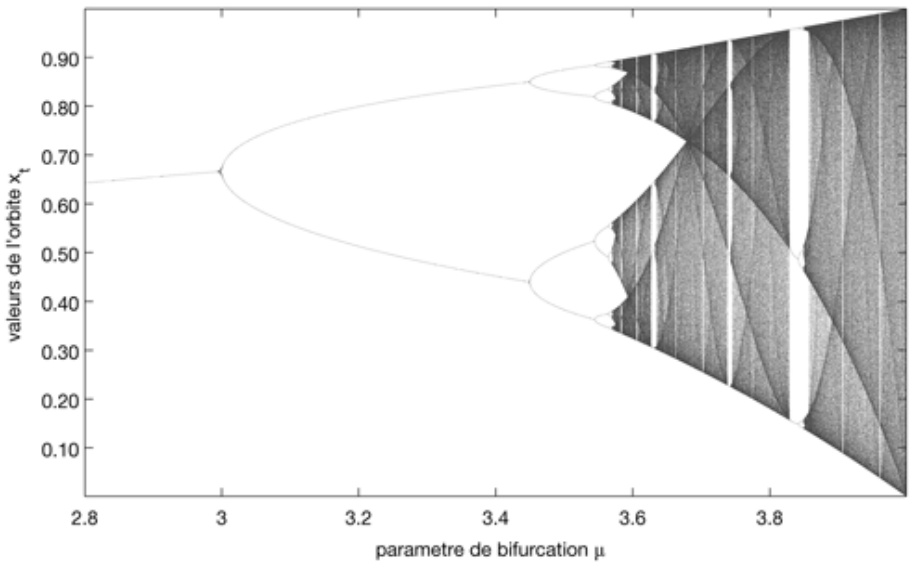

Figure 3. Period doubling bifurcation road to chaos in the logistic map.

maturation and growth times needed to reach reproductive age in a population, signal propagation along neuronal axons, and post-translational protein modifications. Introduction of a discrete delay in an ordinary differential equation can destabilize steady states and generate complex dynamics, from limit cycles to chaos.

The discrete delay differential equation of the form

$$\dot{x} = F\Big(x, x(t-\tau)\Big)$$

is a model paradigm in biology and physics. The first argument of F is the instantaneous part of the loop and the second one, the delayed or retarded part, which closes the feedback loop. The **discrete delay** τ is a positive constant.

discrete delay

Example 6 The Goodwin model with a transcriptional delay

We can assume that there is a lag between initiation of transcription and the synthesis of a mRNA molecule, i.e. that X should depend on the concentration of Z, but at a time τ earlier.

$$\begin{aligned}
\frac{dX}{dt} &= f(Z(t-\tau)) - \alpha X, \\
\frac{dY}{dt} &= \beta(X - Y), \\
\frac{dZ}{dt} &= \beta(Y - Z).
\end{aligned}$$

The package `deSolve` includes a delay differential equation solver called `dede`. Here

is an example of how to solve the Goodwin equation with a transcriptional delay.

```
Goodwin.delay <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {

    # ------------------------------------------------
    # Define the equations here
    # The delayed variable Z(t - tau) is accessed with the
    # function lagvalue: Z(t - tau) = lagvalue(Time - tau,3)
    # The argument Time - tau must be greater than 0.
    # When Time < tau, the delayed variable is specified by its
    # initial condition, here Z = 0.
    # ------------------------------------------------
    if (Time < tau)
      mRNAproductionRate <- k0*K^h/(K^h) # Z = 0 initially
    else
      mRNAproductionRate <- k0*K^h/(K^h + lagvalue(Time - tau,3)^h)

    mRNAdegradationRate    <- alpha * X

    dXdt <- mRNAproductionRate - mRNAdegradationRate
    dYdt <- beta * ( X - Y )
    dZdt <- beta * ( Y - Z )

    return(list(c(dXdt, dYdt, dZdt)))
    # ------------------------------------------------
  })
}

pars  <- c(k0      = 2,    # transcripts per hour, max mRNA
↪  synthesis rate
           alpha   = 1.0,  # per hour, mRNA degradation rate
           beta    = 1.0,  # per hour, kinetic rate
           K       = 1,    # mmol, half-repression concentration
           h       = 3,   # no unit, Hill coefficient
           tau     = 10 )  # transcriptional delay

y0  <- c(X = 1, Y = 0, Z = 0)
timespan <- seq(0, 100, by = 0.1)
Goodwin.delay.sol <- dede(y0, timespan, Goodwin.delay, pars)
plot(Goodwin.delay.sol)
```

