III. Appendices

B. Hazard/RPF

4. R Programs for the Revised Analysis

a. Part 1: Package RBMDS

RBMDS is a package of utility functions written to facilitate the analysis of the cholinesterase activity dose-response data. These functions are made available to the scripts that carry out the data analysis by including the call “require(RBMDS)” or “library(RBMDS)” at the beginning of the scripts.

```
### Generalization of exponential decreasing model
### A, B, and m are constrained to be strictly positive
CexpB <- function (Dose, A, B, m)
{
  ## exp(A)*(1/(1 + exp(-B)) + (exp(-B)/(1 + exp(-B)) * exp(-exp(m)*Dose)))
  .expr1 <- exp(A)
  .expr3 <- exp(-B)
  .expr4 <- 1 + .expr3
  .expr6 <- exp(m)
  .expr9 <- exp(-.expr6 * Dose)
  .expr10 <- .expr3 * .expr9
  .expr11 <- .expr10/.expr4
  .expr13 <- .expr1 * (1/.expr4 + .expr11)
  .expr14 <- .expr4^2
  .value <- .expr13
  .grad <- array(0, c(length(.value), 3), list(NULL, c("A",
                                                       "B", "m")))
  .grad[, "A"] <- .expr13
  .grad[, "B"] <- .expr1 * (.expr3/.expr14 - (.expr11 - .expr10 *
                                                .expr3/.expr14))
  .grad[, "m"] <- -.expr1 * (.expr3 * (.expr9 * (.expr6 *
                                                   Dose))/.expr4)
  attr(.value, "gradient") <- .grad
  .value
}

### CexpB2: same as above, but gradient includes deriv wrt dose.

CexpB2 <-
function (Dose, A, B, m)
{
    .expr1 <- exp(A)
    .expr3 <- exp(-B)
    .expr4 <- 1 + .expr3
    .expr6 <- .expr3/.expr4
    .expr7 <- exp(m)
    .expr10 <- exp(-.expr7 * Dose)
    .expr13 <- .expr1 * (1/.expr4 + .expr6 * .expr10)
    .expr18 <- .expr4^2
    .value <- .expr13
    .grad <- array(0, c(length(.value), 4), list(NULL, c("Dose",
        "A", "B", "m")))
    .grad[, "Dose"] <- -.expr1 * (.expr6 * (.expr10 * .expr7))
    .grad[, "A"] <- .expr13
```

```
    .grad[, "B"] <- .expr1 * (.expr3/.expr18 - (.expr6 - .expr3 *
        .expr3/.expr18) * .expr10)
    .grad[, "m"] <- -.expr1 * (.expr6 * (.expr10 * (.expr7 *
        Dose)))
    attr(.value, "gradient") <- .grad
    .value
}


### CpkexpB: above model, with the additional assumption that there is
### saturable detoxification.  This is implemented by composing CexpB
### (now assuming that Dose in CexpB refers to internal dose), with
### a model that relates administered dose to internal dose:
###
### idose = 0.5*((Dose - S - D) + sqrt((Dose - S - D)^2 + 4*Dose*S))
###
### This model approaches a line with slope 1 and y-intercept -D as Dose
### increases to infinity.  The parameter 'S' controls the shape of the
### low-dose part of the curve.  For values close to 0, the curve looks
### very threshold-like; as S increases, the curve becomes more gradual.
### S and D must be positive
###
### The final function is (Dose refers to administered dose; exponentiation
### used to force positive parameters):
###
###
## exp(A)*(1/(1 + exp(-B)) + (exp(-B)/(1 + exp(-B)) * exp(-exp(m)*0.5*((Dose
- exp(S) - exp(D)) + sqrt((Dose - exp(S) - exp(D))^2 + 4*Dose*exp(S))))))
##


CpkexpB <- function (Dose, A, B, m, S, D)
{
  idose <- CpkB(Dose, S, D)
  .value <- CexpB2(idose, A, B, m)
  .grad <- array(0, c(length(.value), 5),
                 list(NULL,
                      c("A","B","m","S","D")))
  .grad[,c("A","B","m")] <- attr(.value,"gradient")[,c("A","B","m")]
  .grad[,c("S","D")] <- attr(.value,"gradient")[,"Dose"] *
    attr(idose, "gradient")[,c("S", "D")]
  attr(.value, "gradient") <- .grad
  .value
}

### CexpBS(dose, A, B, m, sex, fixed=NULL) allows fixed, a named list of
### vectors of fixed values for the model CexpB.  They can differ by sex.
### used, for example:
### nlme(model=chei ~ CexpBS(dose, A, m, sex,
###      fixed=list(B=c(F=-3.01, B=-2.78))), data=mydata, random=A+m~1 |
mrid)


CexpBS <- function(dose, A, B, m, sex, fixed=NULL) {
  callList <- vector("list",4)
  names(callList) <- c("Dose","A","B","m")
  sex <- switch(length(fixed) + 1,
                sex,
                m,
                B)

  callList[["Dose"]] <- dose
  ### Assume we will always estimate A
  callList[["A"]] <- A
  if ("B" %in% names(fixed)) {
```

```
    callList[["B"]] <- fixed[["B"]][as.character(sex)]
    callList[["m"]] <- if ("m" %in% names(fixed)) {
      fixed[["m"]][as.character(sex)]
    } else {
      B
    }
  } else {
    callList[["B"]] <- B
    callList[["m"]] <- if ("m" %in% names(fixed)) {
      fixed[["m"]][as.character(sex)]
    } else {
      m
    }
  }
  .value <- do.call("CexpB",callList)
  .grad <- attr(.value, "gradient")
  .grad <- .grad[,-match(names(fixed),colnames(.grad)),drop=FALSE]
  attr(.value,"gradient") <- .grad
  .value
}

### This is the same model, reparameterized so that, instead of 'm', we
### estimate 'BMD', for a BMR*100% reduction in mean response relative to
### control.
###
### Model is:
### ~ exp(A)*(1/(1 + exp(-B)) + exp(-B)/(1 + exp(-B))*exp(log(1 - BMR*(1 +
exp(B))) * Dose*exp(-BMD)))

CexpBwD <- function (Dose, A, B, BMD, BMR=0.10)
{
    .expr1 <- exp(A)
    .expr3 <- exp(-B)
    .expr4 <- 1 + .expr3
    .expr6 <- .expr3/.expr4
    .expr7 <- exp(B)
    .expr10 <- 1 - BMR * (1 + .expr7)
    .expr14 <- exp(-BMD)
    .expr15 <- log(.expr10) * Dose * .expr14
    .expr16 <- exp(.expr15)
    .expr19 <- .expr1 * (1/.expr4 + .expr6 * .expr16)
    .expr20 <- .expr4^2
    .value <- .expr19
    .grad <- array(0, c(length(.value), 3), list(NULL, c("A",
        "B", "BMD")))
    .grad[, "A"] <- .expr19
    .grad[, "B"] <- .expr1 * (.expr3/.expr20 - (.expr6 * (.expr16 *
        (BMR * .expr7/.expr10 * Dose * .expr14)) + (.expr6 -
        .expr3 * .expr3/.expr20) * .expr16))
    .grad[, "BMD"] <- -.expr1 * (.expr6 * (.expr16 * .expr15))
    attr(.value, "gradient") <- .grad
    .value
}
## The following returns the hessian, too (for calculating covariances
## from nlme models).
CexpBwDH <- function (Dose, A, B, BMD, BMR=0.10)
{
    .expr1 <- exp(A)
    .expr3 <- exp(-B)
    .expr4 <- 1 + .expr3
    .expr6 <- .expr3/.expr4
    .expr7 <- exp(B)
    .expr10 <- 1 - BMR * (1 + .expr7)
    .expr14 <- exp(-BMD)
    .expr15 <- log(.expr10) * Dose * .expr14
```

```
    .expr16 <- exp(.expr15)
    .expr19 <- .expr1 * (1/.expr4 + .expr6 * .expr16)
    .expr20 <- .expr4^2
    .expr21 <- .expr3/.expr20
    .expr22 <- BMR * .expr7
    .expr23 <- .expr22/.expr10
    .expr25 <- .expr23 * Dose * .expr14
    .expr26 <- .expr16 * .expr25
    .expr28 <- .expr3 * .expr3
    .expr30 <- .expr6 - .expr28/.expr20
    .expr34 <- .expr1 * (.expr21 - (.expr6 * .expr26 + .expr30 *
        .expr16))
    .expr35 <- .expr16 * .expr15
    .expr38 <- -.expr1 * (.expr6 * .expr35)
    .expr40 <- 2 * (.expr3 * .expr4)
    .expr42 <- .expr20^2
    .expr55 <- .expr30 * .expr26
    .value <- .expr19
    .grad <- array(0, c(length(.value), 3), list(NULL, c("A",
        "B", "BMD")))
    .hessian <- array(0, c(length(.value), 3, 3), list(NULL,
        c("A", "B", "BMD"), c("A", "B", "BMD")))
    .grad[, "A"] <- .expr19
    .hessian[, "A", "A"] <- .expr19
    .hessian[, "A", "B"] <- .hessian[, "B", "A"] <- .expr34
    .hessian[, "A", "BMD"] <- .hessian[, "BMD", "A"] <- .expr38
    .grad[, "B"] <- .expr34
    .hessian[, "B", "B"] <- -.expr1 * (.expr21 - .expr3 * .expr40/.expr42 +
        .expr6 * (.expr16 * ((.expr23 + .expr22 * .expr22/.expr10^2) *
            Dose * .expr14) - .expr26 * .expr25) - .expr55 -
            (.expr55 + (.expr30 - ((.expr28 + .expr28)/.expr20 -
                .expr28 * .expr40/.expr42)) * .expr16))
    .hessian[, "B", "BMD"] <- .hessian[, "BMD", "B"] <- .expr1 *
        (.expr30 * .expr35 + .expr6 * (.expr26 + .expr35 * .expr25))
    .grad[, "BMD"] <- .expr38
    .hessian[, "BMD", "BMD"] <- .expr1 * (.expr6 * (.expr35 +
        .expr35 * .expr15))
    attr(.value, "gradient") <- .grad
    attr(.value, "hessian") <- .hessian
    .value
}

### Include the derivative wrt Dose, to combine with the Pk model:

CexpBwD2 <- function (Dose, A, B, BMD, BMR=0.10)
{
    .expr1 <- exp(A)
    .expr3 <- exp(-B)
    .expr4 <- 1 + .expr3
    .expr6 <- .expr3/.expr4
    .expr7 <- exp(B)
    .expr10 <- 1 - BMR * (1 + .expr7)
    .expr11 <- log(.expr10)
    .expr14 <- exp(-BMD)
    .expr15 <- .expr11 * Dose * .expr14
    .expr16 <- exp(.expr15)
    .expr19 <- .expr1 * (1/.expr4 + .expr6 * .expr16)
    .expr24 <- .expr4^2
    .value <- .expr19
    .grad <- array(0, c(length(.value), 4), list(NULL, c("Dose",
        "A", "B", "BMD")))
    .grad[, "Dose"] <- .expr1 * (.expr6 * (.expr16 * (.expr11 *
        .expr14)))
    .grad[, "A"] <- .expr19
    .grad[, "B"] <- .expr1 * (.expr3/.expr24 - (.expr6 * (.expr16 *
```

```
        (BMR * .expr7/.expr10 * Dose * .expr14)) + (.expr6 -
        .expr3 * .expr3/.expr24) * .expr16))
    .grad[, "BMD"] <- -.expr1 * (.expr6 * (.expr16 * .expr15))
    attr(.value, "gradient") <- .grad
    .value
}

### Again, with the hessian:
CexpBwD2H <- function (Dose, A, B, BMD, BMR=0.1)
{
    .expr1 <- exp(A)
    .expr3 <- exp(-B)
    .expr4 <- 1 + .expr3
    .expr6 <- .expr3/.expr4
    .expr7 <- exp(B)
    .expr10 <- 1 - BMR * (1 + .expr7)
    .expr11 <- log(.expr10)
    .expr14 <- exp(-BMD)
    .expr15 <- .expr11 * Dose * .expr14
    .expr16 <- exp(.expr15)
    .expr19 <- .expr1 * (1/.expr4 + .expr6 * .expr16)
    .expr20 <- .expr11 * .expr14
    .expr21 <- .expr16 * .expr20
    .expr23 <- .expr1 * (.expr6 * .expr21)
    .expr27 <- BMR * .expr7
    .expr28 <- .expr27/.expr10
    .expr32 <- .expr28 * Dose * .expr14
    .expr33 <- .expr16 * .expr32
    .expr37 <- .expr3 * .expr3
    .expr38 <- .expr4^2
    .expr40 <- .expr6 - .expr37/.expr38
    .expr45 <- .expr16 * .expr15
    .expr51 <- .expr3/.expr38
    .expr56 <- .expr1 * (.expr51 - (.expr6 * .expr33 + .expr40 *
        .expr16))
    .expr59 <- -.expr1 * (.expr6 * .expr45)
    .expr61 <- 2 * (.expr3 * .expr4)
    .expr63 <- .expr38^2
    .expr76 <- .expr40 * .expr33
    .value <- .expr19
    .grad <- array(0, c(length(.value), 4), list(NULL, c("Dose",
        "A", "B", "BMD")))
    .hessian <- array(0, c(length(.value), 4, 4), list(NULL,
        c("Dose", "A", "B", "BMD"), c("Dose", "A", "B", "BMD")))
    .grad[, "Dose"] <- .expr23
    .hessian[, "Dose", "Dose"] <- .expr1 * (.expr6 * (.expr21 *
        .expr20))
    .hessian[, "Dose", "A"] <- .hessian[, "A", "Dose"] <- .expr23
    .hessian[, "Dose", "B"] <- .hessian[, "B", "Dose"] <- -.expr1 *
        (.expr6 * (.expr16 * (.expr28 * .expr14) + .expr33 *
            .expr20) + .expr40 * .expr21)
    .hessian[, "Dose", "BMD"] <- .hessian[, "BMD", "Dose"] <- -.expr1 *
        (.expr6 * (.expr21 + .expr45 * .expr20))
    .grad[, "A"] <- .expr19
    .hessian[, "A", "A"] <- .expr19
    .hessian[, "A", "B"] <- .hessian[, "B", "A"] <- .expr56
    .hessian[, "A", "BMD"] <- .hessian[, "BMD", "A"] <- .expr59
    .grad[, "B"] <- .expr56
    .hessian[, "B", "B"] <- -.expr1 * (.expr51 - .expr3 * .expr61/.expr63 +
        .expr6 * (.expr16 * ((.expr28 + .expr27 * .expr27/.expr10^2) *
            Dose * .expr14) - .expr33 * .expr32) - .expr76 -
            (.expr76 + (.expr40 - ((.expr37 + .expr37)/.expr38 -
                .expr37 * .expr61/.expr63)) * .expr16))
    .hessian[, "B", "BMD"] <- .hessian[, "BMD", "B"] <- .expr1 *
        (.expr40 * .expr45 + .expr6 * (.expr33 + .expr45 * .expr32))
```

```
    .grad[, "BMD"] <- .expr59
    .hessian[, "BMD", "BMD"] <- .expr1 * (.expr6 * (.expr45 +
        .expr45 * .expr15))
    attr(.value, "gradient") <- .grad
    attr(.value, "hessian") <- .hessian
    .value
}


