eAPPENDIX 1

Retrospective analysis at the end of an outbreak

Uniform release window: Let us assume that the probability of an infected individual having been infected at time t follows a uniform density function given by:

$$u(t;x,y)=\begin{cases}\dfrac{1}{y-x} & \text{for } x\le t\le y\\ 0 & \text{otherwise}\end{cases} \qquad \text{eq. 1}$$

where the parameters x and y represent the release start and end times, respectively. Let us also assume that the probability of an infected individual having an incubation period of duration t follows a gamma density function given by:

$$g(t;a,b)=\begin{cases}\dfrac{t^{a-1}e^{-t/b}}{b^{a}\Gamma(a)} & \text{for } t,a,b>0\\ 0 & \text{otherwise}\end{cases} \qquad \text{eq. 2}$$

where a and b represent the shape and scale parameters, respectively, and Γ is the gamma function:

$$\Gamma(z)=\int_{-\infty}^{\infty} v^{z-1}e^{-v}dv \quad \text{for } z>0\,. \qquad \text{eq. 3}$$

The modelled symptom onset times, t, are derived from the convolution of equations 1 and 2, with density function given by:

$$h_u(t;a,b,x,y) = \int_x^t u(t-v;x,y)g(v;a,b)dv$$

$$= \frac{G(t-x,a,b) - G(t-y,a,b)}{y-x} \qquad \text{eq. 4}$$

where G is the gamma distribution function:

$$G(t;a,b) = \int_{-\infty}^{t} g(v;a,b)dv\,. \qquad \text{eq. 5}$$

The likelihood function for an outbreak of size n is therefore given by:

$$\ell(\tilde{t} \mid n;a,b,x,y) = \prod_{i=1}^{n} h_u(t_i;a,b,x,y)$$

$$= \prod_{i=1}^{n} \frac{G(t_i - x,a,b) - G(t_i - y,a,b)}{y-x} \qquad \text{eq. 6}$$

where $\tilde{t} = \{t_1, t_2, ..., t_n\}$ represent the observed symptom onset times. Assuming that a and b are known *a priori*, x and y can be estimated by maximising equation 6. The authors used the "mle" function in the R stats4 package[1] to calculate such parameter estimates and their associated confidence intervals. No continuity-correction was applied for discrete data and no adjustment was made for interval censoring.

Logistic release window: In order to construct a model that captures the growth of *L. pneumophila* during the early stages of the release we start by considering the logistic distribution function given by:

$$L(t; x_1, x_2) = \frac{1}{1 + e^{-(t-x_1)/x_2}} \qquad \text{for} \qquad x_2 > 0 \qquad \text{eq. 7}$$

where the parameters x_1 and x_2 represent the location and scale parameters, respectively. We then seek to normalise equation 7 (i.e. convert the distribution function into a density function) by firstly performing the following integration:

$$\int_{-\infty}^{y} L(t; x_1, x_2)dt = x_2 \log\left(1 + e^{(y-x_1)/x_2}\right) \qquad \text{eq. 8}$$

where y is the fixed end of the release analogous to the same parameter in equation 1. Dividing equation 7 by equation 8 we obtain the probability of an infected individual having been infected at time t:

$$m(t; x_1, x_2, y) = \begin{cases} \left[x_2\left[1 + e^{-(t-x_1)/x_2}\right]\log\left(1 + e^{(y-x_1)/x_2}\right)\right]^{-1} & \text{for} \quad t \le y \\ 0 & \textit{otherwise} \end{cases}. \qquad \text{eq. 9}$$

Note that equation 9 is not equivalent to the standard logistic probability density function whereas equation 1 is equivalent to the standard uniform probability density function. Epidemic curves are modelled by the convolution of equations 2 and 9:

$$h_l(t;a,b,x_1,x_2,y) = \int_{-\infty}^{t} m(t-v;x_1,x_2,y)g(v;a,b)dv\,. \qquad \text{eq. 10}$$

However, unlike equation 4, the authors were unable to express the solution to equation 10 in terms of easily evaluated gamma functions (see below) and so performed the calculation with the "integrate" function in the R stats package[1]. Also, the authors were unable to maximise the associated likelihood function with the "mle" function in the R stats4 package and, therefore, calculated the parameter estimates and confidence intervals by implementing a grid-search method in R. No continuity-correction was applied for discrete data and no adjustment was made for interval censoring.

eAPPENDIX 2

Prospective analysis during an outbreak

Adapted from the instantaneous release scenario [2,3], the likelihood function for the continuous release scenario, assuming a uniform release window, can be written as:

$$\ell(\tilde{t} \mid \tau,k;n,a,b,x,y) = \binom{n}{k}[1-H_u(\tau;a,b,x,y)]^{n-k}\prod_{i=1}^{k} h_u(t_i;a,b,x,y) \qquad \text{eq. 11}$$

where n is the total number of cases that would present by the end of the outbreak, k is the number of cases with symptom onset at most τ days relative to the first case, $\tilde{t} = \{t_1,t_2,...,t_k\}$ represent the observed symptom onset times, h_u is given by equation

4 and H_u is the distribution function of symptom onset times (i.e. the probability an individual infected uniformly over $[x, y]$ is symptomatic by time t):

$$H_u(t; a, b, x, y) = \int_{-\infty}^{t} h_u(v, a, b, x, y) dv . \qquad \text{eq. 12}$$