6 Stochastic Differential Equations

The Lanvegin equation was introduced by Paul Langevin to describe the random motion of a large particle (a Brownian particle) in a bath of microscopic particles (atoms). The Brownian particle, being much heavier than the atoms, moves much more slowly, and its motion is influence by a large number of atoms. In the over-

damped limit where the viscosity is large (γ is large), the acceleration term can be neglected, and the Langevin equation reduces to a first-order equation

$$\gamma \frac{dx}{dt} = \xi(t).$$

The term $\xi(t)$ is a stochastic (random) noise term that needs to be defined. It is usual, but not necessary, to choose $\xi(t)$ as a **Gaussian white noise**. A Gaussian white noise is a Gaussian random variable with mean 0 (centered) that is completely uncorrelated: if $t \neq t'$, then $\xi(t)$ is completely independent of $\xi(t')$. However, at any given time t, the variance of $\xi(t)$ must be large, otherwise the noise term would have no effect on the Langevin equation.

Gaussian white noise

More generally, an overdamped particle subject to the potential $U(x)$ has the Langevin equation

$$\gamma \frac{dx}{dt} = -\frac{\partial U}{\partial x} + \xi(t).$$

In biological applications, the variable x is not necessarily a position, it can be a density or concentration, or an electrical potential. Likewise, the term $-\frac{\partial U}{\partial x}$ can be any function f describing the "deterministic" part of the movement. The Langevin equation can be re-expressed in a more common way

$$\frac{dx}{dt} = f(t,x) + g(t,x)\xi(t),$$

where the positive function g is the intensity of the noise. This is the most common class of **stochastic differential equation** used in biology.

stochastic differential equation

There is a specific method for solving stochastic differential equations called the Euler-Maruyama scheme. Because the solution is stochastic, the Euler-Maruyama method computes specific realizations of the solution. To compute a realization $x(t)$ over the time interval $0 \leq t \leq T$, we choose a small timestep h, and discretize the time at points $0, h, 2h, ..., T$. We denote by t_k the time point hk, and by x_k the numerical solution at that time point. The numerical scheme computes the solution at time step $k+1$, assuming that the solution is known at time step k:

$$x_{k+1} = x_k + hf(t_k, x_k) + g(t_k, x_k)\sqrt{h}\xi_k,$$

where ξ_k are i.i.d. (independent identically distributed) Gaussian random variables with distribution $N(0,1)$.

6.1 Case study: Electrical properties of cells membranes

We are interested in the electric potential across the membrane of a neuron Let x be the voltage or the difference of potential between inside and outside the cell. The membrane of a cell is a good insulator and electrons cannot freely cross it. For the voltage to change, ions such as sodium and potassium have to go through ion

channels. These ion channels can be open or close, and their state depends on the voltage x itself. The voltage across the membrane is proportional to the electric charge of the cell, with a proportionality constant C,

$$C = Q/x,$$

where Q is the electric charge of the cell, x is the voltage and C is called *capacitance*. Change in electric charge (and voltage) occurs when ions cross the membrane. The rate of change of charge is the current I

$$\frac{dQ}{dt} = I,$$

and from the voltage-charge relation

$$C\frac{dx}{dt} = I,$$

The total current through the membrane is the sum of ionic currents, which is zero at rest ($dx/dt = 0$). At rest, the cell has an electric potential different from zero because active ion pump maintain a difference in ion concentrations across the membrane. When voltage changes, ion channels close or open, and ions flow in or out of the cell, creating a current. The Hodgkin-Huxley model is a detailed model of the different ion channels. A simplified version is the FitzHugh-Nagumo model. If we neglect the recovery variable in the FitzHugh-Nagumo model, we obtain a nonlinear Langevin equation

$$\frac{dx}{dt} = x - x^3 + \sigma\xi(t) = -\frac{\partial U}{\partial x} + \sigma\xi(t),$$

with potential $U(x) = -x^2/2 + x^4/4$, and $\xi(t)$ a Gaussian white noise.

