<div align="center">

Mathematics 4MB3/6MB3 Mathematical Biology
2018 ASSIGNMENT 2

Group Name: Cream

Group Members: Eric Clapton, Ginger Baker, Jack Bruce

</div>

This assignment was due in class on **Monday 5 February 2018 at 11:30am**.

1 Plot P&I mortality in Philadelphia in 1918

(a) Confirm that you have received this data file by e-mail:

<div align="center">

pim_us_phila_city_1918_dy.csv

</div>

This plain text comma-separated-value file can be examined (if you wish) using any plain text editor, such as Emacs.

(b) Read the data into a data frame in R, using the read.csv() function. For example, the following chunk of R code should work:

```
datafile <- "pim_us_phila_city_1918_dy.csv"
philadata <- read.csv(datafile)
philadata$date <- as.Date(philadata$date)
```

The purpose of the last line of code above is to ensure that R encodes character strings such as "1918-10-15" as dates.

(c) Reproduce the Philadelphia 1918 P&I plot:

Solution. . . . beautiful graph here. . . and nicely commented code that produces it, either in a separate .R file or preceding the graph if this is a knitr script. . . □

You'll need to use functions such as plot(), points() and lines(). For a comprehensive list of graphics parameters accepted by these functions, enter ?par into the Console pane in RStudio. There are multiple ways to produce a graph exactly like the above, but the following steps work:

- Use plot() to draw the box and basic annotation and the grey line. Suppress labels when doing this (*e.g.,* xlab=""). The box type is controlled by the bty option and the orientation of annotation is controlled by the las option.

- Use points() to draw the heavy red dots with black borders. The most elegant way to do this is to set the point character type to 21 (pch=21) and the point background colour to red (bg="red"). Alternatively, you can use points() twice (first to draw the red dots and then to draw the black circles around them).

- Use mtext() to add the x and y axis labels in the margins of the plot.

2 Estimate $\mathcal{R}_0$ from the Philadelphia P&I time series

(a) The observed mortality time series $M(t)$ is certainly not equal to the prevalence $I(t)$ that appears in the SIR model. Suppose, however, that $I(t) = \eta M(t - \tau)$ for all time (where η and τ are constants), *i.e.,* that the mortality curve is exactly a scaled and translated version of the prevalence curve. Prove that if both I and M are growing exactly exponentially over some time period then their exponential rates are identical. Thus, if we compare them during the "exponential phase" on a logarithmic scale, then both curves will be perfectly straight with exactly the same slope.

Solution. . . . beautifully clear and concise text to be inserted here. . . □

(b) Fit a straight line to the part of the Philadelphia 1918 mortality time series that looks straight on a logarithmic scale (and show your result in a plot). Once you get the hang of it, the easiest way to do this is to use the `lm()` function in ℝ (lm stands for linear model). Note that the simplest way to draw a straight line with given slope and intercept is with the `abline()` function. If you find `lm()` counter-intutive to understand then experiment with `abline()` until your eyes tell you that you have discovered a line that provides a good fit.

Solution. . . . beautifully clear text and plot(s) here . . . interspersed with ℝ code if this is a `knitr` script □

(c) How is the slope of your fitted line related to the parameters of the SIR model? (*Hint: When I is small, $S \simeq 1$.*) Why do you need an independent measure of the mean infectious period to estimate $\mathcal{R}_0$? If the mean infectious period is 4 days, what is your estimate of $\mathcal{R}_0$?

Solution. . . . beautifully clear text here . . . including some embedded ℝ code to estimate $\mathcal{R}_0$ if this is a `knitr` script □

3 Fit the basic SIR model to the Philadelphia P&I time series

(a) Install the `"deSolve"` package. This is done by typing the following command in the Console pane of `RStudio`:

```
install.packages("deSolve")
```