Evaluating H_u involves integrating the gamma distribution function (equation 5) which can be written in an alternative form:

$$G(t; a, b) = \gamma\left(a, \frac{t}{b}\right) \Big/ \Gamma(a) \qquad \text{eq. 13}$$

where γ is the lower incomplete gamma function:

$$\gamma(z, s) = \int_{-\infty}^{s} v^{z-1} e^{-v} dv \quad \text{for } z > 0 . \qquad \text{eq. 14}$$

It can be shown that:

$$\int_{-\infty}^{r} \gamma(z, w) dw = [r - z]\Gamma(z) - r\Gamma(z, r) + \Gamma(z+1, r) \qquad \text{eq. 15}$$

with the upper incomplete gamma function given by:

$$\Gamma(z,s)=\int_{s}^{\infty} v^{z-1}e^{-v}dv\,. \qquad \text{eq. 16}$$

Thus, after changing variables we have:

$$\int_{-\infty}^{t} G(v-x;a,b)dv=\frac{b[[\lambda(x)-a]\Gamma(a)-\lambda(x)\Gamma(a,\lambda(x))+\Gamma(a+1,\lambda(x))]}{\Gamma(a)} \qquad \text{eq. 17}$$

where $\lambda(x)=(t-x)/b$. eq. 18

Combining equations 4, 6 and 17 allows the likelihood function (equation 11) to be expressed in terms of easily evaluated gamma functions (equations 3, 14 and 16). However, the authors were unable to maximise the likelihood function with the "mle" function in the R stats4 package[1] and, therefore, calculated the parameter estimates and confidence intervals by implementing a grid-search method in R. No continuity-correction was applied for discrete data and no adjustment was made for interval censoring (although τ provided an upper bound for right censoring).

REFERENCES:

1. R Development Core Team. *R: A language and environment for statistical computing.*. Vienna, Austria: R Foundation for Statistical Computing; 2008. Available at: http://www.R-project.org.

