Basic computational analysis of a mathematical model

01 – Using functions

Julien Arino
University of Manitoba
julien.arino@umanitoba.ca

Fall 2024

The University of Manitoba campuses are located on original lands of Anishinaabeg, Ininew, Anisininew, Dakota and Dene peoples, and on the National Homeland of the Red River Métis.
We respect the Treaties that were made on these territories, we acknowledge the harms and mistakes of the past, and we dedicate ourselves to move forward in partnership with Indigenous communities in a spirit of Reconciliation and collaboration.

Using functions ?

To caricature, suppose we have an IVP $x' = f(x, \mu)$, $x(t_0) = x_0$, where μ are parameters

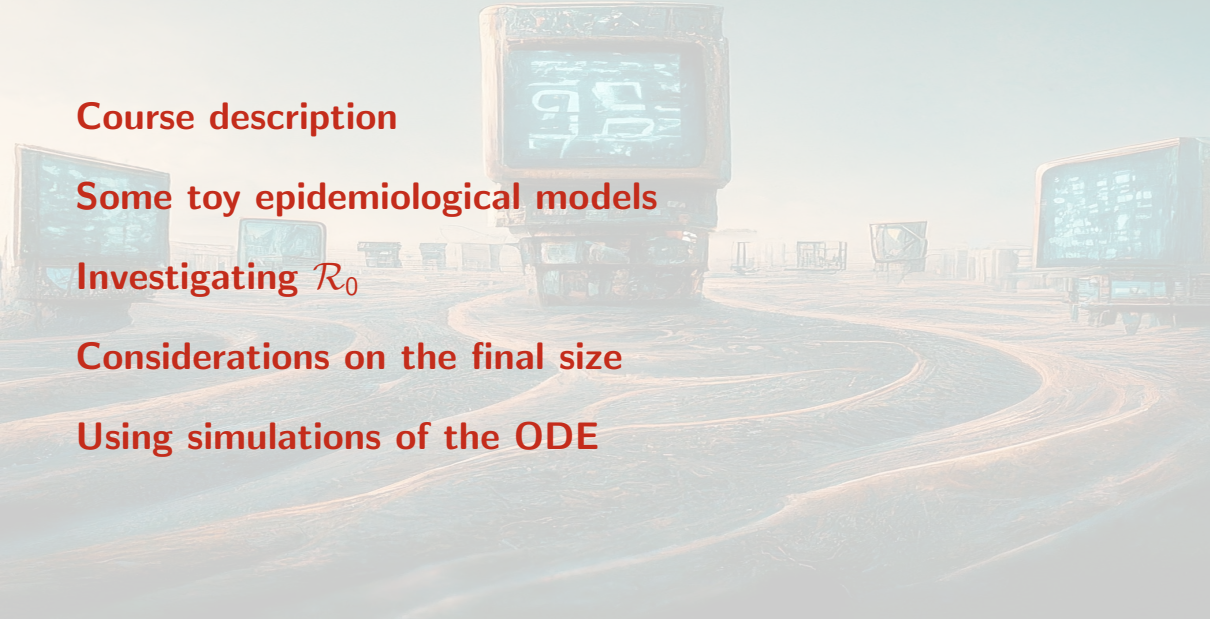
We can find functional expressions telling us that, say, if $\Psi(\mu) < 0$, then the system has a certain behaviour and that this changes when $\Psi(\mu) > 0$

This type of functional relationship between model parameters and behaviour is what we are interested in here

There are cases also where we need to numerically solve the IVP to obtain, say, the value of an equilibrium $x^\star$, because there is no closed-form formula giving $x^\star$ as a function of μ

This type of work *uses simulations* and is the second set of slides in this lecture

Outline



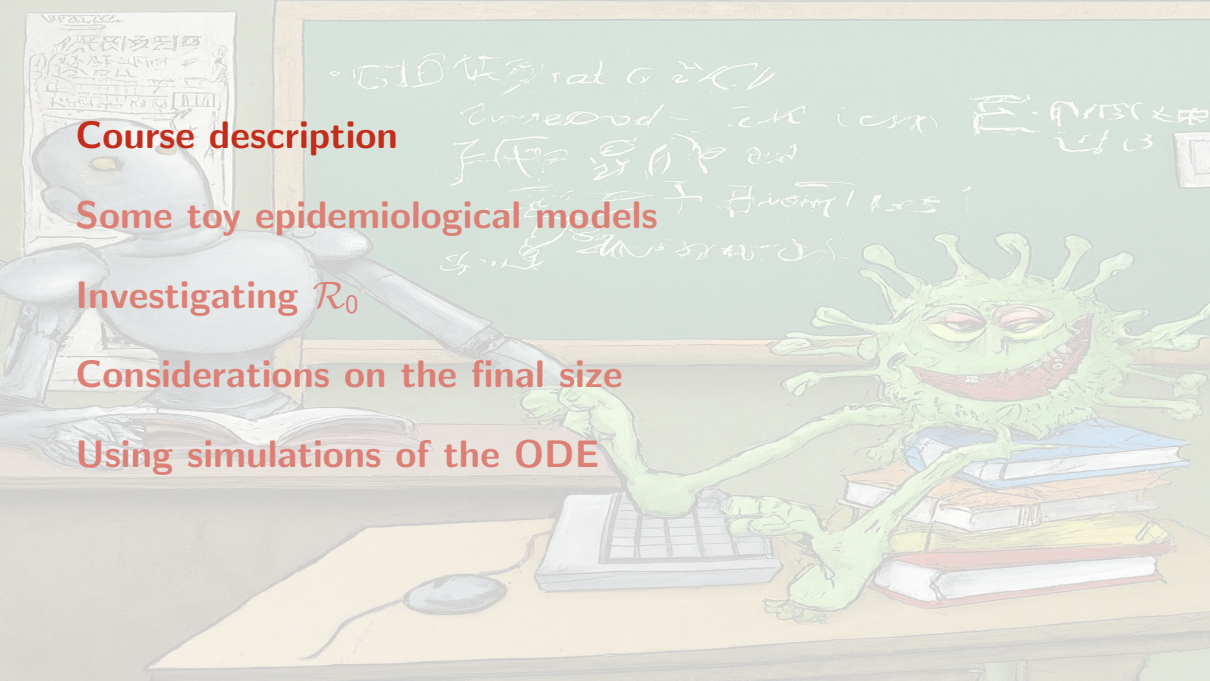
Course description

Some toy epidemiological models

Investigating $\mathcal{R}_0$

Considerations on the final size

Using simulations of the ODE



Course description

Some toy epidemiological models

Investigating $\mathcal{R}_0$

Considerations on the final size

Using simulations of the ODE

This is not a *vignette*!

The vignettes in this repo illustrate how to use R to consider several problems that a modeller is faced with

This is somewhat orthogonal: it takes the information in several of these vignettes and integrates it in the perspective of computational analysis

I am including it in this repo, however, because of the non-empty intersection between the two: this is R and is related to modelling

Course objectives

The objective of the course is to introduce notions used in the computational analysis of a mathematical model

This is not an exhaustive course on the subject, but rather an introduction to some basic concepts

See this as a minimal toolkit to get you started

If you are a graduate student of mine, take this as a hint: this is the type of stuff that I expect to see in your work

Note that I am not doing my job properly: I am skipping a very important part of any computational analysis by not doing a proper *return to biology*. This is an essential part of any computational analysis but is outside the scope of these slides

Course slides

These slides are produced using `knitr` in `Rnw`, i.e., `R` within LaTeX, to illustrate some of the concepts

To generate them, you need to have `R` and LaTeX installed on your computer and, preferably, `RStudio`

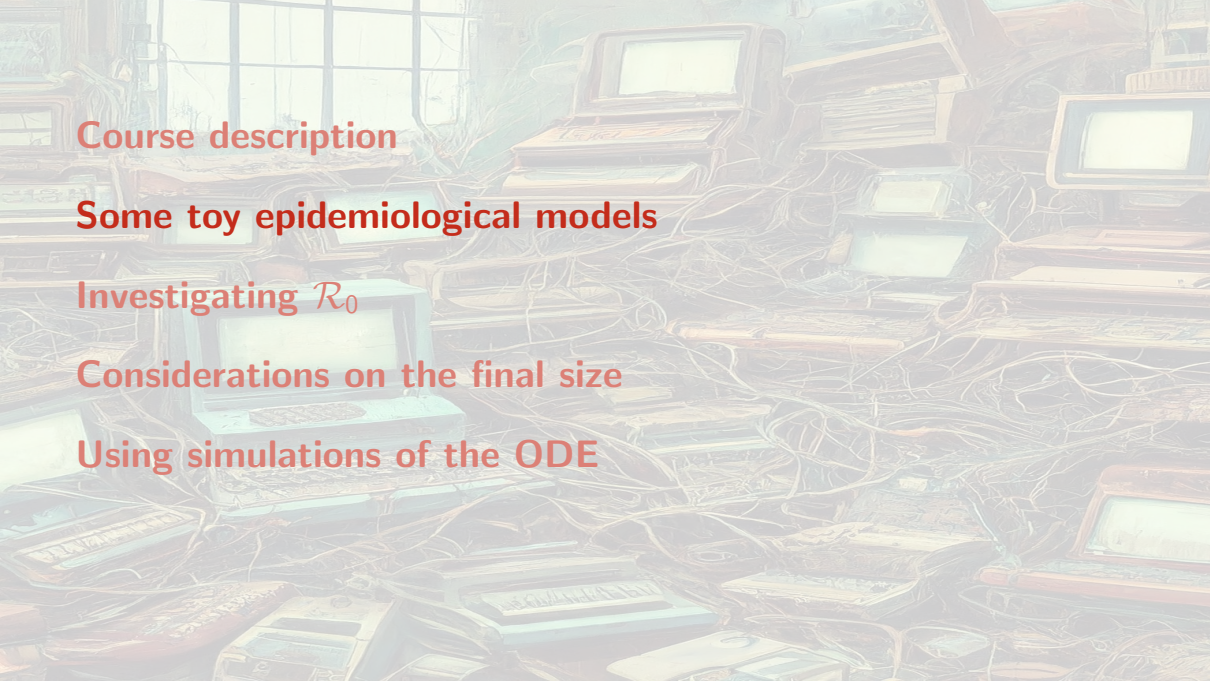
The code ensures that all the required packages are installed

Code chunks

In `Rnw` files, code chunks are delimited by `<<>>=` and `@`

Code chunks are highlighted in the RStudio editor, so you should be able to identify them easily

I also generate an `R` file with all the code (`basic-computational-analysis.Rnw`). It is in the CODE directory of the repo. See the last slide for details

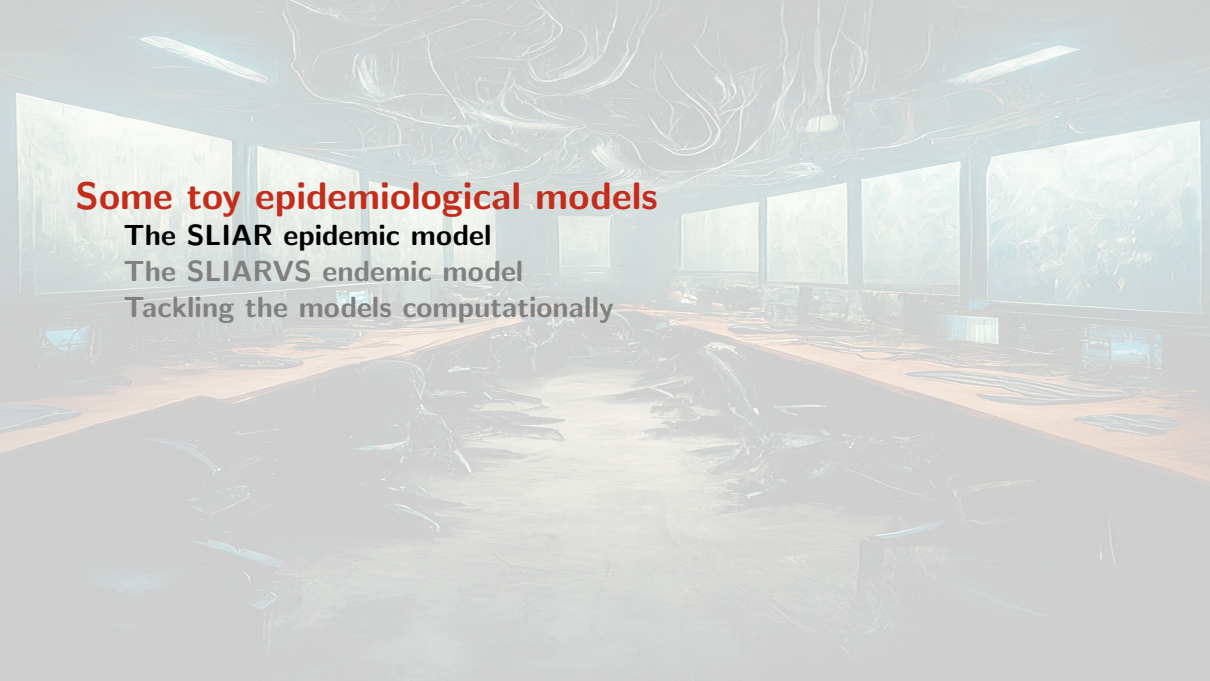

Course description
Some toy epidemiological models
Investigating $\mathcal{R}_0$
Considerations on the final size
Using simulations of the ODE

The toy models

I illustrate the methods using two toy models

Both are epidemiological models I have worked on

Some of the computational analysis is common to both models, some is specific



Some toy epidemiological models

- **The SLIAR epidemic model**
- The SLIARVS endemic model
- Tackling the models computationally

