Cumulative Hazard Assessment: Issues for the FIFRA Scientific Advisory Panel

Appendix 4. R Source code for the simple pharmacokinetic risk assessment model example

“

February 15-18, 2005
National Airport Holiday Inn
Arlington, VA

Prepared by:
Office of Pesticide Programs
US Environmental Protection Agency

Appendix 4: R Source Code for the Simple Pharmacokinetic Risk Assessment Model Example

The example in section IV.C. (Figure 1.5) was generated by the following script. This script requires that the package CarbUtils (which is listed in the next section) be installed. All source code listed in this example is available in binary form from the same web site where this document was found.

RAexamp1.R

The following script generated Figure 1.5:

```
### RAexamp1.R
library(CarbUtils)

### Go from P to tz

P2tz <- function(P) {
  log(P/(1 - P))
}

### Time-course due to 8:00 exposure
### Times from 8 to 24, every 10 minutes
x1 <- seq(8,24,by=1/6)
### Times from 11 to 24, every 10 minutes
x2 <- seq(11,24,by=1/6)
### Times from 12 to 24, every 10 minutes
x3 <- seq(12,24,by=1/6)

### Inhibition fractions due to first exposure

y1a <- tcminhfn(1/70,x1 - 8,P2tz(0.3),lD=log(300),lg=0,lTr=log(0.75),lTmax=log(0.25))
y1b <- tcminhfn(0.5/70,x1 - 8,P2tz(0.3),lD=log(150),lg=0,lTr=log(1.5),lTmax=log(0.25))
y1c <- tcminhfn(0.02/70,x1 -
8,P2tz(0.3),lD=log(100),lg=0,lTr=log(1.25),lTmax=log(0.25))

y1 <- y1a + y1b + y1c
### Make a postscript file for the example, but not everyone using windows can read
### a postscript file
### postscript(file="RAexamp1.ps",width=3,height=3,pointsize=10,horizontal=FALSE)
par(las=1,mar=c(5,5,0,0)+.1)

plot(x1,y1a,type="l",xlab="Time of Day (24 hour clock)",ylab="",
     ylim=c(0,1.5e-05),
     xlim=c(0,24),col="red",yaxt="n")
lines(x1,y1b,col="blue")
lines(x1,y1c,col="green")
## lines(x1,y1)

### Inhibition fractions due to second exposure, at 11:00

Act <- 1 - tcminhfn(1/70,11 -
8,P2tz(0.3),lD=log(300),lg=0,lTr=log(0.75),lTmax=log(0.25)) -
  tcminhfn(0.5/70,11 - 8,P2tz(0.3),lD=log(150),lg=0,lTr=log(1.5),lTmax=log(0.25)) -
  tcminhfn(0.02/70,11 - 8,P2tz(0.3),lD=log(100),lg=0,lTr=log(1.25),lTmax=log(0.25))
y2b <- Act*tcminhfn(0.1/70,x2-
11,P2tz(0.3),lD=log(150),lg=0,lTr=log(1.5),lTmax=log(0.25))

y2 <- y1
y2[x1 >=11] <- y2[x1 >= 11] + y2b

lines(x2,y2b,col="blue")
## lines(x1,y2,col="red")

Act <- 1 - y2[x1 == 12]
```

```
y3 <- y2

y3a <- Act*tcminhfn(2/70,x3 -
12,P2tz(0.3),lD=log(300),lg=0,lTr=log(0.75),lTmax=log(0.25))
y3b <- Act*tcminhfn(0.2/70,x3 -
12,P2tz(0.3),lD=log(150),lg=0,lTr=log(1.5),lTmax=log(0.25))
y3c <- Act*tcminhfn(0.1/70,x3 -
12,P2tz(0.3),lD=log(100),lg=0,lTr=log(1.25),lTmax=log(0.25))
y3abc <- y3a + y3b + y3c

y3[x1 >= 12] <- y3[x1 >= 12] + y3abc

lines(x3,y3a,col="red")
lines(x3,y3b,col="blue")
lines(x3,y3c,col="green")

lines(x1,y3)
axis(2,at=c(0,5e-6,1e-5,1.5e-5),
     labels=expression(0,5 %*% 10^-6,1 %*% 10^-5, 1.5 %*% 10^-5),las=1
     )
mtext("Fractional AChE Inhibition",side=2,line=4,las=0)
### Don't close the Windows device or x11()
### dev.off()
```

Package CarbUtils

CarbUtils is a package of utility functions for modeling *N*-methyl carbamate dose-time response data and related activities. While the entire package is installed when the associated package CarbUtils is installed in R, only the functions actually called by the simple pk-based risk assessment model example are listed here.

