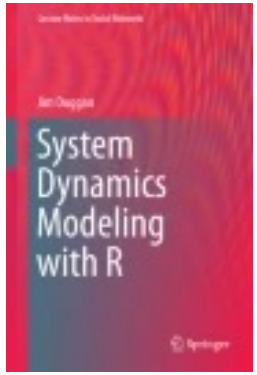
Behavioural OR and System Dynamics

University of Southampton 15 - 19th July 2024

System Dynamics Modelling using R



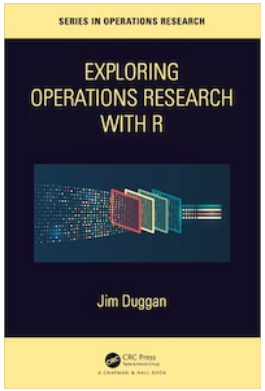
Prof. Jim Duggan,

University of Galway

https://github.com/JimDuggan/SDWorshop

https://github.com/JimDuggan/explore_or

https://github.com/JimDuggan/SDMR



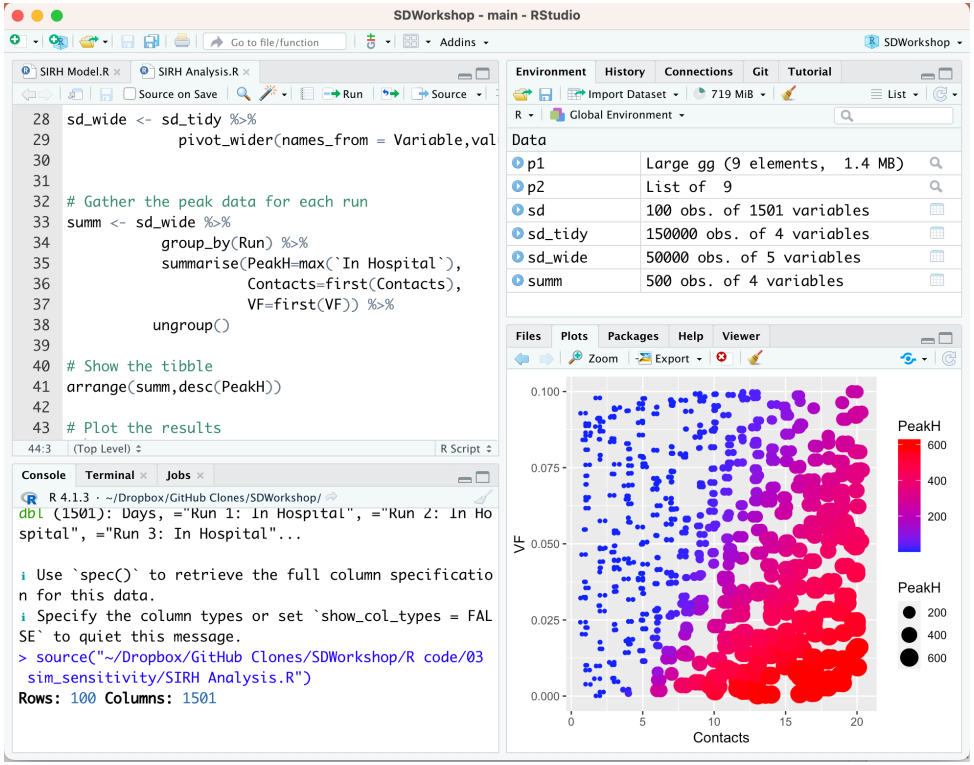
Overall Goals

1. Understand data frames and plot simulation data using **ggplot2**
2. Analyse Stella sensitivity runs with **dplyr**
3. Build a model using **deSolve**
4. Run sensitivity sweep with **purrr, deSolve** and **dplyr**
5. Explore SIR Model with Hospitalisations
6. Challenge: Model implementation and sensitivity analysis

R

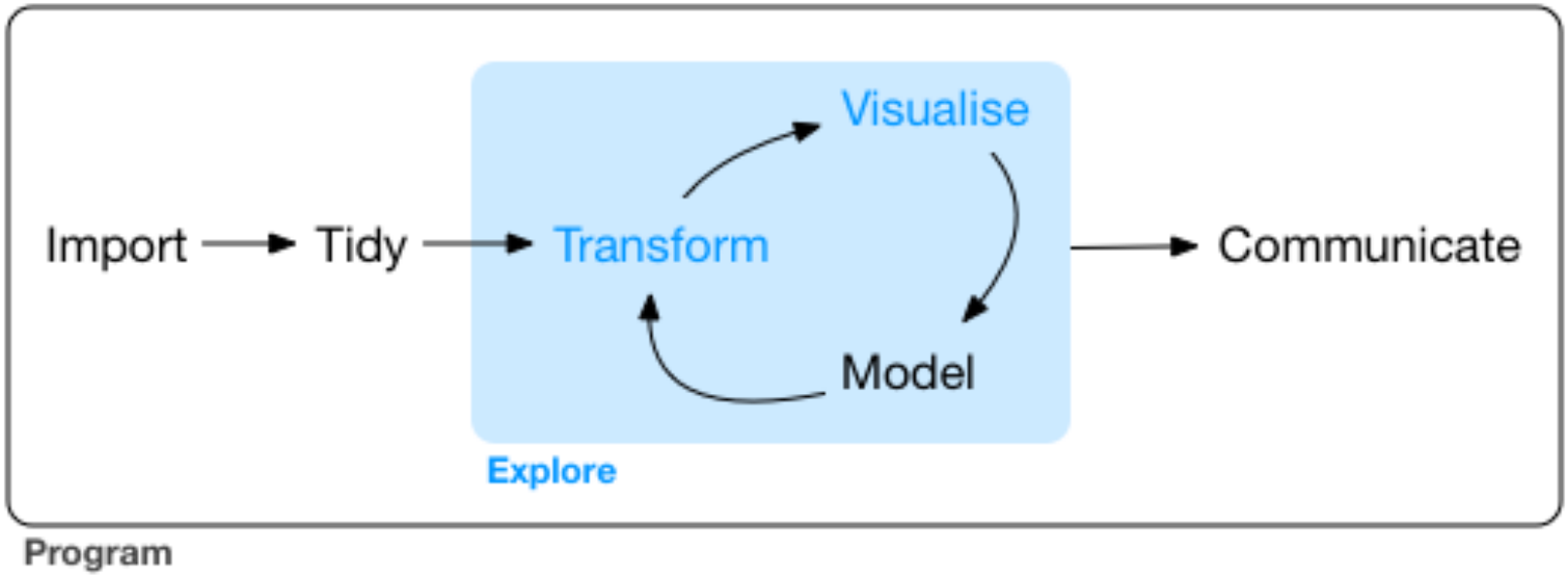
> R is an extremely versatile open source programming language for statistics and data science.
>
> — Norman Matloff (Matloff, 2011)

- R's *mission* is to enable the best and most thorough exploration of data possible (Chambers 2008).
- It is a dialect of the S language, developed at Bell Laboratories
- ACM noted that *S "will forever alter the way people analyze, visualize, and manipulate data"*

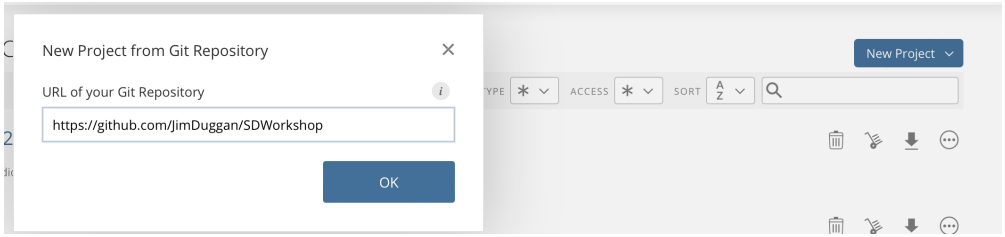


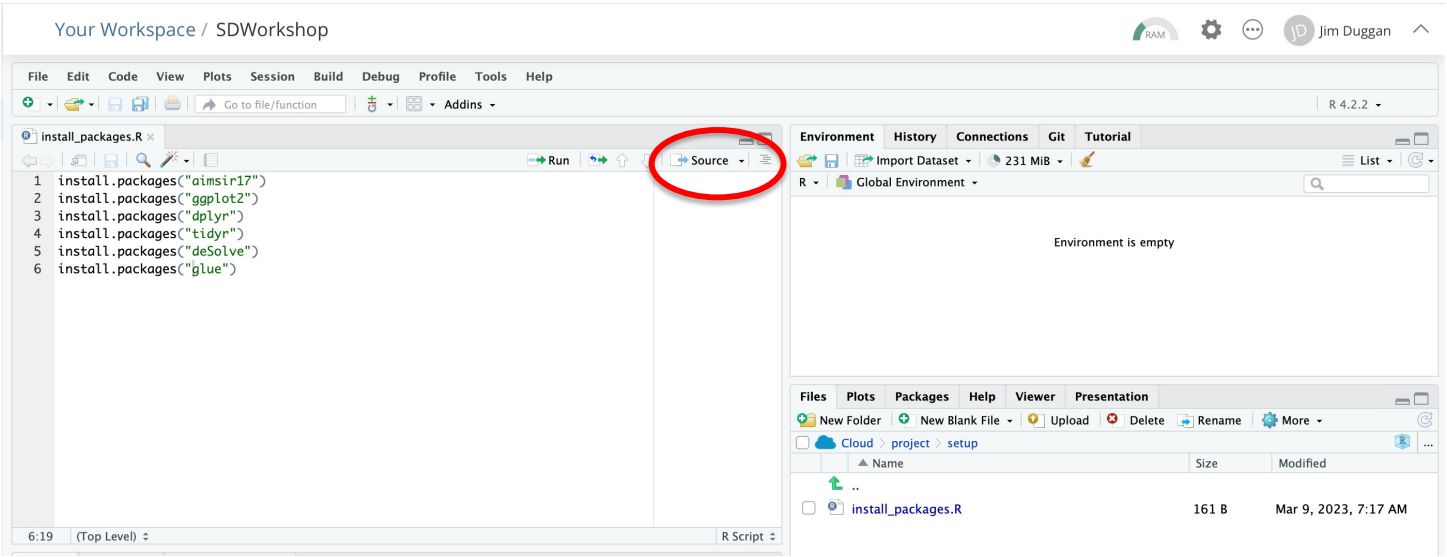
Data Science Process and Workflow

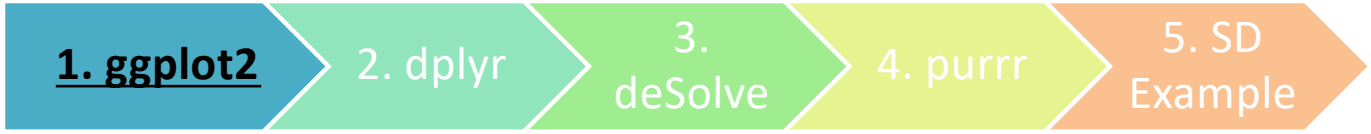
"Data exploration is the art of looking at your data, rapidly generating hypotheses, quickly testing them, then repeating again and again and again." (Wickham and Grolemund 2017).



posit.cloud: Access to workshop resources and setup







- Introducing the data frame (rectangular data)
- Plot graphs
- Import a simulation run from a CSV file (e.g. Stella/Vensim output)
- Plot simulation data

Data Frames/Tibbles

> On an intuitive level, a *data frame* is like a matrix, with a two-dimensional rows-and-columns structure. However, it differs from a matrix in that each column may have a different type.
>
> — Norman Matloff (Matloff, 2011)

- The most common way of storing data in R
- A two-dimensional structure, with rows (observations) and columns (variables)