### CexpB2wD(dose, A, PB, BMD, BMR=0.10) Same as CexpBwD, but PB is on
### original scale.
### Model is:
### ~ exp(A)*(PB + (1-PB)*exp(log((1 - BMR - PB)/(1 - PB)) *
Dose*exp(-BMD)))
### Primarily used to be called from CexpBwDS; grad[,"PB"] is not returned.
CexpB2wD <- function (dose, A, PB, BMD, BMR=0.1)  {
  .expr1 <- exp(A)
  .expr2 <- 1 - PB
  .expr4 <- 1 - BMR - PB
  .expr5 <- .expr4/.expr2
  .expr9 <- exp(-BMD)
  .expr10 <- log(.expr5) * dose * .expr9
  .expr11 <- exp(.expr10)
  .expr14 <- .expr1 * (PB + .expr2 * .expr11)
  .value <- .expr14
  .grad <- array(0, c(length(.value), 2), list(NULL, c("A", "BMD")))
  .grad[, "A"] <- .expr14
  .grad[, "BMD"] <- -.expr1 * (.expr2 * (.expr11 * .expr10))
  attr(.value, "gradient") <- .grad
  .value
}

### Same as above, but return gradient and hessian, and include PB in both
### (for computing standard errors)

CexpB2wDH <- function (Dose, A, PB, BMD, BMR)
{
    .expr1 <- exp(A)
    .expr2 <- 1 - PB
    .expr4 <- 1 - BMR - PB
    .expr5 <- .expr4/.expr2
    .expr6 <- log(.expr5)
    .expr9 <- exp(-BMD)
    .expr10 <- .expr6 * Dose * .expr9
    .expr11 <- exp(.expr10)
    .expr14 <- .expr1 * (PB + .expr2 * .expr11)
    .expr15 <- .expr6 * .expr9
    .expr16 <- .expr11 * .expr15
    .expr18 <- .expr1 * (.expr2 * .expr16)
    .expr23 <- .expr2^2
    .expr25 <- 1/.expr2 - .expr4/.expr23
    .expr26 <- .expr25/.expr5
    .expr30 <- .expr26 * Dose * .expr9
    .expr31 <- .expr11 * .expr30
    .expr38 <- .expr11 * .expr10
    .expr47 <- .expr1 * (1 - (.expr2 * .expr31 + .expr11))
    .expr50 <- -.expr1 * (.expr2 * .expr38)
    .expr51 <- 1/.expr23
    .value <- .expr14
    .grad <- array(0, c(length(.value), 4), list(NULL, c("Dose",
        "A", "PB", "BMD")))
    .hessian <- array(0, c(length(.value), 4, 4), list(NULL,
        c("Dose", "A", "PB", "BMD"), c("Dose", "A", "PB", "BMD")))
    .grad[, "Dose"] <- .expr18
    .hessian[, "Dose", "Dose"] <- .expr1 * (.expr2 * (.expr16 *
```

```
        .expr15))
    .hessian[, "Dose", "A"] <- .hessian[, "A", "Dose"] <- .expr18
    .hessian[, "Dose", "PB"] <- .hessian[, "PB", "Dose"] <- -.expr1 *
        (.expr2 * (.expr11 * (.expr26 * .expr9) + .expr31 * .expr15) +
            .expr16)
    .hessian[, "Dose", "BMD"] <- .hessian[, "BMD", "Dose"] <- -.expr1 *
        (.expr2 * (.expr16 + .expr38 * .expr15))
    .grad[, "A"] <- .expr14
    .hessian[, "A", "A"] <- .expr14
    .hessian[, "A", "PB"] <- .hessian[, "PB", "A"] <- .expr47
    .hessian[, "A", "BMD"] <- .hessian[, "BMD", "A"] <- .expr50
    .grad[, "PB"] <- .expr47
    .hessian[, "PB", "PB"] <- -.expr1 * (.expr2 * (.expr11 *
        (((.expr51 + .expr51 - .expr4 * (2 * .expr2)/.expr23^2)/.expr5 +
            .expr25 * .expr25/.expr5^2) * Dose * .expr9) - .expr31 *
        .expr30) - .expr31 - .expr31)
    .hessian[, "PB", "BMD"] <- .hessian[, "BMD", "PB"] <- .expr1 *
        (.expr38 + .expr2 * (.expr31 + .expr38 * .expr30))
    .grad[, "BMD"] <- .expr50
    .hessian[, "BMD", "BMD"] <- .expr1 * (.expr2 * (.expr38 +
        .expr38 * .expr10))
    attr(.value, "gradient") <- .grad
    attr(.value, "hessian") <- .hessian
    .value
}


### Above, but including derivative of dose, for the pk model

### CexpBwDS(dose, A, B, BMD, sex, fixed=NULL) allows fixed, a named list of
### vectors of fixed values for the model CexpBwD.  They can differ by sex.
### used, for example:
### nlme(model=chei ~ CexpBwDS(dose, A, BMD, sex,
###      fixed=list(PB=c(F=0.05, M=0.06)), data=mydata, random=A+m~1 | mrid)
### This implementation assumes PB is the only fixed parameter, and is on
### its original scale (0 <= PB < 1).

CexpBwDS <- function(dose, A, BMD, sex, fixed=NULL, BMR=0.10) {
  callList <- vector("list",5)
  names(callList) <- c("dose","A","PB","BMD","BMR")
  callList[["dose"]] <- dose
  callList[["A"]] <- A
  callList[["PB"]] <- if ("B" %in% names(fixed)) {
    B <- fixed[["B"]][as.character(sex)]
    1/(1 + exp(-B))
  } else {
    fixed[["PB"]][as.character(sex)]
  }
  callList[["BMD"]] <- BMD
  callList[["BMR"]] <- BMR
  do.call("CexpB2wD",callList)
}

### CpkexpBwD: pk combined with the exp model in terms of BMD:

CpkexpBwD <- function (Dose, A, B, BMD, S, D, BMR=0.10)
{
  idose <- CpkB(Dose, S, D)
  .value <- CexpBwD2(idose, A, B, BMD, BMR=BMR)
  .grad <- array(0, c(length(.value), 5),
                 list(NULL,
                      c("A","B","BMD","S","D")))
  .grad[,c("A","B","BMD")] <- attr(.value,"gradient")[,c("A","B","BMD")]
  .grad[,c("S","D")] <- attr(.value,"gradient")[,"Dose"] *
    attr(idose, "gradient")[,c("S", "D")]
```

```
  attr(.value, "gradient") <- .grad
  .value
}
###: same as above, but include hessian
CpkexpBwDH <- function (Dose, A, B, BMD, S, D, BMR=0.10)
{
  idose <- CpkBH(Dose, S, D)
  .value <- CexpBwD2H(idose, A, B, BMD, BMR=BMR)
  .grad <- array(0, c(length(.value), 5),
                 list(NULL,
                      c("A","B","BMD","S","D")))
  .grad[,c("A","B","BMD")] <- attr(.value,"gradient")[,c("A","B","BMD")]
  .grad[,c("S","D")] <- attr(.value,"gradient")[,"Dose"] *
    attr(idose, "gradient")[,c("S", "D")]
  .hessian <- array(0, c(length(.value),5,5),
                    c("A","B","BMD","S","D"),
                    c("A","B","BMD","S","D"))
  .hessian[,c("A","B","BMD"),c("A","B","BMD")] <-
    attr(.value,"hessian")[,c("A","B","BMD"),c("A","B","BMD")]
  .hessian[,c("S","D"),c("S","D")] <-
    attr(.value,"hessian")[,"Dose","Dose"] *
      (attr(.idose,"gradient")[,c("S","D"),c("S","D")])^2 +
      attr(.idose,"gradient")[,"Dose"] *
        attr(.idose,"hessian")[,c("S","D"),c("S","D")]
  .hessian[,"A","S"] <- .hessian[,"S","A"] <-
    attr(.value,"hessian")[,"Dose","A"] * attr(.idose,"gradient")[,"S"]
  .hessian[,"A","D"] <- .hessian[,"S","D"] <-
    attr(.value,"hessian")[,"Dose","A"] * attr(.idose,"gradient")[,"D"]
  .hessian[,"B","S"] <- .hessian[,"S","B"] <-
    attr(.value,"hessian")[,"Dose","B"] * attr(.idose,"gradient")[,"S"]
  .hessian[,"B","D"] <- .hessian[,"S","D"] <-
    attr(.value,"hessian")[,"Dose","B"] * attr(.idose,"gradient")[,"D"]
  .hessian[,"BMD","S"] <- .hessian[,"S","BMD"] <-
    attr(.value,"hessian")[,"Dose","BMD"] * attr(.idose,"gradient")[,"S"]
  .hessian[,"BMD","D"] <- .hessian[,"S","D"] <-
    attr(.value,"hessian")[,"Dose","BMD"] * attr(.idose,"gradient")[,"D"]

  attr(.value, "gradient") <- .grad
  attr(.value, "hessian") <- .hessian
  .value
}
CpkexpB2wDH <- function (Dose, A, PB, BMD, S, D, BMR=0.10)
{
  idose <- CpkBH(Dose, S, D)
  .value <- CexpB2wDH(idose, A, PB, BMD, BMR=BMR)
  .grad <- array(0, c(length(.value), 5),
                 list(NULL,
                      c("A","PB","BMD","S","D")))
  .grad[,c("A","PB","BMD")] <- attr(.value,"gradient")[,c("A","PB","BMD")]
  .grad[,c("S","D")] <- attr(.value,"gradient")[,"Dose"] *
    attr(idose, "gradient")[,c("S", "D")]
  .hessian <- array(0, c(length(.value),5,5),
                    list(NULL,c("A","PB","BMD","S","D"),
                    c("A","PB","BMD","S","D")))
  .hessian[,c("A","PB","BMD"),c("A","PB","BMD")] <-
    attr(.value,"hessian")[,c("A","PB","BMD"),c("A","PB","BMD")]
  .hessian[,"S","S"] <-
    attr(.value,"hessian")[,"Dose","Dose"] *
      attr(idose,"gradient")[,"S"] * attr(idose,"gradient")[,"S"] +
      attr(.value,"gradient")[,"Dose"] *
        attr(idose,"hessian")[,"S","S"]
  .hessian[,"D","D"] <-
    attr(.value,"hessian")[,"Dose","Dose"] *
      attr(idose,"gradient")[,"D"] * attr(idose,"gradient")[,"D"] +
      attr(.value,"gradient")[,"Dose"] *
```

```
        attr(idose,"hessian")[,"D","D"]
  .hessian[,"S","D"] <- .hessian[,"D","S"] <-
    attr(.value,"hessian")[,"Dose","Dose"] *
      attr(idose,"gradient")[,"S"] * attr(idose,"gradient")[,"D"] +
      attr(.value,"gradient")[,"Dose"] *
        attr(idose,"hessian")[,"S","D"]

  .hessian[,"A","S"] <- .hessian[,"S","A"] <-
    attr(.value,"hessian")[,"Dose","A"] * attr(idose,"gradient")[,"S"]
  .hessian[,"A","D"] <- .hessian[,"S","D"] <-
    attr(.value,"hessian")[,"Dose","A"] * attr(idose,"gradient")[,"D"]
  .hessian[,"PB","S"] <- .hessian[,"S","PB"] <-
    attr(.value,"hessian")[,"Dose","PB"] * attr(idose,"gradient")[,"S"]
  .hessian[,"PB","D"] <- .hessian[,"S","D"] <-
    attr(.value,"hessian")[,"Dose","PB"] * attr(idose,"gradient")[,"D"]
  .hessian[,"BMD","S"] <- .hessian[,"S","BMD"] <-
    attr(.value,"hessian")[,"Dose","BMD"] * attr(idose,"gradient")[,"S"]
  .hessian[,"BMD","D"] <- .hessian[,"S","D"] <-
    attr(.value,"hessian")[,"Dose","BMD"] * attr(idose,"gradient")[,"D"]

  attr(.value, "gradient") <- .grad
  attr(.value, "hessian") <- .hessian
  .value
}