This is admittedly a very schematic model for a neuron, but we can draw analytic insight from it. In absence of noise ($\sigma = 0$), there are three steady states: one unstable steady state at $x = 0$ and two stable ones at $x = \pm 1$. These steady state correspond to local extrema of the potential $U(x)$. In presence of noise, trajectories are not constant, so steady states cannot exist as in the deterministic case. Nevertheless, trajectories starting near one of the local minima of the potential will stay there for a while, until a perturbation large enough can kick the trajectory near the other local minimum. In the long run, expect the probability density of x to be concentrated near the local minima of $U(x)$. Therefore $-U(x)$ gives us an idea of the shape of the probability density of $x(t)$, given large enough times t.

7 Partial Differential Equations

To make things more quantitative, let $p(x, t)$ be the probability density of x at time t, i.e.

$$\Pr(x < a) = \int_{-\infty}^{a} p(x,t)dx.$$

Fokker-Planck

The **Fokker-Planck** equation is an partial differential equation for the probability density of the solution of the Langevin equation. For the equation

$$\frac{dx}{dt} = \mu(x) + \sigma\xi(t),$$

with a nonlinear deterministic part $\mu(x)$ and a noise with constant σ, the Fokker-Planck equation reads

$$\frac{\partial p}{\partial t} = -\frac{\partial \mu p}{\partial x} + \frac{\sigma^2}{2}\frac{\partial^2 p}{\partial x^2},$$

This is a partial differential equation, and analytic solutions only exist for specific cases. Nevertheless, we can try to solve for a stationary distribution $p(x,t) = p(x)$, that is a solution independent of time. Setting $\partial p/\partial t = 0$, the Fokker-Planck equation reduces to an ordinary differential equation

$$\frac{d\mu p}{dx} = \frac{\sigma^2}{2}\frac{d^2 p}{dx^2}.$$

(We have replaced the partial derivative symbol ∂ by the ordinary derivative d.) Integrating on both sides with respect to x,

$$\mu p = \frac{\sigma^2}{2}\frac{dp}{dx} + C.$$

The integration constant $C = 0$ here because of both p and its derivative must vanish at infinity, leaving $C = 0$. This leaves a first order linear ordinary differential equation to solve for. The nonlinear function μ does not depend on p, only on the independent variable x. We find the solution by separation of variables:

$$\int \mu dx = \frac{\sigma^2}{2}\int \frac{1}{p}dp = \frac{\sigma^2}{2}\ln p + C,$$

and

$$p(x) = K\exp\Big(\frac{\int \mu(x)dx}{\sigma^2/2}\Big).$$

The constant $K = e^{2C/\sigma^2}$ is a normalisation constant so that $\int p(x)dx = 1$. The stationary solution $p(x)$ is best written in term of the potential $U(x)$. The potential $U(x) = -\int \mu(x)dx$ (the integration constant does not matter, it will be absorbed in the constant K), and the stationary solution

$$p(x) = K \exp\Big(-\frac{U(x)}{\sigma^2/2}\Big).$$

It is now quite clear how $U(x)$ is related to the stationary density: up to a constant, $U(x)$ is the log of $p(x)$.

For the neuron model defined above, we find that the stationary solution

$$p(x) = K \exp\Big(\frac{x^2 - x^4/6}{\sigma^2}\Big).$$

Exercise 7 The Fokker-Planck equation provides a way to compute the probability density of the solution of a Langevin equation. Conversely, the Langevin equation provides a way to compute the solution of the Fokker-Planck equation. For the bistable model

$$\frac{dx}{dt} = x - x^3 + \sigma\xi(t)$$

with Gaussian white noise, compute several trajectories numerically up to a fixed time t, and plot the histogram of the solution $x(t)$. Compare with the analytical stationary density.

8 Numerical schemes

This section presents widely used and simple numerical schemes for solving difference equations, ordinary differential equations, stochastic differential equations, and stochastic processes.

