

2. Introduction to modelling: Exponential and logistic growth

Last time

- Data analysis of COVID-19 data
- Visualizations

Goals for today

- Simple **models** of dynamics of infection
- Exponential and logistic growth

- **Simulate** to produce **data**
- Visualize for understanding

Dynamics

- Many questions in sciences, engineering, economics... concern **dynamics**:

*how system changes (**evolves**) over time*

Dynamics

- Many questions in sciences, engineering, economics... concern **dynamics**:

*how system changes (**evolves**) over time*

- Examples:
 - stock market
 - chemical reactions
 - noise in jet engine
 - interactions of genes
 - motion of galaxy

Modelling

- What is a **model**?

Mathematical / computational description of world

Modelling

- What is a **model**?

Mathematical / computational description of world

- Full system (= world) is too complicated

Modelling II

- What is a **model**?

*Mathematical / computational description of **small sub-piece** of world*

Modelling II

- What is a **model**?

*Mathematical / computational description of **small sub-piece** of world*

- Isolate effects of interest

Modelling II

- What is a **model**?

*Mathematical / computational description of **small sub-piece** of world*

- Isolate effects of interest
- Start simple, understand behaviour
- Make more complicated (add additional effects), understand how results change
- Repeat!

Modelling infections

- Let's try to model infection spreading through campus

Modelling infections

- Let's try to model infection spreading through campus
- Need to make assumptions to start modelling process
- Assumptions must be **explicit**

- **Suppose** no influence from outside

Modelling infections

- Let's try to model infection spreading through campus
- Need to make assumptions to start modelling process
- Assumptions must be **explicit**

- **Suppose** no influence from outside
- **Suppose** no births or deaths

Modelling infections

- Let's try to model infection spreading through campus
- Need to make assumptions to start modelling process
- Assumptions must be **explicit**

- **Suppose** no influence from outside
- **Suppose** no births or deaths
- **Suppose** infection is transmitted by proximity

Modelling infections

- Let's try to model infection spreading through campus
- Need to make assumptions to start modelling process
- Assumptions must be **explicit**

- **Suppose** no influence from outside
- **Suppose** no births or deaths
- **Suppose** infection is transmitted by proximity
- **Suppose** stay infected once infected

Mathematical model

- **Mathematical model:** math. description of assumptions

Mathematical model

- **Mathematical model:** math. description of assumptions
- Choose a model in **discrete time**: $n = \text{days}$

Mathematical model

- **Mathematical model**: math. description of assumptions
- Choose a model in **discrete time**: n = days
- Notation: I_n is number of **infectious** people on n th day
- S_n is number of **susceptible** people on n th day

Mathematical model

- **Mathematical model**: math. description of assumptions
- Choose a model in **discrete time**: $n = \text{days}$
- Notation: I_n is number of **infectious** people on n th day
- S_n is number of **susceptible** people on n th day

- **Suppose** that everybody is either susceptible or infected:

$$S_n + I_n = N$$

and total population N is constant

Initial conditions

- Need to specify initial values at time $n = 0$
- E.g. $I_0 = 1$ and $S_0 = N - 1$

Initial conditions

- Need to specify initial values at time $n = 0$
- E.g. $I_0 = 1$ and $S_0 = N - 1$
- We will need hypotheses on dynamics of infection:
- Specify how I_{n+1} depends on I_0, \dots, I_n

Initial conditions

- Need to specify initial values at time $n = 0$
- E.g. $I_0 = 1$ and $S_0 = N - 1$
- We will need hypotheses on dynamics of infection:
- Specify how I_{n+1} depends on I_0, \dots, I_n
- E.g. $I_{n+1} = f(I_n)$ **deterministic** model
- Or $I_{n+1} = f(I_n) + \xi$ **stochastic** model with random fluctuations ξ

Output of model

- The output of model will be a **time series**
- i.e. vectors of values of S_n and I_n at times $n = 0, \dots, T$
- Up to some final time T

Output of model

- The output of model will be a **time series**
 - i.e. vectors of values of S_n and I_n at times $n = 0, \dots, T$
 - Up to some final time T
-
- Goal: Obtain time series that “mimics” real-world data
 - Goal: Understand which effects must be included, which not

Exponential growth

- Let's start with one of simplest models:
- **Suppose** Each individual infects c others each day

Exponential growth

- Let's start with one of simplest models:
- **Suppose** Each individual infects c others each day