2. Walden J, Kaplan K. Estimating time and size of bioterror attack. *Emerging infectious diseases*. 2004;10:1202–1205.

3. Brookmeyer R, Blades N. Prevention of inhalational anthrax in the U.S. outbreak. *Science*. 2002;295(5561):1861.

```
# eSCRIPT 1

# Incubation period data-sets

MelbourneIP <-
c(1,1,2,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,4,5,5,5,5
,5,5,5,5,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,7,7,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9
,9,9,9,9,9,10,10,10,10,11,11,11,12,12,13,15,16)

MiyazakiIP <-
c(2,2,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,5,5,5
,6,6,6,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,8,
8,8,8,9,9,9,10,10,11,12,12,14)


#   Symptom onset data-sets

Amsterdam <-
c(0,1,2,3,3,4,4,4,4,4,4,4,4,5,6,7,7,7,7,8,8,8,8,10,10,11,12,19,
23,24)

Barcelona <-
c(0,1,1,2,2,3,3,3,3,3,3,3,4,4,4,5,6,6,6,6,6,6,7,7,7,7,8,8,8,8,8,8
,8,9,9,9,9,10,10,10,10,10,10,10,10,10,10,11,11,11,12,12,14,14)

Barrow <-
c(0,2,4,6,6,7,9,9,9,9,10,11,11,12,12,12,12,12,13,13,13,13,13,13,
14,14,14,14,14,14,15,15,15,15,15,15,15,15,15,15,15,16,16,16,16,16
,16,16,16,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,
17,17,18,18,18,18,18,18,18,18,18,19,19,19,19,19,19,19,19,19,19,20
,20,20,20,20,20,20,20,21,21,21,21,21,21,21,21,22,22,22,22,22,23,
23,24,24,25)

Cerdanyola <- c(
0,10,10,13,15,15,15,16,19,19,19,19,20,20,20,20,20,21,21,21,21,21,
21,22,22,22,22,22,22,22,22,22,22,22,22,22,22,23,23,23,23,23,23,23,23
,23,23,23,23,24,24,24,24,25,25,25,25,25,25,25,25,25,25,25,25,25,25,
26,26,26,26,26,26,26,26,26,27,27,27,27,27,28,28,28,28,28,29,30,30
,30,30,30,30,30,31,31,31,31,32,32,33,33,33,34,34,34,34,34,34,35,
35,35,35,39,39,41)

Christiansburg <-
c(0,1,9,9,9,11,11,11,11,11,12,12,12,13,14,14,15,15,15,17,19,23)

Genova <-
c(0,0,0,1,6,8,9,9,9,9,10,10,10,11,11,11,11,11,12,12,12,12,12,12,
12,13,13,13,13,13,13,13,13,13,13,14,14,14,14,14,14,14,14,14,14,14
,14,15,15,15,15,15,15,15,16,16,16,16,16,16,16,16,16,16,16,17,17,
17,17,17,17,17,17,17,18,18,19,19,19,20,20,21,22,22,22,22,22,23,23
,25,25,28,28,29,29,31,31,35)

Hereford <-
c(0,20,22,22,22,23,24,24,25,25,26,26,27,27,30,30,31,31,31,32,32,
32,32,33,36,40,43)

Bovenkarspel1 <-
c(0,0,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5
,5,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,7,7,
```

```
7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8
,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,11,11,11,11
,11,11,11,11,11,11,11,11,11,11,11,11,11,12,12,12,12,13,13,13,13,
13,14,15,15,15,15,16,16,18,19)

Bovenkarspel2 <-
c(0,0,1,1,2,2,2,2,3,3,3,4,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5
,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,7,8,8,8,
8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,9,9
,9,9,9,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,11,11,11,
11,11,11,11,11,11,11,11,11,11,11,11,11,11,12,12,12,12,13,13,13,13,13
,14,14,15,15,15,15,16,16,18,19)

Kapellen <-
c(0,1,2,3,3,3,3,3,3,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,7,7
,7,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,10,10,10,10,10
,10,10,10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,11,11,11,
11,11,11,11,11,11,11,11,13,14,15)

Melbourne <-
c(0,0,0,0,1,1,1,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4
,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,
7,7,7,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,10,
10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,12,12,12,12,12
,13,13,13,14,16,18)

Miyazaki <-
c(0,2,2,3,3,4,6,10,10,11,11,11,12,12,13,13,13,13,13,13,13,13,14,
14,14,14,14,14,14,14,15,15,15,15,15,15,15,15,16,16,16,16,16,16,16
,16,16,16,16,17,17,17,17,17,17,17,17,17,17,18,18,18,18,18,18,18,
18,18,18,18,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,20
,20,20,20,20,20,21,21,21,21,21,21,21,21,21,21,22,22,22,22,22,
22,22,22,23,23,23,23,23,23,23,23,23,23,24,24,24,24,24,24,24,24
,24,25,25,25,25,25,25,25,25,25,25,26,26,26,26,26,26,26,26,26,26,
26,26,26,26,26,26,27,27,27,27,27,27,27,27,27,27,28,28,28,28,28,28
,28,28,28,28,28,28,28,28,28,29,29,29,29,29,29,29,29,29,29,30,
30,30,30,30,30,30,30,30,30,31,31,31,31,31,31,31,31,31,31,31,31
,31,31,31,32,32,32,32,32,32,32,33,33,33,33,33,33,33,33,33,33,33,
33,33,33,33,34,34,34,34,34,34,35,35,35,35,35,35,36,36,37,37,37,37
,37,38,39,39,39,42,43,43,45,51,53,53)

Murcia <-
c(0,1,1,2,2,3,3,3,3,4,4,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,6
,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,
7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,8
,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10,10,
10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,11,11,11,11
,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,
11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11
,11,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
12,12,12,12,12,12,13,13,13,13,13,13,13,13,13,13,13,13,13,14,14
,14,14,14,14,14,15,15,15,15,15,15,15,15,15,16,16,16,16,16,16,
16,16,17,17,17,17,17,17,18,19,20,20,20,21,21,21,22,22,23)
```

```
Pamplona <-
c(0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2