8.1 Numerical scheme for difference equations

For the equations $x_{t+1} = f(x_t, y_t), y_{t+1} = g(x_t, y_t)$, the pseudo-code of the numerical scheme is a simple loop:

```
x0 = ...   # define the initial value x0
y0 = ...   # define the initial value y0
T  = ...   # define the number of time step
x = x0     # x is the current value of x_t. set the current value of x
↪  to x0
y = y0     # y is the current value of y_t. set the current value of y
↪  to y0
xsol = x   # xsol is the array of x for all time points
ysol = y   # ysol is the array of y for all time points
t = 0      # set time to 0
while(t < T) {
  x = f(x,y)       # update the value of x
  y = g(x,y)       # update the value of y
  xsol.append(x) # append the solution x to xsol
  ysol.append(y) # append the solution y to ysol
```

```
  t = t + 1      # update t
}
```

8.2 Numerical scheme for ordinary differential equations (ODEs)

For the equations

$$\frac{dx}{dt} = f(x, y),$$
$$\frac{dy}{dt} = g(x_t, y_t),$$

we can use the explicit, or forward Euler method (FE method). The pseudo-code of the numerical scheme is a simple loop:

```
x0 = ...   # define the initial value x0
y0 = ...   # define the initial value y0
T  = ...   # define the final time
dt = ...   # define the numerical time step. This value
           # is the temporal resolution of the solution. It should
           # be smaller than the time scale of interest, and small
           # enough to avoid numerical instabilities. Try dt = 0.01
           # for a start.
x = x0     # x is the current value of x_t. set the current value of x
↪  to x0
y = y0     # y is the current value of y_t. set the current value of y
↪  to y0
xsol = x   # xsol is the array of x for all time points
ysol = y   # ysol is the array of y for all time points
t = 0      # set time to 0
while(t < T) {
  x = x + dt*f(x,y)       # update the value of x
  y = y + dt*g(x,y)       # update the value of y
  xsol.append(x) # append the solution x to array xsol
  ysol.append(y) # append the solution y to array ysol
  t = t + dt     # update t
}
```

8.3 Numerical scheme for stochastic differential equations (SDEs)

If we add noise terms ξ_1 and ξ_2, with strengths σ_1, σ_2, the ODEs become stochastic differential equations

$$\frac{dx}{dt} = f(x, y) + \sigma_1 \xi_1,$$
$$\frac{dy}{dt} = g(x_t, y_t)\sigma_2 \xi_1.$$

We can use the Euler-Maruyama method. For a noise strength σ, this scheme adds a Gaussian random term that has a standard deviation equal to $\sigma/\sqrt{h}$. This is the

appropriate form for a noise term that has a zero average, a fixed standard deviation and no memory (i.e. noise is uncorrelated from one moment to the other). This type of noise is called **white noise**. The pseudo-code of the numerical scheme is a simple loop:

white noise

```
x0 = ...   # define the initial value x0
y0 = ...   # define the initial value y0
T  = ...   # define the final time
dt = ...   # define the numerical time step. This value
           # is the temporal resolution of the solution. It should
           # be smaller than the time scale of interest, and small
           # enough to avoid numerical instabilities. Try dt = 0.01
           # for a start.
x = x0     # x is the current value of x_t. set the current value of x
↪  to x0
y = y0     # y is the current value of y_t. set the current value of y
↪  to y0
xsol = x   # xsol is the array of x for all time points
ysol = y   # ysol is the array of y for all time points
t = 0      # set time to 0
while(t < T) {
  x = x + dt*f(x,y) + \sqrt(dt)*sigma1*randn(0,1)   # update the value
  ↪  of x
  y = y + dt*g(x,y) + \sqrt(dt)*sigma2*randn(0,1)   # update the value
  ↪  of y
  xsol.append(x) # append the solution x to array xsol
  ysol.append(y) # append the solution y to array ysol
  t = t + dt     # update t
}
```

8.4 Numerical scheme for stochastic processes

When the dynamical variables describe species that have low counts, say $n < 100$, it can be useful to model the variable as integer numbers, and to count each addition and loss individually.