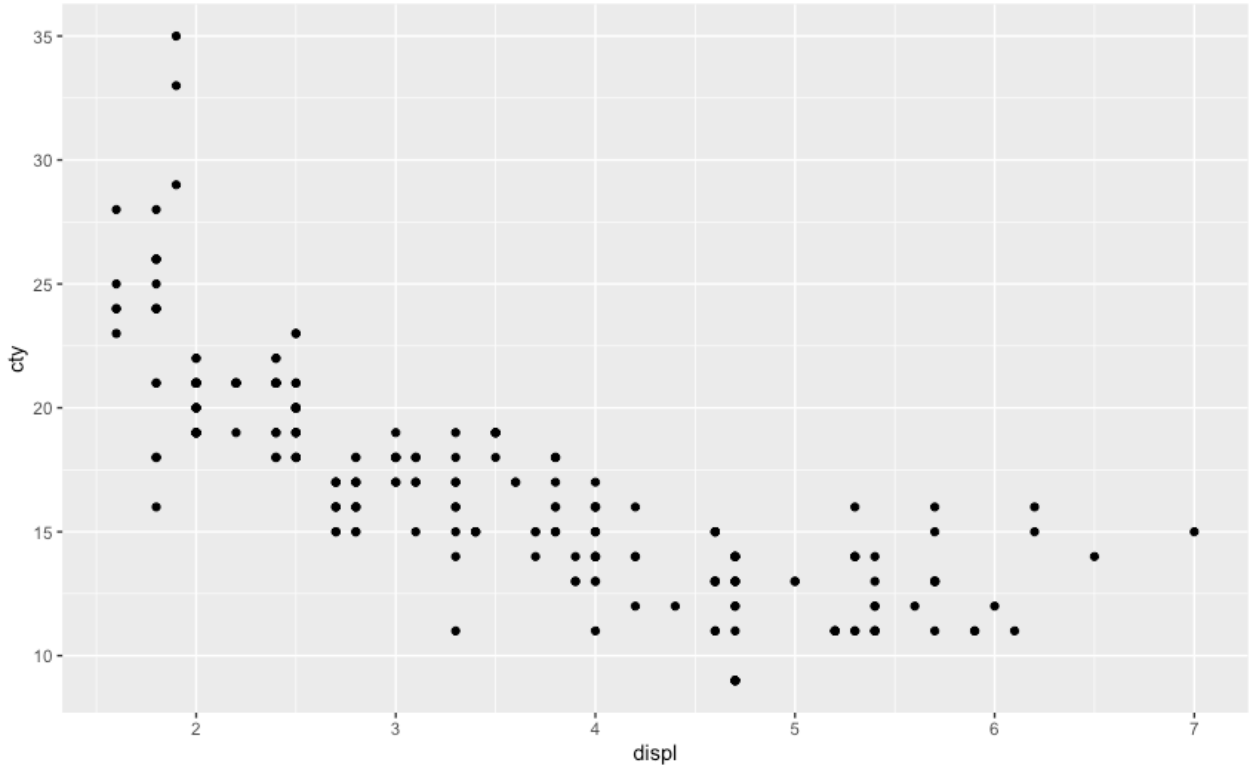
```
> mpg
# A tibble: 234 × 11
   manufacturer model      displ  year   cyl trans      drv     cty   hwy fl    class
   <chr>        <chr>      <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 audi         a4           1.8  1999     4 auto(l5)   f        18    29 p     compact
 2 audi         a4           1.8  1999     4 manual(m5) f        21    29 p     compact
 3 audi         a4           2    2008     4 manual(m6) f        20    31 p     compact
 4 audi         a4           2    2008     4 auto(av)   f        21    30 p     compact
 5 audi         a4           2.8  1999     6 auto(l5)   f        16    26 p     compact
 6 audi         a4           2.8  1999     6 manual(m5) f        18    26 p     compact
 7 audi         a4           3.1  2008     6 auto(av)   f        18    27 p     compact
 8 audi         a4 quattro   1.8  1999     4 manual(m5) 4        18    26 p     compact
 9 audi         a4 quattro   1.8  1999     4 auto(l5)   4        16    25 p     compact
10 audi         a4 quattro   2    2008     4 manual(m6) 4        20    28 p     compact
# … with 224 more rows
# ℹ Use `print(n = ...)` to see more rows
```

Sample Code (engines.R)

```
1  library(ggplot2)
2
3  summary(mpg)
4
5  ggplot(mpg,aes(x=displ,y=cty))+geom_point()
```

```
> summary(mpg)
 manufacturer          model               displ            year          cyl           trans
 Length:234         Length:234         Min.   :1.600   Min.   :1999   Min.   :4.000   Length:234
 Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999   1st Qu.:4.000   Class :character
 Mode  :character   Mode  :character   Median :3.300   Median :2004   Median :6.000   Mode  :character
                                       Mean   :3.472   Mean   :2004   Mean   :5.889
                                       3rd Qu.:4.600   3rd Qu.:2008   3rd Qu.:8.000
                                       Max.   :7.000   Max.   :2008   Max.   :8.000
     drv                 cty              hwy             fl                class
 Length:234         Min.   : 9.00   Min.   :12.00   Length:234         Length:234
 Class :character   1st Qu.:14.00   1st Qu.:18.00   Class :character   Class :character
 Mode  :character   Median :17.00   Median :24.00   Mode  :character   Mode  :character
                    Mean   :16.86   Mean   :23.44
                    3rd Qu.:19.00   3rd Qu.:27.00
                    Max.   :35.00   Max.   :44.00
```

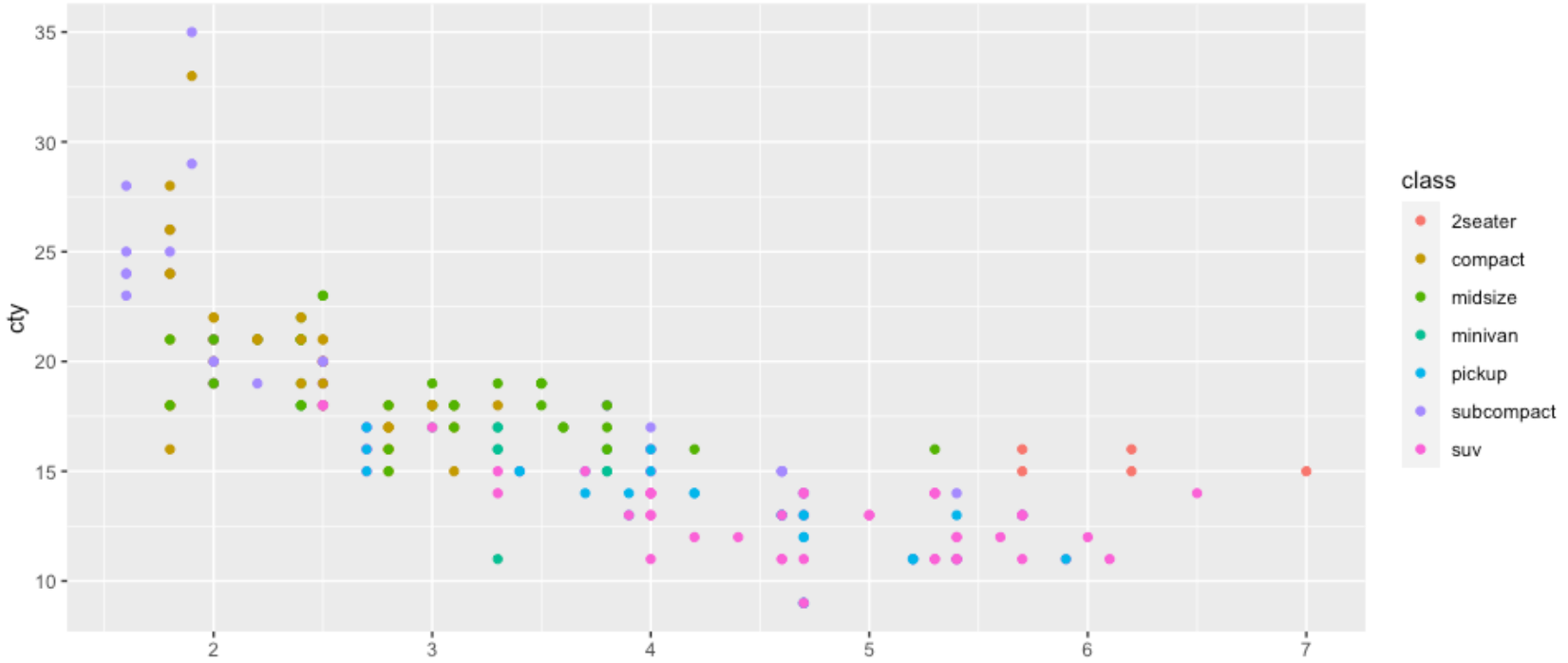
"The simple graph has brought more information to the data analyst's mind that any other device." – John Tukey




```
ggplot(mpg,aes(x=displ,y=cty))+geom_point()
```

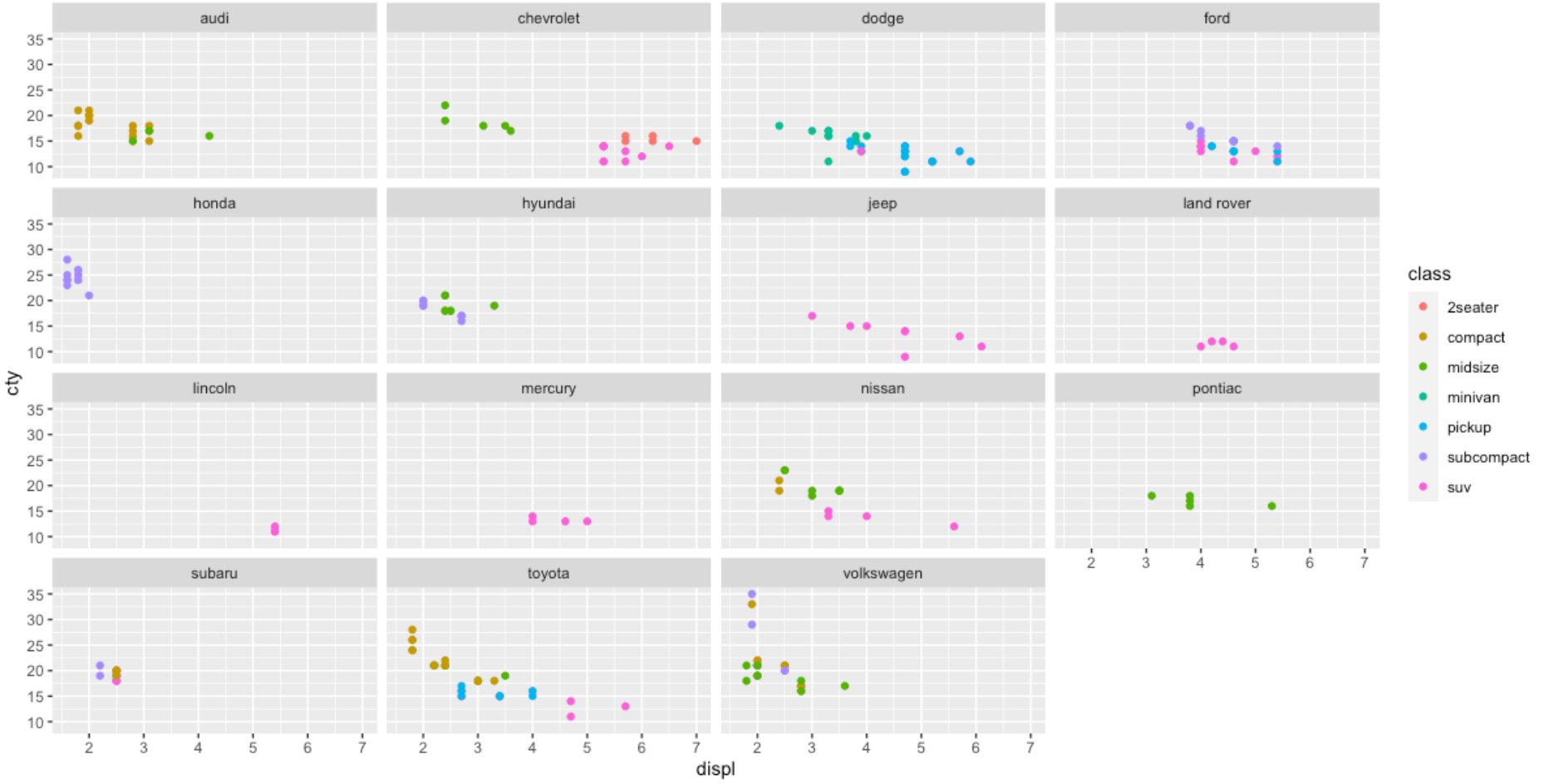
"The greatest value of a picture is when it forces us to notice what we never expected to see" – John Tukey

A third variable can be added to a 2-D plot by mapping it to an aesthetic: colour, size or shape