,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
3,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,5,5,5,5
,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,
8,8,8,8,8,8,9,9,10,10,10,10,10,10,12,12,13)

Sarpsborg <-
c(0,0,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5
,5,5,5,5,6,6,6,7,7,7,7,8,8,8,8,9,10,11,11,11,12,12,12,13)

Sheboygan <-
c(0,1,1,1,2,2,2,2,2,3,3,3,4,4,4,4,4,5,5,6,7,8,8,11,12,12,16,19)

Stafford <-
c(0,3,6,6,6,6,6,7,7,7,7,7,8,8,8,9,10,10,10,10,10,10,11,12,12,12,
12,12,13,13,13,13,13,13,13,13,13,13,14,14,14,14,15,15,15,15,15,15
,16,16,16,16,16,17,17,17,17,18,18,19,19,20)

Torrevieja <-
c(0,1,1,1,2,2,2,3,3,3,5,5,5,5,5,5,5,6,6,6,6,6,6,7,7,7,9,12)

VicGurb <-
c(0,0,0,0,5,7,7,7,7,8,8,9,10,10,10,10,11,11,11,11,12,12,13,14,14,
14,14,15,15,16,17,17,17,18,19,19,20,21,21,21,22,22,22,23,23,23,24
,24,25,26,27,28,35,36)

Zaragoza <-
c(0,1,3,5,5,5,6,8,8,9,9,10,12,13,13,13,14,14,14,15,16,16,16,17,17
,19,20,21,23)
```

```
# eSCRIPT 2

# Back-calculation model that retrospectively estimates an
# assumed uniform release window of Legionella pneumophila using
# symptom onset dates.

# Copyright (C) 2010  Joseph R. Egan, Health Protection Agency.

# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License as
# published by the Free Software Foundation; either version 2 of
# the License, or (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# For a copy of the GNU General Public License write to the Free
# Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
# Boston, MA  02110-1301, USA.


bcm_uniform <- function(input =
c(0,0,0,0,1,1,1,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4
,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,
7,7,7,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,10,
10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,12,12,12,12,12
,13,13,13,14,16,18), location = "melbourne", a = 4.96, b =
1/1.27, begin = -5, finish = 1, xmin = -10, xmax = 5, chi2min =
5, plot.curve = TRUE, plot.pdf = TRUE, plot.window = TRUE,
write.results = FALSE, run.time = TRUE){

      # input: list of symptom onset days (earliest case is day 0)
      # location: location of outbreak
      # a: shape parameter for the gamma incubation period
      # b: rate parameter for the gamma incubation period
      # begin: initial estimate for the start of uniform release
      # window (days before day 0)
      # finish: initial estimate for the end of the uniform
      # release window (days after day 0)
      # xmin: minimum value on the x-axis when plotting
      # xmax: maximum value on the x-axis when plotting
      # chi2min: threshold for expected frequencies in a Chi-
      # squared test
      # plot.curve: plot the continuous pdf function
      # plot.pdf: plot the discretised daily pdf values
      # plot.window: plot the release window including confidence
      # intervals
      # write.results: write the plot to file
      # run.time: print the run-time of the model


      # Start the clock

      if(run.time == TRUE){
            time1 <- Sys.time()
      }
```

```
# Load required packages

library(stats4)


# Useful values

xmax <- (max(input)+xmax)
num_of_cases <- length(input)
epi_curve <- tabulate(input+1)


# Write plot

if(write.results == TRUE){
      pdf(file=paste(location , "_uniform.pdf", sep=""))
}


# Convolved probability density function (PDF) for symptom
# onset dates

convolvedPDF <- function(d,x,y) (pgamma(d-x,a,b)-pgamma(d -
y,a,b))/(y-x)


# Negative log-likelihood function

minusll <- function(x,y) -sum(log(convolvedPDF(input,x,y)))


# Maximum likelihood estimation

fit <- mle(minusll, start=list(x=begin, y=finish), control =
list(ndeps=c(0.01,0.01)))
prms <- coef(fit)      # estimated parameters
conf <- confint(fit)   # estimated confidence intervals
mll <- logLik(fit)[1]  # maximised log-likelihood


# Print useful values to screen

print(paste("x =", round(prms[1],2), "(", round(conf[1],2),
",", round(conf[3],2), ")"), quote=FALSE)
print(paste("y =", round(prms[2],2), "(", round(conf[2],2),
",", round(conf[4],2), ")"), quote=FALSE)


# Plot epidemic curve

drange <- c(xmin,xmax)
hist(input, breaks = seq(drange[1]+0.5, drange[2]+0.5, 0.5),
freq = TRUE, xlim = drange, ylim = c(0, max(epi_curve)+1),
main = "", xlab = "", ylab = "", axes = FALSE, col =
"black")
axis(1, pos = 0, at = c(drange[1]:drange[2]), labels =
c(drange[1]:drange[2]))
title(main = "", xlab = "Symptom onset day",
```

```
ylab = "Number of cases")
axis(2, pos = drange[1], las=2)


# Plot release window

if(plot.window == TRUE){
      lines(c(prms[1],prms[2]),
      c(max(epi_curve)+0.5,max(epi_curve)+0.5), lwd=1)
      points(c(prms[1],prms[2]), c(max(epi_curve)+0.5,
      max(epi_curve)+0.5), cex=2, pch=21, bg="black")
      lines(c(conf[1],conf[3]),
      c(max(epi_curve)+0.5,max(epi_curve)+0.5), lwd=2)
      lines(c(conf[2],conf[4]),
      c(max(epi_curve)+0.5,max(epi_curve)+0.5), lwd=2)
      points(c(conf[1],conf[3]),
      c(max(epi_curve)+0.5,max(epi_curve)+0.5), cex=1.25,
      pch=21, bg="black")
      points(c(conf[2],conf[4]),