tcminhfn()

```
tcminh <- expression(exp(lA)*((1 - (1 - R)/(1 + exp(-tz)))*(1 - exp(log(exp(-tz)*(1 -
R)/(exp(-tz) + R))*(x/exp(lD))^exp(lg)))*
    ((exp(-log(2)*Tpd/exp(lTr)) - exp(-log(2)*Tpd/exp(lTa)))/
     (exp(-log(2)*exp(lTmax)/exp(lTr)) - exp(-log(2)*exp(lTmax)/exp(lTa))))) +
    b * Tc)

## tcminhfn <- ##deriv(tcminh,c("lA","tz","lD","lg","lTr","lTa","lTmax","b"),
##c("x","Tpd","Tc","lA","tz","lD","lg","lTr","lTmax","b","R"))
## formals(tcminhfn)$R <- 0.1
## formals(tcminhfn)$lg <- 0.0
## formals(tcminhfn)$b <- 0.0

tcminhfn <-
  function (x, Tpd, tz, lD, lg = 0, lTr, lTmax,
            R = 0.1)
{
  .expr2 <- 1 - R
  .expr4 <- exp(-tz)
  .expr5 <- 1 + .expr4
  .expr7 <- 1 - .expr2/.expr5
  .expr8 <- .expr4 * .expr2
  .expr9 <- .expr4 + R
  .expr10 <- .expr8/.expr9
  .expr11 <- log(.expr10)
  .expr12 <- exp(lD)
  .expr13 <- x/.expr12
  .expr14 <- exp(lg)
  .expr15 <- .expr13^.expr14
  .expr17 <- exp(.expr11 * .expr15)
  .expr18 <- 1 - .expr17
  .expr19 <- .expr7 * .expr18
  .expr20 <- log(2)
  .expr21 <- -.expr20
  .expr22 <- .expr21 * Tpd
  .expr23 <- exp(lTr)
  .expr25 <- exp(.expr22/.expr23)
```

```
  .expr30 <- exp(lTmax)
  lTa <- lTaest(lTr, .expr30)
  .expr26 <- exp(lTa)
  .expr28 <- exp(.expr22/.expr26)
  .expr29 <- .expr25 - .expr28
  .expr31 <- .expr21 * .expr30
  .expr33 <- exp(.expr31/.expr23)
  .expr35 <- exp(.expr31/.expr26)
  .expr36 <- .expr33 - .expr35
  .expr37 <- .expr29/.expr36
  .expr19 * .expr37
}
```

Calculate log(Ta) from lTr and Tmax:

```
"lTaest" <-
  function (lTr, Tmax)
{
  log2 <- log(2)
  Tr <- exp(lTr)
  arg1 <- (Tmax * log2 + lTr * Tr)
  arg2 <- -Tmax * log2 * exp(-arg1/Tr)
  lTa <- Re((lambertw(arg2, 0) * Tr + arg1)/Tr)
  if (any(abs(lTr - lTa) < 1e-6)) {
    lTam1 <- Re((lambertw(arg2, -1) * Tr + arg1)/Tr)
    lTa <- ifelse(abs(lTa - lTr) < 1e-6,lTam1,lTa)
  }
  lTa
}
```