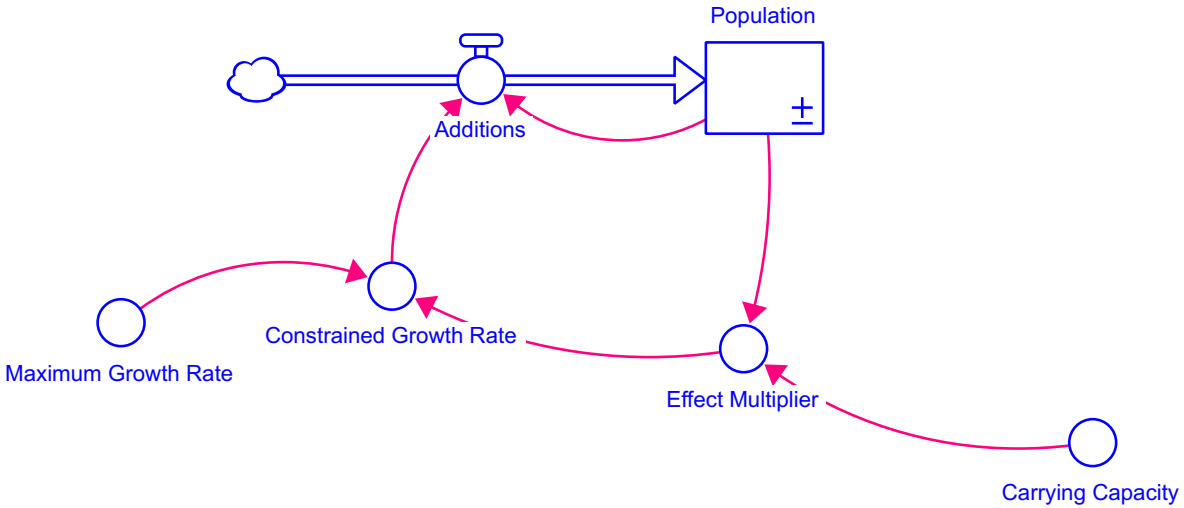


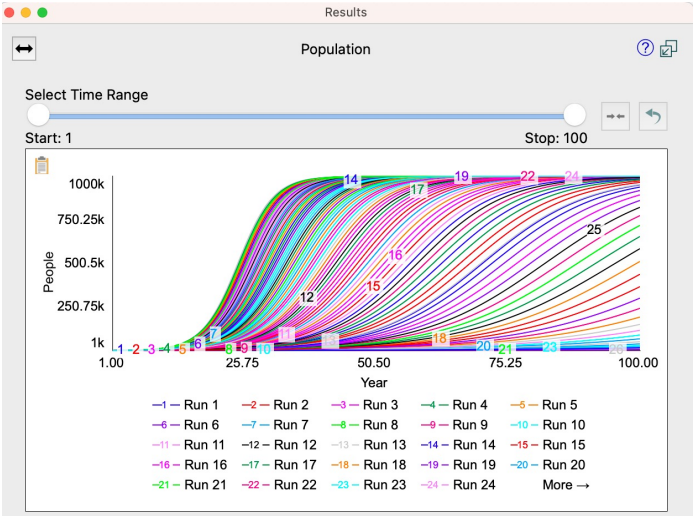

```
ggplot(mpg,aes(x=displ,y=cty,colour=class))+geom_point()
```

```
ggplot(mpg,aes(x=displ,y=cty,colour=class))+geom_point()+facet_wrap(~manufacturer)
```



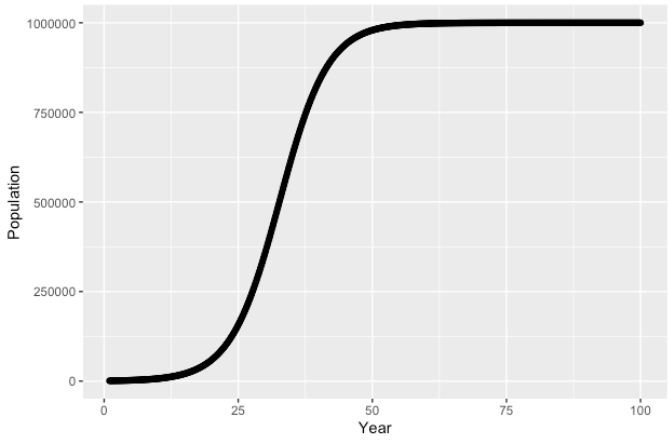
SD Simulation Output – CSV File



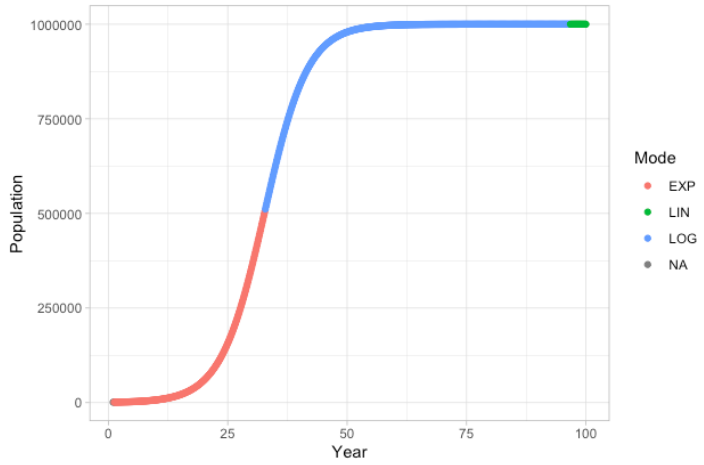


A	B	C	D	E	F	G
Year	Additions	Carrying Capacity	Constrained Growth Rate	Effect Multiplier	Maximum Growth Rate	Population
1	220.2795	1000000	0.2202795	0.999	0.2205	1000
1.125	226.3386436	1000000	0.220273429	0.998972465	0.2205	1027.534938
1.25	232.5641055	1000000	0.22026719	0.998944173	0.2205	1055.827268
1.375	238.9604315	1000000	0.22026078	0.998915102	0.2205	1084.897781
1.5	245.5322907	1000000	0.220254194	0.998885232	0.2205	1114.767835
1.625	252.2844784	1000000	0.220247426	0.998854541	0.2205	1145.459371

```r
library(ggplot2)
library(readr)
library(dplyr)

d <- read_csv("datasets/single runs/LTG.csv")

ggplot(d,aes(x=Year,y=Population)) +
  geom_point()+geom_line()

d <- d %>%
  mutate(ModeValue=round(c(NA,NA,diff(abs(diff(Population)))),3),
         Mode=as.factor(ifelse(ModeValue>0,"EXP",
                       ifelse(ModeValue<0,"LOG","LIN"))))

ggplot(d,aes(x=Year,y=Population,colour=Mode)) +
  geom_point()+theme_light()
```




```
> d
# A tibble: 793 × 9
   Year Additions `Carrying Capacity` `Constrained Growth Rate` Effect …¹ Maxim…² Popul…³ ModeV…⁴ Mode
  <dbl>     <dbl>               <dbl>                     <dbl>     <dbl>   <dbl>   <dbl>   <dbl> <fct>
1  1         220.             1000000                     0.220     0.999   0.220    1000      NA NA
2  1.12      226.             1000000                     0.220     0.999   0.220    1028.     NA NA
3  1.25      233.             1000000                     0.220     0.999   0.220    1056.  0.757 EXP
4  1.38      239.             1000000                     0.220     0.999   0.220    1085.  0.778 EXP
5  1.5       246.             1000000                     0.220     0.999   0.220    1115.  0.8   EXP
6  1.62      252.             1000000                     0.220     0.999   0.220    1145.  0.821 EXP
7  1.75      259.             1000000                     0.220     0.999   0.220    1177.  0.844 EXP
```



A behavioral approach to feedback loop dominance analysis

David N. Ford[a]

linear atomic behavior pattern	$\partial(\lvert(\partial x/\partial t)\rvert)/\partial t = 0$
exponential atomic behavior pattern	$\partial(\lvert(\partial x/\partial t)\rvert)/\partial t > 0$
logarithmic atomic behavior pattern	$\partial(\lvert(\partial x/\partial t)\rvert)/\partial t < 0$

- Visualisation is an important tool for insight generation, but it's rare that you get the data in exactly the right form you need (Wickham and Grolemund 2017)
  - Create new variables
  - Create summaries
  - Order data
- **dplyr** package is designed for data transformation

dplyr Basics: 5 key functions

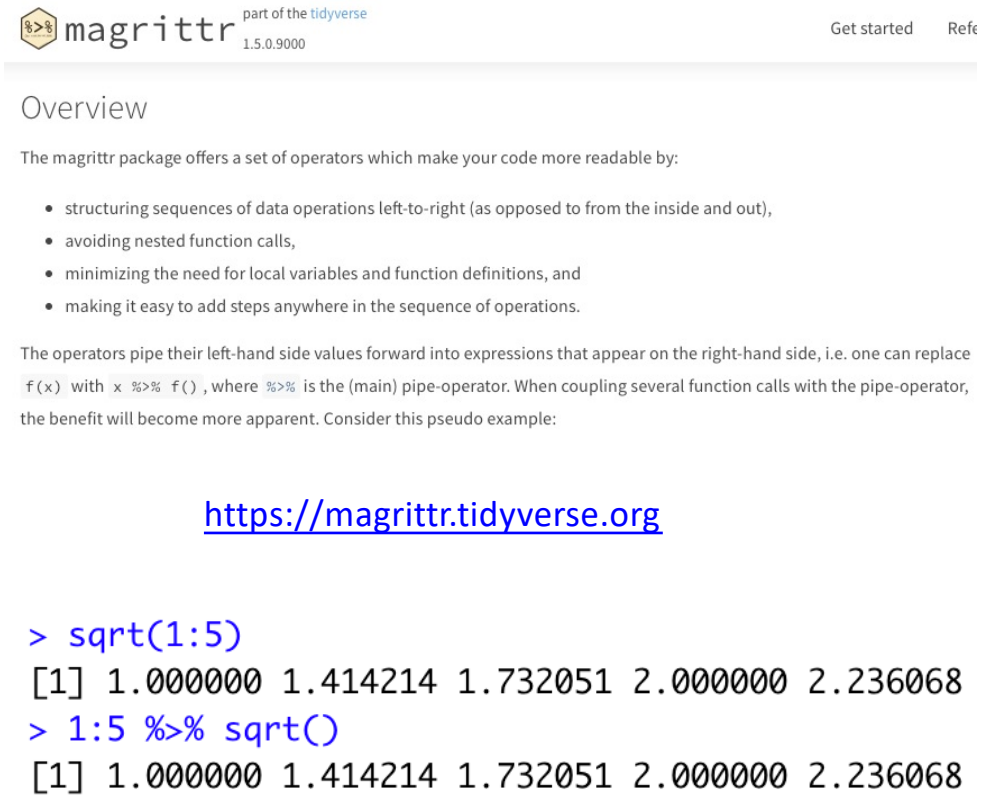
Function	Purpose
filter()	Pick observations by their values
arrange()	Reorder the rows
select()	Pick variables by their names
mutate()	*Create new variables with functions of existing variables*
summarise()	*Collapse many values down to a single summary*

- ”A grammar of data manipulation”
- All verbs (functions) work similarly
  - The first argument is a data frame/tibble
  - The subsequent arguments decide what to do with the data frame/tibble
  - The result (data frame/tibble) supports chaining of steps – NOTE the “pipe operator” which we will cover later.

https://dplyr.tidyverse.org

Useful: Combining operations with the Pipe

- The pipe %>% comes from the magrittr package (Stefan Milton Bache)
- Helps to write code that is easier to read and understand
- x %>% f(y) turns into f(x, y)



https://magrittr.tidyverse.org

```
> sqrt(1:5)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
> 1:5 %>% sqrt()
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

Process Sensitivity Data (Stella)

A	B	C	D	E	F	G	H	I	J
Year	Run 1: Population	Run 2: Population	Run 3: Population	Run 4: Population	Run 5: Population	Run 6: Population	Run 7: Population	Run 8: Population	Run 9: Population
1	1000	1000	1000	1000	1000	1000	1000	1000	1000
2	1104.372133	1108.19136	1172.035949	1231.861465	1056.806787	1249.957467	1271.978986	1001.499482	1085.448046
3	1219.624672	1228.073885	1373.62837	1517.403111	1116.837003	1562.298647	1617.814294	1003.00121	1178.188937
4	1346.888976	1360.907683	1609.840283	1869.011748	1180.273137	1952.539704	2057.489958	1004.505187	1278.843588
5	1487.413409	1508.087943	1886.596445	2301.911204	1247.307964	2440.02602	2616.353021	1006.011418	1388.085504