      c(max(epi_curve)+0.5,max(epi_curve)+0.5), cex=1.25,
      pch=21, bg="black")
}


# Plot discretised PDF values

if(plot.pdf == TRUE){
      for(i in drange[1]:drange[2]){
            rect(i, 0, i+0.5, num_of_cases *
            convolvedPDF(i,coef(fit)[1], coef(fit)[2]),
            col="grey")
      }
}


# Plot continuous PDF function

if(plot.curve == TRUE){
      curve(num_of_cases * convolvedPDF(x,coef(fit)[1],
      coef(fit)[2]) ,drange[1], drange[2], add=TRUE)
}


# Chi-squared goodness-of-fit test

tmp_chi2ob <- matrix(0,100,1)
tmp_chi2ob[1,1] <- 0
for(k in 1:100){
      ex <- num_of_cases * convolvedPDF( -
      k:(max(input)+k),prms[1],prms[2])
      ob <- c(rep(0,k),epi_curve,rep(0,k))
      for(j in 0:length(ex)){
            for(i in 1:length(ex)){
                  if(sum(ex[(1+j):(i+j)])<chi2min){
                        ex[i+j+1]<-ex[i+j+1]+ex[i+j]
                        ob[i+j+1]<-ob[i+j+1]+ob[i+j]
                        ex[i+j]<-0
                        ob[i+j]<--1
                  }else{
```

```
                        break
                  }
            }
            ex <- ex[ex>0]
            ob <- ob[ob>-1]
            for(i in 1:length(ex)){
                  if(sum(ex[(length(ex)-j):(length(ex)+1-i -
                  j)])<chi2min){
                        ex[length(ex)-i-j]<-ex[length(ex)-i -
                        j]+ex[length(ex)-i-j+1]
                        ob[length(ex)-i-j]<-ob[length(ob)-i -
                        j]+ob[length(ob)-i-j+1]
                        ex[length(ex)-i-j+1]<-0
                        ob[length(ob)-i-j+1]<--1
                  }else{
                        break
                  }
            }
            ex <- ex[ex>0]
            ob <- ob[ob>-1]
            if(all(ex>chi2min)==TRUE){break}
      }
      tmp_chi2ob[k+1,1] <- sum(((ob-ex)^2)/ex)
      if(round(tmp_chi2ob[k+1,1],2) ==
      round(tmp_chi2ob[k,1],2)){break}
}

if(length(ex)>3){
      chi2ob <- sum(((ob-ex)^2)/ex)
      chi2 <- qchisq(0.95,(length(ex)-3))
      pval <- 1-pchisq(chi2ob,(length(ex)-3))
      print(paste("Log-likelihood =", round(mll,2)), quote =
      FALSE)
      print(paste("Akaike's Information Criterion =",
      round(4-(2*mll),2)), quote = FALSE)
      print(paste("Degrees of Freedom =", length(ex)-3),
      quote = FALSE)
      print(paste("Chi-Squared Statistic =",
      round(chi2ob,2)), quote = FALSE)
      print(paste("Critical Value =", round(chi2,2)), quote
      = FALSE)
      print(paste("p-Value =", round(pval,3)), quote =
      FALSE)
}else{
      print(paste("Log-likelihood =",
      round(logLik(fit)[1],2)), quote = FALSE)
      print(paste("Akaike's Information Criterion =",
      round(4-(2*mll),2)), quote = FALSE)
      print("Not enough data to perform Chi-squared
      goodness-of-fit test")
}


# Turn off the write plotting function

if(write.results == TRUE){
      dev.off()
}
```

```
      # Unload required packages

      detach("package:stats4")


      # Print run-time of program

      if(run.time == TRUE){
            time2 <- Sys.time()
            run_time <- round(unclass(time2-time1),2)
            print(paste("run time =", run_time[1],
            attr(run_time,"units")), quote = FALSE)
      }
}


# Exemplar run command

# bcm_uniform()
```

```
# eSCRIPT 3

# Back-calculation model that retrospectively estimates an
# assumed normalized logistic release window of Legionella
# pneumophila using symptom onset dates.

# Copyright (C) 2010  Joseph R. Egan, Health Protection Agency

# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License as
# published by the Free Software Foundation; either version 2 of
# the License, or (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# For a copy of the GNU General Public License write to the Free
# Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
# Boston, MA  02110-1301, USA.


bcm_logistic <- function(input =
c(0,0,0,0,1,1,1,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4
,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,
7,7,7,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,10,
10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,12,12,12,12,12
,13,13,13,14,16,18),
location = "melbourne", a = 4.96, b = 1/1.27, xmin = -10, xmax =
5, chi2min = 5, pw1 = -10, pw2 = 15, px1 = 0, px2 = 5, py1 = 0,
py2 = 10, w_split = 1, x_split = 1, y_split = 1, confidence =
0.95, plot_prob = 0.99, plot.curve = TRUE, plot.pdf = TRUE,
write.results = FALSE, run.time = TRUE){

      # input: list of symptom onset days (earliest case is day 0)
      # location: location of outbreak
      # a: shape parameter for the gamma distribution
      # b: rate parameter for the gamma distribution
      # xmin: minimum value on the x-axis when plotting
      # xmax: maximum value on the x-axis when plotting
      # chi2min: threshold for expected frequencies in a Chi-
      # squared test
      # pw1: lower limit for the w parameter during the grid-
      # search
      # pw2: upper limit for the w parameter during the grid-
      # search