You will then be prompted to choose a mirror site from which to download the package. It doesn't matter which mirror you choose, but choosing a site in Ontario might save a fraction of a second. *Note:* This is a one-time operation. You do not want an `install.packages()` command inside your solutions code.
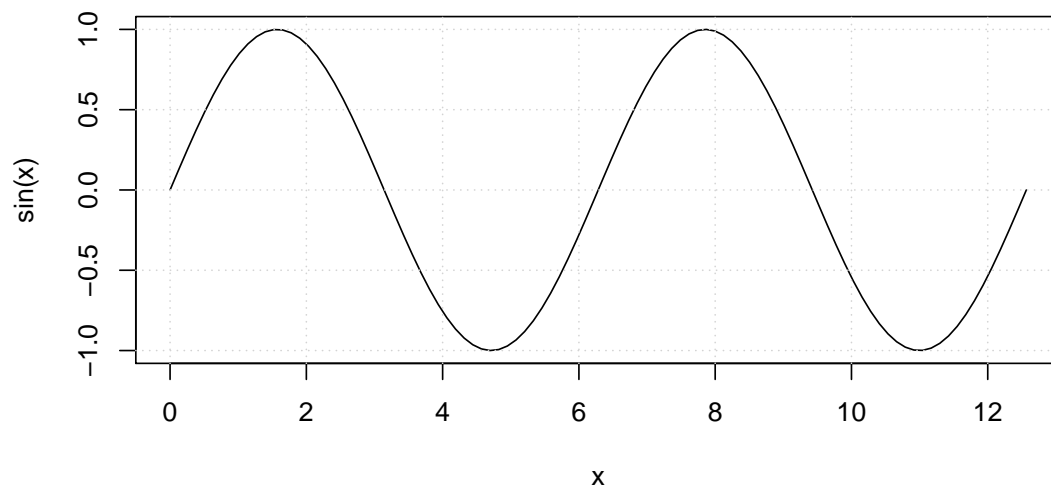
(b) Write an ℝ function that plots the solution $I(t)$ of the SIR model for given parameter values ($\mathcal{R}_0$ and $1/\gamma$) and given initial conditions (S_0, I_0). Use the `ode()` function in the `deSolve` package. A few hints:

- Your code will first need to load the `deSolve` package:

```
library("deSolve")

## Warning:  package 'deSolve' was built under R version 3.3.3
```

- As an example of defining a function (without getting involved with a differential equation), here is a code chunk that defines a function to plot a sine curve, and then executes the function. Note that the default min and max x values are set in the parameter list of the function definition, but the max x value is changed when the function is executed:

```
plot.sine <- function( xmin=0, xmax=2*pi ) {
  x <- seq(xmin,xmax,length=100)
  plot(x, sin(x), typ="l")
  grid() # add a light grey grid
}
plot.sine(xmax=4*pi)
```


- Here's another example. This time we first define the vector field for a differential equation. We then use this function inside another function that plots the solution of the associated differential equation. To understand the construction, you can, as usual, study the help page for the calling function (`?ode` in this case), but the most important issues are the following.

One of the arguments of the `ode()` function is the function that evaluates the vector field at the current time. To avoid confusion, choose the arguments of your vector field function to be `t`, `vars` and `parms` (in that order):

      `t`   The current time, which will be used within the vector field function if the system is non-autonomous.

  `vars`   A named vector of the variables in the system (*e.g.*, S, I). The variables, as named vector passed to this function, are used in the code that defines the vector field within the function.

`parms`   A named vector of the parameters of the system (*e.g.*, β, γ). It is convenient—but not necessary—to specify default values for the parameters.

It is strongly recommended that you follow exactly the style below when defining vector fields for differential equations that you wish to solve with the `ode()` function. In particular, the construction "`with(as.list(c(parms,vars)), ...)`" makes the variables and parameters visible within the section of code between the braces (`{...}`) without having to refer to the vectors or lists in which they are stored. For example, the code would be much harder to read if each instance of `x` were replaced by `vars$x` and each instance of `beta` were replaced by `parms$beta`; this issue becomes extremely important for complicated vector fields.

```
## Vector Field for SI model
SI.vector.field <- function(t, vars, parms=c(beta=2,gamma=1)) {
  with(as.list(c(parms, vars)), {
    dx <- -beta*x*y # dS/dt
    dy <- beta*x*y  # dI/dt
    vec.fld <- c(dx=dx, dy=dy)
    return(list(vec.fld)) # ode() requires a list
  })
}
```

The following function plots a single solution of the ODE for a given initial condition (`ic`), integration time (`tmax`) and times at which the state is to be returned (`times`). The vector field function is passed as the `func` argument and the parameter vector is passed as the `parms` argument. If further arguments are given, they are passed to the `lines()` function that draws the solution.

```
## Draw solution
draw.soln <- function(ic=c(x=1,y=0), tmax=1,
                      times=seq(0,tmax,by=tmax/500),
                      func, parms,
                      col="blue",
                      ... ) {
  soln <- ode(ic, times, func, parms)
```

```
  lines(times, soln[,"y"], col=col, lwd=3, ... )
}
```

Note here that the call to the **ode()** function gives the arguments in the default order so they are interpretted correctly. If we wished to write the arguments in a different order then we would have to be explicit about which argument is which. For example, if we wanted to list the initial conditions last for some deep reason then we would have to write:

```
  soln <- ode(times=times, func=func, parms=parms, y=ic)
```