```
> head(sd)
# A tibble: 6 × 203
   Year ="Run …¹ ="Run…² ="Run…³ ="Run…⁴ ="Run…⁵ ="Run…⁶ ="Run…⁷ ="Run…⁸ ="Run…⁹ ="Run…× ="Run…× ="Run…×
  <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
1     1     1000    1000    1000    1000    1000    1000    1000    1000    1000    1000    1000    1000
2     2     1104.   1108.   1172.   1232.   1057.   1250.   1272.   1001.   1085.   1017.   1152.   1179.
3     3     1220    1228    1374    1517    1117    1562    1618    1003    1178    1033    1326    1390
```

```
1  library(readr)
2  library(dplyr)
3  library(ggplot2)
4  library(stringr)
5
6  # Load in the sensitivity data
7  sd <- read_csv("datasets/sensitivity runs/LTG_Sensitivity.csv")
```

Convert wide data to tidy data

```
> sd_tidy
# A tibble: 20,200 × 4
     Run  Year Variable   Value
   <int> <dbl> <chr>      <dbl>
 1     1     1 Population  1000
 2     2     1 Population  1000
 3     3     1 Population  1000
 4     4     1 Population  1000
 5     5     1 Population  1000
 6     6     1 Population  1000
 7     7     1 Population  1000
 8     8     1 Population  1000
 9     9     1 Population  1000
10    10     1 Population  1000
# … with 20,190 more rows
```

- The tidy data standard is designed to:
  - Facilitate initial exploration and analysis of data
  - Simplify the development of data analysis tools that work well together
- Rules
  - Each variable must have its own column
  - Each observation must have its own row
  - Each value must have its own cell

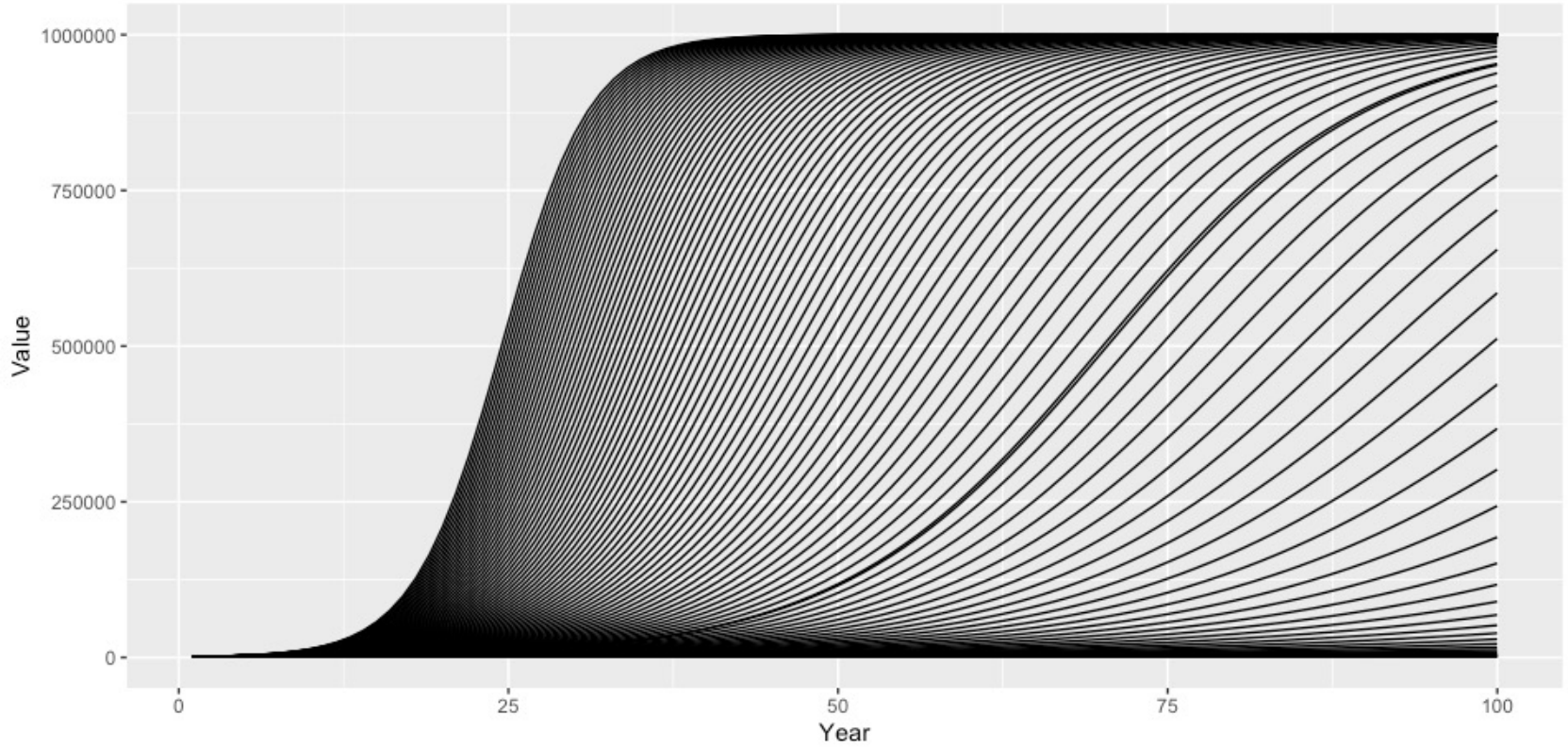
Code (tidyr and dplyr)

```
# Convert to tidy data format
sd_tidy <- sd %>%
             pivot_longer(cols=!Year,names_to="Variable",values_to="Value") %>%
             mutate(Variable=str_replace_all(Variable,'\"',""),
                    Variable=str_replace(Variable,"=","")) %>%
             separate(Variable,into=c("Run","Variable"),sep = ":") %>%
             mutate(Variable=trimws(Variable)) %>%
             separate(Run,c("TempRun","Run"),sep = " ",convert = TRUE) %>%
             select(Run,Year,Variable,Value)
```

Tidy data makes it easier to leverage tools such as ggplot2 and dplyr

Plotting multiple runs with ggplot2

```
19  ggplot(filter(sd_tidy,Variable=="Population"),aes(x=Year,y=Value,group=Run))+geom_line()
```



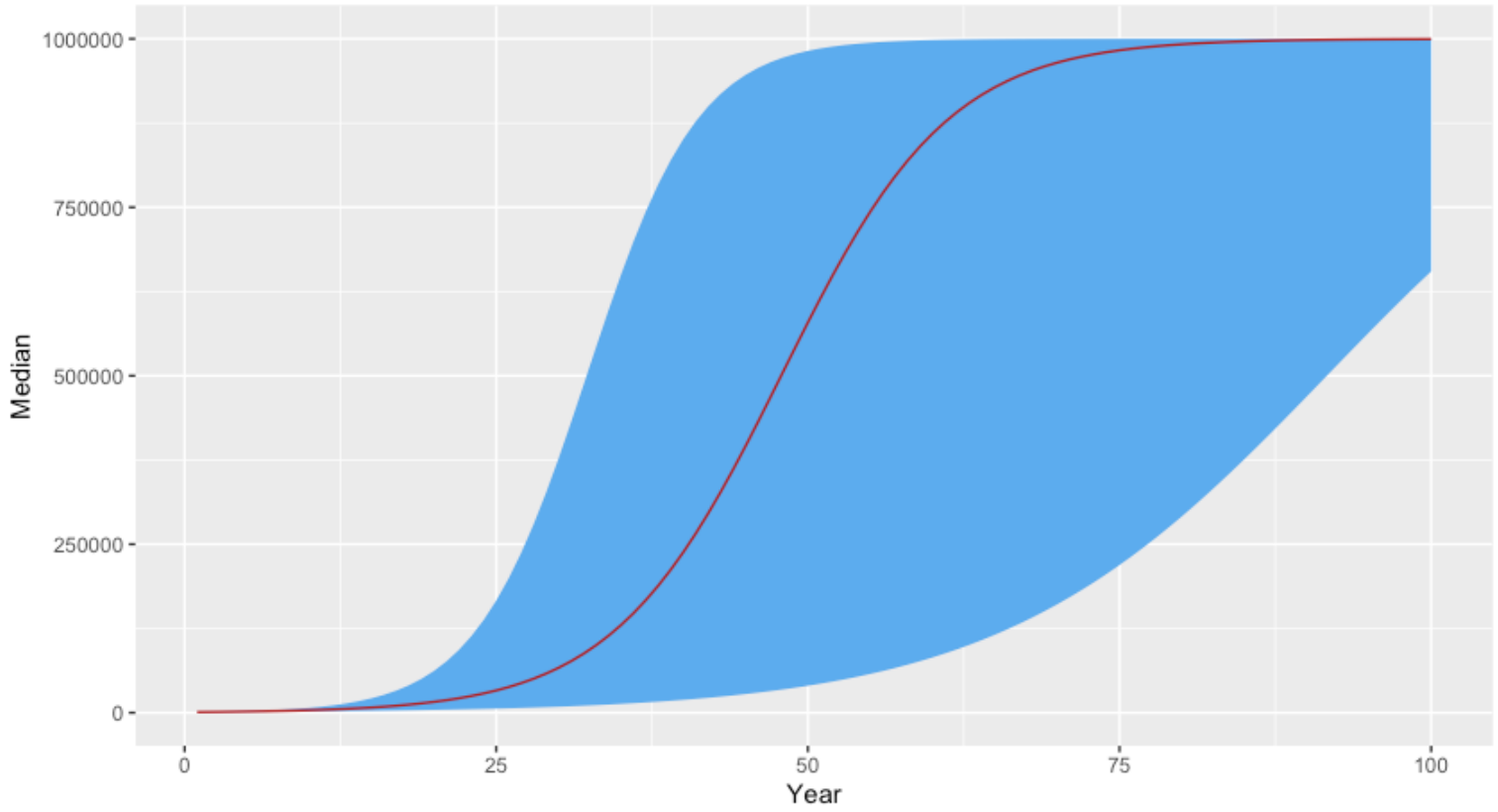
Finding summary data - quantiles

```
21  sum_runs <- sd_tidy %>%
22                filter(Variable=="Population") %>%
23                group_by(Variable,Year) %>%
24                summarise(Q75=quantile(Value,0.75),
25                          Q25=quantile(Value,0.25),
26                          Median=median(Value),
27                          Mean=mean(Value))
```

Finding summary data - quantiles

```
> sum_runs
# A tibble: 100 × 6
# Groups:   Variable [1]
   Variable    Year   Q75   Q25 Median  Mean
   <chr>      <dbl> <dbl> <dbl>  <dbl> <dbl>
 1 Population     1 1000  1000   1000  1000
 2 Population     2 1246. 1079.  1158. 1163.
 3 Population     3 1553. 1164.  1342. 1362.
 4 Population     4 1936. 1256.  1554. 1607.
 5 Population     5 2412. 1356.  1800. 1909.
 6 Population     6 3005. 1463.  2085. 2282.
 7 Population     7 3743. 1578.  2414. 2745.
 8 Population     8 4662. 1703.  2796. 3322.
 9 Population     9 5805. 1837.  3238. 4043.
10 Population    10 7227. 1982.  3749. 4945.
# … with 90 more rows
# ℹ Use `print(n = ...)` to see more rows
```