The SLIAR epidemic model

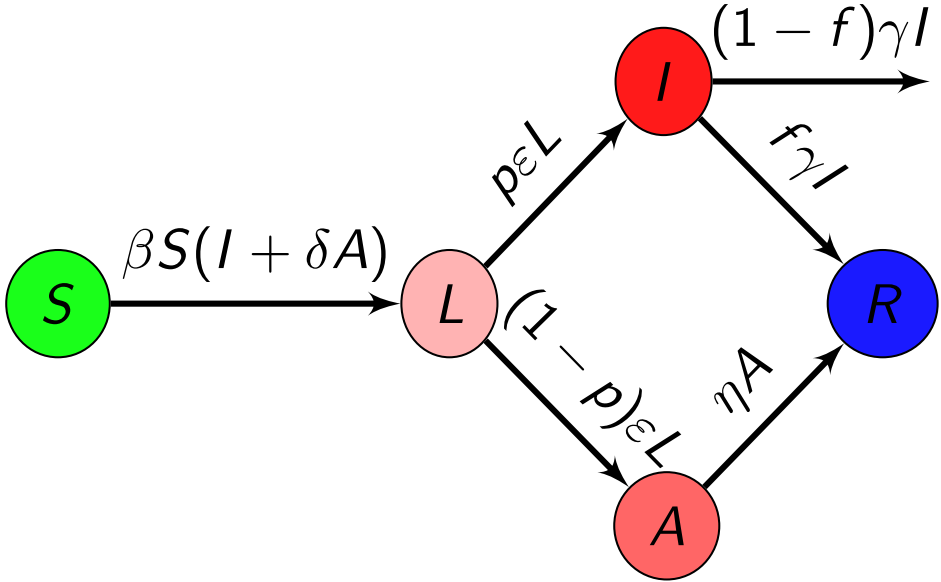
Kermack-McKendrick SIR (susceptible-infectious-removed) is a little too simple for many diseases:

- No incubation period
- A lot of infectious diseases (in particular respiratory) have mild and less mild forms depending on the patient

$\Longrightarrow$ model with SIR but also L(atent) and (A)symptomatic individuals, in which I are now symptomatic individuals

Arino, Brauer, PvdD, Watmough & Wu, Simple models for containment of a pandemic, *Journal of the Royal Society Interface* (2006)

The SLIAR epidemic model – Flow diagram



The SLIAR epidemic model – Equations

$$S' = -\beta S(I + \delta A) \tag{1a}$$
$$L' = \beta S(I + \delta A) - \varepsilon L \tag{1b}$$
$$I' = p\varepsilon L - \gamma I \tag{1c}$$
$$A' = p\varepsilon L - \eta A \tag{1d}$$
$$R' = f\gamma I + \eta A \tag{1e}$$

The SLIAR epidemic model – Behaviour

It's always a good idea to not barge into the computational analysis of a model without an understanding of its behaviour

This is an **epidemic** model: all its solutions go to *a* disease-free equilibrium

There is a **basic reproduction number** $\mathcal{R}_0$ (next slide) that determines whether the disease will spread or not. If $\mathcal{R}_0 < 1$, the disease dies out without first going through an outbreakl if $\mathcal{R}_0 > 1$, the disease goes through an outbreak, then dies out

As with many epidemic models, we can also caracterise the so-called **final size** of the epidemic

The SLIAR epidemic model – Basic reproduction number & Final size

We find the basic reproduction number

$$\mathcal{R}_0 = \beta \left(\frac{p}{\gamma} + \frac{\delta(1-p)}{\eta} \right) S_0 \tag{2}$$

The final size relation takes the form

$$S_0(\ln S_0 - \ln S_\infty) = \mathcal{R}_0(S_0 - S_\infty) + \frac{\mathcal{R}_0 I_0}{\rho} \tag{3}$$

with

$$\rho = \frac{p}{\gamma} + \frac{\delta(1-p)}{\eta}$$

Some toy epidemiological models

- The SLIAR epidemic model
- **The SLIARVS endemic model**
- Tackling the models computationally

The SLIARVS endemic model

The SLIAR model is an epidemic model: all its solutions go to *a* disease-free equilibrium

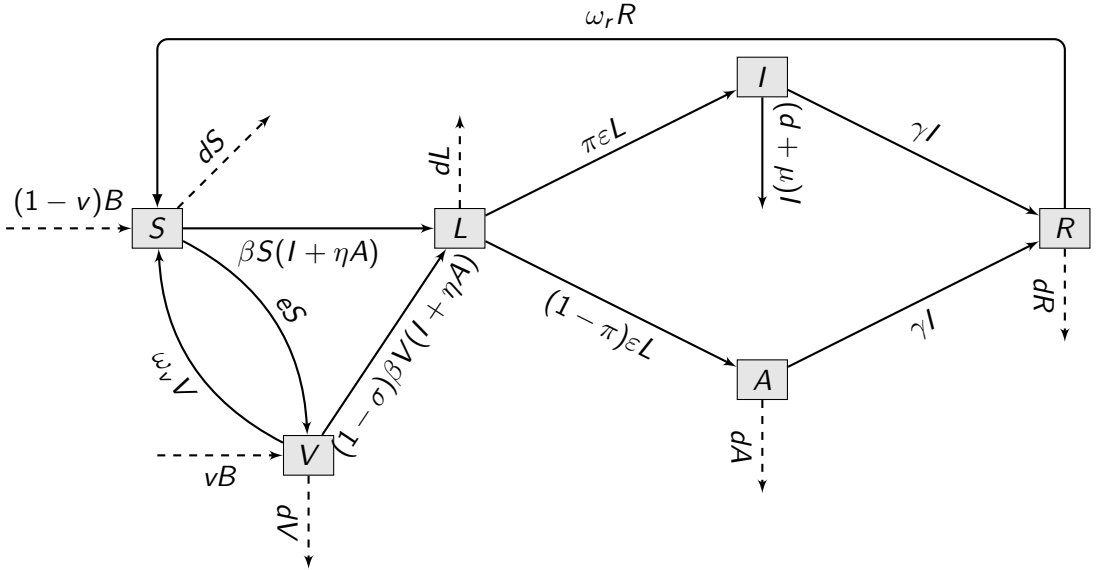
Here we consider a complexification of the SLIAR epidemic model:

- Add vital dynamics (births and deaths), a.k.a. demography
- Add a vaccination compartment V, with imperfect and waning vaccine
- Interpret R as *recovered* (and immune) individuals instead of *removed*
- Add loss of immunity (waning immunity)

This makes the model **endemic**: it has an endemic equilibrium (EEP) and (roughly) $\mathcal{R}_0$ determines if the system goes to the DFE or the EEP

Arino & Milliken, Bistability in deterministic and stochastic SLIAR-type models with imperfect and waning vaccine protection, *Journal of Mathematical Biology* (2022)

The SLIARVS endemic model – Flow diagram



The SLIARVS endemic model – Equations

$$S' = (1-v)B + \omega_v V + \omega_r R - \beta S(I + \eta A) - (e+d)S \tag{4a}$$
$$V' = vB + eS - (1-\sigma)\beta V(I + \eta A) - (\omega_v + d)V \tag{4b}$$
$$L' = \beta(S + (1-\sigma)V)(I + \eta A) - (\varepsilon + d)L \tag{4c}$$
$$I' = \pi\varepsilon L - (\gamma + \mu + d)I \tag{4d}$$
$$A' = (1-\pi)\varepsilon L - (\gamma + d)A \tag{4e}$$
$$R' = \gamma(A + I) - (\omega_r + d)R \tag{4f}$$

The SLIARVS endemic model – DFE

In (4) without equation for V' and with $v = e = \omega_v = 0$, disease-free equilibrium (DFE) has $\bar{S}_0 = B/d$

DFE of full (4) is $E_0 = (S_0, V_0, 0, 0, 0, 0)$, where

$$S_0 = \frac{(1-v)d + \omega_v}{e + \omega_v + d} \frac{B}{d} \quad \text{and} \quad V_0 = \frac{vd + e}{e + \omega_v + d} \frac{B}{d} \tag{5}$$

The SLIARVS endemic model – Reproduction numbers

With the combination parameter

$$\lambda = \beta\varepsilon\frac{(\gamma+\mu+d)\eta(1-\pi)+\pi(\gamma+d)}{(\gamma+d)(\gamma+\mu+d)} \tag{6}$$

we have

$$\mathcal{R}_0 = \frac{\lambda}{\varepsilon+d}\bar{S}_0 \tag{7}$$

$$\mathcal{R}_v = \frac{\lambda}{\varepsilon+d}(S_0+(1-\sigma)V_0) \tag{8}$$

Some toy epidemiological models

- The SLIAR epidemic model
- The SLIARVS endemic model
- **Tackling the models computationally**

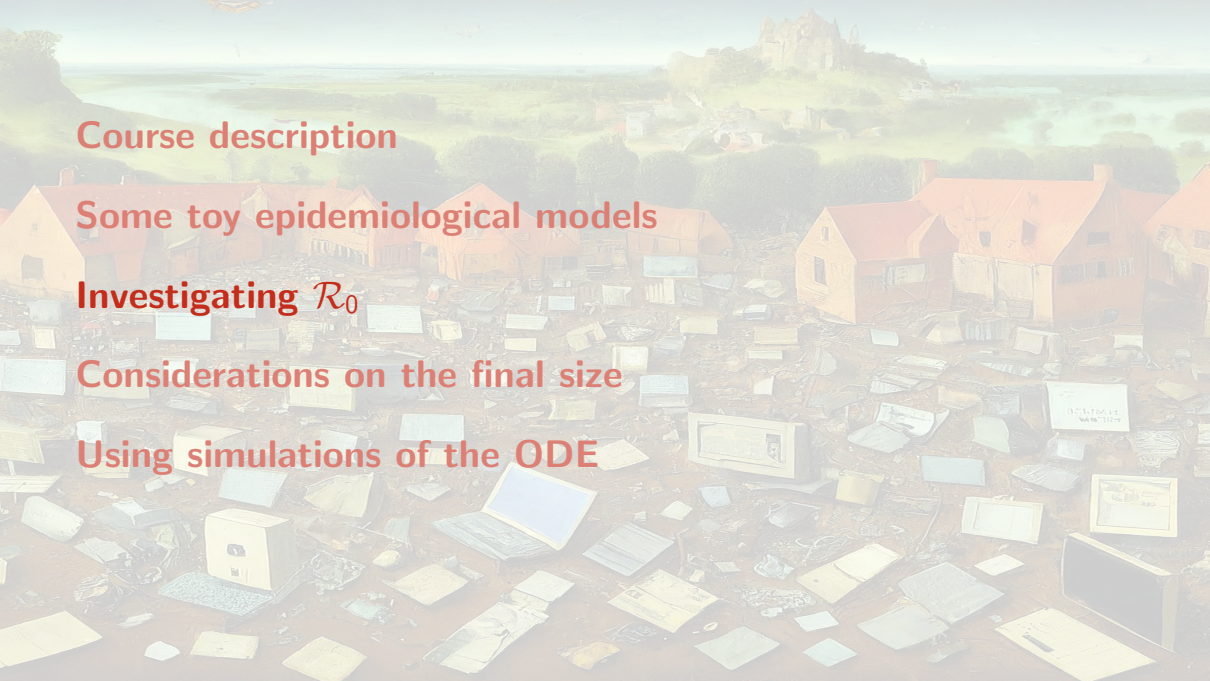
The models are systems of ODEs, we could simulate and show the result, but who cares if we just show the behaviour?

- System (1) goes to the DFE every time, after undergoing (or not) an epidemic depending on the value of $\mathcal{R}_0$
- System (4) goes to the DFE or the EEP, depending on the value of $\mathcal{R}_0$

Booooriiiing!

We can still do things with the solutions, but we'll have to make it worthwhile...

To get more insight into the model, we can use the formula for the reproduction numbers (2), (7) and (8), the final size relation (3) and other quantities to study the model: this will show how these important quantities depend on parameters



Course description

Some toy epidemiological models

Investigating $\mathcal{R}_0$

Considerations on the final size

Using simulations of the ODE

Make a function for $\mathcal{R}_0$

We have

$$\mathcal{R}_0 = \beta \left(\frac{p}{\gamma} + \frac{\delta(1-p)}{\eta} \right) S_0 \tag{2}$$

So write a function, where p is a list of parameters (including the initial number S0 of susceptible individuals)

```
R0_SLIAR = function(p) {
  OUT = p$beta*(p$p/p$gamma+p$delta*(1-p$p)/p$eta)*p$S0
  return(OUT)
}
```

Another way to write the function

```
R0_SLIAR_2 = function(p) {
  with(as.list(p), {
       OUT = beta*(p/gamma+delta*(1-p)/eta)*S0
       return(OUT)
  })
}
```

This can be useful if you have a lot of parameters and want to avoid writing `p$` all the time. However, note that the `return` statement must be within the `with` statement

What we can do with this function

Of course, we can check what $\mathcal{R}_0$ is at a given point in parameter space, but let us go further

We can plot $\mathcal{R}_0$ as a function of one parameter, *ceteris paribus* (all other things remaining the same), to see how it behaves

We can plot $\mathcal{R}_0$ as a function of two parameters, to see how it behaves in a plane

We can conduct a "full fledged" sensitivity analysis, by plotting $\mathcal{R}_0$ as a function of all parameters

Investigating $\mathcal{R}_0$

- $\mathcal{R}_0$ as a function of a single parameter
- $\mathcal{R}_0$ as a function of two parameters
- Sensitivity analysis of $\mathcal{R}_0$
- Now repeat everything for the SLIARVS model