Calculate Lambert's W function. This function is a translation of the function in the octave-forge collection of packages for octave (http://www.octave.org), which is included in the comments at the end of the R function. This code was checked by comparing results from this code against results from the corresponding function in Maple 8 (Maple version 8.01, from Waterloo Maple Inc.).

```
## usage: lambertw(z, n)
##
## This R function is a translation of the octave function of
## the same name, with the exception that the order of the arguments is
## reversed.  The orignal function is included in the extended comments
## at the end of this file.  lambertsw (w()) has is defined as the function
## such that w(x) * exp(e(x)) = x.  It has multiple branches, but only
## branches 0 and -1 give real values, and then only for x > -1/e.
## This implementation is for complex z, and defaults to branch b=0.

lambertw <- function(z, b=0) {
  z <- as.complex(z)
  ## some error checking
  if (any(round(Re(b)) != b))
    stop('branch number for lambertw must be integer')

    ## series expansion about -1/e
    #
    # p = (1 - 2*abs(b))*sqrt(2*e*z + 2);
    # w = (11/72)*p;
    # w = (w - 1/3)*p;
    # w = (w + 1)*p - 1
    #
    # first-order version suffices:
    #
  e <- exp(1)
  eps <- .Machine$double.eps
  w <- (1 - 2*abs(b))*sqrt(2*e*z + 2) - 1;

  ## asymptotic expansion at 0 and Inf
                                      #
  v <- log(z + !(Mod(z) | b)) + 2*pi*(0+1i)*b;
  v <- v - log(v + !Mod(v));
```

```
  ## choose strategy for initial guess
                                        #
  c <- abs(z + 1/e);
  c <- (c > 1.45 - 1.1*abs(b));
  c <- c | (b*Im(z) > 0) | (!Im(z) & (b == 1));
  w <- (1 - c)*w + c*v;

  ## Halley iteration
                                        #
  for (n in 1:10) {
    p = exp(w);
    t = w*p - z;
    f = (w != -1);
    t = f*t/(p*(w + f) - 0.5*(w + 2.0)*t/(w + f));
    w = w - t;
    if (abs(Re(t)) < (2.48*eps)*(1.0 + abs(Re(w)))
        && abs(Im(t)) < (2.48*eps)*(1.0 + abs(Im(w))))
      return (w)
  }
  warning('iteration limit reached, result of lambertw may be inaccurate');
  w
}

### %% usage: lambertw(z) or lambertw(n,z)
### %%
### %% Compute the Lambert W function of z.  This function satisfies
### %% W(z).*exp(W(z)) = z, and can thus be used to express solutions
### %% of transcendental equations involving exponentials or logarithms.
### %%
### %% n must be integer, and specifies the branch of W to be computed;
### %% W(z) is a shorthand for W(0,z), the principal branch.  Branches
### %% 0 and -1 are the only ones that can take on non-complex values.
### %%
### %% If either n or z are non-scalar, the function is mapped to each
### %% element; both may be non-scalar provided their dimensions agree.
### %%
### %% This implementation should return values within 2.5*eps of its
### %% counterpart in Maple V, release 3 or later.  Please report any
### %% discrepancies to the author, Nici Schraudolph <schraudo@inf.ethz.ch>.
### %%
### %% For further details, see:
### %%
### %% Corless, Gonnet, Hare, Jeffrey, and Knuth (1996), "On the Lambert
### %% W Function", Advances in Computational Mathematics 5(4):329-359.

### %% Author:   Nicol N. Schraudolph <schraudo@inf.ethz.ch>
### %% Version:  1.0
### %% Created:  07 Aug 1998
### %% Keywords: Lambert W Omega special transcendental function

### %% Copyright (C) 1998 by Nicol N. Schraudolph
### %%
### %% This program is free software; you can redistribute and/or
### %% modify it under the terms of the GNU General Public
### %% License as published by the Free Software Foundation;
### %% either version 2, 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.
### %%
### %% You should have received a copy of the GNU General Public
### %% License along with Octave; see the file COPYING.  If not,
### %% write to the Free Software Foundation, 59 Temple Place -
### %% Suite 330, Boston, MA 02111-1307, USA.

### function w = lambertw(b,z)
###     if (nargin == 1)
###         z = b;
###         b = 0;
###     else
```

```
###         %% some error checking
###         %
###         if (nargin != 2)
###             usage('result = lambertw(branch, argument)')
###         else
###             if (any(round(real(b)) != b))
###                 usage('branch number for lambertw must be integer')
###             end
###         end
###     end

###     %% series expansion about -1/e
###     %
###     % p = (1 - 2*abs(b)).*sqrt(2*e*z + 2);
###     % w = (11/72)*p;
###     % w = (w - 1/3).*p;
###     % w = (w + 1).*p - 1
###     %
###     % first-order version suffices:
###     %
###     w = (1 - 2*abs(b)).*sqrt(2*e*z + 2) - 1;

###     %% asymptotic expansion at 0 and Inf
###     %
###     v = log(z + ~(z | b)) + 2*pi*I*b;
###     v = v - log(v + ~v);

###     %% choose strategy for initial guess
###     %
###     c = abs(z + 1/e);
###     c = (c > 1.45 - 1.1*abs(b));
###     c = c | (b.*imag(z) > 0) | (~imag(z) & (b == 1));
###     w = (1 - c).*w + c.*v;

###     %% Halley iteration
###     %
###     for n = 1:10
###         p = exp(w);
###         t = w.*p - z;
###         f = (w != -1);
###         t = f.*t./(p.*(w + f) - 0.5*(w + 2.0).*t./(w + f));
###         w = w - t;
###         if (abs(real(t)) < (2.48*eps)*(1.0 + abs(real(w)))
###             && abs(imag(t)) < (2.48*eps)*(1.0 + abs(imag(w))))
###             return
###         end
###     end
###     warning('iteration limit reached, result of lambertw may be inaccurate');
### end
```