```
30 ggplot(sum_runs,aes(Year, Median)) +
31   geom_ribbon(aes(ymin = Q25,ymax = Q75),fill = "steelblue2") +
32   geom_line(color = "firebrick")
```



R's deSolve package solves initial value problems written as ordinary differential equations (ODE), differential algebraic equations (DAE), and partial differential equations (PDE)

Journal of Statistical Software
February 2010, Volume 33, Issue 9. *http://www.jstatsoft.org/*

Solving Differential Equations in R: Package deSolve

Karline Soetaert
Netherlands Institute of Ecology

Thomas Petzoldt
Technische Universität Dresden

R. Woodrow Setzer
US Environmental Protection Agency

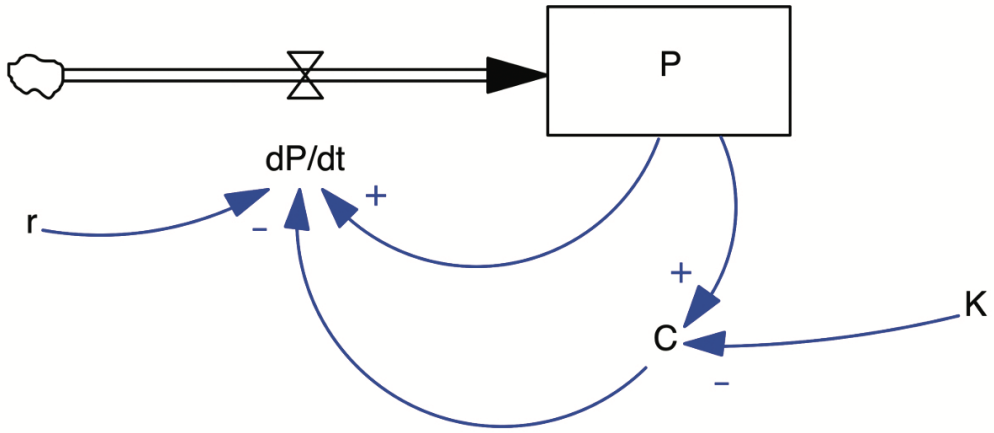
The model is implemented as an R function, and called via ode()

ode function in deSolve

Argument	Description
`y`	The initial (state) values for the ODE system, a vector. If *y* has a name attribute, the names will be used to label the output matrix
`times`	Time sequence for which output is wanted; the first value of times must be the initial time
`func`	• An R function that computes the values of the derivatives in the ODE system at time *t*. It must be defined as: `func <- function(t, y, parms,...)`, where **t** is the current time point in the integration and **y** is the current estimate of the variables in the ODE system. • If the initial value **y** has a names attribute, the names will be available inside func. • `parms` is a vector or list of parameters; ... (optional) are any other arguments passed to the function. • The return value of `func` should be a list, whose first element is a vector containing the derivatives of `y` with respect to time, and whose next elements are global values that are required at each point in times. The derivatives must be specified in the same order as the state variables `y`.
`parms`	Parameters passed to `func`.
`method`	Normally a string to indicate the integration method, for example, `"euler"`, `"rk4"`, `"ode23"`, `"ode45"`.
`returns`	A matrix of class `deSolve` with up to as many rows as elements in times and as many columns as elements in *y* plus the number of "global" values returned in the second element of the return from `func`, plus an additional column (the first) for the time value. This can be easily converted to a data frame object using the function `data.frame()`

```
res <- ode(y=stocks,
           times=simtime,
           func = ltg,
           parms=auxs,
           method="euler")
```

Limits to Growth Model (Verhulst Equation)



$$\frac{dP}{dt} = r\,P\,(1 - C) \quad (1)$$

$$C = {}^{P}/_{K} \quad (2)$$

$$r = 0.15 \quad (3)$$

$$K = 100{,}000 \quad (4)$$

$$P_{INIT} = 100 \quad (5)$$

Include R libraries

```
library(deSolve)
library(dplyr)
library(ggplot2)
library(tidyr)
library(purrr)
library(ggpubr)
```

The LTG model implementation

```
ltg <- function(time, stocks, auxs){
  with(as.list(c(stocks, auxs)),{
    C <- P/K                # Eq (2)
    dP_dt <- r*P*(1-C)      # Eq (1)
    return (list(c(dP_dt),
                 r=r,
                 K=K,
                 C=C,
                 Flow=dP_dt))
  })
}
```

Setting up parameters and initial conditions

```
simtime <- seq(0,100,by=0.25)
stocks  <- c(P=100)              # Eq (5)
auxs    <- c(r=0.15,K=100000)    # Eq (3) and Eq (4)
```

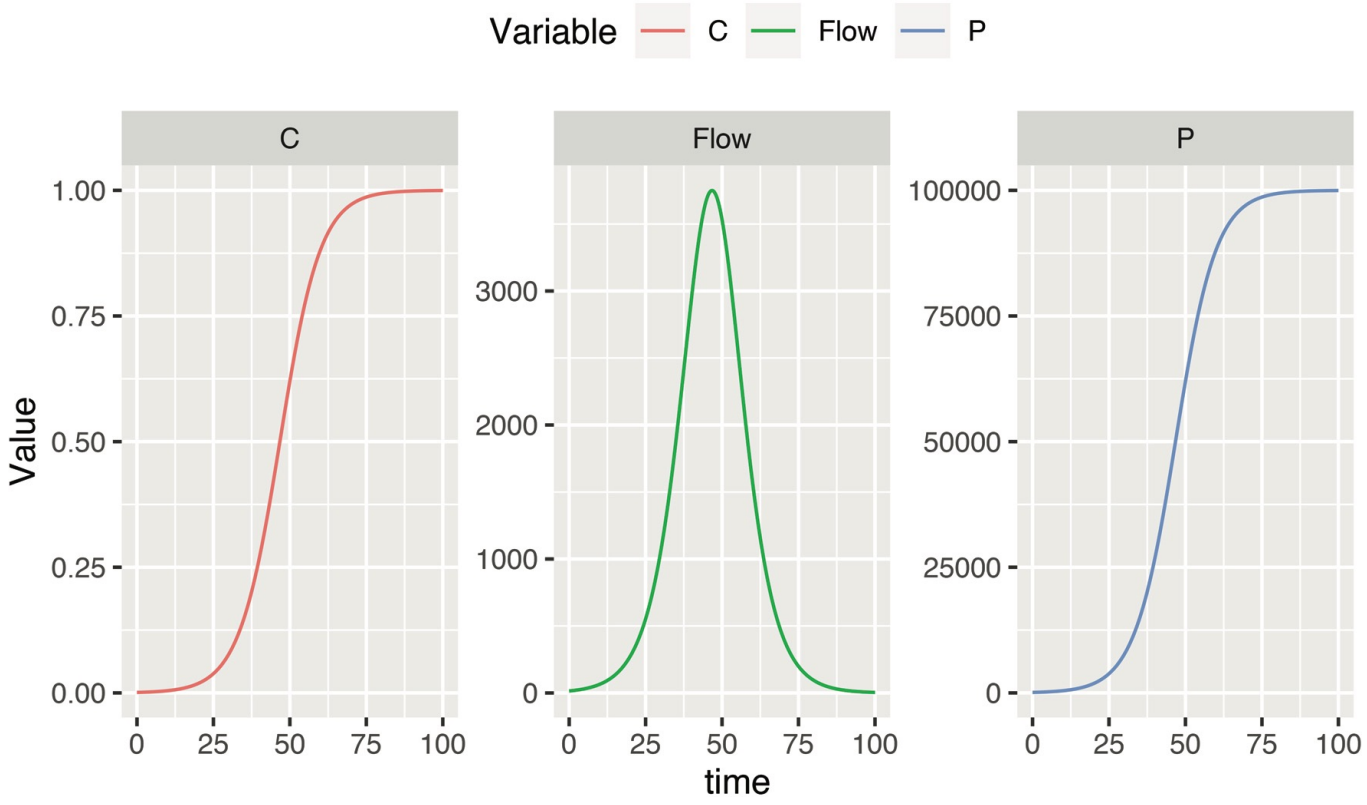
Running the model

```
res <- ode(y=stocks,
           times=simtime,
           func = ltg,
           parms=auxs,
           method="euler") %>%
        data.frame() %>%
        dplyr::as_tibble()
res
#> # A tibble: 401 x 6
#>     time     P     r      K       C  Flow
#>    <dbl> <dbl> <dbl>  <dbl>   <dbl> <dbl>
#>  1  0     100   0.15 100000 0.001    15.0
#>  2  0.25  104.  0.15 100000 0.00104  15.5
#>  3  0.5   108.  0.15 100000 0.00108  16.1
#>  4  0.75  112.  0.15 100000 0.00112  16.7
#>  5  1     116.  0.15 100000 0.00116  17.4
```

Visualising the Results

```
res_long <- res %>%
              dplyr::select(time,C,P,Flow) %>%
              tidyr::pivot_longer(names_to = "Variable",
                                  values_to = "Value",
                                  -time)

ggplot(res_long,aes(x=time,y=Value,color=Variable)) +
  geom_line() + facet_wrap(~Variable,scales = "free")+
  theme(legend.position = "top")
```

Model output



- A **map function** is one that applies the same action/function to every element of an object (e.g. each entry of a list or a vector, or each of the columns of a data frame)
- The naming convention of the map functions are such that the type of the **output** is specified by the term that follows the underscore in the function name
- Consistent with the way of the tidyverse, the **first argument** of each mapping function is always the **data object** that you want to map over, and the **second argument** is always the ***function*** that you want to iteratively apply to each element of the input object

- `map(.x, .f)` is the main mapping function and returns a list
- `map_df(.x, .f)` returns a data frame
- `map_dbl(.x, .f)` returns a numeric (double) vector
- `map_chr(.x, .f)` returns a character vector
- `map_lgl(.x, .f)` returns a logical vector

purrr's map functions support iteration Sensitivity Analysis for SD Models

Examples

```
library(purrr)

o1 <- purrr::map(c(1,2,3,2),function(x)x^2)
str(o1)
#> List of 4
#>  $ : num 1
#>  $ : num 4
#>  $ : num 9
#>  $ : num 4
```

```
o2 <- purrr::map(c(1,2,3,2),~.x^2)
str(o2)
#> List of 4
#>  $ : num 1
#>  $ : num 4
#>  $ : num 9
#>  $ : num 4
```

(1) Wrap the SD model in a flexible function

```
run_scenario <- function(run_id=1,
                         P=100,
                         simtime=seq(0,100,by=0.25),
                         r=0.15,
                         K=100000){

  auxs    <- c(r=r,K=K)
  stocks <- c(P=P)

  res <- ode(y=stocks,
             times=simtime,
             func = ltg,
             parms=auxs,
             method="euler") %>%
    data.frame() %>%
    dplyr::as_tibble() %>%
    dplyr::mutate(RunID=as.integer(run_id)) %>%
    dplyr::select(RunID, everything())
}
```

(2) Setup the inputs for the experiment (21^2 = 441)

```
r_vals <- seq(0,.20,length.out=21)
p_vals  <- seq(0,1000,length.out=21)
sim_inputs <- expand.grid(r_vals,p_vals)
```

```
> head(sim_inputs)
  Var1 Var2
1 0.00    0
2 0.01    0
3 0.02    0
4 0.03    0
5 0.04    0
6 0.05    0
>
>
> tail(sim_inputs)
    Var1 Var2
436 0.15 1000
437 0.16 1000
438 0.17 1000
439 0.18 1000
440 0.19 1000
441 0.20 1000
```

(3) Use map to run the simulations

```
run_id <- 1

sim_res <- map2(sim_inputs[,1],
                sim_inputs[,2],~{
                  res <- run_scenario(run_id  = run_id,
                                      r    = .x,
                                      P    = .y,
                                      simtime = seq(0,100,by=0.25))
                  run_id <<- run_id + 1
                  res
                }) %>% dplyr::bind_rows()
sim_res
```

Results (a large tibble)

```
> sim_res
# A tibble: 176,841 × 7
   RunID  time     P     r      K     C  Flow
   <int> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>
 1     1  0         0     0 100000     0     0
 2     1  0.25      0     0 100000     0     0
 3     1  0.5       0     0 100000     0     0
 4     1  0.75      0     0 100000     0     0
 5     1  1         0     0 100000     0     0
 6     1  1.25      0     0 100000     0     0
 7     1  1.5       0     0 100000     0     0
 8     1  1.75      0     0 100000     0     0
 9     1  2         0     0 100000     0     0
10     1  2.25      0     0 100000     0     0
# … with 176,831 more rows
# ℹ Use `print(n = ...)` to see more rows
```