The parameter list

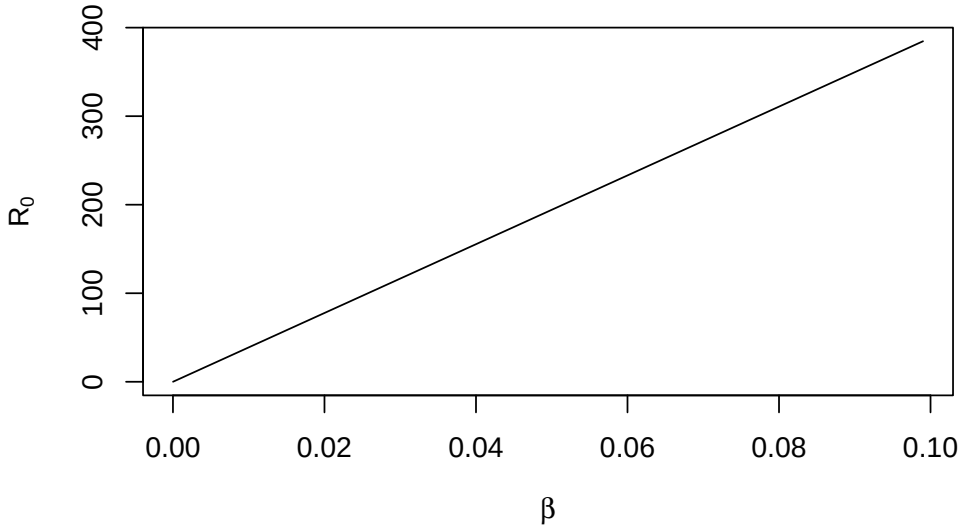
Before we begin, we have to set default values for the parameters

```
p = list()
p$kappa = 1/3 # incubation
p$p = 1/3 # fraction going to I vs A
p$beta = 1 # contact parameter
p$delta = 1/3 # infectivity differential A
p$gamma = 1/7 # recovery rate I
p$eta = 1/7 # recovery rate A
p$f = 0.5 # fraction symptomatic not dying
p$S0 = 999 # Initial S
```

Plotting $\mathcal{R}_0$ as a function of one parameter

We can plot $\mathcal{R}_0$ as a function of one parameter, say β

```
beta = seq(1e-5, 1e-1, by=1e-3)
R0_values = c()
for (i in 1:length(beta)) {
  p$beta = beta[i]
  R0_values = c(R0_values, R0_SLIAR(p))
}
plot(beta, R0_values,
     type="l", xlab=expression(beta), ylab=expression(R[0]))
```


0
100
200
300
400
R_0
0.00
0.02
0.04
0.06
0.08
0.10
β

An old trick

Clearly, we're getting super large values of $\mathcal{R}_0$. Finding the "right" values for β is hard. One way to solve the problem is to solve (2) as a function of $\mathcal{R}_0$ for β and set an interval for $\mathcal{R}_0$ instead of one for β

$$\mathcal{R}_0 = \beta \left(\frac{p}{\gamma} + \frac{\delta(1-p)}{\eta} \right) S_0 \tag{2}$$

So we can write

$$\beta = \frac{\mathcal{R}_0}{\left(\frac{p}{\gamma} + \frac{\delta(1-p)}{\eta} \right) S_0} \tag{9}$$

```
beta_SLIAR = function(R0, p) {
  return(R0/((p$p/p$gamma+p$delta*(1-p$p)/p$eta)*p$S0))
}
```

But there's no point plotting $\mathcal{R}_0$ as a function of $\mathcal{R}_0$...

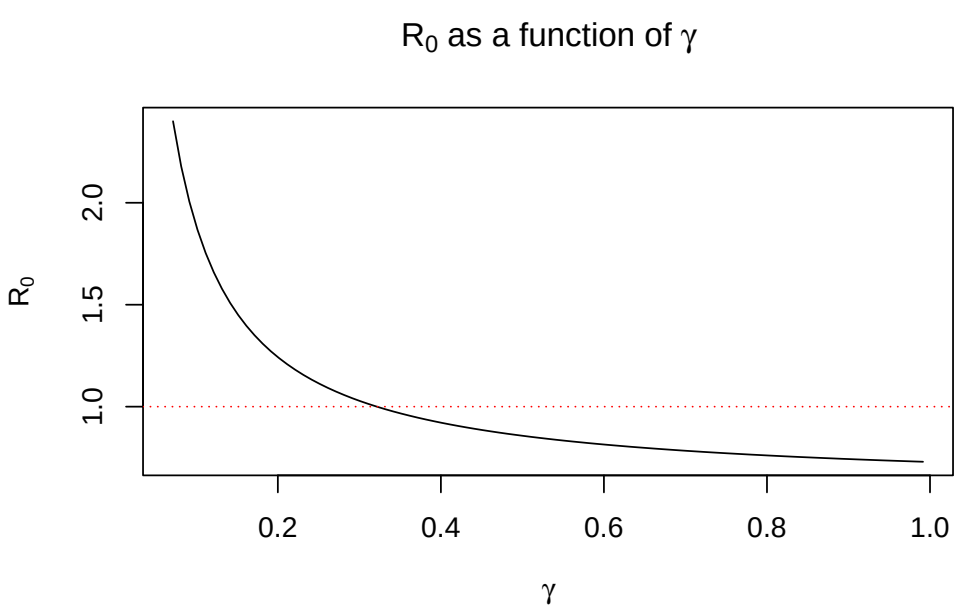
This function will nonetheless be useful later

For now, let's vary another parameter. For this

- choose a sensible value of β using (9), all other parameters being fixed
- vary the parameter of interest
- compute $\mathcal{R}_0$ for each value of the parameter of interest

Plotting $\mathcal{R}_0$ as a function of another parameter

```
p$beta = beta_SLIAR(R0=1.5, p)
gamma = seq(from=1/14, to=1, by=0.01)
R0_values = c()
for (i in 1:length(gamma)) {
  p$gamma = gamma[i]
  R0_values = c(R0_values, R0_SLIAR(p))
}
plot(gamma, R0_values,
     type="l", xlab=expression(gamma), ylab=expression(R[0]),
     main = TeX("$R_0$ as a function of $\\gamma$"))
# Throw in a line at R0=1 for good measure
abline(h=1, col="red", lty = 3)
```


R_0 as a function of γ
R_0
2.0
1.5
1.0
0.2
0.4
0.6
0.8
1.0
γ

Making axes that make sense

A substantial part of a modeller's job is to communicate with non-modellers (e.g., your favourite biologist/public health/medical person)

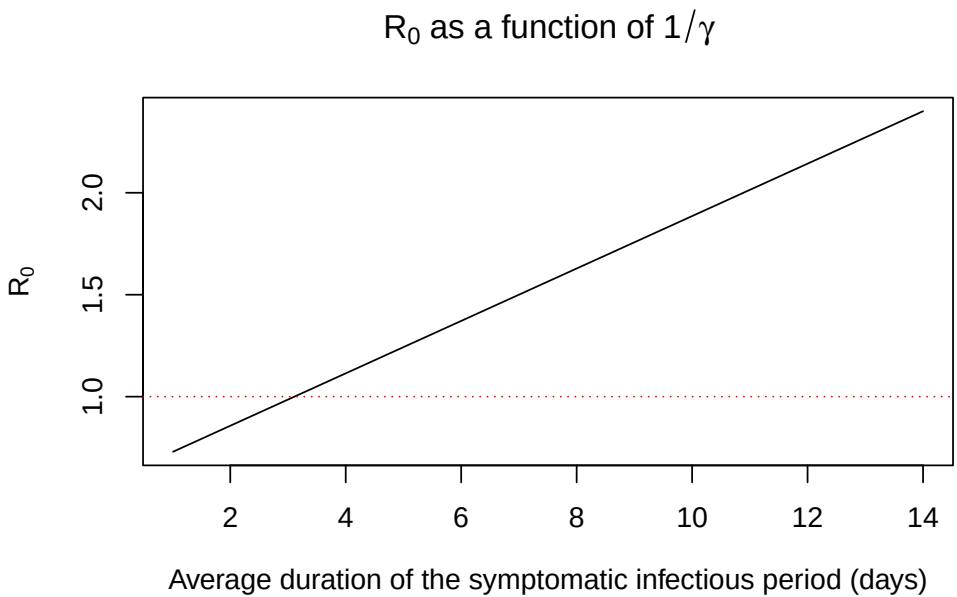
It is important to use axes that make sense to them: they are your *end users*

Remember: numerics is here to help you bridge the gap between the model and the real world

Reworking the previous figure

γ is the recovery rate of symptomatic individuals, so the average duration of the symptomatic infectious period is $1/\gamma$

```
inv_gamma = 1/gamma
plot(inv_gamma, R0_values,
     type="l",
     xlab=TeX("Average duration of the symptomatic infectious period (days)
     ylab=expression(R[0]),
     main = TeX("$R_0$ as a function of $1/\\gamma$"))
abline(h=1, col="red", lty = 3)
```


R_0 as a function of $1/\gamma$
R_0
1.0
1.5
2.0
2
4
6
8
10
12
14
Average duration of the symptomatic infectious period (days)

Looks way less cool

Indeed, but it is way more informative

And expected ...

$$\mathcal{R}_0 = \beta \left(\frac{p}{\gamma} + \frac{\delta(1-p)}{\eta} \right) S_0 \tag{2}$$

So as a function of $1/\gamma$, $\mathcal{R}_0$ is an affine function

Make it even easier to read

A good figure can "sell your work", it is worth spending some time thinking about it

Don't overload with information, but ensure the important stuff is there

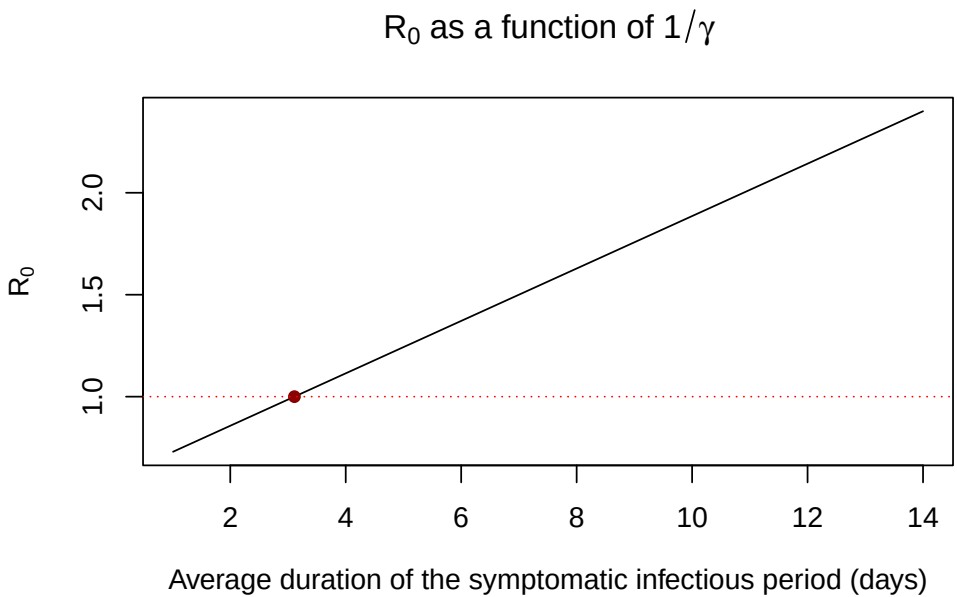
For instance, the red line at $\mathcal{R}_0 = 1$ is a good idea: in your work, you are probably describing how the situation changes when $\mathcal{R}_0 = 1$

Let's make the point of intersection even easier to see

Finding where $\mathcal{R}_0 = 1$

Recall that as we computed it, $\mathcal{R}_0$ is decreasing, so look for when $\mathcal{R}_0 < 1$ the first time

```
idx_switch = which(R0_values<=1)[1]
inv_gamma_switch = inv_gamma[idx_switch]
plot(inv_gamma, R0_values,
     type="l",
     xlab=TeX("Average duration of the symptomatic infectious period (days)
     ylab=expression(R[0]),
     main = TeX("$R_0$ as a function of $1/\\gamma$"))
points(x = inv_gamma_switch, y = 1, col = "darkred", pch = 16)
abline(h=1, col="red", lty = 3)
```


R_0 as a function of $1/\gamma$
R_0
1.0
1.5
2.0
2
4
6
8
10
12
14
Average duration of the symptomatic infectious period (days)

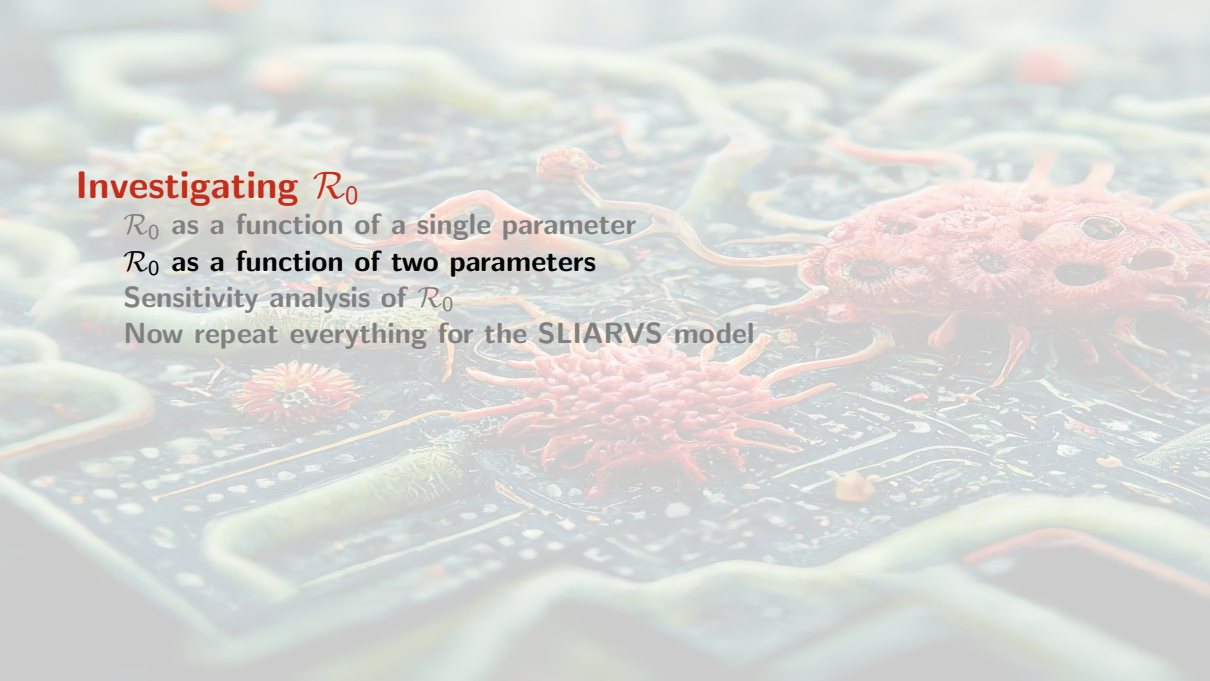
About figure titles (IMOBO)

(in my own biased opinion)

Here, I am using `main=` to add titles to the figures

This is because I am including the figures as full size images in slides without titles (see the `Rnw` file)

In a paper, usually, you do not need to provide a title for the figure: the caption should be enough and serves the same role. Check the journal you are submitting to!