We can now use our **draw.soln()** function to plot a few solutions of the SI model.

```
## Plot solutions of the SI model
tmax <- 10 # end time for numerical integration of the ODE
## draw box for plot:
plot(0,0,xlim=c(0,tmax),ylim=c(0,1),
     type="n",xlab="Time (t)",ylab="Prevalence (I)",las=1)
## initial conditions:
I0 <- 0.001
S0 <- 1 - I0
## draw solutions for several values of parameter beta:
betavals <- c(1.5,2,2.5)
for (i in 1:length(betavals)) {
  draw.soln(ic=c(x=S0,y=I0), tmax=tmax,
            func=SI.vector.field,
            parms=c(beta=betavals[i],gamma=1),
            lty=i # use a different line style for each solution
            )
}
```
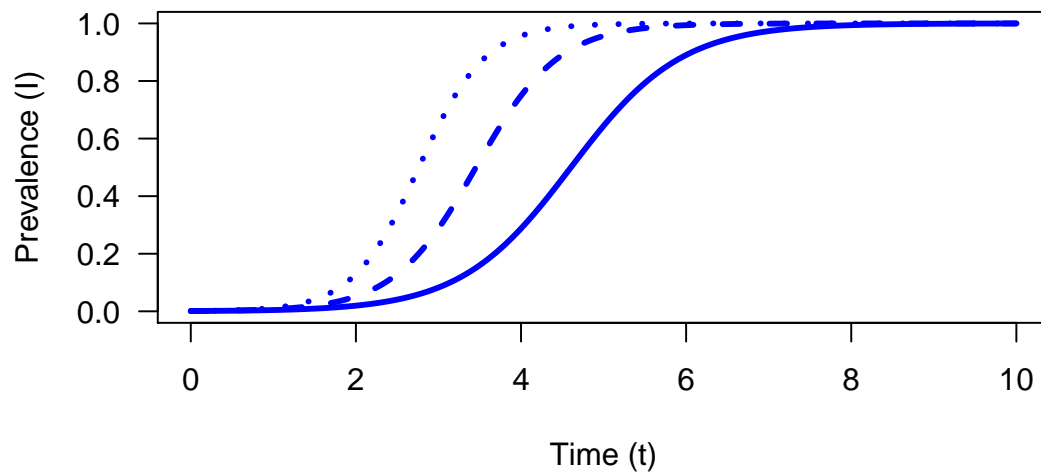
Solution. First, we start by defining a function for SIR vector field:

```
## Vector Field for SIR model
SIR.vector.field <- function(t, vars, parms=c(beta=2,gamma=1)) {
    with(as.list(c(parms, vars)), {
        inf <- beta*x*y
        return(list(c(dx=-inf, dy=inf-gamma*y)))
    })
}
```

Then, we can define a function that draws the solution:

```
##' @param S0 initial proportion of susceptible individuals
##' @param I0 initial proportion of infected individuals
##' @param R0 basic reproductive number
##' @param inf.period length of infectious period
##' @param ... additional arguments passed to `draw.soln`
draw.sir <- function(S0,
                     I0,
                     R0,
                     inf.period,
                     tmax, ## max time
                     ...) {
    draw.soln(ic=c(x=S0, y=I0),
              tmax=tmax,
              func=SIR.vector.field,
```

```
            parms=c(beta=R0/inf.period,
                    gamma=1/inf.period),
            ...)
}
```