```
> summary(sim_res)
     RunID          time           P                r                K
 Min.   :  1   Min.   :  0   Min.   :     0   Min.   :0.00   Min.   :1e+05
 1st Qu.:111   1st Qu.: 25   1st Qu.:  1189   1st Qu.:0.05   1st Qu.:1e+05
 Median :221   Median : 50   Median : 11023   Median :0.10   Median :1e+05
 Mean   :221   Mean   : 50   Mean   : 36747   Mean   :0.10   Mean   :1e+05
 3rd Qu.:331   3rd Qu.: 75   3rd Qu.: 87652   3rd Qu.:0.15   3rd Qu.:1e+05
 Max.   :441   Max.   :100   Max.   :100000   Max.   :0.20   Max.   :1e+05
       C                Flow
 Min.   :0.00000   Min.   :   0.0
 1st Qu.:0.01189   1st Qu.:  15.5
 Median :0.11023   Median : 159.2
 Mean   :0.36747   Mean   : 673.9
 3rd Qu.:0.87652   3rd Qu.: 985.7
 Max.   :1.00000   Max.   :5000.0
```

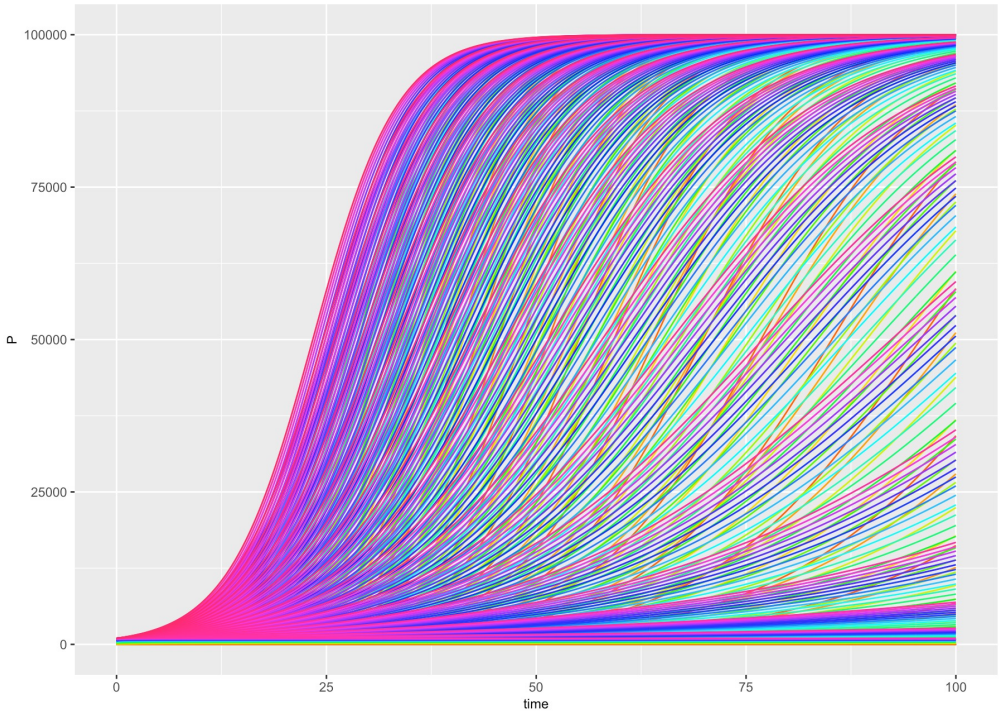
Calculating quantiles

```
sum_vals <- sim_res %>%
  dplyr::group_by(time) %>%
  dplyr::summarize(MeanP=mean(P),
                   MedianP=median(P),
                   Q75=quantile(P,0.75),
                   Q25=quantile(P,0.25))
sum_vals
```

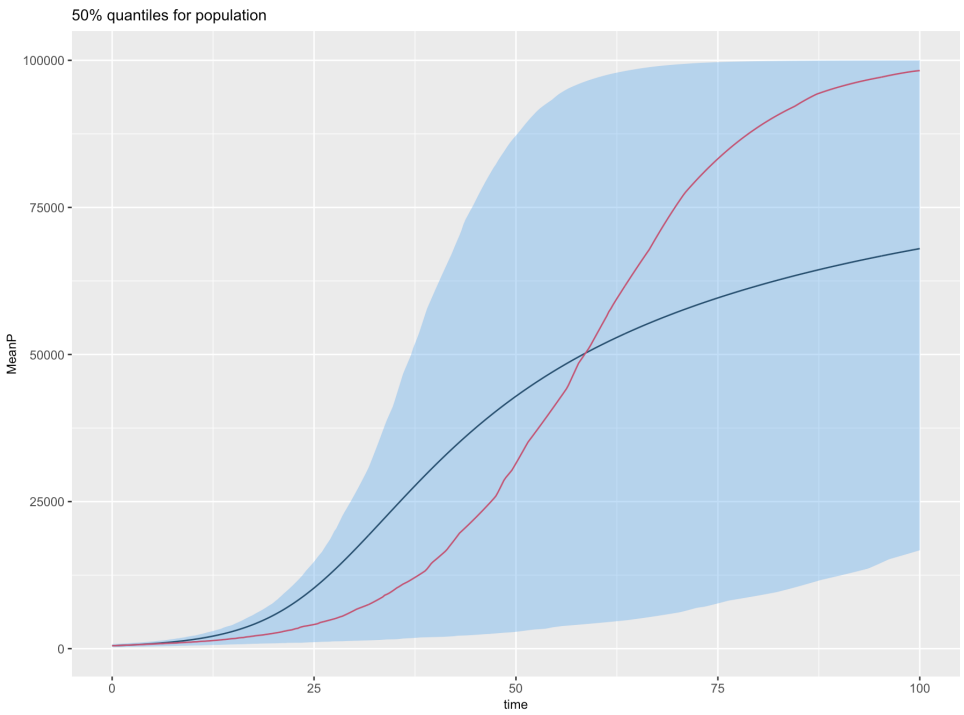
```
> sum_vals
# A tibble: 401 × 5
    time MeanP MedianP   Q75   Q25
   <dbl> <dbl>   <dbl> <dbl> <dbl>
 1  0     500     500   750   250
 2  0.25  512.    512.  778.  253.
 3  0.5   525.    525.  803.  256.
 4  0.75  539.    538.  818.  259.
 5  1     552.    550   840.  263.
 6  1.25  566.    564.  861.  266.
 7  1.5   581.    577.  876.  269.
 8  1.75  596.    591.  900   277.
 9  2     612.    603.  920.  287.
10  2.25  628.    615.  941.  297.
# … with 391 more rows
# ℹ Use `print(n = ...)` to see more rows
```

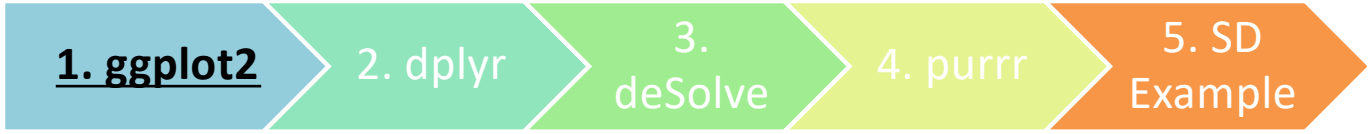
Plots

```
ggplot(sim_res,aes(x=time,y=P,color=RunID,group=RunID))+
  geom_line()+
  scale_color_gradientn(colors=rainbow(14))+
  theme(legend.position = "none")+
  labs(title="Population")+
  theme(title = element_text(size=9))
```



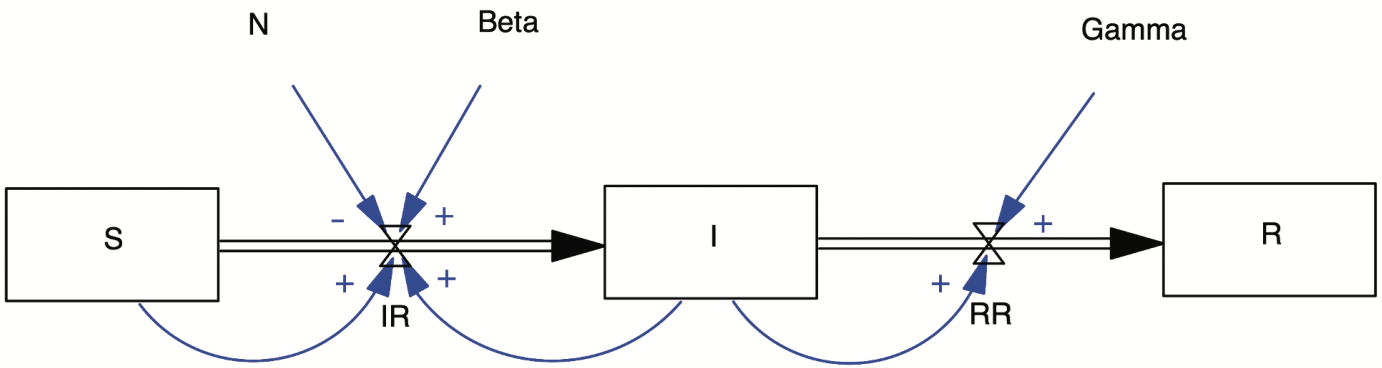

```
ggplot(sum_vals,aes(x=time,y=MeanP))+geom_line()+
  geom_line(aes(y=MedianP),colour="red")+
  geom_ribbon(aes(x=time,ymin=Q25,ymax=Q75),
              alpha=0.4,fill="steelblue2")+
  labs(title="50% quantiles for population")+
  theme(title = element_text(size=9))
```





- SIR Model
- SIRH Model
- Policy Exploration using R

(a) SIR stock and flow model



(b) SIR differential equation model

$$\frac{dS}{dt} = -IR \quad (6) \qquad \frac{dI}{dt} = IR - RR \quad (7) \qquad \frac{dR}{dt} = RR \quad (8)$$

$$IR = \beta\, I\, \frac{S}{N} \quad (9) \qquad RR = I\,\gamma \quad (10) \qquad N = 10{,}000 \quad (11)$$

$$\gamma = 0.25 \quad (12) \qquad \beta = 1.0 \quad (13) \qquad S_{INIT} = 9999 \quad (14)$$

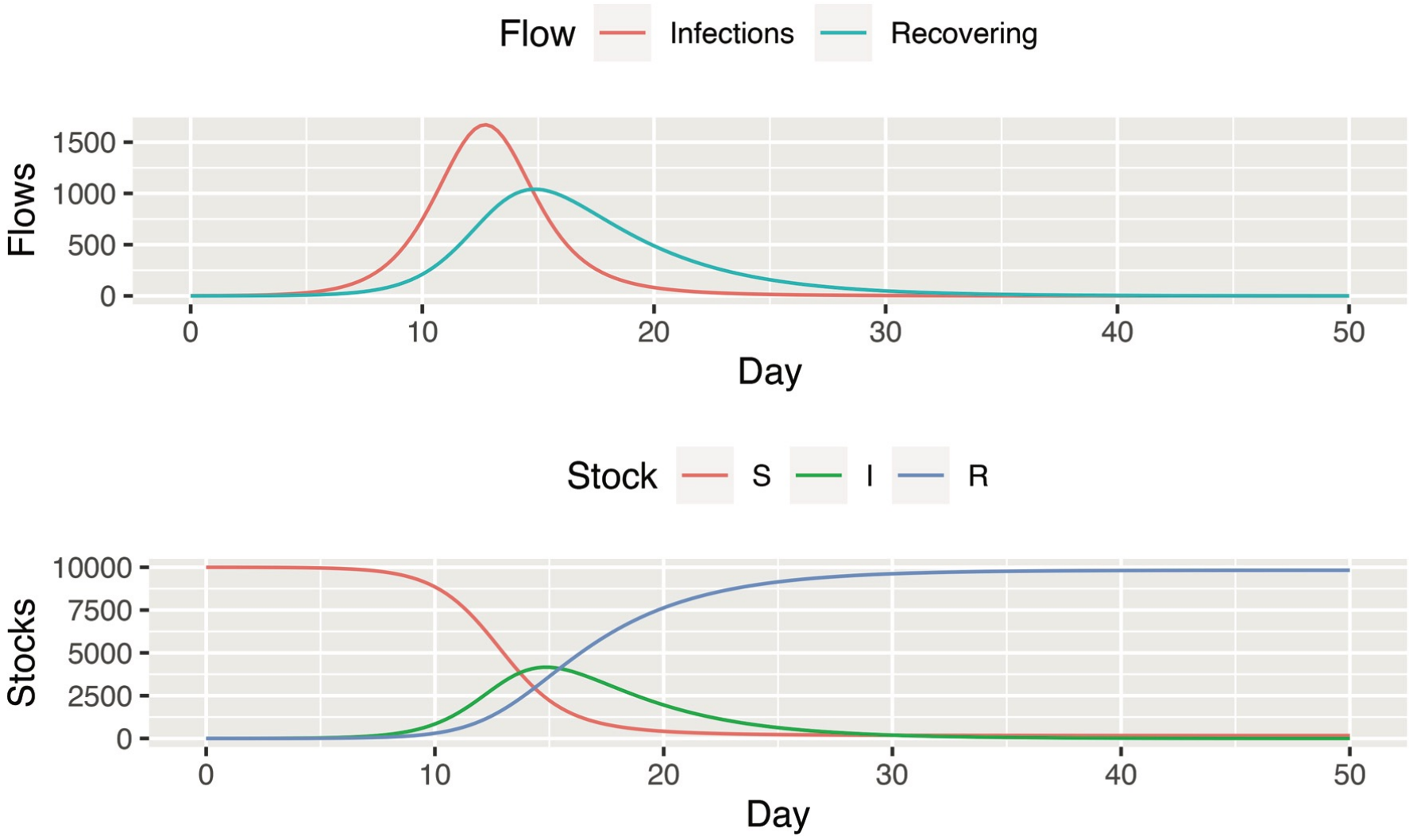
$$I_{INIT} = 1 \quad (15) \qquad R_{INIT} = 0 \quad (16)$$