      # px1: lower limit for the x parameter during the grid-
      # search
      # px2:     upper limit for the x parameter during the grid-
      # search
      # py1: lower limit for the y parameter during the grid-
      # search
      # py2: upper limit for the y parameter during the grid-
      # search
      # w_split: grid division for the w parameter
      # x_split: grid division for the x parameter
      # y_split: grid division for the y parameter
      # confidence: required confidence level
```

```
# plot_prob: determines plot ranges (quantile)
# plot.curve: plot the continuous pdf function
# plot.pdf: plot the discretised daily pdf values
# write.results: write the plot to file
# run.time: print the run-time of the model


# Start the clock

if(run.time == TRUE){
      time1 <- Sys.time()
}


# Useful values and functions

xmax <- (max(input)+xmax)
num_of_cases <- length(input)
epi_curve <- tabulate(input+1)
ci <- c((1-confidence)/2,1-(1-confidence)/2)
plot_range <- c((1-plot_prob)/2,1-(1-plot_prob)/2)

w_total <- (pw2-pw1)*w_split
x_total <- (px2-px1)*x_split
y_total <- (py2-py1)*y_split

fw <- function(x) pw1+(x/w_split)
fx <- function(x) px1+(x/x_split)
fy <- function(x) py1+(x/y_split)


# Write plot to file

if(write.results == TRUE){
      pdf(file = paste(location ,"_logistic.pdf", sep=""))
}


# Probability density function (PDF) for infection dates

inf <- function(t,w,x,y)
ifelse(t>y,0,1/((x*log(1+exp((y-w)/x)))*(1+exp(-(t-w)/x))))



# Convolved PDF for symptom onset dates

convolvedPDF <- function(d,w,x,y){
      result <- c()
      for(i in 1:length(d)){
            integrand <- function(s) dgamma(d[i] -
            s,a,b)*inf(s,w,x,y)
            result[i] <- as.numeric(integrate(integrand, -
            20,d[i])[1])
      }
      #print(-sum(log(result)))
      return(result)
}
```

```
# Log-likelihood function

loglikelihood <- function(w,x,y){
      if(x<=0) return(-Inf)
      result <- sum(log(convolvedPDF(input,w,x,y)))
      return(result)
}


# Maximum likelihood estimation

results <- array(0,c(w_total,x_total,y_total))
for(i in 1:w_total){
      for(j in 1:x_total){
            for(l in 1:y_total){
                  results[i,j,l] <- loglikelihood(fw(i),
                  fx(j), fy(l))
                  print(paste(i,j,l,fw(i),fx(j),fy(l),
                  results[i,j,l]),quote=FALSE)
                  if(results[i,j,l]>0) stop()
            }
      }
}

mll <- max(results)
results <- results - max(results)
results <- exp(results)
if(sum(results) == Inf) stop('Infinite denominator')
results <- results / sum(results)


# Generate estimates for w

w_scale <- fw(1:w_total)
fw_post <- vector(mode='numeric', w_total)
for(i in 1:w_total) fw_post[i] <- sum(results[i,,])
fw_cum <- cumsum(fw_post)
fw_est <- round(fw(which.max(fw_post)))
fw_ci <- c(round(fw(length(fw_cum[fw_cum<ci[1]]))),
round(fw(length(fw_cum[fw_cum<ci[2]])+1)))
fw_lim <- c(round(fw(length(fw_cum[fw_cum<plot_range[1]]))),
round(fw(length(fw_cum[fw_cum<plot_range[2]])+1)))
print(paste("w =", fw_est), quote=FALSE)
print(paste("w (lower) =", fw_ci[1]), quote=FALSE)
print(paste("w (upper) =", fw_ci[2]), quote=FALSE)


# Generate estimates for x

x_scale <- fx(1:x_total)
fx_post <- vector(mode='numeric', x_total)
for(i in 1:x_total) fx_post[i] <- sum(results[,i,])
fx_cum <- cumsum(fx_post)
fx_est <- round(fx(which.max(fx_post)),2)
fx_ci <- c(round(fx(length(fx_cum[fx_cum<ci[1]])),2),
round(fx(length(fx_cum[fx_cum<ci[2]])+1),2))
```

```
fx_lim <-
c(round(fx(length(fx_cum[fx_cum<plot_range[1]])),2),
round(fx(length(fx_cum[fx_cum<plot_range[2]])+1),2))
print(paste("x =", fx_est), quote=FALSE)
print(paste("x (lower) =", fx_ci[1]), quote=FALSE)
print(paste("x (upper) =", fx_ci[2]), quote=FALSE)


# Generate estimates for y

y_scale <- fy(1:y_total)
fy_post <- vector(mode='numeric', y_total)
for(i in 1:y_total) fy_post[i] <- sum(results[,,i])
fy_cum <- cumsum(fy_post)
fy_est <- round(fy(which.max(fy_post)),2)
fy_ci <- c(round(fy(length(fy_cum[fy_cum<ci[1]]))),
round(fy(length(fy_cum[fy_cum<ci[2]])+1),2))
fy_lim <- c(round(fy(length(fy_cum[fy_cum<plot_range[1]]))),
round(fy(length(fy_cum[fy_cum<plot_range[2]])+1),2))
print(paste("y =", fy_est), quote=FALSE)
print(paste("y (lower) =", fy_ci[1]), quote=FALSE)
print(paste("y (upper) =", fy_ci[2]), quote=FALSE)


# Plot epidemic curve

drange <- c(xmin,xmax)
hist(input, breaks = seq(drange[1]+0.5,drange[2]+0.5,0.5),
freq=TRUE, xlim=drange, ylim=c(0,max(epi_curve)+1), main="",
xlab="", ylab="", axes=FALSE, col="black")
axis(1, pos=0, at=c(drange[1]:drange[2]),
labels=c(drange[1]:drange[2]))
title(main="", xlab="Symptom onset day",
ylab="Number of cases")
axis(2, pos=drange[1], las=2)


# Plot discretised PDF values

if(plot.pdf == TRUE){
      for(i in drange[1]:drange[2]){
            rect(i, 0, i+0.5, num_of_cases *
            convolvedPDF(i,fw_est,fx_est,fy_est),
            col="grey")
      }
}


# Plot continuous PDF function

if(plot.curve == TRUE){
      curve(num_of_cases *
      convolvedPDF(x,fw_est,fx_est,fy_est),
      drange[1],drange[2],add=TRUE)