Investigating $\mathcal{R}_0$

- $\mathcal{R}_0$ as a function of a single parameter
- **$\mathcal{R}_0$ as a function of two parameters**
- Sensitivity analysis of $\mathcal{R}_0$
- Now repeat everything for the SLIARVS model

Heatmap plots

The last plot shows how $\mathcal{R}_0$ varies as a function of γ

In many circumstances, we are interested in how $\mathcal{R}_0$ varies as a function of two parameters

Setting up the heatmap

The easiest way (IMOBO) to set things up is to use the function `expand.grid` to make a table with all combinations of the two parameters and compute the value of the function for each of the combinations

Ideally, the function evaluated should be vectorised, so that you can compute all values at once, but this is not always possible. Here, we will do both the direct approach and the vectorised one to illustrate the difference

Setting up the heatmap – calling expand.grid

```
values = expand.grid(gamma = seq(1/14, 1, by=0.01),
                     p = seq(0, 1, by = 0.01))
head(values)
```

```
##        gamma p
## 1 0.07142857 0
## 2 0.08142857 0
## 3 0.09142857 0
## 4 0.10142857 0
## 5 0.11142857 0
## 6 0.12142857 0
```

Setting up the heatmap – computing the values

```
R0_values = c()
for (i in 1:nrow(values)) {
  p$gamma = values$gamma[i]
  p$p = values$p[i]
  R0_values = c(R0_values, R0_SLIAR(p))
}
values$R0 = R0_values
head(values, 4)

##        gamma p  R0
## 1 0.07142857 0 0.9
## 2 0.08142857 0 0.9
## 3 0.09142857 0 0.9
## 4 0.10142857 0 0.9
```

Setting up the heatmap – computing the values (vectorised)

Revise the $\mathcal{R}_0$ function so it makes the difference between default parameter values and those changed for the heatmap. We could of course just redefine `p$p` and `p$gamma`

```
R0_SLIAR_vect = function(p, v) {
  return(p$beta*(v$p/v$gamma + p$delta*(1-v$p)/p$eta)*p$S0)
}
values = expand.grid(gamma = seq(1/14, 1, by=0.01),
                     p = seq(0, 1, by = 0.01))
values$R0 = R0_SLIAR_vect(p, values)
head(values, 4)

##        gamma p  R0
## 1 0.07142857 0 0.9
## 2 0.08142857 0 0.9
## 3 0.09142857 0 0.9
## 4 0.10142857 0 0.9
```

Plotting the heatmap

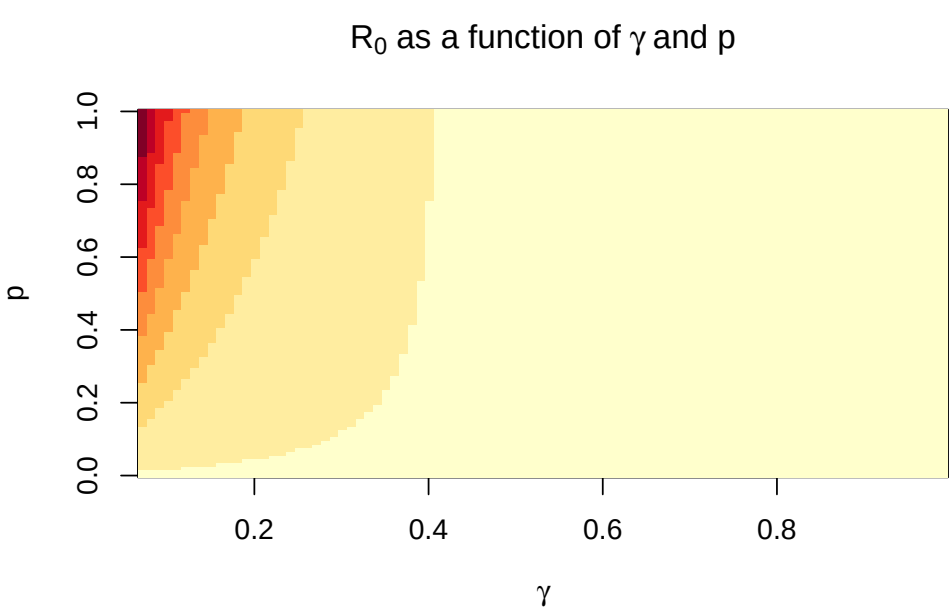
There are many different ways to plot heatmaps in R

- The `image` function is the simplest, but it is not very flexible
- The `ggplot2` package has a `geom_tile` function that is more flexible
- The `pheatmap` package is very good for publication quality heatmaps
- Contour plots are also a good option

Plotting the heatmap – using image

We need `require(RColorBrewer)` for the colour palette

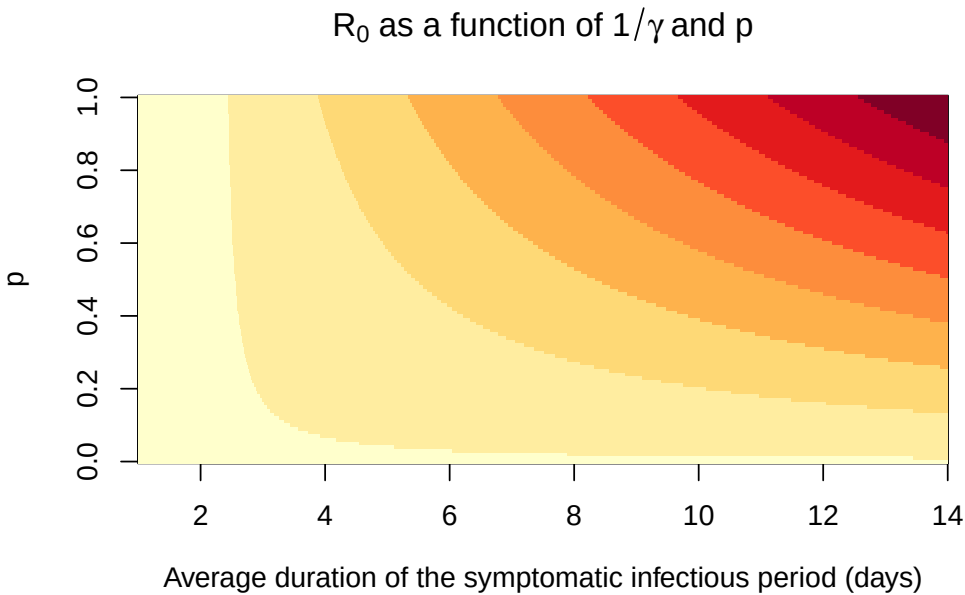
```
image(x = unique(values$gamma), y = unique(values$p),
      z = matrix(values$R0,
                 nrow = length(unique(values$gamma))),
      col = brewer.pal(9, "YlOrRd"),
      xlab = TeX("$\\gamma$"),
      ylab = TeX("$p$"),
      main = TeX("$R_0$ as a function of $\\gamma$ and $p$"))
```


R_0 as a function of γ and p
p
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8
γ

As before: axes need to make sense to normal people

Here, it is easier to actually recompute with $1/\gamma$

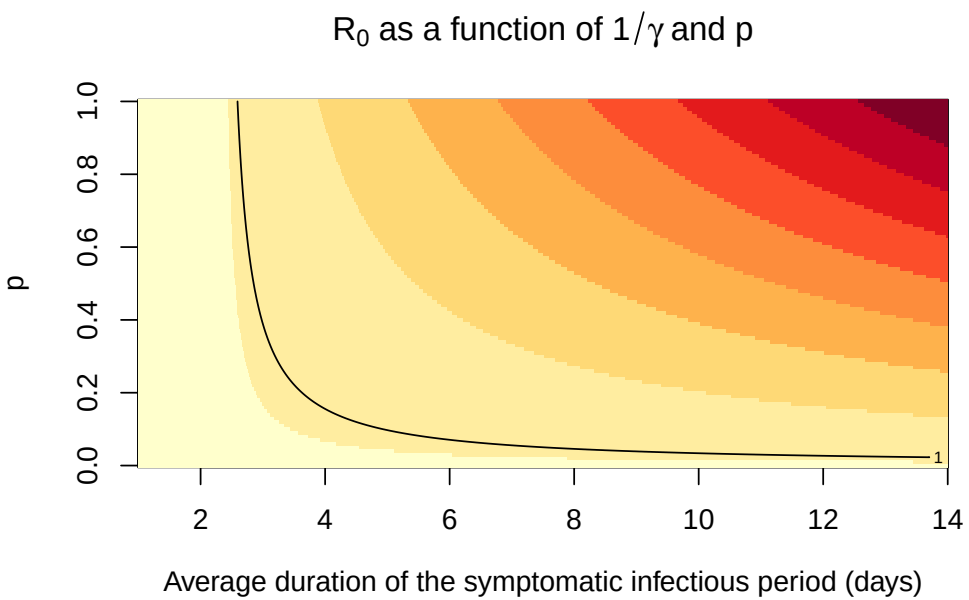
```
R0_SLIAR_vect_inv_gamma = function(p, v) {
  return(p$beta*(v$p*v$inv_gamma + p$delta*(1-v$p)/p$eta)*p$S0)
}
values = expand.grid(inv_gamma = seq(1, 14, by=0.01),
                     p = seq(0, 1, by = 0.01))
values$R0 = R0_SLIAR_vect_inv_gamma(p, values)
image(x = unique(values$inv_gamma), y = unique(values$p),
      z = matrix(values$R0,
                 nrow = length(unique(values$inv_gamma))),
      col = brewer.pal(9, "YlOrRd"),
      xlab = TeX("Average duration of the symptomatic infectious period (da
      ylab = TeX("$p$"),
      main = TeX("$R_0$ as a function of $1/\\gamma$ and $p$"))
```


R_0 as a function of $1/\gamma$ and p
p
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
12
14
Average duration of the symptomatic infectious period (days)

Let us add useful information

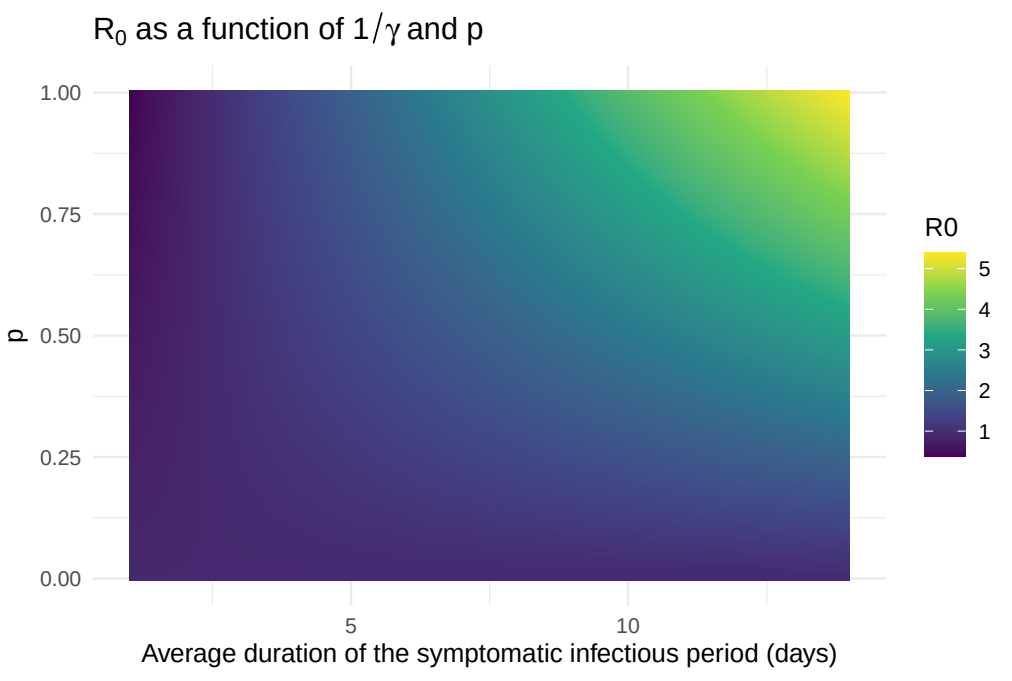
A curve showing where $\mathcal{R}_0 = 1$ would be useful

```
image(x = unique(values$inv_gamma), y = unique(values$p),
      z = matrix(values$R0,
                 nrow = length(unique(values$inv_gamma))),
      col = brewer.pal(9, "YlOrRd"),
      xlab = TeX("Average duration of the symptomatic infectious period (da
      ylab = TeX("$p$"),
      main = TeX("$R_0$ as a function of $1/\\gamma$ and $p$"))
contour(x = unique(values$inv_gamma), y = unique(values$p),
        z = matrix(values$R0,
                   nrow = length(unique(values$inv_gamma))),
        levels = c(1), add = TRUE)
```


R_0 as a function of $1/\gamma$ and p
p
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
12
14
1
Average duration of the symptomatic infectious period (days)