### CpkexpB2wD: pk combined with the exp model in terms of BMD.  Assume PB,
### S, and D
### are fixed, and do not return their components of the gradient.

CpkexpB2wD <- function (dose, A, PB, BMD, S, D, BMR=0.10)
{
  idose <- CpkB(dose, S, D)
  .value <- CexpB2wD(idose, A, PB, BMD, BMR=BMR)
  .grad <- array(0, c(length(.value), 2),
                 list(NULL,
                      c("A","BMD")))
  .grad[,c("A","BMD")] <- attr(.value,"gradient")[,c("A","BMD")]
  attr(.value, "gradient") <- .grad
  .value
}

### CpkexpBS: pk combined with exp model, with fixed B and/or S values:
CpkexpBS <- function(dose, A, B, m, S, D, sex, fixed=NULL) {
  callList <- vector("list",6)
  names(callList) <- c("Dose","A","B","m","S","D")
  callList[["Dose"]] <- dose
  callList[["A"]] <- A
  if (length(fixed) == 0) sx <- sex
  if (length(fixed) == 1) sx <- D
  if (length(fixed) == 2) sx <- S
  if (length(fixed) == 3) sx <- m
  if ("B" %in% names(fixed)) {
    callList[["B"]] <- fixed[["B"]][as.character(sx)]
    callList[["m"]] <- B
    if ("S" %in% names(fixed)) {
      callList[["S"]] <- fixed[["S"]][as.character(sx)]
      if ("D" %in% names(fixed)) {
        callList[["D"]] <- fixed[["D"]][as.character(sx)]
      } else {
        callList[["D"]] <- m
      }
    } else {
      callList[["S"]] <- m
      if ("D" %in% names(fixed)) {
        callList[["D"]] <- fixed[["D"]][as.character(sx)]
```

```
      } else {
        callList[["D"]] <- S
      }
    }
  } else {
    callList[["B"]] <- B
    callList[["m"]] <- m
    if ("S" %in% names(fixed)) {
      callList[["S"]] <- fixed[["S"]][as.character(sx)]
      if ("D" %in% names(fixed)) {
        callList[["D"]] <- fixed[["D"]][as.character(sx)]
      } else {
        callList[["D"]] <- S
      }
    } else {
      callList[["S"]] <- S
      if ("D" %in% names(fixed)) {
        callList[["D"]] <- fixed[["D"]][as.character(sx)]
      } else {
        callList[["D"]] <- D
      }
    }
  }
  do.call("CpkexpB",callList)
}
### CpkexpBwDS: pk combined with exp model, with fixed B and/or S values:
CpkexpBwDS <- function(dose, A, B, BMD, S, D, sex, fixed=NULL) {
  callList <- vector("list",6)
  names(callList) <- c("Dose","A","B","BMD","S","D")
  callList[["Dose"]] <- dose
  callList[["A"]] <- A
  if (length(fixed) == 0) sx <- sex
  if (length(fixed) == 1) sx <- D
  if (length(fixed) == 2) sx <- S
  if (length(fixed) == 3) sx <- BMD
  if ("B" %in% names(fixed)) {
    callList[["B"]] <- fixed[["B"]][as.character(sx)]
    callList[["BMD"]] <- B
    if ("S" %in% names(fixed)) {
      callList[["S"]] <- fixed[["S"]][as.character(sx)]
      if ("D" %in% names(fixed)) {
        callList[["D"]] <- fixed[["D"]][as.character(sx)]
      } else {
        callList[["D"]] <- BMD
      }
    } else {
      callList[["S"]] <- BMD
      if ("D" %in% names(fixed)) {
        callList[["D"]] <- fixed[["D"]][as.character(sx)]
      } else {
        callList[["D"]] <- S
      }
    }
  } else {
    callList[["B"]] <- B
    callList[["BMD"]] <- BMD
    if ("S" %in% names(fixed)) {
      callList[["S"]] <- fixed[["S"]][as.character(sx)]
      if ("D" %in% names(fixed)) {
        callList[["D"]] <- fixed[["D"]][as.character(sx)]
      } else {
        callList[["D"]] <- S
      }
    } else {
      callList[["S"]] <- S
```

```
      if ("D" %in% names(fixed)) {
        callList[["D"]] <- fixed[["D"]][as.character(sx)]
      } else {
        callList[["D"]] <- D
      }
    }
  }
  do.call("CpkexpBwD",callList)
}

### CpkexpB2wDS: pk combined with exp model, with fixed PB, S, and D values:

CpkexpB2wDS <- function(dose, A, BMD, sex, fixed=NULL) {
  callList <- vector("list",7)
  names(callList) <- c("dose","A","PB", "BMD","S","D","BMR")
  callList[["dose"]] <- dose
  callList[["A"]] <- A
  callList[["PB"]] <- fixed[["PB"]][as.character(sex)]
  callList[["BMD"]] <- BMD
  callList[["S"]] <- fixed[["S"]][as.character(sex)]
  callList[["D"]] <- fixed[["D"]][as.character(sex)]
  callList[["BMR"]] <- 0.1

  do.call("CpkexpB2wD",callList)
}

### Compute log(BMD) (for a BMR * 100% decrease in the mean) for the
### exponential
### model, based on the parameter transformations used here.
### This returns the gradient of log(BMD), to use in computing standard
### errors.
###
### log(BMD) <- expression(log(-log(1 - BMR*(1 + exp(B)))) - m)
### in terms of the transformation used in the CexpB models.
### fixed is a vector of strings, listing the parameters that should not
### be in the gradient.

CexplBMD <- function (BMR, m, B, fixed=NULL)
{
    .expr1 <- exp(B)
    .expr4 <- 1 - BMR * (1 + .expr1)
    .expr5 <- log(.expr4)
    .value <- log(-.expr5) - m
    .grad <- array(0, c(length(.value), 2), list(NULL, c("m",
        "B")))
    .grad[, "m"] <- -1
    .grad[, "B"] <- -BMR * .expr1/.expr4/.expr5
    if (!is.null(fixed) > 0) {
      .grad <- .grad[,-match(fixed,colnames(.grad)),drop=FALSE]
    }
    attr(.value, "gradient") <- .grad
    .value
}

### Essentially the same function, but takes as input our model object
### (xx) and returns a list with elements lBMD and lBMD.se,
### each with components for "F" and "M".


explBMD <- function(object, BMR) {
  fit <- object$Fitm
  if (inherits(fit, "nlme")) {
    Sigma <- fit$varFix
    Coefs <- fit$coefficients$fixed
  } else {
```

```
    Sigma <- fit$varBeta
    Coefs <- fit$coefficients
  }
### Fixed gives the fixed parameters
  tmp <- names(fit$call$model[[3]])
  Fixed <- NULL
  if ("fixed" %in% tmp) {
    tmp <- names(fit$call$model[[3]][["fixed"]])
    Fixed <- c(Fixed,c("m","B")[c("m","B") %in% tmp])
  }
  m.F <- if ("m" %in% Fixed)
    fit$call$model[[3]][["fixed"]][["m"]][["F"]]
  else
    if ("m.sexF" %in% names(Coefs)) Coefs["m.sexF"] else Coefs["m"]
  m.M <- if ("m" %in% Fixed)
    fit$call$model[[3]][["fixed"]][["m"]][["M"]]
  else
    if ("m.sexM" %in% names(Coefs)) Coefs["m.sexM"] else Coefs["m"]

  B.F <- if ("B" %in% Fixed)
    fit$call$model[[3]][["fixed"]][["B"]][["F"]]
  else
    if ("B.sexF" %in% names(Coefs)) Coefs["B.sexF"] else Coefs["B"]
  B.M <- if ("B" %in% Fixed)
    fit$call$model[[3]][["fixed"]][["B"]][["M"]]
  else
    if ("B.sexM" %in% names(Coefs)) Coefs["B.sexM"] else Coefs["B"]
  ### Females
  lBMD.F <- CexplBMD(BMR=BMR,m=m.F,B=B.F,fixed=Fixed) +
log(object$Dosescale)
  ### Males
  lBMD.M <- CexplBMD(BMR=BMR,m=m.M,B=B.M,fixed=Fixed) +
log(object$Dosescale)

  ## Get the standard errors
  ## 1) Get the 'Female' and 'Male' Sigmas
  indx <- match(c("m.sexF","B.sexF"),rownames(Sigma),nomatch=0)
  Sigma.F <- Sigma[indx,indx,drop=FALSE]

  ## Strip off the '.F' from the rownames and colnames
  rownames(Sigma.F) <- colnames(Sigma.F) <-
sub("\\.sexF","",rownames(Sigma.F))

  indx <- match(c("m.sexM","B.sexM"),rownames(Sigma),nomatch=0)
  Sigma.M <- Sigma[indx,indx,drop=FALSE]

  ## Strip off the '.M' from the rownames and colnames
  rownames(Sigma.M) <- colnames(Sigma.M) <-
sub("\\.sexM","",rownames(Sigma.M))

  ## Get the 'Female' and 'Male' gradients

  grad.F <- attr(lBMD.F,"gradient")
  grad.F <- grad.F[,rownames(Sigma.F)]
  grad.M <- attr(lBMD.M,"gradient")
  grad.M <- grad.M[,rownames(Sigma.M)]
  list(lBMD = c(F=c(lBMD.F),M=c(lBMD.M)),
       lBMD.se = c(F=sqrt(t(grad.F) %*% Sigma.F %*% grad.F),
         M=sqrt(t(grad.M) %*% Sigma.M %*% grad.M)))
}

### Compute BMD (for a BMR * 100% decrease in the mean) for the
### exponential
### model, based on the parameter transformations used here.
### This returns the gradient of BMD, to use in computing standard
```