}


# Chi-squared goodness-of-fit test
```

```
tmp_chi2ob <- matrix(0,100,1)
tmp_chi2ob[1,1] <- 0
for(k in 1:100){
      ex <- num_of_cases * convolvedPDF( -
      k:(max(input)+k),fw_est,fx_est,fy_est)
      ob <- c(rep(0,k),epi_curve,rep(0,k))
      for(j in 0:length(ex)){
            for(i in 1:length(ex)){
                  if(sum(ex[(1+j):(i+j)])<chi2min){
                        ex[i+j+1]<-ex[i+j+1]+ex[i+j]
                        ob[i+j+1]<-ob[i+j+1]+ob[i+j]
                        ex[i+j]<-0
                        ob[i+j]<--1
                  }else{
                        break
                  }
            }
            ex <- ex[ex>0]
            ob <- ob[ob>-1]
            for(i in 1:length(ex)){
                  if(sum(ex[(length(ex)-j):(length(ex)+1-i -
                  j)])<chi2min){
                        ex[length(ex)-i-j]<-ex[length(ex)-i -
                        j]+ex[length(ex)-i-j+1]
                        ob[length(ex)-i-j]<-ob[length(ob)-i -
                        j]+ob[length(ob)-i-j+1]
                        ex[length(ex)-i-j+1]<-0
                        ob[length(ob)-i-j+1]<--1
                  }else{
                        break
                  }
            }
            ex <- ex[ex>0]
            ob <- ob[ob>-1]
            if(all(ex>chi2min)==TRUE){break}
      }
      tmp_chi2ob[k+1,1] <- sum(((ob-ex)^2)/ex)
      if(round(tmp_chi2ob[k+1,1],2) ==
      round(tmp_chi2ob[k,1],2)){break}
}

if(length(ex)>4){
      chi2ob <- sum(((ob-ex)^2)/ex)
      chi2 <- qchisq(0.95,(length(ex)-4))
      pval <- 1-pchisq(chi2ob,(length(ex)-4))
      print(paste("Maximised log-likelihood =",
      round(mll,2)), quote = FALSE)
      print(paste("Akaike's Information Criterion =",
      round(6-(2*mll),2)), quote = FALSE)
      print(paste("Degrees of Freedom =", length(ex)-4),
      quote = FALSE)
      print(paste("Chi-Squared Statistic =",
      round(chi2ob,2)), quote = FALSE)
      print(paste("Critical Value =", round(chi2,2)), quote
      = FALSE)
      print(paste("p-Value =", round(pval,3)), quote =
      FALSE)
}else{
```

```
            print(paste("Maximised log Likelihood =", mll), quote
            = FALSE)
            print(paste("Akaike's Information Criterion =",
            round(6-(2*mll),2)), quote = FALSE)
            print("Not enough data to perform Chi-squared
            goodness-of-fit test")
      }


      # Turn off write plotting function

      if(write.results == TRUE){
            dev.off()
      }


      # Print run-time of program

      if(run.time == TRUE){
            time2 <- Sys.time()
            run_time <- round(unclass(time2-time1),2)
            print(paste("run time =", run_time[1],
            attr(run_time,"units")), quote = FALSE)
      }
}


# Exemplar run command

# bcm_logistic()
```

```
# eSCRIPT 4

# Real-time model that prospectively estimates an
# assumed uniform release window of Legionella pneumophila using
# symptom onset dates.

# Copyright (C) 2010  Joseph R. Egan, Health Protection Agency.

# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License as
# published by the Free Software Foundation; either version 2 of
# the License, or (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# For a copy of the GNU General Public License write to the Free
# Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
# Boston, MA  02110-1301, USA.


# For simulated outbreaks only

simulate <- function(n, w, tau, p1 = 4.96, p2 = 1/1.27){
       t <- runif(n,0,w)+rgamma(n,p1,p2)
       x <- -min(t)
       y <- x+w
       y <- ifelse(y>tau,tau,y)
       t <- t+x
       t <- t[t<tau]
       print(paste("k =",length(t)),quote = FALSE)
       print(paste("x =",round(x, 1)),quote = FALSE)
       print(paste("y =",round(y, 1)),quote = FALSE)
       return(t)
}


############################################################


rtm_uniform <- function(t, tau, a = 4.96, b = 1/1.27, confidence
= 0.95, plot_prob = 0.99, pn1 = 0, pn2 = 200, px1 = -5, px2 = 0,
py1 = -5, py2 = tau, x_split = 4, y_split = 4, n_split = 1,
plot.results = TRUE, write.results = TRUE, run.time = TRUE){

      # t: list of symptom onset dates (earliest case is day 0)
      # tau: censoring time
      # a: shape parameter for the gamma distribution
      # b: rate parameter for the gamma distribution
      # confidence: required confidence level
      # plot_prob: determines plot ranges (quantile)
      # pn1: lower limit for the n parameter during the grid-
      # search
      # pn2: upper limit for the n parameter during the grid-
      # search
      # px1: lower limit for the x parameter during the grid-
      # search
```

```
# px2: upper limit for the x parameter during the grid-
# search
# py1: lower limit for the y parameter during the grid-
# search
# py2: upper limit for the y parameter during the grid-
# search
# n_split: grid division for the n parameter
# x_split: grid division for the x parameter
# y_split: grid division for the y parameter
# write.results: write results to file
# plot.results: plot results
# run.time: print the run-time of the model


# Start the clock

if(run.time == TRUE){