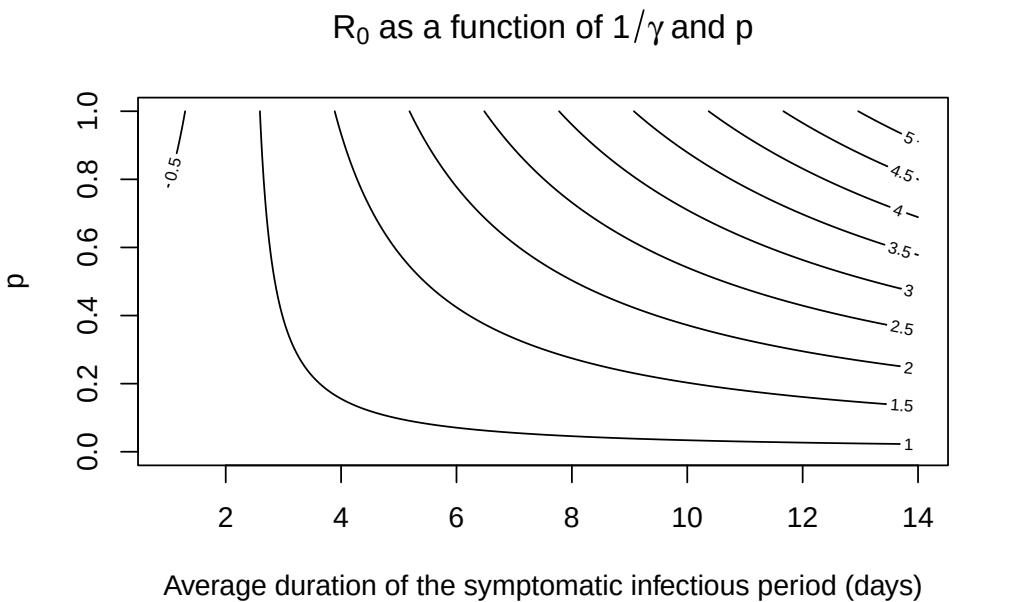
Plotting the heatmap – using ggplot2

```
require(ggplot2)
ggplot(values, aes(x = inv_gamma, y = p, fill = R0)) +
  geom_tile() +
  scale_fill_viridis_c() +
  #scale_fill_gradientn(colours = brewer.pal(9, "YlOrRd")) +
  xlab("Average duration of the symptomatic infectious period (days)") +
  ylab(TeX("$p$")) +
  ggtitle(TeX("$R_0$ as a function of $1/\\gamma$ and $p$")) +
  theme_minimal()
```


R_0 as a function of $1/\gamma$ and p
1.00
0.75
0.50
0.25
0.00
p
5
10
Average duration of the symptomatic infectious period (days)
R0
5
4
3
2
1

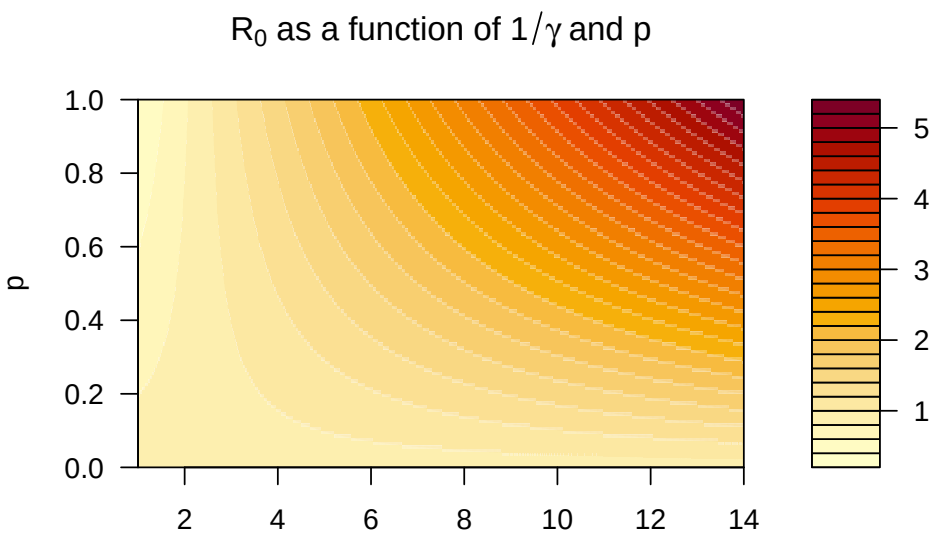
Plotting the heatmap – using contour

```
contour(x = unique(values$inv_gamma), y = unique(values$p),
        z = matrix(values$R0,
                   nrow = length(unique(values$inv_gamma))),
        xlab = "Average duration of the symptomatic infectious period (days
        ylab = TeX("$p$"),
        main = TeX("$R_0$ as a function of $1/\\gamma$ and $p$"))
```


R_0 as a function of $1/\gamma$ and p
p
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
12
14
Average duration of the symptomatic infectious period (days)
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Plotting the heatmap – using `filled.contour`

```
filled.contour(x = unique(values$inv_gamma), y = unique(values$p),
        z = matrix(values$R0,
                   nrow = length(unique(values$inv_gamma))),
        xlab = "Average duration of the symptomatic infectious period (days
        ylab = TeX("$p$"),
        main = TeX("$R_0$ as a function of $1/\\gamma$ and $p$"))
```


R_0 as a function of $1/\gamma$ and p
p
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
12
14
5
4
3
2
1
Average duration of the symptomatic infectious period (da

Last remark on heatmaps – `expand.grid` is cool

We can use `expand.grid` for more than two variables

The only problem is visualisation...

Investigating $\mathcal{R}_0$

- $\mathcal{R}_0$ as a function of a single parameter
- $\mathcal{R}_0$ as a function of two parameters
- **Sensitivity analysis of $\mathcal{R}_0$**
- Now repeat everything for the SLIARVS model

Sensitivity analysis of $\mathcal{R}_0$

To perform a sensitivity analysis, there are several steps to follow:

1. Define the parameters of interest
2. Define the range of variation of each parameter
3. Define the method to use to vary the parameters
4. Define the function to evaluate
5. Evaluate the function for each combination of parameters
6. Choose a method to evaluate the sensitivity

SLIAR model – Sensitivity analysis of $\mathcal{R}_0$

We perform a sensitivity analysis of $\mathcal{R}_0$ with respect to all parameters involved in (2) except S_0 (which we take to be 999 here)

Note that we cannot use the same trick as before, where we set β using (9): it would defeat the purpose of the sensitivity analysis to assume a value of $\mathcal{R}_0$ to compute β to then compute $\mathcal{R}_0$

Instead, we set overall acceptable bounds for $\mathcal{R}_0$ and compute a resulting range of values of β based on the set ranges for the other parameters and these bounds for $\mathcal{R}_0$

SLIAR model – Finding the range of β

$$\beta = \frac{\mathcal{R}_0}{\left(\frac{p}{\gamma} + \frac{\delta(1-p)}{\eta}\right) S_0} \tag{9}$$

Then

$$\frac{\partial \beta}{\partial \mathcal{R}_0} > 0, \frac{\partial \beta}{\partial \gamma} > 0, \frac{\partial \beta}{\partial \eta} > 0$$

$$\frac{\partial \beta}{\partial \delta} < 0$$

while

$$\text{sgn}\left(\frac{\partial \beta}{\partial p}\right) = \text{sgn}(\gamma\delta - \eta)$$

So the maximum of β can be evaluated using (9) with the maximum values of $\mathcal{R}_0$, γ, η and the minimum value of δ

On the other hand, we always vary $p \in [0, 1]$; for such p, we have $\beta > 0$

$$\beta|_{p=0} = \frac{\eta \mathcal{R}_0}{\delta S_0} \quad \beta|_{p=1} = \frac{\gamma \mathcal{R}_0}{S_0}$$

with $\beta(p)$

- $\nearrow$ if $\gamma\delta > \eta \implies$ use $p = 0$ and $p = 1$ for the min and max values of β, respectively
- $\searrow$ if $\gamma\delta < \eta \implies$ use $p = 1$ and $p = 0$ for the min and max values of β, respectively

So we can further simplify the computation of the range of β

Thus, in practice, if $\gamma^{\min}\delta^{\min} > \eta^{\min}$, then $\beta(p)$ is increasing and

$$\beta^{\min} = \frac{\eta^{\min}\mathcal{R}_0^{\min}}{\delta^{\min}S_0^{\min}}$$

while if $\gamma^{\min}\delta^{\min} \leq \eta^{\min}$, $\beta(p)$ is decreasing and

$$\beta^{\min} = \frac{\gamma^{\min}\mathcal{R}_0^{\min}}{S_0^{\min}}$$

Similarly, if $\gamma^{\max}\delta^{\max} > \eta^{\max}$, then $\beta(p)$ is increasing and

$$\beta^{\max} = \frac{\gamma^{\max}\mathcal{R}_0^{\max}}{S_0^{\max}}$$

while if $\gamma^{\max}\delta^{\max} \leq \eta^{\max}$, $\beta(p)$ is decreasing and

$$\beta^{\max} = \frac{\eta^{\max}\mathcal{R}_0^{\max}}{\delta^{\max}S_0^{\max}}$$

SLIAR model – Setting parameters

The ranges are shown on the next page, but let us here show how we set up the code

```
params = list()
params$N = 1000
params$I0 = 1
params$S0 = params$N-params$I0
params$R0 = c(0.5, 5)
params.vary = list(
  delta = c(0.05, 1),
  p = c(0, 1),
  gamma = c(1/10, 1/2),
  eta = c(1/7,1))
params.vary$beta =
  c(params.vary$gamma[1]*params$R0[1]/params$S0,
    params.vary$eta[2]*params$R0[2]/(params.vary$delta[2]*params$S0))
```

SLIAR model – Parameter ranges used with $\mathcal{R}_0$

Parameter	Minimum	Maximum
$\mathcal{R}_0$	0.5	5
δ	0.05	1
p	0	1
γ	$1/10$	$1/2$
η	$1/7$	1

$$\gamma^{\min}\delta^{\min} \leq \eta^{\min}\ (0.005 \leq 0.143) \implies \beta^{\min} \simeq 5 \times 10^{-5}$$

$$\gamma^{\max}\delta^{\max} \leq \eta^{\max}\ (0.5 \leq 1) \implies \beta^{\max} \simeq 0.005005$$

SLIAR model – Parameter ranges used with β

After this computation of β, we thus consider the following ranges for the parameters

Parameter	Minimum	Maximum
β	5×10^{-5}	0.005005
δ	0.05	1
p	0	1
γ	$1/10$	$1/2$
η	$1/7$	1

An important question – How to sample parameter space

We want to generate `nb_samples` values of the function we are evaluating

However, we can't just pick random values for each parameter

Indeed, if we did, we would have no guarantee that we would cover the whole parameter space (in particular, various combinations of random values of the different parameters)

Of course, we cannot test all possible combinations of parameter values, but we can try to approach this by using a **sampling method**

Illustrating sampling methods

We use a function adapted from one (`graph2LHS`) in an official CRAN vignette (not my vignettes) (link to Rmd version of the vignette) to show the different sampling schemes

In all illustrations, we use two parameters whose values are in $[0, 1]$

In practice, we have more parameters and their values are in different ranges

Investigating $\mathcal{R}_0$

Sensitivity analysis of $\mathcal{R}_0$

- **Grid Sampling**
- Sobol sampling
- Latin Hypercube Sampling
- Comparing results
- Using a parallel version

The "naive" method – Grid sampling

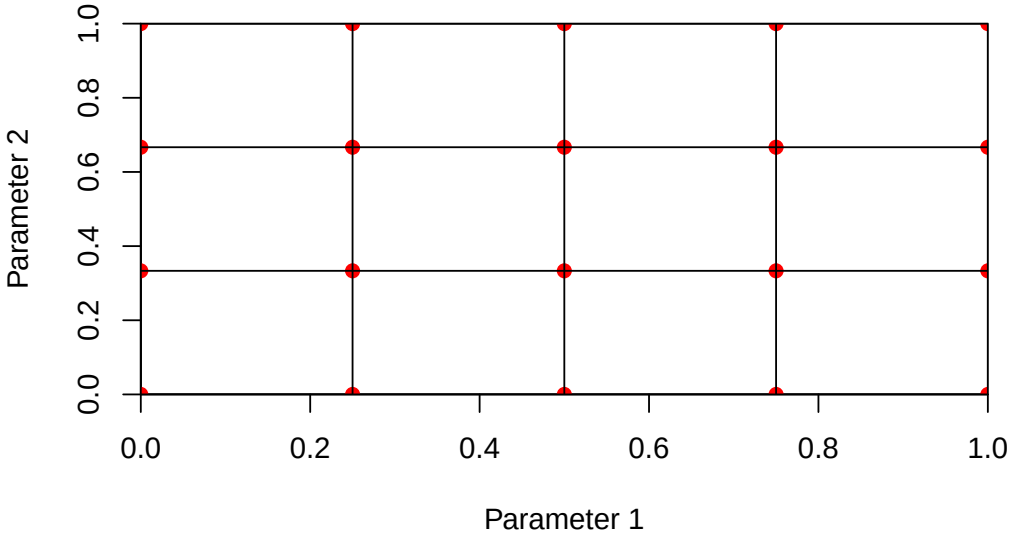
We can use `expand.grid` to generate all possible combinations of parameter values

This gets large a little too quickly: in the SLIAR model (1), we want to investigate 5 parameters

If we pick 10 values for each parameter, we have to evaluate $10^5 = 100,000$ combinations

This is viable if the function being evaluated ($\mathcal{R}_0$ here) is not costly or there aren't too many parameters; otherwise, this quickly becomes messy or too long

Grid sampling (20 samples)



Select a sampling method – Grid sampling

```
nb_points_per_param = 10
params.grid = expand_grid(
  delta = seq(params.vary$delta[1],
              params.vary$delta[2],
              length.out = nb_points_per_param),
  p = seq(params.vary$p[1], params.vary$p[2], length.out = nb_points_per_pa
  gamma = seq(params.vary$gamma[1], params.vary$gamma[2], length.out = nb_p
  eta = seq(params.vary$eta[1], params.vary$eta[2], length.out = nb_points_
  beta = seq(params.vary$beta[1], params.vary$beta[2], length.out = nb_poin
```

Vectorised or parallel? Both?