- So

$$I_{n+1} = I_n + cI_n$$

- So

$$I_{n+1} = \lambda I_n,$$

where $\lambda := 1 + c$ is **growth rate**

Exponential growth II

- After day 1: $I_1 = \lambda \cdot I_0$
- After step 2: $I_2 = \lambda \cdot I_1$

Exponential growth II

- After day 1: $I_1 = \lambda \cdot I_0$
- After step 2: $I_2 = \lambda \cdot I_1$

- How *map* model \rightarrow **computational experiment**?
- Create a small “world” inside computer
- Start: represent **initial data** as variables

Exponential growth II

- After day 1: $I_1 = \lambda \cdot I_0$
- After step 2: $I_2 = \lambda \cdot I_1$

- How *map* model \rightarrow **computational experiment**?
- Create a small “world” inside computer
- Start: represent **initial data** as variables

- Will need a way to store all the I_n

Computational thinking III

- Solution: math I_n corresponds to computation $\mathcal{I}[n]$, where $[n]$ means **index**

Computational thinking III

- Solution: math I_n corresponds to computation $I[n]$, where $[n]$ means **index**
- Need a suitable **data structure** to store data

Computational thinking III

- Solution: math I_n corresponds to computation $I[n]$, where $[n]$ means **index**
- Need a suitable **data structure** to store data
- Options: Dictionary, **vector**

Creating data over time

- Often create / read data and store incrementally, e.g. when don't know how much storage we need
- Need a data structure that knows how to automatically grow
- `Dict` allows this
- Common alternative if ordered data, e.g. in time: create empty `Vector` and add data to it
- `push!(vv, a)`; cf. `vv.append(a)` in Python.
- `!` is convention in Julia: function that *modifies its argument*

- Argument must be *mutable*, e.g. Vector
- Disadvantage: loses explicit index n ; implicit as “the index of the last element that was added”
- Advantage: *Think in a new way*, less mathematically and more computationally:
 - “current value of x ”
 - “next value of x that will be produced”
- Suggestion: Call vector x_s since stores many values of x ; allows using x for current value:

```
x = x0          # current value
xs = [x0]       # initialize Vector with initial value

for n in 1:N
    next_x =  $\lambda$  * x
    push!(xs, next_x)

    x = next_x
end
```

- Key step: *update current value* at end of loop
- New value will be used as input in next loop iteration.
- Mathematically: often write x' for next value: $x' = \lambda x$
- Julia: $x' = \lambda * x$ – write as `\prime<TAB>`
- **Julia looks like math**

Anonymous functions

- Math: $f_{\lambda}(x) = \lambda x$
- Function of one variable, x , with one parameter, λ
- Julia: “the function that takes x to $2x$ ” is written
 $x \rightarrow 2x$
- Called an **anonymous function** since it has no name.

- Now can write `julia f(λ) = x -> λ * x`
- So `f(3)` *is an (anonymous) function*

Exploring data: Visualization

- Now want to explore *how data behaves*.
- “Exploratory data analysis”: visualize by plotting
- We choose `Plots.jl` package (out of several options).
- Install package (one time) with

```
using Pkg  
Pkg.add("Plots")
```

or

```
]add Plots
```

- `Pkg` is *built-in* Julia package manager.
- In REPL, `]` enters package mode: prompt `(v1.2) pkg>`.

Plotting

- Once installed, load in *each* session:

```
using Plots
```

- Can now plot data as *points* (scatter) or lines (plot):

```
scatter(x)
```

- If x coordinates not given, Julia plots the data against time.

Automate: put it into a function

- Have generated data for specific values of λ and x_0 .
- How is dynamics *affected* by x_0 and λ ?
- Must change them and recalculate.
- *Never* copy, paste and modify values by hand!
- Instead, introduce *abstraction*: **create a function**

```
"""
```

```
Simulate growth with rate ` $\lambda$ `, initial value `x0` and time `N`.
```

```
"""
```

```
function growth( $\lambda$ , x0, N)
```

```
    x = zeros(N+1)
```

```
    x[1] = x0
```

```
    for n in 1:N
```

```
        x[n+1] =  $\lambda$  * x[n]
```

```
    end
```

```
    return x
```

```
end
```

- Docstrings (within `"""`) are placed *above* function body.
- Can now easily experiment with different values by executing the function:

```
growth(1.2, 10.0, 10)
```

```
growth(2.0, 10.0, 10)
```