```
### errors.
###
### BMD <- expression(-log(1 - BMR*(1 + exp(B)))*exp( - m ))
### in terms of the transformation used in the CexpB models.
### fixed is a vector of strings, listing the parameters that should not
### be in the gradient.

CexpBMD <- function (BMR, m, B, fixed)
{
    .expr1 <- exp(B)
    .expr4 <- 1 - BMR * (1 + .expr1)
    .expr5 <- log(.expr4)
    .expr8 <- exp(-m)
    .value <- -.expr5 * .expr8
    .grad <- array(0, c(length(.value), 2), list(NULL, c("m",
        "B")))
    .grad[, "m"] <- .expr5 * .expr8
    .grad[, "B"] <- BMR * .expr1/.expr4 * .expr8
    if (!is.null(fixed) > 0) {
      .grad <- .grad[,-match(fixed,colnames(.grad)),drop=FALSE]
    }
    attr(.value, "gradient") <- .grad
    .value
}

### Compute lBMD and its standard error for pkexpBS
### This calculates the value for both sexes at the same time

pkexpSlBMD.se <- function(object,BMR) {
  fitpk <- object$Fitpk
  if (inherits(fitpk, "nlme")) {
    Sigma <- fitpk$varFix
    Coefs <- fitpk$coefficients$fixed
  } else {
    Sigma <- fitpk$varBeta
    Coefs <- fitpk$coefficients
  }

### Fixed1 gives the fixed parameters for the CexpBMD part (i.e., 'B')
### Fixed2 gives the fixed parameters for the CpkBMD part (i.e., 'S')

  tmp <- names(fitpk$call$model[[3]])
  Fixed1 <- Fixed2 <- NULL
  if ("fixed" %in% tmp) {
    tmp <- names(fitpk$call$model[[3]][["fixed"]])
    Fixed1 <- c(Fixed1,c("m","B")[c("m","B") %in% tmp])
    Fixed2 <- c(Fixed2,c("S","D")[c("S","D") %in% tmp])
  }

  S.F <- if ("S" %in% Fixed2)
    fitpk$call$model[[3]][["fixed"]][["S"]][["F"]]
  else
    if ("S.sexF" %in% names(Coefs)) Coefs["S.sexF"] else Coefs["S"]
  S.M <- if ("S" %in% Fixed2)
    fitpk$call$model[[3]][["fixed"]][["S"]][["M"]]
  else
    if ("S.sexM" %in% names(Coefs)) Coefs["S.sexM"] else Coefs["S"]

  D.F <- if ("D" %in% Fixed2)
    fitpk$call$model[[3]][["fixed"]][["D"]][["F"]]
  else
    if ("D.sexF" %in% names(Coefs)) Coefs["D.sexF"] else Coefs["D"]
  D.M <- if ("D" %in% Fixed2)
    fitpk$call$model[[3]][["fixed"]][["D"]][["M"]]
  else
```

```
    if ("D.sexM" %in% names(Coefs)) Coefs["D.sexM"] else Coefs["D"]

  m.F <- if ("m" %in% Fixed1)
    fitpk$call$model[[3]][["fixed"]][["m"]][["F"]]
  else
    if ("m.sexF" %in% names(Coefs)) Coefs["m.sexF"] else Coefs["m"]
  m.M <- if ("m" %in% Fixed1)
    fitpk$call$model[[3]][["fixed"]][["m"]][["M"]]
  else
    if ("m.sexM" %in% names(Coefs)) Coefs["m.sexM"] else Coefs["m"]

  B.F <- if ("B" %in% Fixed1)
    fitpk$call$model[[3]][["fixed"]][["B"]][["F"]]
  else
    if ("B.sexF" %in% names(Coefs)) Coefs["B.sexF"] else Coefs["B"]
  B.M <- if ("B" %in% Fixed1)
    fitpk$call$model[[3]][["fixed"]][["B"]][["M"]]
  else
    if ("B.sexM" %in% names(Coefs)) Coefs["B.sexM"] else Coefs["B"]

  ### Females
  idose.F <- CexpBMD(BMR,m.F,B.F,fixed=Fixed1)
  lBMD.F <- CpkBi(as.vector(idose.F),S.F,D.F,fixed=Fixed2) +
    log(object$Dosescale)

  ### Males
  idose.M <- CexpBMD(BMR,m.M,B.M,fixed=Fixed1)
  lBMD.M <- CpkBi(as.vector(idose.M),S.M,D.M,fixed=Fixed2) +
    log(object$Dosescale)

  ## Get the standard errors
  ## 1) Get the 'Female' and 'Male' Sigmas
  indx <- match(c("m.sexF","B.sexF","S","D"),rownames(Sigma),nomatch=0)
  Sigma.F <- Sigma[indx,indx,drop=FALSE]

  ## Strip off the '.F' from the rownames and colnames
  rownames(Sigma.F) <- colnames(Sigma.F) <-
sub("\\.sexF","",rownames(Sigma.F))

  indx <- match(c("m.sexM","B.sexM","S","D"),rownames(Sigma),nomatch=0)
  Sigma.M <- Sigma[indx,indx,drop=FALSE]

  ## Strip off the '.M' from the rownames and colnames
  rownames(Sigma.M) <- colnames(Sigma.M) <-
sub("\\.sexM","",rownames(Sigma.M))

  ## 2) Get the 'Female' and 'Male' gradients

  grad.F <- c(attr(idose.F,"gradient")*attr(lBMD.F,"gradient")[,"idose"],
              attr(lBMD.F,"gradient")[,-match("idose",
                                              colnames(attr(lBMD.F,
                                                            "gradient"))),
                                      drop=FALSE])
  nm <- c(colnames(attr(idose.F,"gradient")),
          colnames(attr(lBMD.F,"gradient")))
  names(grad.F) <- nm[-match("idose",nm)]

  ## Make sure the elements are in the right order for Sigma.F

  grad.F <- grad.F[rownames(Sigma.F)]

  grad.M <- c(attr(idose.M,"gradient")*attr(lBMD.M,"gradient")[,"idose"],
              attr(lBMD.M,"gradient")[,-match("idose",
                                              colnames(attr(lBMD.M,
                                                            "gradient"))),
```

```
                                        drop=FALSE])
  nm <- c(colnames(attr(idose.M,"gradient")),
          colnames(attr(lBMD.M,"gradient")))
  names(grad.M) <- nm[-match("idose",nm)]
  ##Make sure the elements are in the right order for Sigma.M

  grad.M <- grad.M[rownames(Sigma.M)]

  list(lBMD=c(F=as.vector(lBMD.F),M=as.vector(lBMD.M)),
       lBMD.se=c(F = sqrt(t(grad.F) %*% Sigma.F %*% grad.F),
         M = sqrt(t(grad.M) %*% Sigma.M %*% grad.M)))
}




### Compute lBMD and its standard error for pkexpBwDS
### This calculates the value for both sexes at the same time

pkexpwDSlBMD.se <- function(object,BMR) {
  fitpk <- object$Fitpk
  if (inherits(fitpk, "nlme")) {
    Sigma <- fitpk$varFix
    Coefs <- fitpk$coefficients$fixed
  } else {
    Sigma <- fitpk$varBeta
    Coefs <- fitpk$coefficients
  }

### Fixed1 gives the fixed parameters for the CexpBMD part (i.e., 'B')
### Fixed2 gives the fixed parameters for the CpkBMD part (i.e., 'S')

  tmp <- names(fitpk$call$model[[3]])
  Fixed1 <- Fixed2 <- NULL
  if ("fixed" %in% tmp) {
    tmp <- names(fitpk$call$model[[3]][["fixed"]])
    Fixed1 <- c(Fixed1,c("BMD","B")[c("BMD","B") %in% tmp])
    Fixed2 <- c(Fixed2,c("S","D")[c("S","D") %in% tmp])
  }
## Extract the parameter estimates
  S.F <- if ("S" %in% Fixed2)
    fitpk$call$model[[3]][["fixed"]][["S"]][["F"]]
  else
    if ("S.sexF" %in% names(Coefs)) Coefs["S.sexF"] else Coefs["S"]
  S.M <- if ("S" %in% Fixed2)
    fitpk$call$model[[3]][["fixed"]][["S"]][["M"]]
  else
    if ("S.sexM" %in% names(Coefs)) Coefs["S.sexM"] else Coefs["S"]

  D.F <- if ("D" %in% Fixed2)
    fitpk$call$model[[3]][["fixed"]][["D"]][["F"]]
  else
    if ("D.sexF" %in% names(Coefs)) Coefs["D.sexF"] else Coefs["D"]
  D.M <- if ("D" %in% Fixed2)
    fitpk$call$model[[3]][["fixed"]][["D"]][["M"]]
  else
    if ("D.sexM" %in% names(Coefs)) Coefs["D.sexM"] else Coefs["D"]

  BMD.F <- if ("BMD" %in% Fixed1)
    fitpk$call$model[[3]][["fixed"]][["BMD"]][["F"]]
  else
    if ("BMD.sexF" %in% names(Coefs)) Coefs["BMD.sexF"] else Coefs["BMD"]
  BMD.M <- if ("BMD" %in% Fixed1)
    fitpk$call$model[[3]][["fixed"]][["BMD"]][["M"]]
```

```
  else
    if ("BMD.sexM" %in% names(Coefs)) Coefs["BMD.sexM"] else Coefs["BMD"]


### Calculate the log BMDs
  ## Females
  idose.F <- exp(BMD.F)
  lBMD.F <- CpkBi(as.vector(idose.F),S.F,D.F,fixed=Fixed2) +
    log(object$Dosescale)

  ### Males
  idose.M <- exp(BMD.M)
  lBMD.M <- CpkBi(as.vector(idose.M),S.M,D.M,fixed=Fixed2) +
    log(object$Dosescale)

### Calculate the standard errors
  ## 1) Get the 'Female' and 'Male' Sigmas
  indx <- match(c("BMD.sexF","S","D"),rownames(Sigma),nomatch=0)
  Sigma.F <- Sigma[indx,indx,drop=FALSE]

  ## Strip off the '.F' from the rownames and colnames
  rownames(Sigma.F) <- colnames(Sigma.F) <-
sub("\\.sexF","",rownames(Sigma.F))

  indx <- match(c("BMD.sexM","S","D"),rownames(Sigma),nomatch=0)
  Sigma.M <- Sigma[indx,indx,drop=FALSE]

  ## Strip off the '.M' from the rownames and colnames
  rownames(Sigma.M) <- colnames(Sigma.M) <-
sub("\\.sexM","",rownames(Sigma.M))

  ## 2) Get the 'Female' and 'Male' gradients

  grad.F <- c(idose.F*attr(lBMD.F,"gradient")[,"idose"],
              attr(lBMD.F,"gradient")[,-match("idose",
                                              colnames(attr(lBMD.F,
                                                            "gradient"))),
                                      drop=FALSE])
  nm <- c("BMD",
          colnames(attr(lBMD.F,"gradient")))
  names(grad.F) <- nm[-match("idose",nm)]

  ## Make sure the elements are in the right order for Sigma.F

  grad.F <- grad.F[rownames(Sigma.F)]

  grad.M <- c(idose.M*attr(lBMD.M,"gradient")[,"idose"],
              attr(lBMD.M,"gradient")[,-match("idose",
                                              colnames(attr(lBMD.M,
                                                            "gradient"))),
                                      drop=FALSE])
  nm <- c("BMD",
          colnames(attr(lBMD.M,"gradient")))
  names(grad.M) <- nm[-match("idose",nm)]
  ##Make sure the elements are in the right order for Sigma.M

  grad.M <- grad.M[rownames(Sigma.M)]

  list(lBMD=c(F=as.vector(lBMD.F),M=as.vector(lBMD.M)),
       lBMD.se=c(F = sqrt(t(grad.F) %*% Sigma.F %*% grad.F),
         M = sqrt(t(grad.M) %*% Sigma.M %*% grad.M)))
}

### CpkB: a low-dose modification to simulate the effect of first-pass
### metabolism on the relationship between administered dose and
```

```
### internal dose.
### Model expression is:
###  ~0.5*(dose - exp(S) - exp(D) + sqrt((dose - exp(S) - exp(D))^2 +
4*dose*exp(S)))
### This maps administered dose (dose) to internal dose (CpkB)
"CpkB" <-
function (dose, S, D)
{
    .expr1 <- exp(S)
    .expr3 <- exp(D)
    .expr4 <- dose - .expr1 - .expr3
    .expr7 <- 4 * dose * .expr1
    .expr8 <- .expr4^2 + .expr7
    .expr15 <- .expr8^-0.5
    .value <- 0.5 * (.expr4 + sqrt(.expr8))
    .grad <- array(0, c(length(.value), 2), list(NULL, c("S",
        "D")))
    .grad[, "S"] <- 0.5 * (0.5 * ((.expr7 - 2 * (.expr1 * .expr4)) *
        .expr15) - .expr1)
    .grad[, "D"] <- -0.5 * (0.5 * (2 * (.expr3 * .expr4) * .expr15) +
        .expr3)
    attr(.value, "gradient") <- .grad
    .value
}

### CpkBH: just like CpkB, but also returns the hessian
CpkBH <- function (dose, S, D)
{
    .expr1 <- exp(S)
    .expr3 <- exp(D)
    .expr4 <- dose - .expr1 - .expr3
    .expr7 <- 4 * dose * .expr1
    .expr8 <- .expr4^2 + .expr7
    .expr12 <- .expr1 * .expr4
    .expr14 <- .expr7 - 2 * .expr12
    .expr15 <- .expr8^-0.5
    .expr25 <- .expr8^-1.5
    .expr36 <- .expr3 * .expr4
    .expr37 <- 2 * .expr36
    .expr39 <- -0.5 * (.expr37 * .expr25)
    .value <- 0.5 * (.expr4 + sqrt(.expr8))
    .grad <- array(0, c(length(.value), 2), list(NULL, c("S",
        "D")))
    .hessian <- array(0, c(length(.value), 2, 2), list(NULL,
        c("S", "D"), c("S", "D")))
    .grad[, "S"] <- 0.5 * (0.5 * (.expr14 * .expr15) - .expr1)
    .hessian[, "S", "S"] <- 0.5 * (0.5 * ((.expr7 - 2 * (.expr12 -
        .expr1 * .expr1)) * .expr15 + .expr14 * (-0.5 * (.expr14 *
        .expr25))) - .expr1)
    .hessian[, "S", "D"] <- .hessian[, "D", "S"] <- 0.5 * (0.5 *
        (2 * (.expr1 * .expr3) * .expr15 - .expr14 * .expr39))
    .grad[, "D"] <- -0.5 * (0.5 * (.expr37 * .expr15) + .expr3)
    .hessian[, "D", "D"] <- -0.5 * (0.5 * (2 * (.expr36 - .expr3 *
        .expr3) * .expr15 - .expr37 * .expr39) + .expr3)
    attr(.value, "gradient") <- .grad
    attr(.value, "hessian") <- .hessian
    .value
}