☐

(c) For $I_0 = 10^{-3}$ and $S_0 = 1 - I_0$, plot the solutions of the SIR model assuming $1/\gamma = 4$ days and $\mathcal{R}_0 \in \{1.2, 1.5, 1.8, 2, 3, 4\}$. Use the legend() command to make a legend on the plot that shows which curves correspond to which values of $\mathcal{R}_0$.
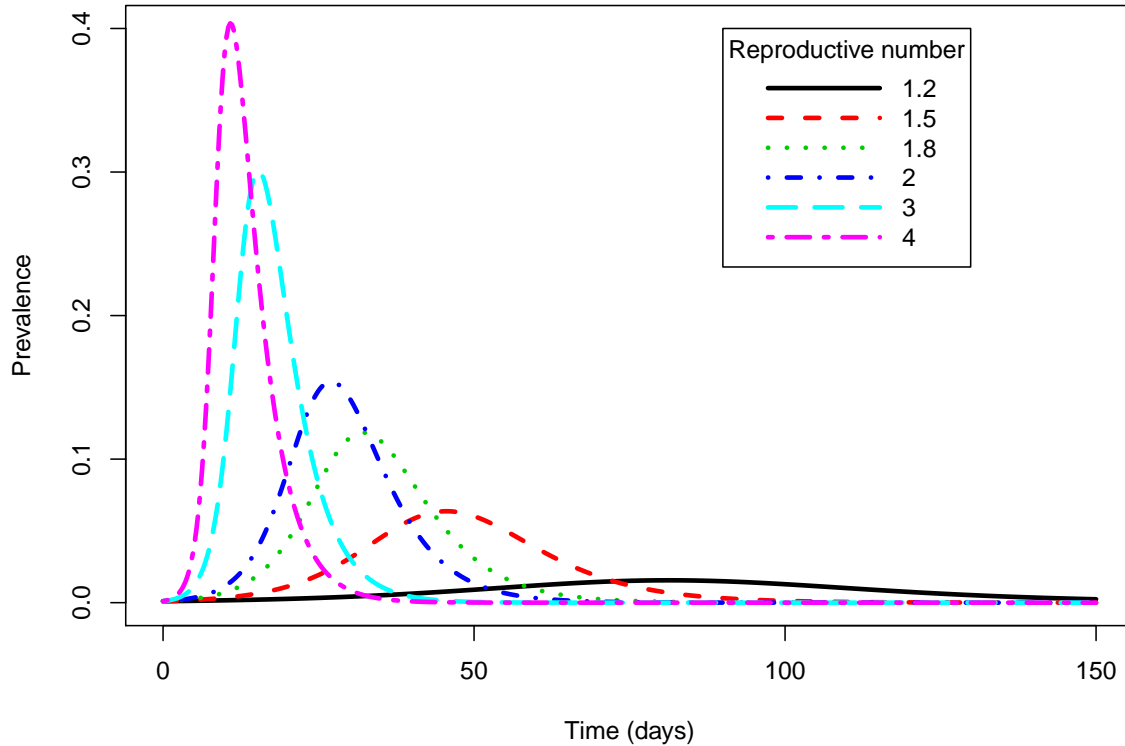
Solution.
```
I0 <- 1e-3
S0 <- 1 - I0
inf.period <- 4
R0 <- c(1.2, 1.5, 1.8, 2, 3, 4)
n <- length(R0)
tmax <- 150

plot(NA, xlim=c(0,tmax), ylim=c(0,0.4), xlab="Time (days)", ylab="Prevalence")
SIR.plot <- Map(draw.sir, S0=S0, I0=I0, inf.period=inf.period,
    R0=R0, lty=1:n, col=1:n,
    tmax=tmax
)

legend(
    x=90, y=0.4,
    legend=R0, lty=1:n, col=1:n,
    lwd=3,
    title="Reproductive number",
    seg.len=5
)
```

(d) By trial and error, find values of $\mathcal{R}_0$ and γ that yield a solution of the SIR model that fits the Philadelphia P&I times series reasonably well. You can assess the quality of fit using the Euclidean distance between the model solution and the data. (*Note:* The trial and error approach is a valuable exercise, but not a suggestion of a method you would really use in practice. We'll discuss better methods for fitting ODE models to data later.)

Solution. In section 2 (a), it was assumed that mortality and prevalence has the following relationship:

$$I(t) = \eta M(t - \tau).$$

Rearranging, we have

$$M(t) = \frac{1}{\eta} I(t + \tau)$$

So we will have to run the model for $t + \tau$ time steps to compare and estimate η, τ, $S(0)$, $I(0)$, $\mathcal{R}_0$, and infectious period.