- **Exercise:** What happens if `x0` and `\lambda` are integers instead of `Float64`?
- Which argument is which? It's difficult to remember.
- Use **keyword arguments** (NB: semicolon, `;`):

```
growth(;  $\lambda$ =1.1, x0=20.0, N=10) = growth( $\lambda$ , x0, N)
```

- Call with

```
growth(N=20)
```

```
growth(N=20, x0=1.0)
```

- Note that we have *added a method* to the function `growth`:

```
methods(growth)
```

Automating plotting

- It's tempting to add plotting to the data generation.
- But we shouldn't do so:
always separate data generation and plotting
- Reasons for this:
 - Data generation is slow; may want to plot data in different ways.
 - Plotting is slow or inconvenient – may not want to plot, e.g. if running non-interactively
- **Exercise:** Write a function `plot_growth` that takes the same parameters, generates the data and then plots.

Interactive visualizations

- It's still clumsy to modify values and re-plot by hand. Can we do this more intuitively and *interactively*?
- Interact.jl package: interactive widgets
- Can be used inside Juno and Jupyter notebook
- Install and load Interact.jl. Then try

```
@manipulate for i in 1:10  
    i^2  
end
```

- *Looks like* standard `for` loop, with i^2 calculated at each iteration.
- But it is doing something different
- `@manipulate` is a **macro** – think of as a “super-function”
- Macro takes piece of Julia code and modifies it
- Returns new piece of Julia code, which is executed *in place of* old code; Julia never sees old code, only modified code.
- NB: In normal `for` would need to output; in a `@manipulate` the last calculated value is displayed automatically

- Can use `@macroexpand` to see new code. Here it generates the widgets.
- Can manipulate more than one variable at a time:

```
using Interact
```

```
@manipulate for a in 0:10, b in 1:11
```

```
    HTML( (a, b) )    # use HTML representation of output  
end
```

- **Exercise:** Use `@manipulate` to visualize the dynamics of the simple growth model as λ and x_0 are varied. What are suitable (physically realistic) bounds for those variables?

There is a key *critical value* of λ where the *dynamics changes qualitatively*. What is it? This is called a **bifurcation**.

Nonlinear dynamics

- General dynamics with single-step function f :

$$x_{n+1} = f(x_n)$$

- **Discrete-time dynamical system**
- Apply f to previous output at each step.
- Example: $f = \cos$.
- “Repeatedly press the cos key on your calculator”. What happens?
- **Exercise:** Implement this.

Fixed points

- Behaviour is *completely different* from previous: iterates **converge** to a **fixed point**:

$$x_n \rightarrow x^* \text{ as } n \rightarrow \infty$$

- **Fixed point**: value that *does not change* when f is applied:

$$x^* = f(x^*)$$

- **Transcendental equation**: *impossible* to find explicit form for solution.
- But iterative method successfully solves this equation (with certain precision).

Rate of convergence

- How “good” is this method?
- Measure *distance* of x_n from x^* , i.e. $\delta_n := |x_n - x^*|$
- **Exercise:** Implement this.
- Find that it converges “quickly”.
- How characterize *rate* of convergence, i.e. how *fast* does δ_n decrease as function of n ?
- Plot data differently: **log scale**
- `yscale=:log` in `Plots.jl`

Stability analysis

- What can we say about $\delta_n := |x_n - x^*|$ analytically?
- For large n , we know x_n is *close to* x^* , so

$$x_{n+1} = x^* + \delta_{n+1} = f(x_n) = f(x^* + \delta_n) \quad (1)$$

$$\simeq f(x^*) + \delta_n f'(x^*), \quad (2)$$

- Approximately satisfies *linear* dynamics:

$$\delta_{n+1} = \lambda \delta_n,$$

with $\lambda = f'(x^*)$ (constant).

- *We already understand this!*
- **Computing and mathematics often consist of trying to reduce a new problem to a problem you already know how to solve!**
- Behaviour of nonlinear systems near fixed points can often be reduced to analyzing the *linearized* system.

Review

- Modelling allows us to study isolated phenomenon in the computer
- Produce data incrementally and store it in a `Vector` using `push!`
- Examine its behaviour by plotting
- Make it interactive to quickly scan possible behaviours
- Reduce a problem to one you already know how to solve if you can!
- Some Julian tips:
 - Return from function, don't print
 - Types are important, but don't worry too much right now
 - Multiple dispatch: Different *methods* of same function for related functionality