Functions like the ones here are easily vectorised. We show an example here

More complex functions (for instance, requiring the use of `ode`) may not be vectorised and can benefit from parallelisation

We will use the `parallel` package to parallelise the evaluation of the function

It is possible to parallelise a vector function as well, but we will not do that here

Vectorised version of the $\mathcal{R}_0$ function

The first function we wrote actually works with vectors, but we need to allow S0 as an argument (as it is not varying). So we distinguish between two parameter sets, the fixed ones and the varying ones

```
R0_SLIAR = function(p, p_fixed) {
  OUT = p$beta*(p$p/p$gamma+p$delta*(1-p$p)/p$eta)*p_fixed$S0
  return(OUT)
}
R0_values = R0_SLIAR(params.grid, params)
```

Now compute PRCC

Partial rank correlation coefficients (PRCC) are a measure of the correlation between a parameter and the output of a function, controlling for the effect of the other parameters

We make a function just to simplify the call

```
compute_PRCC = function(v, pars) {
  x = pcc(pars, as.numeric(v),
          rank = TRUE, semi = FALSE)
  return(x)
}
R0_SLIAR_PRCC_grid = compute_PRCC(R0_values, params.grid)
```

The PRCC values

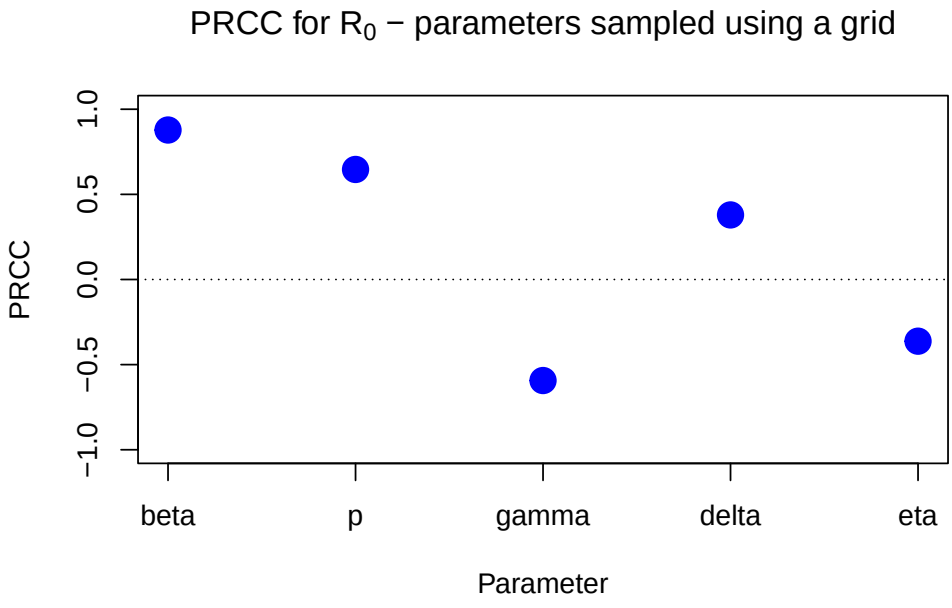
```
R0_SLIAR_PRCC_grid

##
## Call:
## pcc(X = pars, y = as.numeric(v), rank = TRUE, semi = FALSE)
##
## Partial Rank Correlation Coefficients (PRCC):
##         original
## delta  0.3792029
## p      0.6462309
## gamma -0.5934708
## eta   -0.3623526
## beta   0.8778737
```

Plotting the results using base graphics

To make things a bit easier to read, let's sort the PRCC values in decreasing absolute value

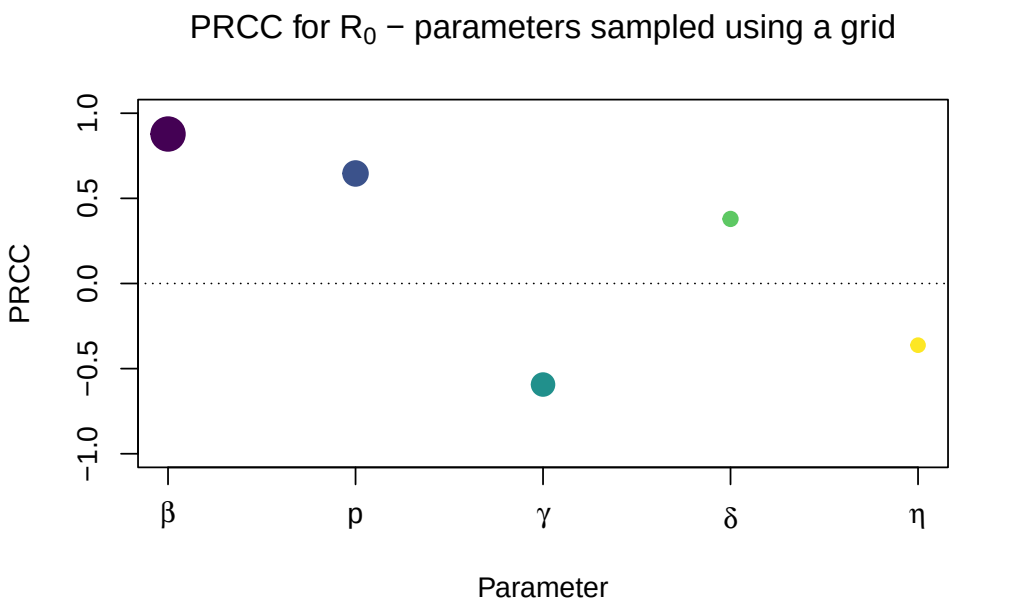
```
idx = order(abs(R0_SLIAR_PRCC_grid$PRCC$original),
            decreasing = TRUE)
plot(R0_SLIAR_PRCC_grid$PRCC$original[idx],
     ylim = c(-1,1), xaxt='n',
     xlab = "Parameter", ylab = "PRCC",
     main = TeX("PRCC for $R_0$ - parameters sampled using a grid"),
     pch = 19, col = "blue", cex = 2)
axis(1, at = 1:length(idx),
     labels = rownames(R0_SLIAR_PRCC_grid$PRCC)[idx])
abline(h=0, lty = 3)
```


PRCC for R_0 − parameters sampled using a grid
PRCC
1.0
0.5
0.0
−0.5
−1.0
beta
p
gamma
delta
eta
Parameter

Refining the plot a little

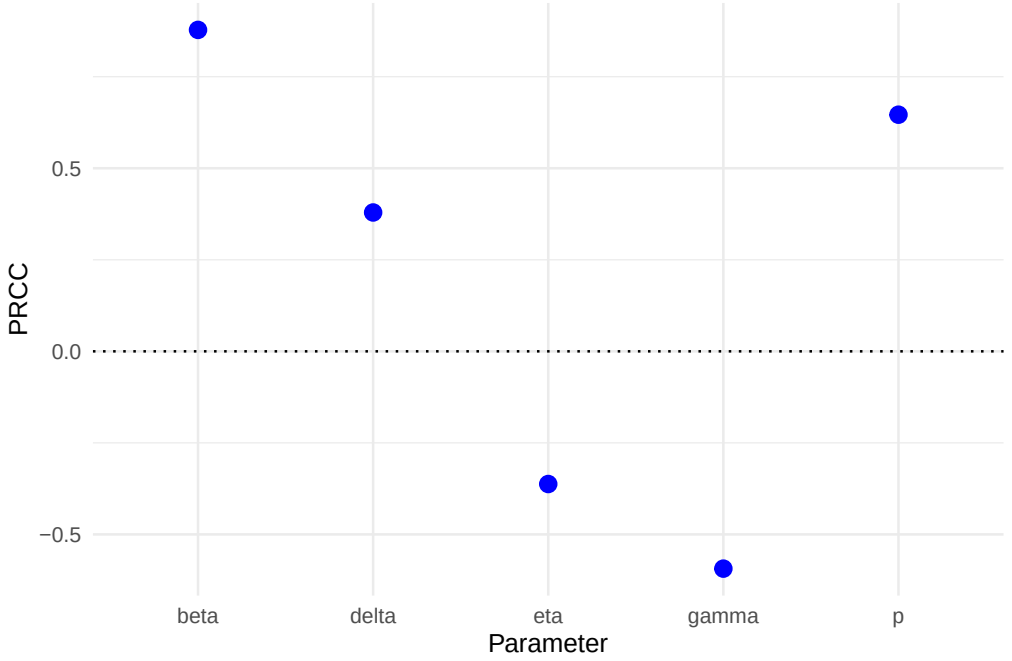
Let's use different colours for the points, make their size proportional to the PRCC values and use LaTeXfor the labels

```
colour = viridis::viridis(length(idx))
labels =  sprintf("$\\%s$", rownames(R0_SLIAR_PRCC_grid$PRCC)[idx])
labels = gsub("\\\\p", "p", labels)
plot(R0_SLIAR_PRCC_grid$PRCC$original[idx],
     ylim = c(-1,1), xaxt='n', pch = 19,
     xlab = "Parameter", ylab = "PRCC",
     col = colour,
     main = TeX("PRCC for $R_0$ - parameters sampled using a grid"),
     cex = 3 * abs(R0_SLIAR_PRCC_grid$PRCC$original)[idx])
axis(1, at = 1:length(idx), labels = TeX(labels))
abline(h=0, lty = 3)
```


PRCC for R_0 − parameters sampled using a grid
PRCC
1.0
0.5
0.0
−0.5
−1.0
β
p
γ
δ
η
Parameter

Plotting the results using ggplot2

```
df = data.frame(Parameter = rownames(R0_SLIAR_PRCC_grid$PRCC)[idx],
                PRCC = R0_SLIAR_PRCC_grid$PRCC$original[idx])
ggplot(df, aes(x = Parameter, y = PRCC)) +
  geom_point(colour = "blue", size = 3) +
  geom_hline(yintercept = 0, linetype = "dotted") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  theme_minimal()
```


PRCC
0.5
0.0
−0.5
beta
delta
eta
gamma
p
Parameter

The problem with grid sampling

Suppose we decide that since 100,000 evaluations are done easily, we can afford to do 1,000,000 evaluations

How do we implement that?

We have

$$1,000,000^{1/5} \simeq 15.8489$$

so we could decide to cut each range in 16 parts

If the range of each parameter is relatively large, this could quickly become unmanageable

What if I do want grid sampling nonetheless?

The function `parameterSets` from the `sensitivity` package can be used to generate a grid of parameter sets

```
nb_samples = 1000000
length_grid_side = floor(nb_samples^(1/5))
params.grid = parameterSets(par.ranges = params.vary,
                            samples = rep(length_grid_side, 5),
                            method = "grid")
params.grid = as.data.frame(params.grid)
colnames(params.grid) = names(params.vary)
R0_values = R0_SLIAR(params.grid, params)
R0_SLIAR_PRCC_grid_2 = compute_PRCC(R0_values, params.grid)
```

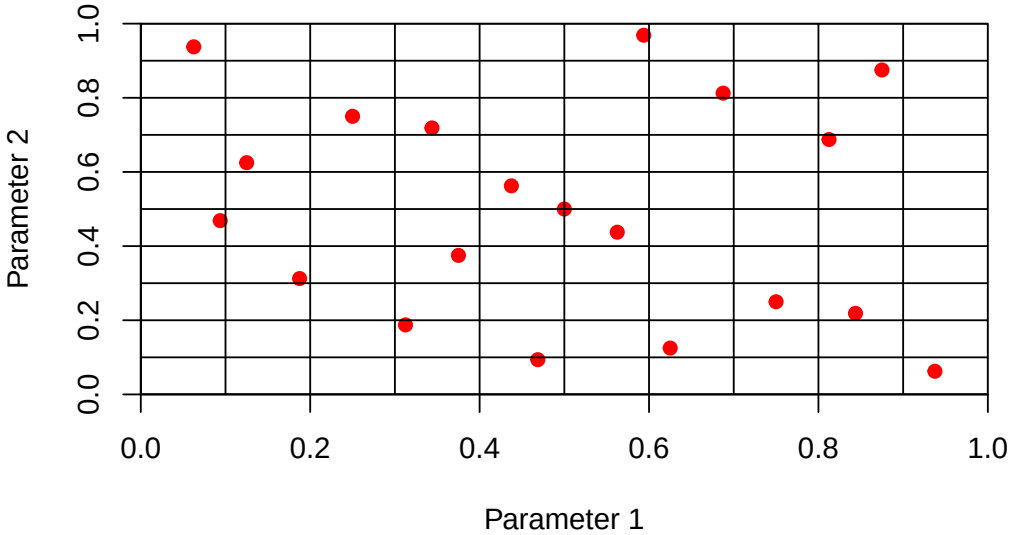
You can also use `method="innergrid"` to generate a grid of parameter sets that are offset from the sides of the hypecube

Investigating $\mathcal{R}_0$

Sensitivity analysis of $\mathcal{R}_0$

- Grid Sampling
- **Sobol sampling**
- Latin Hypercube Sampling
- Comparing results
- Using a parallel version

Sobol sampling (20 samples)



Select a sampling method – Sobol

This uses the `sensitivity` package

```
nb_samples = 100000
params.sobol = parameterSets(par.ranges = params.vary,
                             samples = nb_samples,
                             method = "sobol")
params.sobol = as.data.frame(params.sobol)
colnames(params.sobol) = names(params.vary)
```

Compute PRCC with Sobol sampling

Partial rank correlation coefficients (PRCC) are a measure of the correlation between a parameter and the output of a function, controlling for the effect of the other parameters

```
R0_values = R0_SLIAR(params.sobol, params)
compute_PRCC = function(v, pars) {
  x = pcc(pars, as.numeric(v),
          rank = TRUE, semi = FALSE)
  return(x)
}
R0_SLIAR_PRCC_sobol = compute_PRCC(R0_values, params.sobol)
```