```r
sir <- function(time, stocks, auxs){
  with(as.list(c(stocks, auxs)),{
    N     <- 10000          # Eq (11)
    IR    <- beta*I*S/N     # Eq (9)
    RR    <- gamma*I        # Eq (10)
    dS_dt <- -IR            # Eq (6)
    dI_dt <- IR - RR        # Eq (7)
    dR_dt <- RR             # Eq (8)
    return (list(c(dS_dt,dI_dt,dR_dt),
                 Beta=beta,
                 Gamma=gamma,
                 Infections=IR,
                 Recovering=RR))
  })
}
```

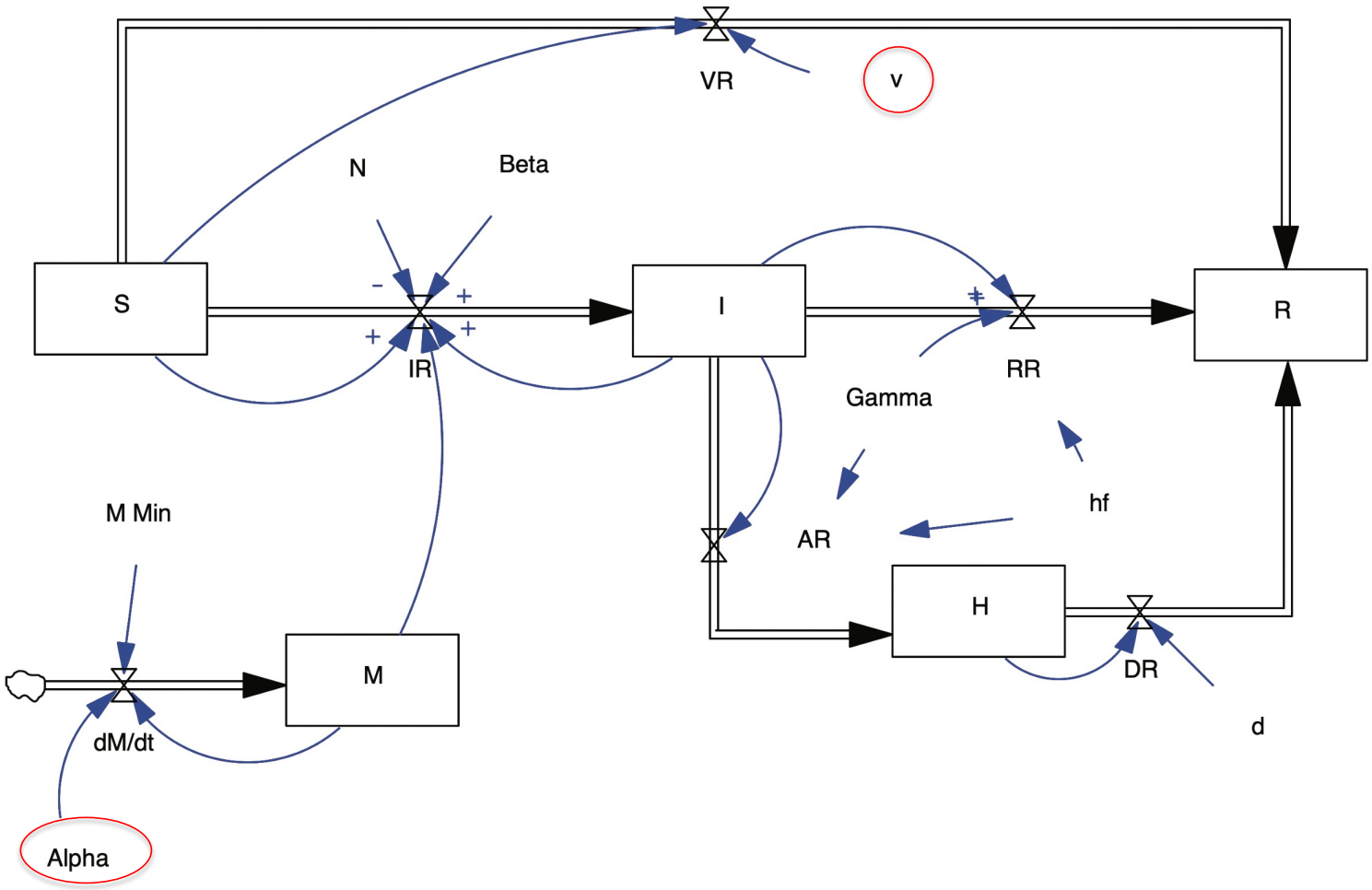
```r
simtime <- seq(0,50,by=0.25)
stocks  <- c(S=9999,I=1,R=0)
auxs    <- c(gamma=0.25,beta=1)
```

```r
res <- ode(y=stocks,
           times=simtime,
           func = sir,
           parms=auxs,
           method="euler") %>%
       data.frame() %>%
       dplyr::as_tibble()
res
```

```
res
#> # A tibble: 201 x 8
#>     time     S     I      R  Beta Gamma Infections Recovering
#>    <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>      <dbl>      <dbl>
#>  1  0    9999   1    0          1  0.25       1.00      0.25
#>  2  0.25 9999.  1.19 0.0625     1  0.25       1.19      0.297
#>  3  0.5  9998.  1.41 0.137      1  0.25       1.41      0.353
#>  4  0.75 9998.  1.67 0.225      1  0.25       1.67      0.419
#>  5  1    9998.  1.99 0.329      1  0.25       1.99      0.497
#>  6  1.25 9997.  2.36 0.454      1  0.25       2.36      0.590
#>  7  1.5  9997.  2.80 0.601      1  0.25       2.80      0.701
#>  8  1.75 9996.  3.33 0.777      1  0.25       3.33      0.832
#>  9  2    9995.  3.95 0.985      1  0.25       3.95      0.988
#> 10  2.25 9994.  4.69 1.23       1  0.25       4.69      1.17
#> # ... with 191 more rows
```


Flow
Infections
Recovering
Flows
1500
1000
500
0
0
10
20
30
40
50
Day
Stock
S
I
R
Stocks
10000
7500
5000
2500
0
0
10
20
30
40
50
Day

(a) SIRH stock and flow model



(b) SIRH differential equation model

$$\frac{dS}{dt} = -IR - VR \quad (17)$$

$$\frac{dI}{dt} = IR - RR - AR \quad (18)$$

$$\frac{dH}{dt} = AR - DR \quad (19)$$

$$\frac{dR}{dt} = RR + DR + VR \quad (20)$$

$$\frac{dM}{dt} = \alpha\,(M_{min} - M) \quad (21)$$

$$IR = \beta\, M\, I\, \frac{S}{N} \quad (22)$$

$$VR = v\, S \quad (23)$$

$$AR = h_f I\, \gamma \quad (24)$$

$$RR = (1 - h_f) I\, \gamma \quad (25)$$

$$DR = H\, d \quad (26)$$

$$\beta = 1.0 \quad (27)$$

$$\gamma = 0.25 \quad (28)$$

$$v = 0.1 \quad (29)$$

$$h_f = 0.1 \quad (30)$$

$$N = 10{,}000 \quad (31)$$

$$d = 0.10 \quad (32)$$

$$\alpha = 0.5 \quad (33)$$

$$S_{INIT} = 9999 \quad (34)$$

$$I_{INIT} = 1 \quad (35)$$

$$H_{INIT} = 0 \quad (36)$$

$$R_{INIT} = 0 \quad (37)$$

$$M_{INIT} = 1 \quad (38)$$

$$M_{MIN} = 0.3 \quad (39)$$

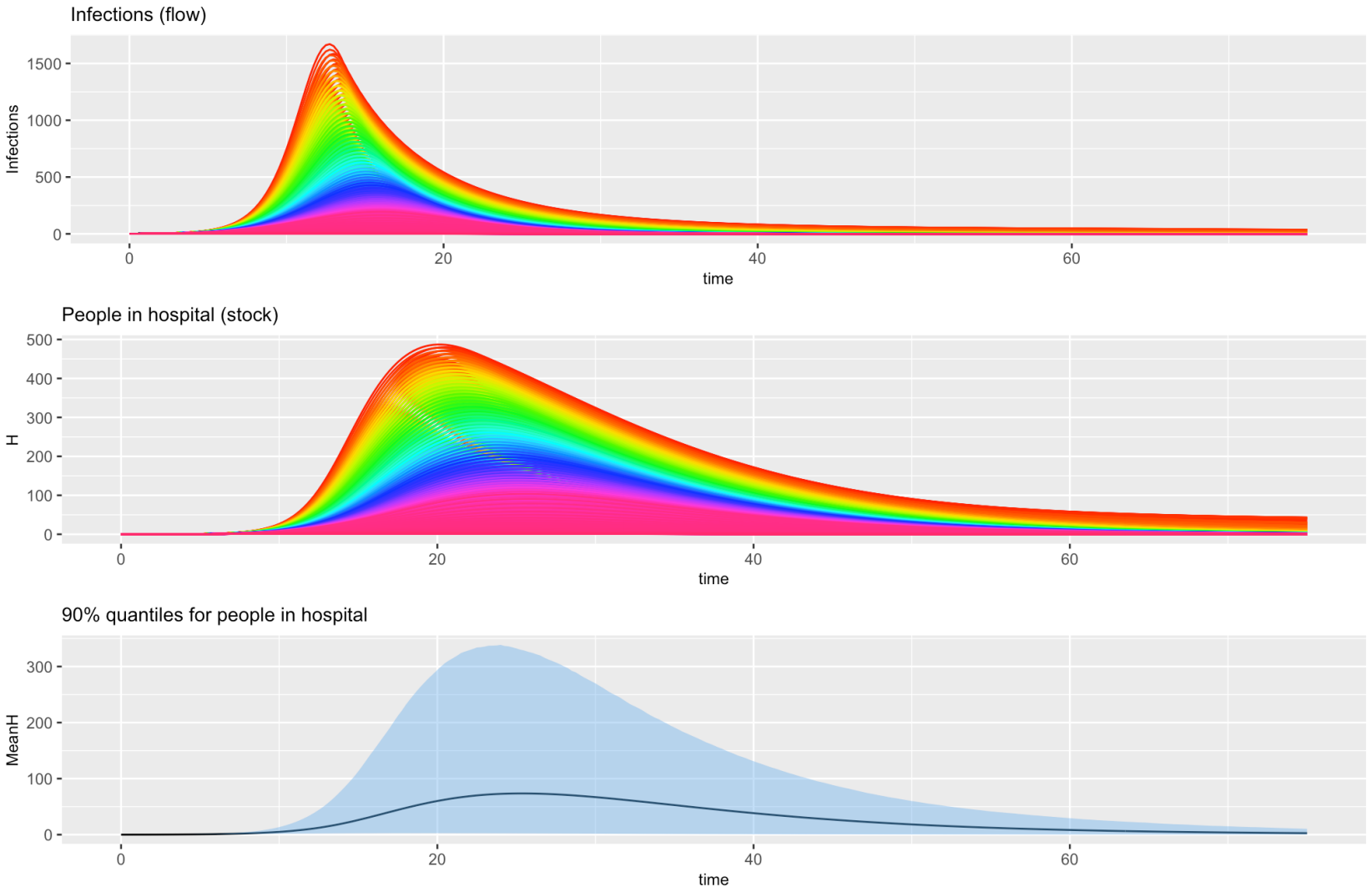
Running scenarios: *2 policy levers*

The advantage of creating the function `run_scenario()` is that we can now call this for a range of parameter values. Here, we are going to sample two policy variables:

- α, which models the speed of mobility restriction implementations. For example, a higher value of α would mean that the population responds quickly to the request for social mobility reductions. In our simulations $0 \leq \alpha \leq 0.20$.
- v, which models the speed of vaccination. A higher value of v means that people transfer more quickly from S to R, and therefore the burden on the hospital sector should be reduced. In our simulations $0 \leq v \leq 0.05$, which indicates that the minimum vaccination duration is 20 days $(1/0.05)$.

Dataset of results