The simplest numerical scheme assume low counts and a discrete time. To implement the stochastic process, it is necessary to explicitly provide the gain and loss terms for each species in the model. This means that the difference equation needs to be written as $x_{t+1} = gain(x_t) - loss(x_t)$. The time step needs to be chosen small enough so that the gain and loss terms are much smaller than 1. In this case, over on time step, it is unlikely that many gain or loss event would occur, and it can be safely assumed that the value of x can only change by one unit.

```
x0 = ...   # define the initial value x0
T  = ...   # define the number of time step
x = x0     # x is the current value of x_t. set the current value of x
↪  to x0
xsol = x   # xsol is the array of x for all time points
```

```
t = 0       # set time to 0
while(t < T) {
  addx = (rand() < gain(x))     # test for adding one to x
  removex = (rand() < loss(x))  # test for removing one to x
  x = x + addx - removex  # update the value of x
  xsol.append(x) # append the solution x to xsol
  t = t + 1      # update t
}
```

For example, a cells in a colony divide randomly in average every 12 hours, and have a lifespan of around 48 h. During a time interval τ hours, we expect to that a colony of x cells has $\tau x/12$h divisions, and $\tau x/48$h cell deaths. We choose a time step τ small enough that these values are much less than one. If we expect x to be less than 50, then taking $\tau < 12/50$ is necessary. If we take $\tau = 0.01$h, the probability of one division is exactly the same as the number of expected divisions (so long as we ignore the rare occurences of many divisions). The means that $gain(x) = \tau x/12 = 8.33e{-}4x$ and $loss(x) = \tau/48x = 2.083e{-}4x$. The gain and loss terms are zero when $x = 0$. If x reaches zero, then it will stay there forever. In this example, cell divisions will quickly outnumber cell deaths; x will grow to values larger than 50, and the choice of the time step τ will not be appropriate anymore. Another scheme, called the stochastic simulation algorithm can be used to deal with fluctuating counts.

Stochastic Simulation Algorithm

The **Stochastic Simulation Algorithm** is a numerical algorithm that takes advantage of the discrete nature of jumps in counts. For the numerical implementation, we need to define

- the initial conditions x_0 y_0, ...,
- a list of all events (gain, loss, change of state...), numbered from 1 to r,
- for each event k, $k \in \{1, ..., r\}$, the propensity $\lambda^{(k)}$ of the event k. Propensities can have nonlinear dependence on the counts of each species, but cannot depend on the occurrence of other events; event probabilities are independent.
- the law of birth or death conditional to event k, given the counts before the event, $\Pr(\text{add } j \text{ members}) = w_k^{(N)}(j)$, $j = ..., -2, -1, 0, 1, 2, ...$

The propensites define a memoryless process. That is, the time to the next event has an exponential distribution. Neglecting all other events, the time τ_k to event k has an exponential distribution with parameter λ_k. When the r events are considered together, the time τ to the next event will be just $\min_{1 \le k \le r}\{\tau_k\}$. This is because during the interval $[t, t + \tau)$, the event probabilities are independent. At time t, the law of τ is therefore exponential with parameter λ, with

$$\lambda = \sum_{k=1}^{r} \lambda_k. \tag{1}$$

Independence also ensures that no two events can occur at the same time. When the event k is realised, the solution to the process will be $x(t + \theta) = x(t)$ for $\theta \in [0, \tau)$, and $x(t + \tau) = j$, with probability $w_k^{(x(t^-))}(j)$. The probability to choose event k is

proportional to its propensity $\lambda_k^{(N)}$. After re-normalization,

$$\Pr(\text{choose event } k) = \frac{\lambda_k}{\lambda}. \tag{2}$$

A realization of $x(t), y(t), ...$ can be computed iteratively, by advancing in time by steps of size τ.