### and this is the inverse function: maps internal dose to the
corresponding
### administered dose.
### CpkBi <- expression((idose^2 + idose*(exp(S) + exp(D)))/(idose +
exp(S)))
###
```

```
### Since I want to give log(BMD), use:
### CpkBi <- expression(log(idose^2 + idose*(exp(S) + exp(D))) - log(idose +
exp(S)))
### fixed: a vector of strings of parameters that will not be included in
### the gradient
CpkBi <- function (idose, S, D, fixed=NULL)
{
    .expr2 <- exp(S)
    .expr3 <- exp(D)
    .expr4 <- .expr2 + .expr3
    .expr6 <- idose^2 + idose * .expr4
    .expr8 <- idose + .expr2
    .value <- log(.expr6) - log(.expr8)
    .grad <- array(0, c(length(.value), 3), list(NULL, c("idose",
        "S", "D")))
    .grad[, "idose"] <- (2 * idose + .expr4)/.expr6 - 1/.expr8
    .grad[, "S"] <- idose * .expr2/.expr6 - .expr2/.expr8
    .grad[, "D"] <- idose * .expr3/.expr6
    if (!is.null(fixed))
      .grad <- .grad[,-match(fixed,colnames(.grad)),drop=FALSE]
    attr(.value, "gradient") <- .grad
    .value
}
"mypkPredict" <- function (object, newdata, level) {
  fparms <- object$coefficients$fixed
  rparms <- object$coefficients$random
  B.est <- !("fixed" %in% names(object$call$model[[3]])) ||
            !("B" %in% names(object$call$model[[3]][["fixed"]]))
  D.est <- !("fixed" %in% names(object$call$model[[3]])) ||
            !("D" %in% names(object$call$model[[3]][["fixed"]]))
  S.est <- !("fixed" %in% names(object$call$model[[3]])) ||
            !("S" %in% names(object$call$model[[3]][["fixed"]]))
  B.sex <- if (B.est) "B.sexF" %in% names(fparms) else NA
  indx <- paste("A.s.U", as.character(newdata$s.U), sep = "")
  A <- fparms[indx]
  B <- if (B.est) {
    indx <- if (B.sex) {
      paste("B.", "sex", as.character(newdata$sex), sep = "")
    } else {
      rep("B",nrow(newdata))
    }
    fparms[indx]
  } else {
    eval(object$call$model[[3]][["fixed"]])$B[as.character(newdata$sex)]
  }
  indx <- paste("BMD.", "sex", as.character(newdata$sex),
                sep = "")
  BMD <- fparms[indx]
### S is always fixed in my application
  S <- if (S.est) {
    rep(fparms["S"],nrow(newdata))
  } else {

eval(object$call$model[[3]][["fixed"]][["S"]])[as.character(newdata$sex)]
  }

### for simplicity, I'm assuming what we actually did, that is, model D ~ 1
### but fixed=list(D=c(F=,M=))
  D <- if (D.est) {
    rep(fparms["D"],nrow(newdata))
  } else {

eval(object$call$model[[3]][["fixed"]][["D"]])[as.character(newdata$sex)]
  }
  if (level >= 1) {
```

```
    grpname <- names(rparms)[1]
    if ("A.(Intercept)" %in% colnames(rparms[[1]]))
      A <- A + rparms[[1]][as.character(newdata[,grpname]),
                           "A.(Intercept)"]
    if (B.est) {
      B <- B + if (B.sex) {
        rparms[[1]][as.character(newdata[,grpname]), "B.(Intercept)"]
      } else {
        rparms[[1]][as.character(newdata[,grpname]), "B"]
      }
    }

    BMD <- BMD + rparms[[1]][as.character(newdata[,grpname]),
"BMD.(Intercept)"]
  }
  if (level == 2) {
    indx <- paste(as.character(newdata$mrid), as.character(newdata$set),
                  sep = "/")
    if ("A.(Intercept)" %in% colnames(rparms[[2]]))
      A <- A + rparms[[2]][indx, "A.(Intercept)"]
    if (B.est)
      B <- B + if (B.sex) {
        rparms[[2]][indx, "B.(Intercept)"]
      } else {
        rparms[[2]][indx, "B"]
      }
    BMD <- BMD + rparms[[2]][indx, "BMD.(Intercept)"]
  }
  CpkexpBwD(newdata$Dose.scaled, A, B, BMD, S, D)
}
### The following function is not general, but applies only to the model
### as used in the analysis of the OP data

mypkPredict2 <- function (object, newdata, level) {
  fparms <- object$coefficients$fixed
  rparms <- object$coefficients$random

  ## Fixed Effects
  indx <- paste("A.s.M.t", as.character(newdata$s.M.t), sep = "")
  A <- fparms[indx]

  PB <-
eval(object$call$model[[3]][["fixed"]])$PB[as.character(newdata$sex)]

  indx <- paste("BMD.", "sex", as.character(newdata$sex),
                sep = "")
  BMD <- fparms[indx]
### S is always fixed in my application
  S <-
eval(object$call$model[[3]][["fixed"]][["S"]])[as.character(newdata$sex)]

### for simplicity, I'm assuming what we actually did, that is, model D ~ 1
### but fixed=list(D=c(F=,M=))
  D <-
eval(object$call$model[[3]][["fixed"]][["D"]])[as.character(newdata$sex)]
  if (level >= 1) {
    grpname <- names(rparms)[1]
    BMD <- BMD + rparms[[1]][as.character(newdata[,grpname]),
"BMD.(Intercept)"]
  }
  if (level == 2) {
    indx <- paste(as.character(newdata$mrid), as.character(newdata$set),
                  sep = "/")
    BMD <- BMD + rparms[[2]][indx, "BMD.(Intercept)"]
  }
```

```
  CpkexpB2wD(newdata$Dose.scaled, A, PB, BMD, S, D)
}
myPredict <- function(object, newdata,level) {
  fparms <- object$coefficients$fixed
  rparms <- object$coefficients$random
  ## Did we estimate Bs?
  B.est <- !("fixed" %in% names(object$call$model[[3]]))
  ## Did the Bs differ between sexes"
  B.sex <- if (B.est) "B.sexM" %in% names(fparms) else NA
  B.REname <- if (B.est) {
    if (B.sex) "B.(Intercept)" else "B"
  } else NA
  ## is  this 'm' or 'BMD'?
  is.m <- "m.sexF" %in% names(fparms)

  ## Take care of the fixed effects
  indx <- paste("A.","s.U",as.character(newdata$s.U),sep="")
  A <- fparms[indx]
  if (B.est) {
    B <- if (B.sex) {
      indx <- paste("B.","sex",as.character(newdata$sex),sep="")
      fparms[indx]
    } else {
      rep(fparms["B"],nrow(newdata))
    }
  } else {
    B <- object$call$model[[3]]["fixed"][[1]]$B[as.character(newdata$sex)]
  }
  if (is.m) {
    indx <- paste("m.","sex",as.character(newdata$sex),sep="")
    thirdcol <- "m.(Intercept)"
  } else {
    indx <- paste("BMD.sex",as.character(newdata$sex),sep="")
    thirdcol <- "BMD.(Intercept)"
  }
  m <- fparms[indx]
  if (level >= 1) {
    grpname <- names(rparms)[1]
    if ("A.(Intercept)" %in% colnames(rparms[[1]]))
      A <- A + rparms[[1]][as.character(newdata[,grpname]),"A.(Intercept)"]
    if (B.est)
      B <- B + rparms[[1]][as.character(newdata[,grpname]),B.REname]
    m <- m + rparms[[1]][as.character(newdata[,grpname]),thirdcol]
  }
  if (level == 2) {
    indx <- paste(as.character(newdata$mrid),
                  as.character(newdata$set),sep="/")
    if ("A.(Intercept)" %in% colnames(rparms[[2]]))
      A <- A + rparms[[2]][indx,"A.(Intercept)"]
    if (B.est)
      B <- B + rparms[[2]][indx,B.REname]
    m <- m + rparms[[2]][indx,thirdcol]
  }
  if (is.m) {
    CexpB(newdata$Dose.scaled,A,B,m)
  } else {
    CexpBwD(newdata$Dose.scaled,A,B,m)
  }

}
myPredict2 <- function(object, newdata,level) {
  fparms <- object$coefficients$fixed
  rparms <- object$coefficients$random

  ## Take care of the fixed effects
```

```
  indx <- paste("A.s.M.t",as.character(newdata$s.M.t),sep="")
  A <- fparms[indx]

  B <- object$call$model[[3]]["fixed"][[1]][[1]][as.character(newdata$sex)]

  indx <- paste("BMD.sex",as.character(newdata$sex),sep="")
  thirdcol <- "BMD.(Intercept)"
  m <- fparms[indx]
  if (level >= 1) {
    grpname <- names(rparms)[1]
    if ("A.(Intercept)" %in% colnames(rparms[[1]]))
      A <- A + rparms[[1]][as.character(newdata[,grpname]),"A.(Intercept)"]
    m <- m + rparms[[1]][as.character(newdata[,grpname]),thirdcol]
  }
  if (level == 2) {
    indx <-
paste(as.character(newdata$mrid),as.character(newdata$set),sep="/")
    if ("A.(Intercept)" %in% colnames(rparms[[2]]))
      A <- A + rparms[[2]][indx,"A.(Intercept)"]
    m <- m + rparms[[2]][indx,thirdcol]
  }
  if ("B" %in% names(object$call$model[[3]]["fixed"][[1]]))
    CexpBwD(newdata$Dose.scaled,A,B,m)
  else
    CexpB2wD(newdata$Dose.scaled, A, B, m)
}
Expr <- expression(exp(A)*(1/(1 + exp(-B)) + (exp(-B)/(1 +
exp(-B)))*exp(log((1 - BMR - B)/(1-B))*Dose/BMD)))


### Function to produce a phony dataset for input into gls and gnls
### Dose, N, M, and SD can be vectors of the same length
### DoseName and RespName are strings that give the names for the
### corresponding
### variables

PhonyDF <- function(Dose,N,M,SD,DoseName,RespName,chem=NULL,ChemName=NULL){
  tmp <- data.frame(rep(Dose,N),
                    qlnorm01(N,M,SD))
  names(tmp) <- c(DoseName,RespName)
  if (!is.null(ChemName)) tmp[,ChemName] <- factor(rep(chem,N))
  tmp
}
### Possible values are "Best","Biggest","Both","None"
options(BMDSplot="Best")

assign("%inint%",function(x,interval) (min(interval) <= x && max(interval)
>= x))
plot.BMDS <- function(X, which=getOption("BMDSplot"),
                      LogX=c("auto","log","linear"),
                      ...) {
  dots <- list(...)
  reduceddots <- dots
  if ((indx <- match("ylim",names(reduceddots),nomatch=0)) > 0)
    reduceddots <- reduceddots[-indx]
  if ((indx <- match("xlim",names(reduceddots),nomatch=0)) > 0)
    reduceddots <- reduceddots[-indx]
  if ((indx <- match("Title",names(reduceddots),nomatch=0)) > 0)
    reduceddots <- reduceddots[-indx]
  switch(which,
         Best={
           Agg <- !is.null(X$Data[,X$varNames["SS"]])
           if (!Agg) {
             ## if there is replication, make an aggregated dataset
             ## and set MyAgg to be true
```

```
              tmp <- rle(sort(X$Data[,X$varNames["Dose"]]))$lengths
              MyAgg <- sum(tmp > 3) >= length(tmp) - 2
              if (MyAgg) {
                indx <- order(X$Data[,X$varNames["Dose"]])
                dose <- unique(X$Data[indx,X$varNames["Dose"]])
                resp <- tapply(X$Data[indx,X$varNames["Resp"]],
                               factor(X$Data[indx,X$varNames["Dose"]]),
                               mean)
                sd <- tapply(X$Data[indx,X$varNames["Resp"]],
                               factor(X$Data[indx,X$varNames["Dose"]]),
                               function(x) sqrt(var(x)))
                MyData <- data.frame(dose,resp,sd,tmp)
                names(MyData) <- X$varNames[c("Dose","Resp","SD","SS")]
              } else MyData <- X$Data
            } else MyData <- X$Data
            LogX <- match.arg(LogX)
            if (LogX == "auto") {
              if (Agg || MyAgg) {
                dose <- sort(MyData[,X$varNames["Dose"]])
                nd <- length(dose)
                LogX <- if ((dose[nd] - dose[1])/(dose[2] - dose[1]) > 20)
                  "log" else "linear"
              } else {
                ## This is a copout for epi data, but works for now
                LogX <- "linear"
              }
            }
            drange <- range(c(MyData[,X$varNames["Dose"]],X$BMD))
            doses <- seq(min(drange),max(drange),length=101)
            if (LogX == "log" && 0 %inint% doses) doses <- doses[doses > 0]
            if (!is.null(X$Fit)) {
              newdata <- data.frame(Dose=doses/X$DoseScale)
              names(newdata) <- X$varNames["Dose"]
              predcrv <-predict(X$Fit,newdata=newdata)*X$RespScale
              if (!is.null(X$RR)) {
                newdata <- data.frame(c(0,X$BMD/X$DoseScale))
                names(newdata) <- X$varNames["Dose"]
                critresp <- predict(X$Fit,newdata=newdata)*X$RespScale
                BMR <- critresp[1]*(1 - X$RR)
              }
              else {
                critresp <- BMR <- NULL
                }
              } else predcrv <- critresp <- BMR <- NULL
              if (Agg || MyAgg) {
                mn <- MyData[,X$varNames["Resp"]]
                ss <- MyData[,X$varNames["SS"]]
                sd <- MyData[,X$varNames["SD"]]
                mnlcl <- mn - qt(0.975,ss - 1)*sd/sqrt(ss)
                mnucl <- mn + qt(0.975,ss - 1)*sd/sqrt(ss)
              } else {
                mnlcl <- mnucl <- MyData[,X$varNames["Resp"]]
              }
              ylim <- if (is.null(dots$ylim))
range(c(predcrv,mnlcl,mnucl,BMR))
              else dots$ylim
              xlim <- if (is.null(dots$xlim)) drange else dots$xlim
              Title <- if (is.null(dots$Title)) X$RunName else dots$Title
              if (Agg || MyAgg)
                do.call("plotCI",c(list(MyData[,X$varNames["Dose"]],
                                        MyData[,X$varNames["Resp"]],
                                        aui=mnucl,ali=mnlcl,err="y",
                                        ylim=ylim,xlim=xlim,
                                        xlab=X$varNames["Dose"],
                                        ylab=X$varNames["Resp"]),
```

```
                                    reduceddots))
              else
                do.call("plot",c(list(MyData[,X$varNames["Dose"]],
                                      MyData[,X$varNames["Resp"]],
                                      ylim=ylim,xlim=xlim,
                                      xlab=X$varNames["Dose"],
                                      ylab=X$varNames["Resp"]),
                                 reduceddots))
              if (MyAgg) points(X$Data[,X$varNames["Dose"]],
                                X$Data[,X$varNames["Resp"]])
              if (!is.null(X$Fit)) lines(doses,predcrv,col="green")
###             } else {
###             }
            if (!is.null(X$RR) && !is.null(X$Fit)) {
              segments(par("usr")[1],BMR,X$BMD,BMR,col="red")
              segments(X$BMD,par("usr")[3],X$BMD,BMR,col="red")
              segments(X$BMDL,par("usr")[3],X$BMDL,BMR,col="red")
            }
            title(main=Title,sub=X$ModelName)
          },
          Biggest={
            if (is.null(X$FitAllDoses)) plot(X,which="Best",...)
            else plot(X$FitAllDoses,which="Best",...)