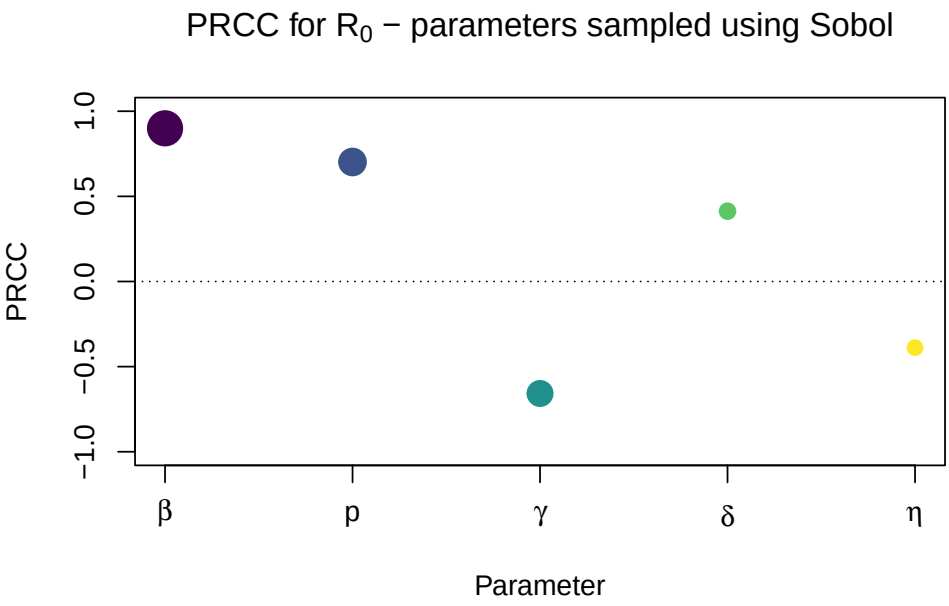
The PRCC values

```
R0_SLIAR_PRCC_sobol

##
## Call:
## pcc(X = pars, y = as.numeric(v), rank = TRUE, semi = FALSE)
##
## Partial Rank Correlation Coefficients (PRCC):
##         original
## delta  0.4129675
## p      0.7018390
## gamma -0.6578156
## eta   -0.3885156
## beta   0.8996126
```

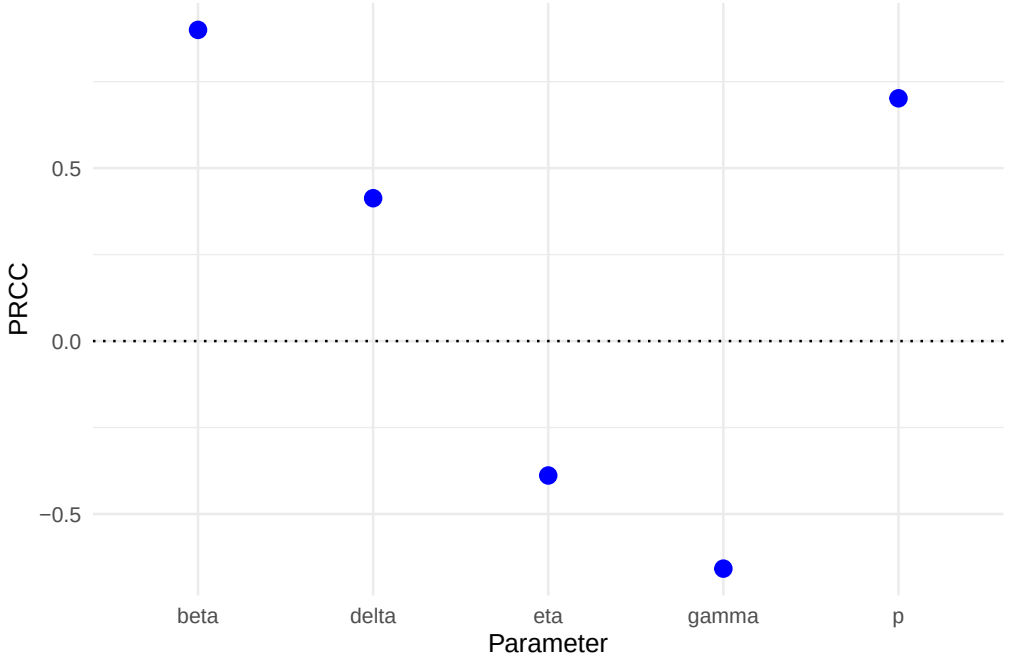
Plotting the results using base graphics

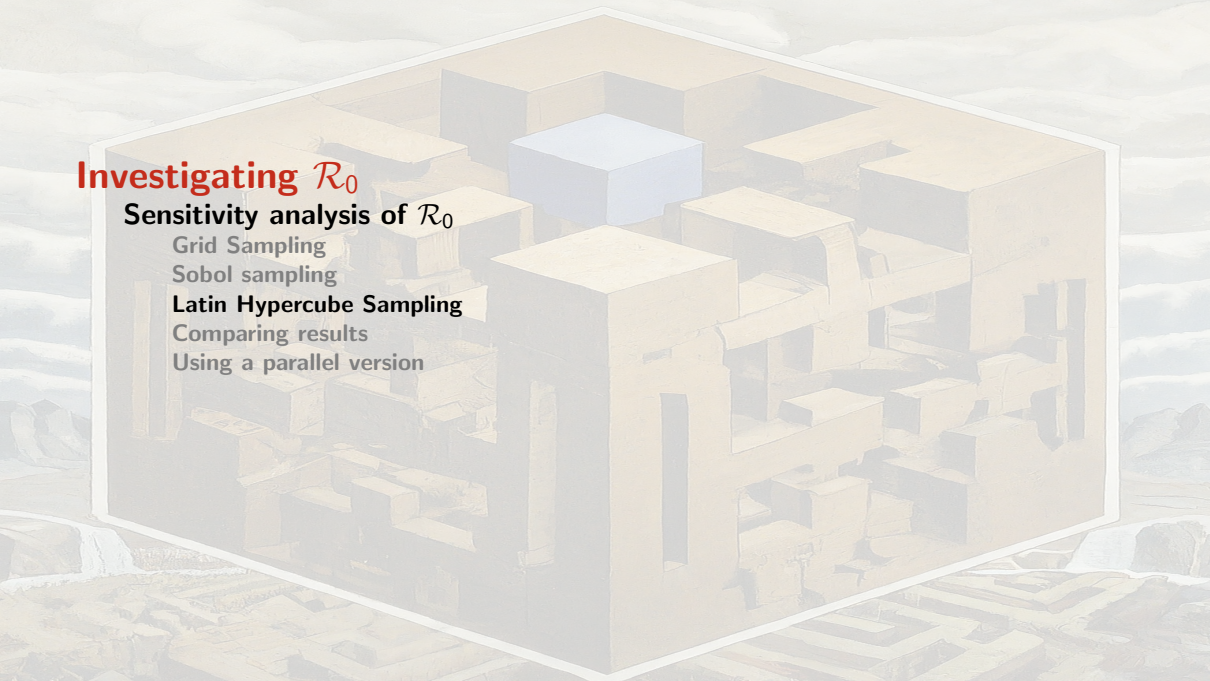
```
idx = order(abs(R0_SLIAR_PRCC_sobol$PRCC$original),
            decreasing = TRUE)
colour = viridis::viridis(length(idx))
labels =  sprintf("$\\%s$", rownames(R0_SLIAR_PRCC_sobol$PRCC)[idx])
labels = gsub("\\\\p", "p", labels)
plot(R0_SLIAR_PRCC_sobol$PRCC$original[idx],
     ylim = c(-1,1), xaxt='n', pch = 19,
     xlab = "Parameter", ylab = "PRCC",
     col = colour,
     main = TeX("PRCC for $R_0$ - parameters sampled using Sobol"),
     cex = 3 * abs(R0_SLIAR_PRCC_sobol$PRCC$original)[idx])
axis(1, at = 1:length(idx), labels = TeX(labels))
abline(h=0, lty = 3)
```


PRCC for R_0 − parameters sampled using Sobol
PRCC
1.0
0.5
0.0
−0.5
−1.0
β
p
γ
δ
η
Parameter

Plotting the results using ggplot2

```
df = data.frame(Parameter = rownames(R0_SLIAR_PRCC_sobol$PRCC)[idx],
                PRCC = R0_SLIAR_PRCC_sobol$PRCC$original[idx])
ggplot(df, aes(x = Parameter, y = PRCC)) +
  geom_point(colour = "blue", size = 3) +
  geom_hline(yintercept = 0, linetype = "dotted") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  theme_minimal()
```


PRCC
0.5
0.0
−0.5
beta
delta
eta
gamma
p
Parameter



Investigating $\mathcal{R}_0$

Sensitivity analysis of $\mathcal{R}_0$

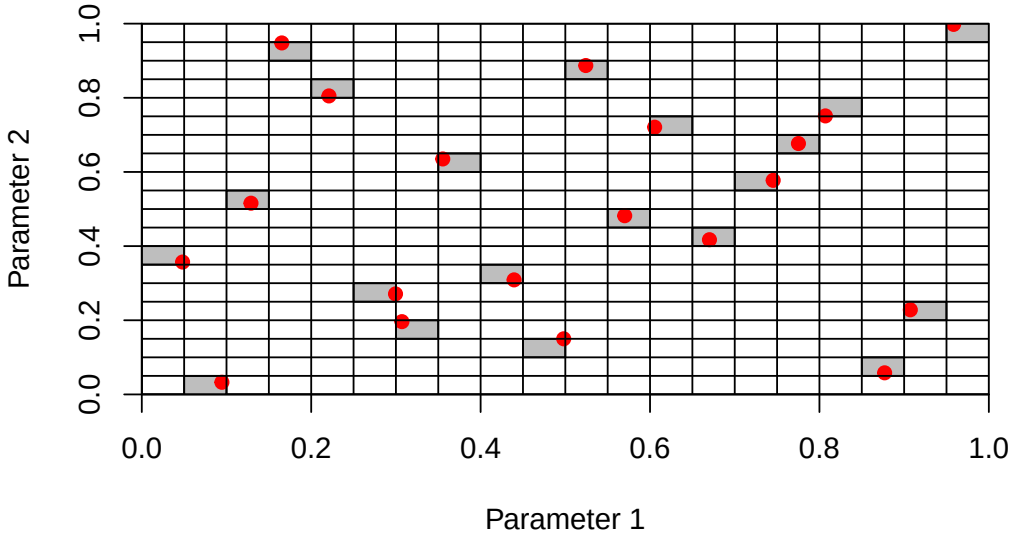
- Grid Sampling
- Sobol sampling
- **Latin Hypercube Sampling**
- Comparing results
- Using a parallel version

Select a sampling method – Latin Hypercube

This is another method to sample the parameter space

> *From Wikipedia In the context of statistical sampling, a square grid containing sample positions is a Latin square if (and only if) there is only one sample in each row and each column. A Latin hypercube is the generalisation of this concept to an arbitrary number of dimensions, whereby each sample is the only one in each axis-aligned hyperplane containing it.*

Latin hypercube sampling (20 samples)



Select a sampling method – Latin Hypercube

Call the function with arguments for the number of samples and the number of parameters

```
params.lhs = lhs::randomLHS(nb_samples, length(params.vary))
params.lhs = as.data.frame(params.lhs)
colnames(params.lhs) = names(params.vary)
head(params.lhs)
```

```
##       delta          p     gamma       eta      beta
## 1 0.8071189 0.82593010 0.7240572 0.1682574 0.4777269
## 2 0.5592378 0.02339682 0.1671804 0.7311842 0.1715592
## 3 0.5878560 0.72620811 0.3368387 0.4749755 0.8139687
## 4 0.6416634 0.71584999 0.2499241 0.3093368 0.5962585
## 5 0.6691014 0.12847221 0.1019496 0.1287732 0.8552293
## 6 0.3453870 0.85983830 0.8939880 0.4173159 0.1367271
```

Results are uniform on $[0, 1]$ for each parameter and need to be transformed prior to use

Transforming the uniform distribution

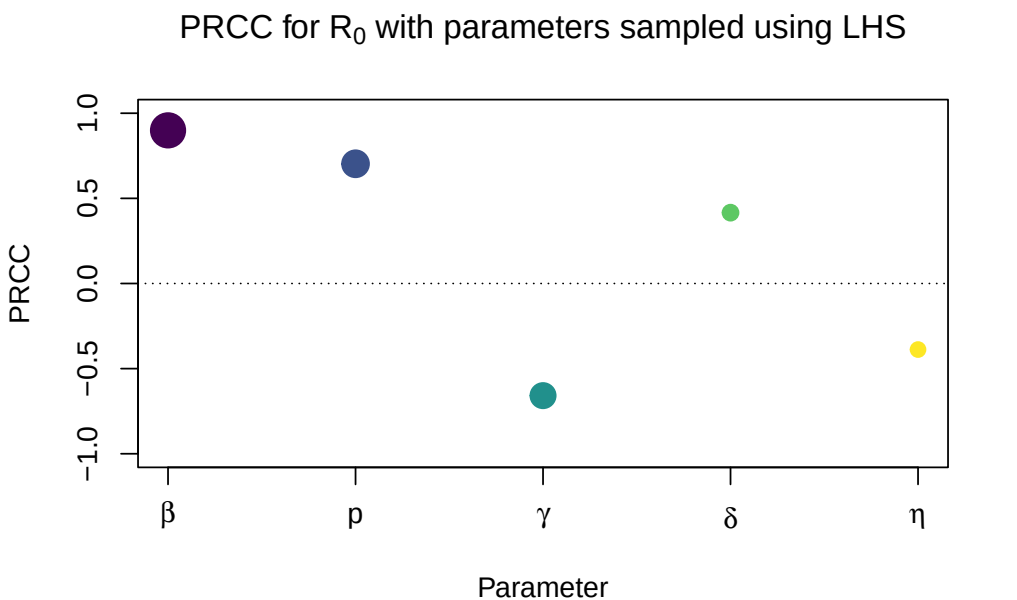
Suppose values ξ uniformly distributed on $[0, 1]$ are to be transformed to the range $[x_{\min}, x_{\max}]$