The algorithm goes as follows

```
################## SSA algorithm #################
# input:
#   X0: array of integers of size n, initial conditions
#   t0: real value, initial time
#   T : positive real value, time interval
# output:
#   X: array of integers of size n, the solution at N(t0 + T)
X = X0;
t = t0;
while ( t < t0 + T ) {
  lambda = sum(lambda_k);
  tau = draw from exponential distribution with parameter lambda;
  k = draw from distribution with probabilities lambda_k/lambda;
  j = draw from distribution with probabilities w_k_j;
  X = X + j;
  t = t + tau;
}
```

Notes. Parameters `tau`, `k`, `j` are drawn from specific distributions that can be easily reproduced using only a standard pseudo-random number generator by using the reciprocal of the repartition function. If the random variable X has a repartition function $F(x) = \Pr(X < x)$ and U is random variable with a uniform density on the $[0, 1]$ interval, then the random variable $F^{-1}(U)$ has the same distribution as X. The time to the next event can be computed with these steps

```
### Generate an exponential covariate ###
  u = rand01(); # u drawn from a uniform distribution on [0,1]
  tau = -log(u)/lambda; # log: natural log
  return tau;
```

For discrete probability laws, the repartition function is piecewise constant, and cannot be inverted. However, the repartition functon induces a partition of the unit interval. The i-th sub-interval has the size of the i-th probability. This means that drawing a integer i with probability p_i is equivalent to identifying which sub-interval contains a random variable drawn from a uniform distribution in the unit interval.

```
### Generate a discrete covariate ###
# p: array of size I of probabilities, p[i] = prob(choose i)
#    If needed, the values of i are rescaled between 1 and I
  u = rand01();  # u drawn from a uniform distribution on [0,1]
  c = cumsum(p); # c array with right-ends of sub-intervals
  i = find_first( u < c ); # find first index i such that u < c[i]
  return i;
```

The algorithm can be adapted for distributions with an infinite number of possibe values.

9 Glossary (English/French)

Definition 1 Dynamical Model/modèle dynamique

A dynamical model is a system in which equations describe the time dependence of a set of variables in a geometrical space. Difference equations and ODEs are dynamical models when the independent variable is time.

Definition 2 Dynamical variables/variables dynamiques

Dynamical variables are variables that change over time, by opposition to constant parameters. Also called state variables.

Definition 3 Difference equation/équation aux différences

A difference equation is an equation that sets a relationship between the values of state variable at finite differences an independent variable (here the independent variable is time). It is usual to denote the value of the variable x at time t by x_t for $t = 1, ...$, to indicate that the time t takes discrete values.

Definition 4 Ordinary differential equations/équations différentielles ordinaires

An ordinary differential equation is an equation that sets a relationship between a set of variables and their derivatives with respect to continuous independent variable (here the independent variable is time).

Definition 5 Model parameter/paramètre du modèle

Model parameters are constant value contained in dynamical models.

Definition 6 Initial conditions/conditions initiales

Initial conditions are the values of the state variables at the beginning of the simulation (usually at t=0, but not necessarily). Initial conditions are needed because dynamical models only provide relations between states, not absolute values.

Definition 7 Steady state/état d'équilibre

A steady state is a special solution of a dynamical system such that, if the initial conditions are on the steady state, the solution remains on the steady. For an ODE $\bar{x}$ is a steady state if $d\bar{x}/dt = 0$. For a difference equation $x_{t+1} = f(x_t)$, $\bar{x}$ is a steady

state if $\bar{x} = f(\bar{x})$. A steady state is stable if solutions with initial conditions close to the steady state will stay close to the steady state.

Definition 8 Oscillations/oscillations

Oscillations is a type of non-constant solution where at least one of the variables comes back through a certain value regularly, for any amount of time.

Definition 9 Bistability/bistabilité

Bistability is a property of a system where there exists two stable states. Which stable state will attract solution depends on the initial condition. Switch between stable states can be obtained by perturbing the system.

Definition 10 Motifs/motifs

Motifs are small blocks of regulation that can be used to distill all the complexity of biology into simple parametric term.

10 References

- RM May, Simple mathematical models with very complicated dynamics (1976) Nature 261:459–467 - Lotka-Volterra as individual-based model <http://www.ahahah.eu/trucs/pp/>
- An Introduction to R <https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf>
- E Klipp et al., Systems Biology, Wiley-VCH Weinheim 2009 - U Alon, An introduction to Systems Biology: Design Principles of Biological Circuits, Chapman & Hall/CRC 2007

Index