          },
          Both={
            plot(X,which="Best",...)
            if (!is.null(X$FitAllDoses)) plot(X$FitAllDoses,which="Best",...)
          },
          None={
            invisible(NULL)
          })
}
### Function to generate N approximately lognormal quantiles with exactly
### mean M and sd SD.  Does reasonable things if N,M,and SD are vectors
### that have the same length, and exits otherwise.  If N[i] == 1,
### then just returns the value of M[i].  If SD[i] == 0, returns
### rep(M[i],N[i]).

qlnorm01 <- function(N,M,SD) {
  if((length(N) != length(M) || (length(N) != length(SD))))
    stop("N, M, and SD must have the same length")
  out <- NULL
  for (i in 1:length(N)) {
    if (N[i] > 1) {
      if (SD[i] > 0) {
        sdlog <- sqrt(log(SD[i]^2/M[i]^2 + 1))
        mnlog <- log(M[i]) - sdlog^2/2
        y <- qlnorm(ppoints(N[i]),mnlog,sdlog)
        sd <- sqrt(var(y))
        mn <- mean(y)
        out <- c(out,SD[i]*(y - mn)/sd + M[i])
      } else {
        out <- c(out,rep(M[i],N[i]))
      }
    } else out <- c(out, M)
  }
  out
}
```

b. Part 2: Modifications to the package nlme.

Two small modifications to the package nlme (version 3.1-24 was used here) were required to facilitate convergence of the models.

```
R/reStruct.R
300c300
<       PACKAGE = "nlme")$loglik
---
>       PACKAGE = "nlme",NAOK=TRUE)$loglik

src/nlme.c
292c292
<             -1. /*dlt*/, pow(epsm, 1.0/3.0) /*gradtl*/, 0. /*stepmx*/,
---
>             -1. /*dlt*/, pow(epsm, 1.0/3.0) /*gradtl*/, 1.0 /*stepmx*/,
```

c. Part 3: Estimating parameters.

Parameters for both the basic and expanded models were estimated by maximizing a profile likelihood. Since this required repeated optimizations, each of which could take a significant amount of time, the optimizations were carried out on a cluster of computers. A single "master" script passed out tasks to be completed to "slave" scripts running on the different nodes of the cluster. The R package *rpvm*, which is an interface to the software package *pvm*, was used to facilitate communication between the master program and the slaves. After the *virtual machine* was initiated using the pvm console program, the simulations were started using the command line:

```
R --slave < master.R > master.Rout 2>&1 &
```

d. Part 3a: Basic model.

Before running 'master.R', another script, 'getStartVals.R' was used to set up directory structures and some basic data structures that were then used by the simulation programs. After running this sequence of scripts once, the grid is refined, 'master.R' and 'slave.R' are updated, and the whole thing run again.

```
### GetStartVals.R
### Construct skeletons of the Fit files that includes everything we can
### determine in advance: data set, pseudodata, etc, including skeletons
### of the LL and Fits vectors for holding the results.  Reorder the Bgrid
### data frame so that we start at the lower left corner (Pf = Pm = 0.001)
### and end up in the upper right corner.  When we do the fits, the start
### value for the fit will be the previous set of estimates
### In particular, set up a structure that holds the initial estimates
###
### From the pre-SAP-review data, extract the best parameter estimates from
### the basic model, and build a list.
###
### This is a list instead of a dataframe because the number of parameters
### will vary, because of the number of background parameters, and whether
### we have one or two estimates of B.  The entry for each chemical is a
### list of two elements, named "A", and "BMD".  Each element is
### a vector of parameter estimates, named appropriately.

### Make sure parameters are expressed on the original data
### scale, not the values based on the rescaled data.  New datasets may
### change the scale!  New datasets may also add new background values.

### The following function takes as its argument one of the Basic Model
### objects and returns a list with the proper format.
###
### Finally, we execute this code only once for any chemical.  Another
script
### will be used in case we need to updata data for some chemicals and rerun

require(nlme)
require(RBMDS)

getparms <- function(x,dta) {
  ## Extract the parameters
  parms <- if(!is.null(x)) {
    if (inherits(x$FitBMD,"nlme")) {
      x$FitBMD$coefficients$fixed
    } else {
      x$FitBMD$coefficients
    } } else {
      NULL
    }

  ## Break them into A, and BMD
  out <- vector("list",2)
  names(out) <- c("A","BMD")
  ## Get the values of s.M.t
  out[["A"]] <- log(unlist(unclass(by(dta,dta$s.M.t,function(x) {
    mean(x$chei[x$dose == min(x$dose)],na.rm=TRUE)
  }))))
  nm <- names(out[["A"]])
  out[["A"]] <- as.vector(out[["A"]])
  names(out[["A"]]) <- nm


  ## Get BMD from x if possible, else figure it is about 1/4 way between
  ## the maximum and minimum dose
  if (!is.null(parms)) {
    tmp <- parms[grep("^BMD",names(parms))]
    ## Rescale using x$Dosescale
    out[["BMD"]] <- tmp + log(x$Dosescale)
  } else {
    out[["BMD"]] <- log(rep(max(dta$dose)/4,2))
    names(out[["BMD"]]) <- c("BMD.sexF","BMD.sexM")
  }
```

```
  ## There is no point is getting "B", since that will be fixed.
  out
}


attach("../Data/opdata.rda")

Nchems <- length(Chemicals)
BasicStarts <- vector("list",Nchems)
names(BasicStarts) <- Chemicals
Nsteps <- 16
Bgrid <- expand.grid(B.F=seq(0.001,0.8,length=Nsteps),
                     B.M=seq(0.001, 0.8, length=Nsteps))
Bgrid$LL <- numeric(nrow(Bgrid))
Bgrid$LL[] <- NA

for (chem in Chemicals) {
  if (file.exists(file.path("Skeletons",paste(chem,"rda",sep=".")))) next
  f <- file.path("../../pre-Feb-02/BasicModelFits/Basic-Model",
                 paste(chem,"rda",sep="."))
  if (file.exists(f)) load(f) else xx <- NULL
  xxsave <- xx
  seldata <- BRAINdata[sel <- (BRAINdata$chemical == chem),]
  seldata$s.M.t <- factor(seldata$s.M.t)
  seldata$sex <- factor(seldata$sex)
  seldata$block <- factor(seldata$block)

### loop through seldata (by block), using PhonyDF to expand it to
### synthetic individual data.


  tmp <- by(seldata,list(seldata$block),function(x) {
    PhonyDF(x$dose,x$n,x$chei,x$sd,"Dose","AChE",
            chem=rep(as.character(x$chunit[1]),nrow(x)),
            ChemName="Unit")
  })

  for (i in seq(along=tmp)) {
    flds <- unlist(strsplit(names(tmp[i]),"-"))
    N <- nrow(tmp[[i]])
    tmp[[i]]$mrid <- rep(flds[2],N)
    tmp[[i]]$sex <- rep(flds[4],N)
    tmp[[i]]$set <- rep(paste(flds[2],flds[5],flds[6],sep=":"),N)
    tmp[[i]]$s.M.t <- paste(flds[4],flds[2],flds[5],sep="-")
  }

  Pseldata <- do.call("rbind",unclass(tmp))
  row.names(Pseldata) <- seq(along=Pseldata[[1]])
  Pseldata$set <- factor(Pseldata$set)
  Pseldata$s.M.t <- factor(Pseldata$s.M.t)
  Respscale <- max(seldata$chei)
  Dosescale <- max(seldata$dose)

  Pseldata$AChE.scaled <- Pseldata$AChE/Respscale
  Pseldata$Dose.scaled <- Pseldata$Dose/Dosescale
### Random Effects
  RnDoM1 <- if (length(levels(Pseldata$mrid)) == 1) {
    if (length(levels(Pseldata$set)) == 1) {
      NULL
    } else {
      list(set=pdDiag(form=BMD ~ 1)) ## BMD~1 | set
    }
  } else {
    if (length(levels(Pseldata$set)) > length(levels(Pseldata$mrid))) {
      list(mrid=pdDiag(form=BMD ~ 1),
```

```
            set=pdDiag(form=BMD~1)) ## BMD~1|mrid/set
    } else {
      list(mrid=pdDiag(form=BMD ~ 1)) ## BMD~1|mrid
    }
  }



  xx <- list(Chemical=chem,Start=getparms(xxsave,seldata),Data=seldata,
             Pdata=Pseldata,Respscale=Respscale,
             Dosescale=Dosescale,Random=RnDoM1)
  save(xx,file=file.path("Skeletons",paste(chem,"rda",sep=".")))
}
### master.R
### Assume that pvm has already been started, and assign work to each node
### in the cluster.  This is a general purpose master; be sure to change
### the value of slavename.  Also, the value of Chemicals can be reassigned
### if you want to do just a subset.  All the real work is done in the slave
### program.


require(rpvm)

attach("/home/setzer/tasks/CumRisk/post-Feb-SAP/Data/opdata.rda")
griddb <- read.csv("griddb.csv",row.names=1)
Chemicals <- row.names(griddb[griddb$Rerun == 1,])
cat("Running for Chemicals:\n")
print(Chemicals)

MOREWORK <- 22
TASKFAIL <- 99
HERESWORK <- 33
workingdir <- outputdir <- getwd()
Hosts <- "/home/setzer/.xpvm_hosts"
print(.PVM.start.pvmd(hosts=Hosts,block=TRUE))
### Here is where changes are most likely

slavename <- "slave" # name of program to do the work for a particular
                     # chemical


### To here


cfg <- .PVM.config()

nodenames <- as.character(cfg$name)

#ntasks <- max(min(nrow(cfg),length(Chemicals)-1),1)
ntasks <- min(nrow(cfg),length(Chemicals))

mytid <- .PVM.mytid()

children <- NULL
# for (i in seq(along=ntasks)) {
#   tmp <- .PVM.spawn(task="slaveR.sh",ntask=ntasks[i],
#                     flag="Host", where=nodenames[i],
#                     arglist=c(slavename,workingdir,
#                     outputdir))
#   cat(tmp, fill=TRUE)
#   if (length(tmp) > 0) children <- c(children,tmp)
# }
children <- .PVM.spawn(task="slaveR.sh",ntask=ntasks,
                     flag="Default",
```

```
                        arglist=c(slavename,workingdir,
                          outputdir))

Sys.sleep(10)

if (length(Chemicals) > 1) {
  tmp <- .PVM.spawn(task="slaveR.sh",ntask=1,flag="Host",
                    where="gandalf.localdomain",
                    arglist=c(slavename,workingdir,outputdir))
  children <- c(children,tmp)
}
### Check for and delete any -1's (system errors)

if (any(children == -1)) {
  warning(paste(sum(children == -1),"failed starts out of",length(children),
                "potential"))
  children <- children[children != -1]
}

if ((Nchild <- length(children)) == 0) stop("No children started\n")

### Get the nodenames of the tasks

NodeNames <- character(length(children))
for (i in seq(along=NodeNames))
  NodeNames[i] <-
as.character(cfg$name[match(.PVM.tasks(where=children[i])$host,
                                  cfg$host.id)])

### Request notification of task exiting

.PVM.notify(msgtag=TASKFAIL,what="ExitTask",children)

### Start looping

i <- 0
j <- 1
Nrunning <- Nchild
repeat {
  i <- (i %% Nchild) + 1
  while (buf <- .PVM.nrecv(-1, TASKFAIL) > 0) {
    tmp <- .PVM.upkint()
    kk <- match(tmp, children)
    if (!is.na(kk)) {
      cat(paste("\n>>> Task",tmp,"on",NodeNames[kk],"has exited\n"))
      Sys.sleep(5)
      children[kk] <- .PVM.spawn(task="slaveR.sh",ntask=1,
                                  flag="Host", where=NodeNames[kk],
                                  arglist=c(slavename,workingdir,
                                    outputdir))
      cat(paste("Replaced by task",children[kk],"\n\n"))
      Sys.sleep(3)
      .PVM.notify(msgtag=TASKFAIL,what="ExitTask",children[kk])
      next
    }
  }
  buf <- .PVM.nrecv(children[i], MOREWORK)
  if ( buf > 0) {
#    tmp <- .PVM.upkstrvec()
#    cat(paste("\n++",tmp,"completed at",date(),"\n"))
    if (j <= length(Chemicals)) {
      cat(paste("\n++ Sending",Chemicals[j],"to task",children[i],
                "on",NodeNames[i],"at",date(),"\n"))
      .PVM.initsend()
      .PVM.pkstrvec(Chemicals[j])
```

```
      .PVM.send(children[i],HERESWORK)
      j <- j + 1
    } else {
      Nrunning <- Nrunning - 1
    }
  }
  if (Nrunning <= 0) {
    break
  }
}
cat(paste("\n!!!!!! All Done!",date(),"\n"))
for (i in seq(along=children)) .PVM.kill(children[i])

rm(Chemicals) ## so we can use the one in opdata.rda
source("plotProfiles.R")

.PVM.exit()



### slave.R
### Profile likelihoods for exponential model and male,female values of B
###
### Get the command line arguments (to specify which chemicals to run):


invisible(options(echo=FALSE))
require(rpvm)
cat("\n================================\n")
mytid <- .PVM.mytid()
cat(paste("I am task",mytid,"on",system("uname -n",intern=TRUE),"\n"))
cat("\n================================\n\n")

attach("/home/setzer/tasks/CumRisk/post-Feb-SAP/Data/opdata.rda")
require(RBMDS)
require(nlme)
setwd("/home/setzer/tasks/CumRisk/post-Feb-SAP/ProfilesForB")
savepath <- "Fits"
griddb <- read.csv("griddb.csv",row.names=1)
MOREWORK <- 22
TASKFAIL <- 99
HERESWORK <- 33


myparent <- .PVM.parent()
chem <- "START"

### BMR is set here
BMR <- 0.1

if (myparent <= 0) stop("PVM error!")