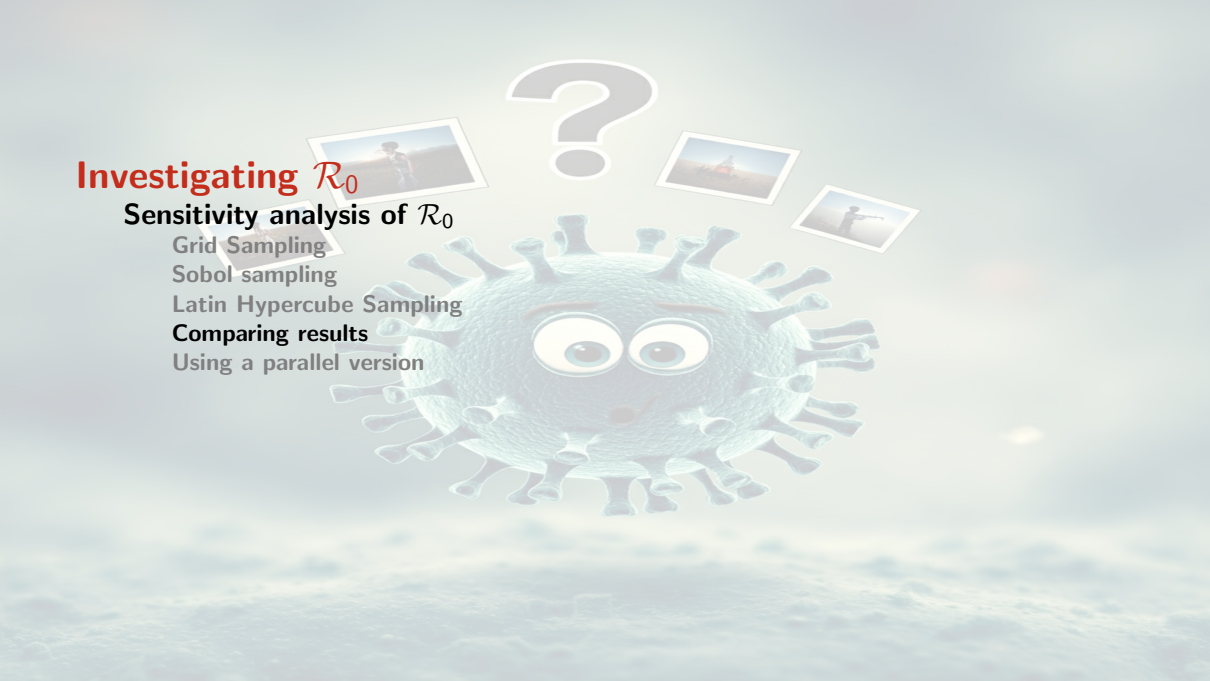
$$x = x_{\min} + \xi(x_{\max} - x_{\min})$$

```
for (c in 1:length(params.vary)) {
  params.lhs[,c] = params.vary[[c]][1] + params.lhs[,c] *
    (params.vary[[c]][2] - params.vary[[c]][1])
}
R0_values = R0_SLIAR(params.lhs, params)
R0_SLIAR_PRCC_lhs = compute_PRCC(R0_values, params.lhs)
```

Plotting the results using base graphics

```
idx = order(abs(R0_SLIAR_PRCC_lhs$PRCC$original),
            decreasing = TRUE)
colour = viridis::viridis(length(idx))
labels =  sprintf("$\\%s$", rownames(R0_SLIAR_PRCC_lhs$PRCC)[idx])
labels = gsub("\\\\p", "p", labels)
plot(R0_SLIAR_PRCC_lhs$PRCC$original[idx],
     ylim = c(-1,1), xaxt='n', pch = 19,
     xlab = "Parameter", ylab = "PRCC",
     main = TeX("PRCC for $R_0$ with parameters sampled using LHS"),
     col = colour,
     cex = 3 * abs(R0_SLIAR_PRCC_lhs$PRCC$original)[idx])
axis(1, at = 1:length(idx), labels = TeX(labels))
abline(h=0, lty = 3)
```


PRCC for R_0 with parameters sampled using LHS
PRCC
1.0
0.5
0.0
−0.5
−1.0
β
p
γ
δ
η
Parameter



Investigating $\mathcal{R}_0$

Sensitivity analysis of $\mathcal{R}_0$

- Grid Sampling
- Sobol sampling
- Latin Hypercube Sampling
- **Comparing results**
- Using a parallel version

Comparing the PRCC values

```
R0_SLIAR_PRCC = cbind(R0_SLIAR_PRCC_grid$PRCC$original,
                      R0_SLIAR_PRCC_grid_2$PRCC$original,
                      R0_SLIAR_PRCC_sobol$PRCC$original,
                      R0_SLIAR_PRCC_lhs$PRCC$original)
rownames(R0_SLIAR_PRCC) = rownames(R0_SLIAR_PRCC_grid$PRCC)
colnames(R0_SLIAR_PRCC) = c("Grid 1", "Grid 2", "Sobol", "LHS")
knitr::kable(R0_SLIAR_PRCC, digits = 3, booktabs = TRUE)
```

	Grid 1	Grid 2	Sobol	LHS
delta	0.379	0.389	0.413	0.416
p	0.646	0.663	0.702	0.703
gamma	-0.593	-0.614	-0.658	-0.659
eta	-0.362	-0.370	-0.389	-0.388
beta	0.878	0.885	0.900	0.899

One last remark on sensitivity

We have used γ and η

We saw when plotting $\mathcal{R}_0$ as a function of γ that the situation is very different when using γ and $1/\gamma$

The same is true, to a lesser extent, of η and $1/\eta$

We would therefore get a very different picture if we considered $1/\gamma$ and $1/\eta$ for the sensitivity analysis

Nobody expects the Spanish Inquisition!

Here, we used PRCC. There are many other types of sensititivities that can be computed

The `sensitivity` packages has quite an array of these. See the package documentation for details

There are other `R` packages that also do sensitivity analysis

Investigating $\mathcal{R}_0$

Sensitivity analysis of $\mathcal{R}_0$

- Grid Sampling
- Sobol sampling
- Latin Hypercube Sampling
- Comparing results
- **Using a parallel version**

Why parallelise?

Numerical problems like the evaluation of a function at a bunch of points in parameter space when all evaluations are independent from one another are **embarassingly parallel**

In the case of the PRCC for $\mathcal{R}_0$ like here, parallel is not necessarily a good idea, as there is a lot of overhead

However, if you are computing the response of a more complex function, e.g., simulating the solutions to an ODE then performing a bunch of computations, then parallelising can greatly speed stuff up

Setting up the parallel version

The easiest way is to use `parLapply`. For this, we need a list in which each entry is a list with a specific point in parameter space

```
params.sobol.list = split(params.sobol, seq(nrow(params.sobol)))
head(params.sobol.list, n = 2)
```

```
## $`1`
##   delta   p gamma       eta        beta
## 1 0.525 0.5   0.3 0.5714286 0.002527528
##
## $`2`
##    delta    p gamma       eta        beta
## 2 0.7625 0.25   0.4 0.3571429 0.003766266
```

Testing iteratively first

If your code is going to run in parallel using `parLapply`, it also needs to run sequentially using `lapply`

Using `lapply` also is good to debug, as debugging parallel code is not easy

```
R0_values = lapply(X = params.sobol.list,
                   FUN = function(x) R0_SLIAR(x, params))
head(R0_values, n = 2)

## $`1`
## [1] 5.368255
##
## $`2`
## [1] 8.376266
```

Note the little trick here

The function `R0_SLIAR` takes two arguments, `p` and `p_fixed`

However, `lapply` (or `parLapply` later) only allow dependence on a single variable

Suppose we have a function `f(a,b)`

We use the construct `function(x) f(x,b)` to make `f` depend only its first argument

Similarly, `function(x) f(a,x)` would make `f` depend only on its second argument

Sequential works, let's go parallel

To run your code in parallel using `parLapply`, you need several steps:

- Create a *cluster*, i.e., a group of *workers* that will each execute the function passed as argument to `parLapply` (`R0_SLIAR` here). Typically, you want to use all compute cores but one (unless you are on a headless server), otherwise your machine may become unresponsive
- Provide each worker with all they need to execute the function. Each worker is an instance of `R` running independently
- Call `parLapply` to have the workers run the code
- Close the cluster to "free up" the workers

Let's go parallel! (and time our run)

```
nb_cores <- parallel::detectCores() - 1
nb_cores <- ifelse(nb_cores > 122, 122, nb_cores)
tictoc::tic("whole parallel phase")
cl <- parallel::makeCluster(nb_cores)
parallel::clusterExport(cl, c("params", "RO_SLIAR"))
tictoc::tic("parLapply")
result = parallel::parLapply(cl = cl, X = params.sobol.list,
                             fun =  function(x) RO_SLIAR(x, params))
tictoc::toc()

## parLapply: 0.722 sec elapsed

parallel::stopCluster(cl)
tictoc::toc()

## whole parallel phase: 1.117 sec elapsed
```

Just for comparison

```
tictoc::tic("lapply")
result = lapply(X = params.sobol.list,
                FUN =  function(x) RO_SLIAR(x, params))
tictoc::toc()

## lapply: 0.988 sec elapsed

tictoc::tic("vectorised")
result = RO_SLIAR(params.sobol, params)
tictoc::toc()

## vectorised: 0.003 sec elapsed
```

Remarks about running in parallel

Here, the gain was not massive and the overhead of setting up the cluster was not negligible (cluster setup is sequential so setup time depends on the number of cores)

One of the machines I run this on has 128 threads. `R` has hard coded limit of 128 threads and uses a few of them for other things $\implies$ limit to 122 threads. If you try to use more, your parallel code will not run

If you want to get rid of this limitation, you need to recompile `R` with a higher limit. See a blog post I made about this (link)

That post also explains how to set up a cluster using multiple machines

Investigating $\mathcal{R}_0$

- $\mathcal{R}_0$ as a function of a single parameter
- $\mathcal{R}_0$ as a function of two parameters
- Sensitivity analysis of $\mathcal{R}_0$
- **Now repeat everything for the SLIARVS model**

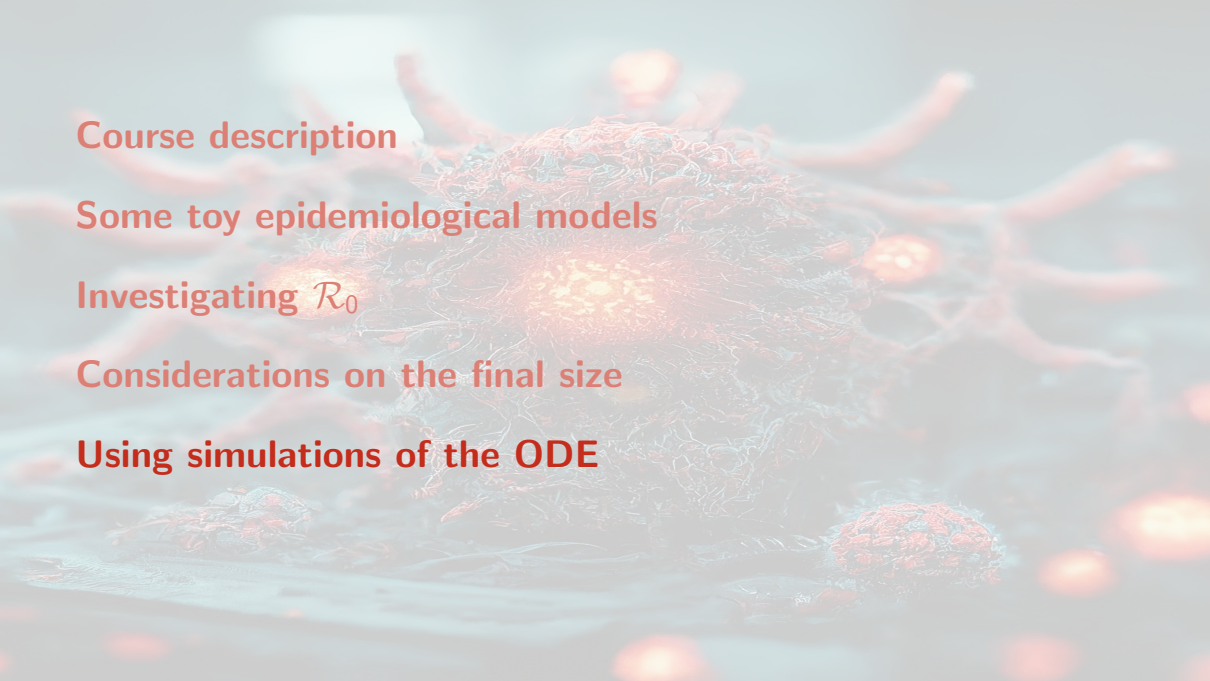
Course description

Some toy epidemiological models

Investigating $\mathcal{R}_0$

Considerations on the final size

Using simulations of the ODE



Course description

Some toy epidemiological models

Investigating $\mathcal{R}_0$

Considerations on the final size

Using simulations of the ODE

In all we have done so far, ε and f have not been used

Indeed, they do not play a role in the computation of the basic reproduction number $\mathcal{R}_0$ given by (2) or the final size relation given by (3)

ε does play a role in determining the "speed" of the system, but we have not considered this aspect in our analysis so far

f helps determine how many individuals die of the disease and won't be discussed here

Can I have this wrapped up to go?

To finish, we use the command `purl` to generate an `R` file (`basic-computational-analysis.R`) in the CODE directory with all the code chunks in this `Rnw` file

```
# From https://stackoverflow.com/questions/36868287/purl-within-knit-dupli
rmd_chunks_to_r_temp <- function(file){
  callr::r(function(file, temp){
    out_file = sprintf("../CODE/%s", gsub(".Rnw", ".R", file))
    knitr::purl(file, output = out_file, documentation = 1)
  }, args = list(file))
}
rmd_chunks_to_r_temp("basic-computational-analysis-1-functions.Rnw")

## [1] "../CODE/basic-computational-analysis-1-functions.R"
```

About that R file

Source the file CODE/basic-computational-analysis.R in R to reproduce all the results in these slides

Some small changes are required; for instance, when sourcing (instead of knitting or interactively), ggplot figures are created but not printed, so in the R file, you need to print them "manually"

```
pp = ggplot(...)
print(pp)
```