Here is the function:

```r
simulate_mortality <- function(S0,
                               I0,
                               R0,
                               inf.period,
                               eta,
                               tau,
                               tmax=nrow(philadata)) {
    soln <- as.data.frame(ode(
        y=c(x=S0, y=I0),
        times=1:(tmax+tau),
        func=SIR.vector.field,
        parms=c(beta=R0/inf.period,
                    gamma=1/inf.period)))

    data.frame(
        date=philadata$date,
        pim=tail(soln$y, -tau)/eta
    )
}
```
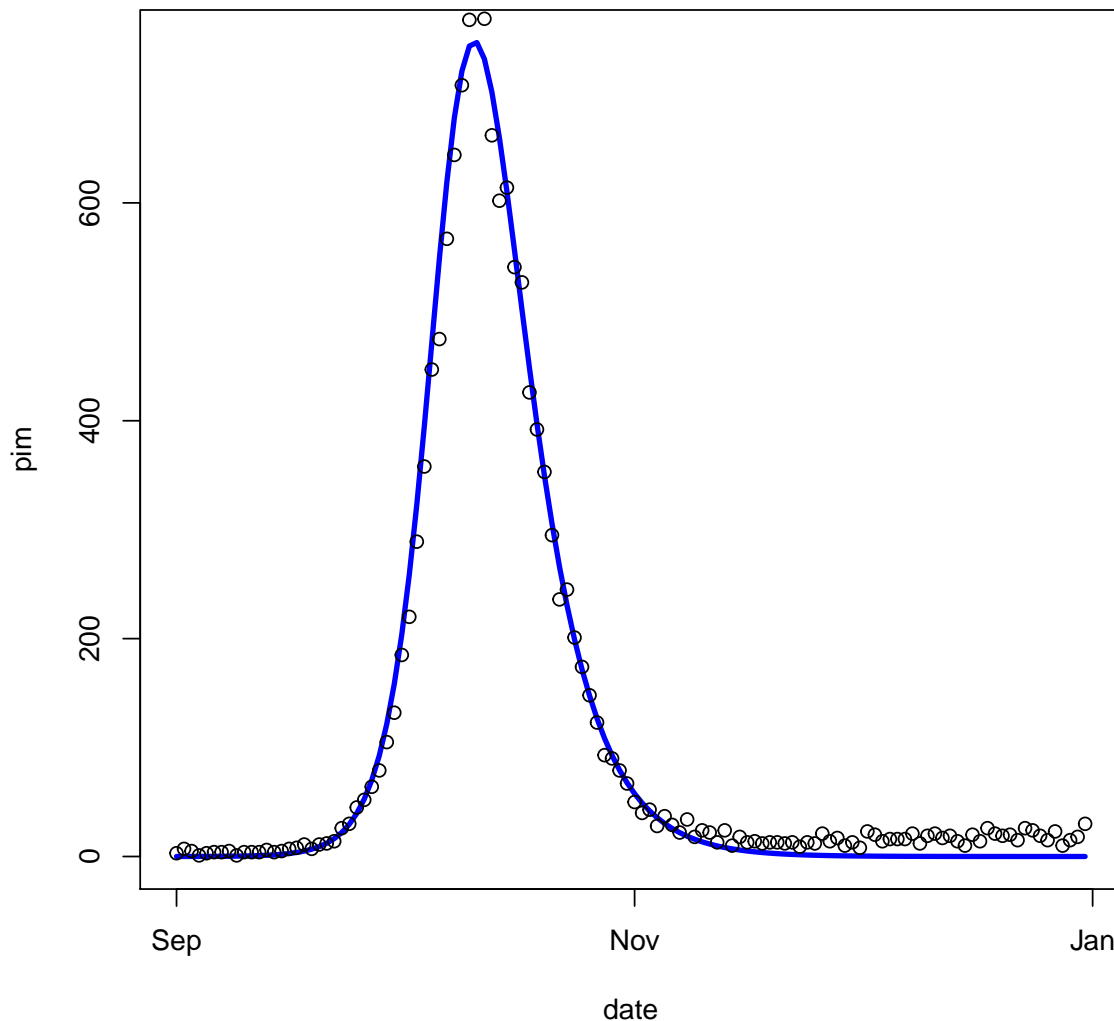
Now here is the estimate that we found trial and error:

```r
mfit <- simulate_mortality(S0=1-1e-7, I0=1e-7, R0=2.2, inf.period = 4, eta=0.00025, t
plot(mfit, type="l", lwd=3, col="blue")
points(philadata)
```

4 Executive summary for the Public Health Agency

The Public Health Agency of Canada (PHAC) is revising their pandemic plan and has asked your group to summarize what you learned from analyzing the 1918 Philadelphia P&I time series. Besides explaining what inferences you feel you can make from your analysis so far, PHAC wants to know what you would investigate if they were to fund you to continue your work full time for a month. They want a maximum of one page from your group.

Incidentally, you might be interested to know that rumour has it that all of the members of the pandemic planning committee took Math 2C03 at McMaster University between 1980 and 2003, but they all failed. Also, when the chair of the committee was recently asked

"What is a differential equation?" he apparently bent over and vomited (it is hard to know quite what to make of this given that PHAC was investigating a norovirus outbreak at the time).

Note: When submitting your assignment solution, it is imperative that the one-page executive summary be printed on its own page. To start a new page in $\LaTeX$, use the `\newpage` *command. Also, as usual, your summary should be in 12 point font. Don't try to cram in as much as possible. Make that page as clear and concise as you can, so that a public health planner can absorb its content quickly and easily.*

Solution. . . . beautifully clear executive summary here, on its own page. . . □

— END OF ASSIGNMENT —

Compile time for this document: February 3, 2018 @ 13:13