repeat {
  .PVM.initsend()
#  .PVM.pkstrvec(chem)
  .PVM.send(myparent,MOREWORK)
  buf <- .PVM.recv(myparent,HERESWORK)
  chem <- .PVM.upkstrvec()
  cat(paste("-------\n",chem,"\n\n"))
  OldModel <- file.path("Skeletons", paste(chem,"rda",sep="."))
  load(OldModel)
  Firststart<- c(xx$Start[["A"]] - log(xx$Respscale),
                 xx$Start[["BMD"]] - log(xx$Dosescale))
  start <- Firststart
### Compute the grid of values.
```

```
  for (i in 1:nrow(xx$Bgrid)) {
    xx$Models[[i]] <- eval(substitute(AChE.scaled ~
CexpBwDS(Dose.scaled,A,BMD,sex,
                                                          fixed=x),

list(x=list(B=c(F=as.vector(xx$Bgrid$B.F[i]),

M=as.vector(xx$Bgrid$B.M[i])))))
  }

  Mrids <- unique(xx$Data$mrid)
  if (length(Mrids) > 1) {
    Power <- rep(0.5, length(Mrids))
    names(Power) <- as.character(Mrids)
    Weights <- varComb(varIdent(form=~1|factor(mrid)),
                       varPower(form=~fitted(.)|factor(mrid),
                                fixed=Power))
  } else {
    Weights <- varPower(form=~fitted(.),fixed=1)
  }

  for (i in 1:nrow(xx$Bgrid)) {

cat(paste("\n------\n",i,":",xx$Bgrid$B.F[i],xx$Bgrid$B.M[i],date(),"\n\n"))
    Model <- xx$Models[[i]]
    RnDoM1 <- xx$Random


     xx$Fits[[i]] <- if (!is.null(RnDoM1)) {
         try(eval(substitute(nlme(Model,data=xx$Pdata,
                                  fixed=list(A ~ s.M.t - 1, BMD ~ sex-1),
                                  random=xxxx,
                                  start=start,
                                  weights=Weights,
                                  method="ML"),

list(Model=Model,xxxx=RnDoM1,Weights=Weights))))
     } else {
        try(eval(substitute(gnls(Model, data=xx$Pdata,
                                 params=list(A ~ s.M.t - 1, BMD ~ sex-1),
                                 start=start,
                                 weights=Weights),
                            list(Model=Model,Weights=Weights))))
    }
    xx$Bgrid$LL[i] <-
      if (!inherits(xx$Fits[[i]], "try-error")) logLik(xx$Fits[[i]]) else NA
    j <- if (i %% xx$Nsteps == 0) {
      i - xx$Nsteps + 1
    } else {
      i
    }
    start <- if (!inherits(xx$Fits[[j]],"try-error")) {
      if (inherits(xx$Fits[[j]],"nlme")) xx$Fits[[j]]$coefficients$fixed
      else xx$Fits[[j]]$coefficients
    } else Firststart

    save(xx,file=file.path(savepath,paste(chem,"rda",sep=".")))
  }
  cat(paste("-------",chem,"complete",date(),"\n\n"))
}


### makefinegriddb.R
loadpath <- "FineFits"
griddb <- read.csv("griddb.csv",row.names=1)
```

```
attach("../Data/opdata.rda")

griddb$Npoints[] <- 11
griddb$Rerun[] <- 1

for (chem in Chemicals) {
  load(file.path(loadpath,paste(chem,"rda",sep=".")))
  griddb[chem,"B.Fmin"] <- min(xx$Bgrid$B.F)
  griddb[chem,"B.Fmax"] <- max(xx$Bgrid$B.F)
  griddb[chem,"B.Mmin"] <- min(xx$Bgrid$B.M)
  griddb[chem,"B.Mmax"] <- max(xx$Bgrid$B.M)
}

write.table(griddb,file="finegriddb.csv",sep=",",col.names=NA)
```

e. Part 3b: Expanded model.

For profile likelihood estimation of S and D in the expanded model, first 'master.R' and 'slave.R' are run, which use the best values found in the estimation of P_B for the basic model as initial values. Then successive versions of 'master' and 'slave' are produced (see below, 'finemaster.R', 'fineslave.R') that result from successive refinements of the grid.

```
### master.R
### Assume that pvm has already been started, and assign work to each node
### in the cluster.  This is a general purpose master; be sure to change
### the value of slavename.  Also, the value of Chemicals can be reassigned
### if you want to do just a subset.  All the real work is done in the slave
### program.


require(rpvm)

attach("/home/setzer/tasks/CumRisk/post-Feb-SAP/Data/opdata.rda")
griddb <- read.csv("griddb.csv",row.names=1)
Chemicals <- row.names(griddb[griddb$DoIt == 1,])
cat("Running for Chemicals:\n")
print(Chemicals)

MOREWORK <- 22
TASKFAIL <- 99
HERESWORK <- 33
GETSTARTED <- 1

workingdir <- outputdir <- getwd()
Hosts <- "/home/setzer/.xpvm_hosts"
#print(.PVM.start.pvmd(hosts=Hosts,block=TRUE))
#Sys.sleep(10)
### Here is where changes are most likely

slavename <- "slave" # name of program to do the work for a particular
                     # chemical


### To here
```

```
cfg <- .PVM.config()

nodenames <- as.character(cfg$name)

#ntasks <- max(min(nrow(cfg),length(Chemicals)-1),1)
ntasks <- min(nrow(cfg),length(Chemicals))

mytid <- .PVM.mytid()
#children <- NULL
# for (i in seq(along=ntasks)) {
#   tmp <- .PVM.spawn(task="slaveR.sh",ntask=ntasks[i],
#                     flag="Host", where=nodenames[i],
#                     arglist=c(slavename,workingdir,
#                     outputdir))
#   cat(tmp, fill=TRUE)
#   if (length(tmp) > 0) children <- c(children,tmp)
# }
children <- .PVM.spawn(task="slaveR.sh",ntask=ntasks,
                       flag="Default",
                       arglist=c(slavename,workingdir,
                         outputdir))
Sys.sleep(10)
print(children)

if (length(Chemicals) > 1) {
  tmp <- .PVM.spawn(task="slaveR.sh",ntask=1,flag="Host",
                    where="gandalf.localdomain",
                    arglist=c(slavename,workingdir,outputdir))
  children <- c(children,tmp)
}
Sys.sleep(10)
print(children)
### Check for and delete any -1's (system errors)

if (any(children == -1)) {
  warning(paste(sum(children == -1),"failed starts out of",length(children),
                "potential"))
  children <- children[children != -1]
}

if ((Nchild <- length(children)) == 0) stop("No children started\n")

### Get the nodenames of the tasks

NodeNames <- character(length(children))
for (i in seq(along=NodeNames)) {
  NodeNames[i] <-
as.character(cfg$name[match(.PVM.tasks(where=children[i])$host,
                                    cfg$host.id)])
}
### Request notification of task exiting

.PVM.notify(msgtag=TASKFAIL,what="ExitTask",children)

### Start them

.PVM.initsend()
.PVM.pkintvec(1:3)
.PVM.mcast(children,GETSTARTED)

### Start looping

i <- 0
j <- 1
```

```
Nrunning <- Nchild
repeat {
  i <- (i %% Nchild) + 1
  while (buf <- .PVM.nrecv(-1, TASKFAIL) > 0) {
    tmp <- .PVM.upkint()
    kk <- match(tmp, children)
    if (!is.na(kk)) {
      cat(paste("\n>>> Task",tmp,"on",NodeNames[kk],"has exited\n"))
      Sys.sleep(5)
      children[kk] <- .PVM.spawn(task="slaveR.sh",ntask=1,
                                 flag="Host", where=NodeNames[kk],
                                 arglist=c(slavename,workingdir,
                                   outputdir))
      cat(paste("Replaced by task",children[kk],"\n\n"))
      Sys.sleep(3)
      .PVM.notify(msgtag=TASKFAIL,what="ExitTask",children[kk])
      next
    }
  }
  buf <- .PVM.nrecv(children[i], MOREWORK)
  if ( buf > 0) {
    tmp <- .PVM.upkstrvec()
    cat(paste("\n++",tmp,"completed at",date(),"\n"))
    if (j <= length(Chemicals)) {
      cat(paste("\n++ Sending",Chemicals[j],"to task",children[i],
                "on",NodeNames[i],"at",date(),"\n"))
      .PVM.initsend()
      .PVM.pkstrvec(Chemicals[j])
      .PVM.send(children[i],HERESWORK)
      j <- j + 1
    } else {
      Nrunning <- Nrunning - 1
    }
  }
  if (Nrunning <= 0) {
    break
  }
}
cat(paste("\n!!!!!! All Done!",date(),"\n"))
for (i in seq(along=children)) .PVM.kill(children[i])

rm(Chemicals) ## so we can use the one in opdata.rda
source("plotProfiles.R")

.PVM.exit()
### slave.R

invisible(options(echo=FALSE))
require(rpvm)
cat("\n================================\n")
mytid <- .PVM.mytid()
cat(paste("I am task",mytid,"on",system("uname -n",intern=TRUE),"\n"))
cat("\n================================\n\n")



attach("/home/setzer/tasks/CumRisk/post-Feb-SAP/Data/opdata.rda")
require(RBMDS)
require(nlme)
setwd("/home/setzer/tasks/CumRisk/post-Feb-SAP/ProfilesForSD")
savepath <- "Fits"

griddb <- read.csv("griddb.csv",row.names=1)
### Walks diagonals of a grid, starting at the upper right hand corner.
walkgrid <- function(Nsteps) {
```

```
  mx <- matrix(1:(Nsteps*Nsteps),nrow=Nsteps,byrow=TRUE)
  GridIndex <- numeric(Nsteps*Nsteps)
  indx <- 1
  for (i in 1:Nsteps) {
    for (k in 1:i) {
      if (i %% 2 == 1) {
        ii <- i - k + 1
        jj <- Nsteps - k + 1
      } else {
        ii <- k
        jj <- Nsteps - i + k
      }
      GridIndex[indx] <- mx[ii,jj]
      indx <- indx + 1
    }
  }
  for (i in 2:Nsteps) {
    for (k in i:Nsteps) {
      if ((Nsteps - i + 1) %% 2 == 1) {
        ii <- Nsteps + i - k
        jj <- Nsteps - k + 1

      } else {
        ii <- k
        jj <- k - i + 1
      }
      GridIndex[indx] <- mx[ii,jj]
      indx <- indx + 1
    }
  }

  GridIndex
}

MOREWORK <- 22
TASKFAIL <- 99
HERESWORK <- 33
GETSTARTED <- 1
myparent <- .PVM.parent()

chem <- "START"
.PVM.recv(myparent, msgtag=GETSTARTED)
tmp <- .PVM.upkintvec()

if (myparent <=  0) stop("PVM error")
repeat {
  .PVM.initsend()
  .PVM.pkstrvec(chem)
  .PVM.send(myparent,MOREWORK)
  buf <- .PVM.recv(myparent,HERESWORK)
  chem <- .PVM.upkstrvec()
  fname <- paste(chem,"rda",sep=".")