      time1 <- Sys.time()
}


# Useful values and functions

if(confidence>plot_prob) plot_prob <- confidence
ci <- c((1-confidence)/2, 1-(1-confidence)/2)
plot_range <- c((1-plot_prob)/2, 1-(1-plot_prob)/2)
k <- length(t)

x_total <- (px2-px1)*x_split
y_total <- (py2-py1)*y_split
n_total <- (pn2-pn1)*n_split

fx <- function(x) px1+(x/x_split)
fy <- function(x) py1+(x/y_split)
fn <- function(x) pn1+(x/n_split)


# Convolved probability density function for symptom onset
# dates

f <- function(t,x,y) (pgamma(t-x,a,b)-pgamma(t-y,a,b))/(y-x)


# Convolved probability distribution function for symptom
# onset dates

igf <- function(u,v) pgamma(v,u, lower=FALSE) * gamma(u)
sol <- function(z) (z-a)*gamma(a)-z*igf(a,z)+igf(a+1,z)
F <- function(t,x,y) (sol(b*(t-x))-sol(b*(t-y)))/(b*(y -
x)*gamma(a))


# Log-likelihood function

loglikelihood <- function(n,x,y){
      if(n<k) return(-Inf)
      if(x>=0) return(-Inf)
      if(y-x<0.5) return(-Inf)
      if(y>tau) return(-Inf)
```

```
      if(n==k){
            result <- sum(log(f(t,x,y)))
      }else{
            result <- lgamma(n+1) - lgamma(n-k+1) -
            lgamma(k+1) + (n-k)*log(1-F(tau,x,y)) +
            sum(log(f(t,x,y)))
      }
      return(result)
}


# Maximum likelihood estimation

results <- array(0,c(n_total,x_total,y_total))
for(i in 1:n_total){
      for(j in 1:x_total){
            for(l in 1:y_total){
                  results[i,j,l] <- loglikelihood(fn(i),
                  fx(j), fy(l))
                  print(paste(i,j,l, fn(i), fx(j), fy(l),
                  results[i,j,l]), quote=FALSE)
                  if(results[i,j,l]>0) stop()
            }
      }
}

results <- results - max(results)
results <- exp(results)
if(sum(results) == Inf) stop('Infinite denominator')
results <- results / sum(results)


# Generate estimates for n

n_scale <- fn(1:n_total)
fn_post <- vector(mode='numeric', n_total)
for(i in 1:n_total) fn_post[i] <- sum(results[i,,])
fn_cum <- cumsum(fn_post)
fn_est <- round(fn(which.max(fn_post)))
fn_ci <- c(round(fn(length(fn_cum[fn_cum<ci[1]]))),
round(fn(length(fn_cum[fn_cum<ci[2]])+1)))
fn_lim <- c(round(fn(length(fn_cum[fn_cum<plot_range[1]]))),
round(fn(length(fn_cum[fn_cum<plot_range[2]])+1)))
print(paste("n =", fn_est), quote=FALSE)
print(paste("n (lower) =", fn_ci[1]), quote=FALSE)
print(paste("n (upper) =", fn_ci[2]), quote=FALSE)


# Generate estimates for x

x_scale <- fx(1:x_total)
fx_post <- vector(mode='numeric',x_total)
for(i in 1:x_total) fx_post[i] <- sum(results[,i,])
fx_cum <- cumsum(fx_post)
fx_est <- round(fx(which.max(fx_post)),2)
fx_ci <- c(round(fx(length(fx_cum[fx_cum < ci[1]])),2),
round(fx(length(fx_cum[fx_cum < ci[2]])+1),2))
```

```
fx_lim <- c(round(fx(length(fx_cum[fx_cum <
plot_range[1]])),2), round(fx(length(fx_cum[fx_cum <
plot_range[2]])+1),2))
print(paste("x =", fx_est), quote=FALSE)
print(paste("x (lower) =", fx_ci[1]), quote=FALSE)
print(paste("x (upper) =", fx_ci[2]), quote=FALSE)


# Generate estimates for y

y_scale <- fy(1:y_total)
fy_post <- vector(mode='numeric',y_total)
for(i in 1:y_total) fy_post[i] <- sum(results[,,i])
fy_cum <- cumsum(fy_post)
fy_est <- round(fy(which.max(fy_post)),2)
fy_ci <- c(round(fy(length(fy_cum[fy_cum<ci[1]]))),
round(fy(length(fy_cum[fy_cum<ci[2]])+1),2))
fy_lim <- c(round(fy(length(fy_cum[fy_cum<plot_range[1]]))),
round(fy(length(fy_cum[fy_cum<plot_range[2]])+1),2))
print(paste("y =", fy_est), quote=FALSE)
print(paste("y (lower) =", fy_ci[1]), quote=FALSE)
print(paste("y (upper) =", fy_ci[2]), quote=FALSE)


# Plot results

if(plot.results == TRUE){
      x11()
      layout(matrix(c(1,2,3),3,1,byrow = TRUE))
      plot(n_scale, fn_post, xlim=c(fn_lim[1],fn_lim[2]),
      xlab='Outbreak size', ylab='Likelihood', type='l')
      abline(v=fn_ci, lty=2, col='grey')
      abline(v=fn_est, col='grey')

      plot(x_scale, fx_post, xlim=c(fx_lim[1],fx_lim[2]),
      xlab='Release start', ylab='Likelihood', type='l')
      abline(v=fx_ci, lty=2, col='grey')
      abline(v=fx_est, col='grey')

      plot(y_scale, fy_post, xlim=c(fy_lim[1],fy_lim[2]),
      xlab='Release end', ylab='Likelihood', type='l')
      abline(v=fy_ci, lty=2, col='grey')
      abline(v=fy_est, col='grey')
}


# Write results to file if required

if(write.results == TRUE){
      write.table(cbind(n_scale,fn_post),
      paste(tau,'_fn_post.csv',sep=""), sep=',',
      row.names=FALSE, col.names=FALSE)
      write.table(cbind(x_scale,fx_post),
      paste(tau,'_fx_post.csv',sep=""), sep=',',
      row.names=FALSE, col.names=FALSE)
      write.table(cbind(y_scale,fy_post),
      paste(tau,'_fy_post.csv',sep=""), sep=',',
      row.names=FALSE, col.names=FALSE)
}
```

```
      # Print run-time of program

      if(run.time == TRUE){
            time2 <- Sys.time()
            run_time <- round(unclass(time2-time1),2)
            print(paste("run time =", run_time[1],
            attr(run_time,"units")), quote = FALSE)
      }
}


# Exemplar run commands

# input <- simulate(100,10,5)
# rtm_uniform(input,5)
```