```
> sim_res
# A tibble: 752,500 × 19
   RunID  time     S     I       H      R     M  Beta Gamma    HF     V Alpha M_Min Infections Recover…¹ Vacci…² Hospi…³ Disch…⁴
   <int> <dbl> <dbl> <dbl>   <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>      <dbl>     <dbl>   <dbl>   <dbl>   <dbl>
 1     1  0    9999   1    0       0         1     1  0.25   0.1     0     0   0.3        1.00     0.225       0  0.025  0
 2     1  0.25 9999.  1.19 0.00625 0.0562    1     1  0.25   0.1     0     0   0.3        1.19     0.267       0  0.0297 6.25e-4
 3     1  0.5  9998.  1.41 0.0135  0.123     1     1  0.25   0.1     0     0   0.3        1.41     0.317       0  0.0353 1.35e-3
 4     1  0.75 9998.  1.67 0.0220  0.203     1     1  0.25   0.1     0     0   0.3        1.67     0.377       0  0.0419 2.20e-3
 5     1  1    9998.  1.99 0.0319  0.298     1     1  0.25   0.1     0     0   0.3        1.99     0.447       0  0.0497 3.19e-3
 6     1  1.25 9997.  2.36 0.0435  0.410     1     1  0.25   0.1     0     0   0.3        2.36     0.531       0  0.0590 4.35e-3
 7     1  1.5  9997.  2.80 0.0572  0.544     1     1  0.25   0.1     0     0   0.3        2.80     0.631       0  0.0701 5.72e-3
 8     1  1.75 9996.  3.33 0.0733  0.703     1     1  0.25   0.1     0     0   0.3        3.33     0.749       0  0.0832 7.33e-3
 9     1  2    9995.  3.95 0.0923  0.892     1     1  0.25   0.1     0     0   0.3        3.95     0.889       0  0.0988 9.23e-3
10     1  2.25 9994.  4.69 0.115   1.12      1     1  0.25   0.1     0     0   0.3        4.69     1.06        0  0.117  1.15e-2
# … with 752,490 more rows, 1 more variable: CheckSum <dbl>, and abbreviated variable names ¹Recovering, ²Vaccinated,
#   ³Hospitalised, ⁴Discharged
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

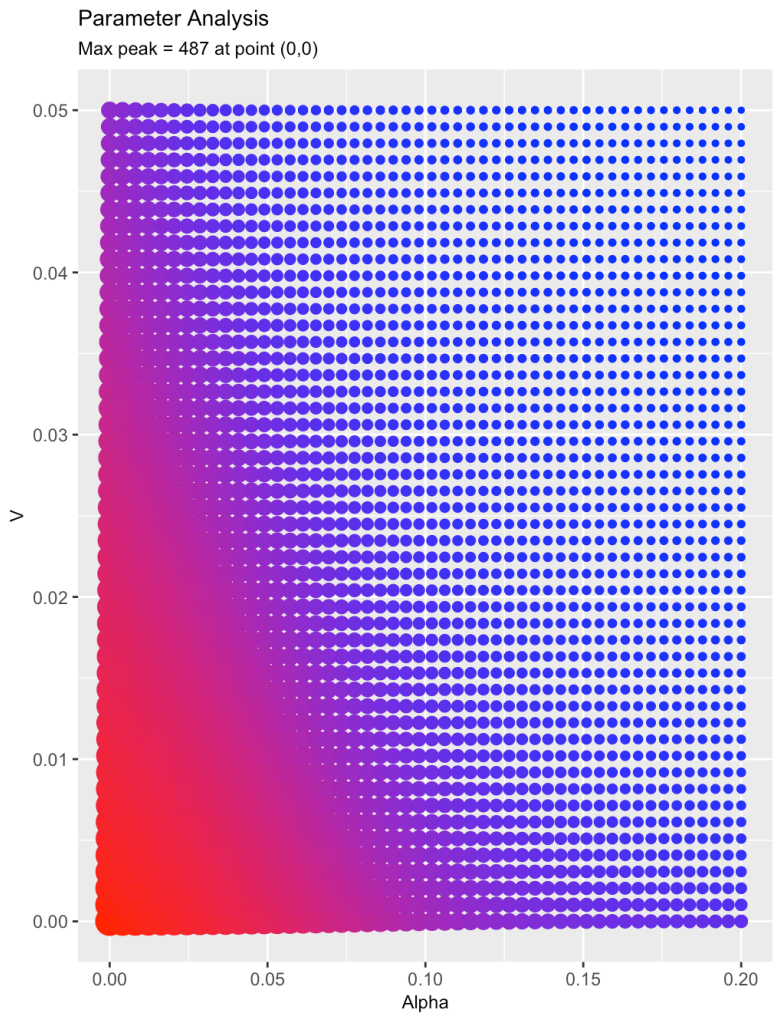

Infections (flow)
Infections
1500
1000
500
0
0
20
40
60
time
People in hospital (stock)
H
500
400
300
200
100
0
0
20
40
60
time
90% quantiles for people in hospital
MeanH
300
200
100
0
0
20
40
60
time

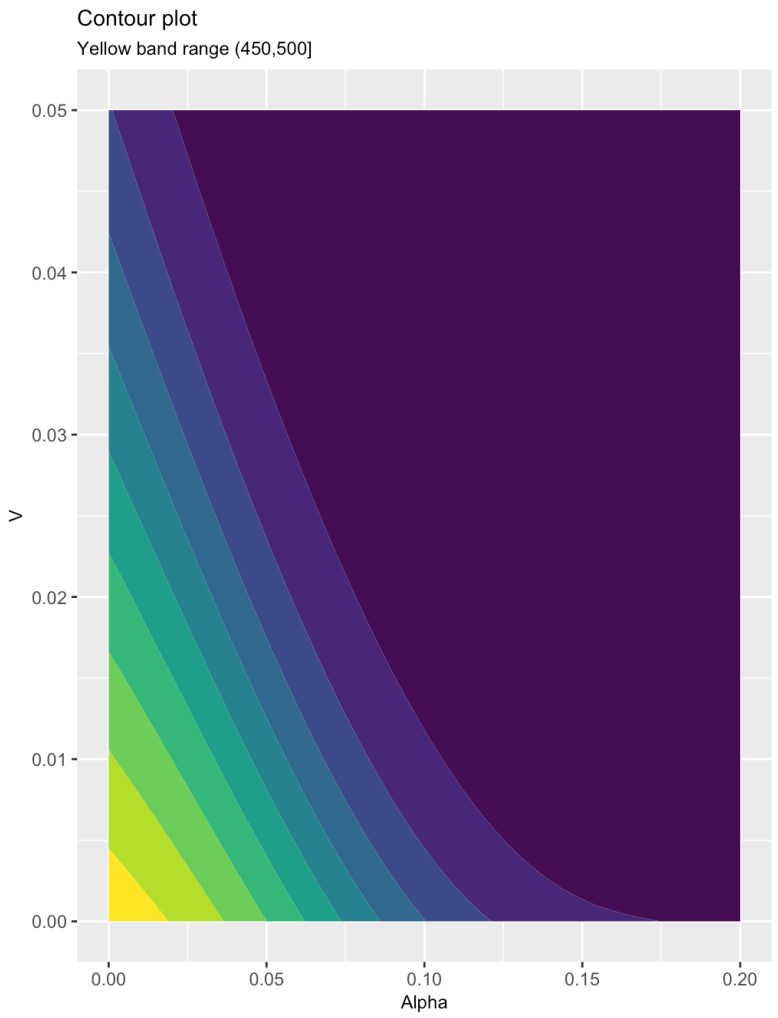
Further Analysis

```
max_h <- sim_res %>%
  dplyr::group_by(RunID) %>%
  dplyr::summarize(MH=max(H),
                   V=first(V),
                   Alpha=first(Alpha))
```

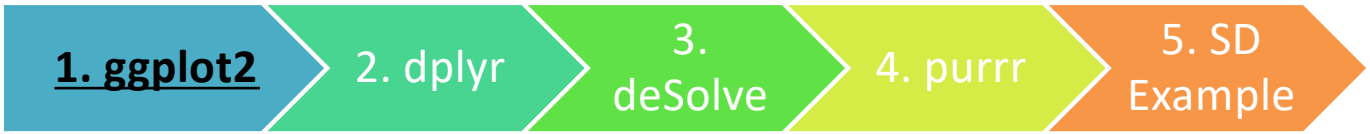
```
> max_h %>% dplyr::arrange(MH) %>% dplyr::slice(1:3)
# A tibble: 3 × 4
  RunID    MH      V Alpha
  <int> <dbl>  <dbl> <dbl>
1  2500  1.68 0.05   0.2
2  2450  1.72 0.0490 0.2
3  2499  1.73 0.05   0.196
```

```
> max_h %>% dplyr::arrange(desc(MH)) %>% dplyr::slice(1:3)
# A tibble: 3 × 4
  RunID    MH       V   Alpha
  <int> <dbl>   <dbl>   <dbl>
1     1  487. 0       0
2     2  480. 0       0.00408
3    51  479. 0.00102 0
```

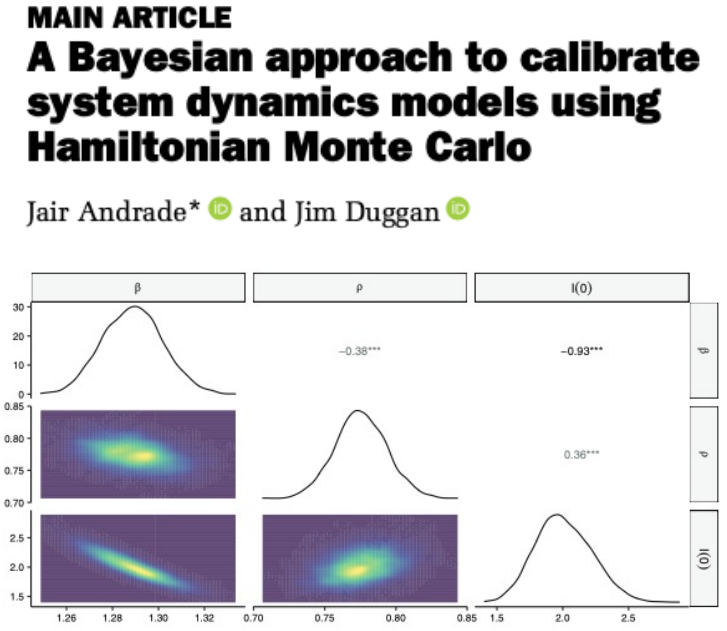

Parameter Analysis
Max peak = 487 at point (0,0)
0.05
0.04
0.03
0.02
0.01
0.00
V
0.00
0.05
0.10
0.15
0.20
Alpha


Contour plot
Yellow band range (450,500]
0.05
0.04
0.03
0.02
0.01
0.00
V
0.00
0.05
0.10
0.15
0.20
Alpha

Conclusion



- R can integrate in a flexible way within your System Dynamics workflow
- Visualization, summaries, modelling, sensitivity and web apps
- Can also support calibration workflows using stan and readsdr - Andrade and Duggan 2021


MAIN ARTICLE

A Bayesian approach to calibrate system dynamics models using Hamiltonian Monte Carlo

Jair Andrade* and Jim Duggan

Workshop Challenge

- Based on the codebase in SDWorkshop:
  - Implement an SD model of your choice in deSolve
  - Identify two parameters that can be varied for sensitivity analysis
  - Run 250 simulations, storing the results in a tibble
  - Generate a scatterplot of the two parameters for one summary variable