  cat(paste("\n-----------------",chem,"--------------------\n\n"))

  load(file.path("../ProfilesForB/FinerFits",fname))

### Get the best fitting model
  indx <- which.max(xx$Bgrid$LL)
  StartLL <- xx$Bgrid$LL[indx]

### Get initial start value, the coefficients from xx$Fit[[indx]]
  start <- if (inherits(xx$Fit[[indx]], "nlme")) {
    xx$Fit[[indx]]$coefficients$fixed
  } else {
    xx$Fit[[indx]]$coefficients
```

```
  }

Bests <- as.vector(c(xx$Bgrid[indx,"B.F"],xx$Bgrid[indx,"B.M"]))

### Get the model for the random parameters used in the basic model
  RandomParms <- xx$Random


  Pseldata <- xx$Pdata

### Compute the grid of values.

  SDgrid <- expand.grid(S = seq(griddb[chem,"Smin"],
                          griddb[chem,"Smax"],
                          length=griddb[chem,"Npoints"]),
                        D = seq(griddb[chem,"Dmin"],
                          griddb[chem, "Dmax"],
                          length=griddb[chem,"Npoints"]))
  SDgrid$LL <- numeric(nrow(SDgrid))
  SDgrid$LL[] <- as.numeric(NA)
  Fits <- vector("list",nrow(SDgrid))
  GridIndex <- walkgrid(griddb[chem,"Npoints"])

  Mrids <- unique(xx$Data$mrid)
  if (length(Mrids) > 1) {
    Power <- rep(0.5, length(Mrids))
    names(Power) <- as.character(Mrids)
    Weights <- varComb(varIdent(form=~1|factor(mrid)),
                       varPower(form=~fitted(.)|factor(mrid),
                                fixed=Power))
  } else {
    Weights <- varPower(form=~fitted(.),fixed=1)
  }

  xx$Fit <- Fits
  xx$Bgrid <- NULL
  xx$SDgrid <- SDgrid

  ## If our Dmin exceeds 0.001, walk up from 0.001 to Dmin in steps of 0.05

  if (griddb[chem,"Dmin"] > 0.001) {
    cat("\n------- Walking up to the parameter grid -----\n")
    Dseq <- seq(0.001, griddb[chem,"Dmin"], by=0.05)
    Smax <- griddb[chem,"Smax"]

    for (i in 1:length(Dseq)) {
      cat("\n-------------\n")
      cat(paste("i:",i,"D[i]:",Dseq[i],", S:",Smax,"\n"))
      cat("\n-------------\n")
      ## set up the model
       Model <- eval(substitute(AChE.scaled ~
CpkexpB2wDS(Dose.scaled,A,BMD,sex,

fixed=list(PB=c(F=xxxx,

M=yyyy),
                                                                S=c(F=zzzz,

M=zzzz),
                                                                D=c(F=wwww,

M=wwww))),
                                  list(xxxx=Bests[1],yyyy=Bests[2],
                                       zzzz=log(Smax),
                                       wwww=log(Dseq[i]))))
     ## estimate it
      if (!is.null(RandomParms)) {
```

```
        try(fitpk <-
            eval(substitute(nlme(Model,data=Pseldata,
                                 fixed=list(A ~ s.M.t - 1, BMD ~ sex - 1),
                                 random=xxxx,
                                 start=zzzz,
                                 weights=Weights,
                                 method="ML"),
                            list(Model=Model,xxxx=RandomParms,
                                 zzzz=start,
                                 Weights=Weights))))
      } else {
        fitpk <-
          try(eval(substitute(gnls(Model, data=Pseldata,
                                   params=list(A ~ s.M.t - 1, BMD ~ sex -
1),
                                   start=zzzz,
                                   weights=Weights),
                              list(Model=Model,
                                   zzzz=start,
                                   Weights=Weights))))
      }
      if (!inherits(fitpk, "try-error")) {
        ## Use the current fit to give the start value for the next
        start <- if (inherits(fitpk, "nlme")) {
          fitpk$coefficients$fixed
        } else {
          fitpk$coefficients
        }
      }
    }
  }

  for (iii in 1:nrow(SDgrid)) {
    i <- GridIndex[iii]
    cat("\n-------------\n")
    cat(paste("i:",i,"D[i]:",SDgrid$D[i],", S[i]:",SDgrid$S[i],date(),"\n"))
    cat("\n-------------\n")

    ## set up the model
    Model <- eval(substitute(AChE.scaled ~
CpkexpB2wDS(Dose.scaled,A,BMD,sex,

fixed=list(PB=c(F=xxxx,
                                                                  M=yyyy),
                                                        S=c(F=zzzz, M=zzzz),
                                                        D=c(F=wwww,
M=wwww))),
                              list(xxxx=Bests[1],yyyy=Bests[2],
                                   zzzz=log(SDgrid$S[i]),
                                   wwww=log(SDgrid$D[i])))))
    ## estimate it
    if (!is.null(RandomParms)) {
      try(fitpk <-
        eval(substitute(nlme(Model,data=Pseldata,
                                 fixed=list(A ~ s.M.t - 1, BMD ~ sex - 1),
                                 random=xxxx,
                                 start=zzzz,
                                 weights=Weights,
                                 method="ML"),
                            list(Model=Model,xxxx=RandomParms,
                                 zzzz=start,
                                 Weights=Weights))))
     } else {
       fitpk <-
        try(eval(substitute(gnls(Model, data=Pseldata,
```

```
                                    params=list(A ~ s.M.t - 1, BMD ~ sex - 1),
                                    start=zzzz,
                                    weights=Weights),
                               list(Model=Model,
                                    zzzz=start,
                                    Weights=Weights))))
  }

    if (!inherits(fitpk, "try-error")) {
      xx$SDgrid$LL[i] <- logLik(fitpk)
      cat(paste("\nLL:",xx$SDgrid$LL[i],"\n"))
      ## Use the current fit to give the start value for the next
      start <- if (inherits(fitpk, "nlme")) {
        fitpk$coefficients$fixed
      } else {
        fitpk$coefficients
      }
    }
    xx$Fit[[i]] <- fitpk
    save(xx,file=file.path("Fits",paste(chem,"rda",sep=".")))
  }
  cat(paste(chem,"finished",date(),"\n\n"))

}
### plotProfiles.R
require(akima)
pdf(file="Profiles-4-SD2.pdf")
par(xpd=NA)
dirname <- "Fits"
attach("../Data/opdata.rda")
for (chem in Chemicals){
  fname <- paste(chem,"rda",sep=".")
  load(file.path("../ProfilesForB/FinerFits",fname))
  B.BestLL <- max(xx$Bgrid$LL,na.rm=TRUE)
  xx <- list()
  ## Get the best-fitting Basic model

  if (!file.exists(file.path(dirname,fname))) next
  tmp <- try(load(file.path(dirname,fname)))
  if (inherits(tmp,"try-error")) {
    cat(paste("Problem reading",chem,"\n"))
    next
  }
  LLgrid <- xx$SDgrid
  LLgrid <- na.omit(LLgrid)
  if (length(LLgrid$LL) == 0) {
    plot(c(0,1),c(0,1),type="n",axes=FALSE, xlab="", ylab="")
    text(0.5,0.5,paste(chem,"no fits"),adj=0.5)
  } else {
    indx <- which.max(LLgrid$LL)
    BestLL <- max(LLgrid$LL[indx],B.BestLL)
    ScaledLL <- 2*(BestLL - LLgrid$LL)
    out <- try(interp(LLgrid$S,LLgrid$D,ScaledLL,
                  xo=seq(min(LLgrid$S), max(LLgrid$S),length=100),
                  yo=seq(min(LLgrid$D), max(LLgrid$D),length=100)))
    if (!inherits(out, "try-error")) {
      res <- try(image(out,xlab="S",ylab="D",main=chem,
                       col = rev(heat.colors(9)),
                       breaks =
c(0,qchisq(c(0.05,.10,.25,.50,.75,.90,.95,.99),2),1e26)))
      if (inherits(res, "try-error")) {
        plot(c(0,1),c(0,1),type="n",axes=FALSE, xlab="", ylab="")
        text(0.5,0.5,paste(chem,"not enough fits"),adj=0.5)
      } else {
        critX2 <- qchisq(0.95,2)
```

```
        points(LLgrid$S,LLgrid$D,
               pch=ifelse(2*(BestLL - LLgrid$LL) < critX2, 19, 3))
        points(LLgrid$S[indx],LLgrid$D[indx],pch=4,cex=1.5)
      }
    } else {
    plot(c(0,1),c(0,1),type="n",axes=FALSE, xlab="", ylab="")
    text(0.5,0.5,paste(chem,"not enough fits"),adj=0.5)
    }
  }
}
dev.off()

### makeFineGrid.R --
### Uses the information in the files in ./Fits to create a new set of
templates
### in FineFits.  Each new template already contains SDgrid and Fits
###
### The best expanded fit for the following chemicals is essentially the
basic model,
### so they are excluded from further action.  The criteria were:
### 1) Pvalue for the difference in log likelihoods was greater than 0.05
AND
### 2) BOTH BMDs were no more than 10% different between the expanded and
basic models
###    with the expanded model being the point of comparison (in the
denominator).

K2IJ <- function(k, Npoints) {
  c(Sindx = (i <- ((k-1)%%Npoints) + 1),
    Dindx = (k - i)/Npoints + 1)
}

IJ2K <- function(i, j, Npoints) {
  i + Npoints * (j - 1)
}

indxmin <- function(i, N) {
  if (i == 1) i else i - 1
}
indxmax <- function(i, N) {
  if (i == N) i else i + 1
}
Neighbors <- function(k, Npoints) {
  ij <- K2IJ(k, Npoints)
  i <- ij[1]
  j <- ij[2]
  iseq <- indxmin(i,Npoints):indxmax(i,Npoints)
  jseq <- indxmin(j, Npoints):indxmax(j, Npoints)

as.vector(apply(data.matrix(expand.grid(i=iseq,j=jseq)),1,function(x)IJ2K(x[
1],x[2],Npoints)))
}


require(RBMDS)
require(nlme)

DropChems <-
c("ACEPHATE","CHLORPYRIPHOSMETHYL","DICROTOPHOS","DIMETHOATE","ETHOPROP",

"FENTHION","METHAMIDOPHOS","METHIDATHION","NALED","OXYDEMETONMETHYL",
                "PIRIMIPHOSMETHYL","PROFENOFOS")

attach("../Data/opdata.rda")
savedir <- "Skel2"
```

```
finegriddb <- read.csv("griddb.csv",row.names=1)

finegriddb[DropChems,"DoIt"] <- 0
finegriddb$Npoints[] <- 5

Dochem <- which(finegriddb$DoIt == 1)


for (chem in Chemicals) {
  fname <- paste(chem,"rda",sep=".")
  load(file.path("Fits",fname))
  ##  'xx' is the new version
  if (chem %in% DropChems) {
    save(xx, file=file.path("FineFits",fname))
    next
  } else {
    oldxx <- xx
    xx$Nsteps <- 5
    xx$Fit <- list()
    Slist <- sort(unique(oldxx$SDgrid$S))
    Dlist <- sort(unique(oldxx$SDgrid$D))

    ## Build the new SDgrid centered around the maximum LL on the old one
    indx <- which.max(oldxx$SDgrid$LL)
    SDindx <- K2IJ(indx,oldxx$Nsteps)
    Smin <- Slist[indxmin(SDindx[1],oldxx$Nsteps)]
    Smax <- Slist[indxmax(SDindx[1],oldxx$Nsteps)]
    Dmin <- Dlist[indxmin(SDindx[2],oldxx$Nsteps)]
    if (SDindx[2] < oldxx$Nsteps) {
      Dmax <- Dlist[indxmax(SDindx[2], oldxx$Nsteps)]
    } else {
      ## If the old D was on the upper border of the grid, expand it by one
step
      Delta <- (max(Dlist) - min(Dlist))/oldxx$Nsteps
      Dmax <- max(Dlist) + Delta
    }
    finegriddb[chem,"Smin"] <- Smin
    finegriddb[chem,"Smax"] <- Smax
    finegriddb[chem,"Dmin"] <- Dmin
    finegriddb[chem,"Dmax"] <- Dmax
    xx$SDgrid <- expand.grid(S=seq(Smin,Smax,length=xx$Nsteps),
                             D=seq(Dmin,Dmax,length=xx$Nsteps))
    xx$SDgrid$LL <- numeric(nrow(xx$SDgrid))
    xx$SDgrid$LL[] <- NA
    xx$Start <- if (inherits(oldxx$Fit[[indx]],"nlme")) {
      oldxx$Fit[[indx]]$coefficients$fixed
    } else {
      oldxx$Fit[[indx]]$coefficients
    }

    xx$Fit <- vector("list",nrow(xx$SDgrid))

    ## Finally, fill in the LLs and Fits we already know.
    ## What are all the indexes?
    nearby <- Neighbors(indx,oldxx$Nsteps)

    ## There must be a better way to do this, but I'm tired ... .
    for (i in seq(along=nearby)) {
      K <- nearby[i]
      a1 <- sapply(xx$SDgrid$S,
                   function(x)identical(all.equal(x,
oldxx$SDgrid$S[K]),TRUE))
      a2 <- sapply(xx$SDgrid$D,
                   function(x)identical(all.equal(x,
oldxx$SDgrid$D[K]),TRUE))
```

```
      j <- which(a1 & a2)
      if (length(j) == 1) {
        xx$SDgrid$LL[j] <- oldxx$SDgrid$LL[K]
        xx$Fit[[j]] <- oldxx$Fit[[K]]
      }
    }

    save(xx,file=file.path(savedir,fname))
  }
}

write.table(finegriddb,file="finegriddb.csv",sep=",",col.names=NA)

### master.R
### Assume that pvm has already been started, and assign work to each node
### in the cluster.  This is a general purpose master; be sure to change
### the value of slavename.  Also, the value of Chemicals can be reassigned
### if you want to do just a subset.  All the real work is done in the slave
### program.


require(rpvm)

attach("/home/setzer/tasks/CumRisk/post-Feb-SAP/Data/opdata.rda")
griddb <- read.csv("finegriddb.csv",row.names=1)
Chemicals <- row.names(griddb[griddb$DoIt == 1,])
cat("Running for Chemicals:\n")
print(Chemicals)

MOREWORK <- 22
TASKFAIL <- 99
HERESWORK <- 33
GETSTARTED <- 1

workingdir <- outputdir <- getwd()
Hosts <- "/home/setzer/.xpvm_hosts"
#print(.PVM.start.pvmd(hosts=Hosts,block=TRUE))
#Sys.sleep(10)
### Here is where changes are most likely

slavename <- "fineslave" # name of program to do the work for a particular
                         # chemical


### To here


cfg <- .PVM.config()

nodenames <- as.character(cfg$name)

#ntasks <- max(min(nrow(cfg),length(Chemicals)-1),1)
ntasks <- min(nrow(cfg),length(Chemicals))

mytid <- .PVM.mytid()
#children <- NULL
# for (i in seq(along=ntasks)) {
#   tmp <- .PVM.spawn(task="slaveR.sh",ntask=ntasks[i],
#                     flag="Host", where=nodenames[i],
#                     arglist=c(slavename,workingdir,
#                     outputdir))
#   cat(tmp, fill=TRUE)
#   if (length(tmp) > 0) children <- c(children,tmp)
# }
children <- .PVM.spawn(task="slaveR.sh",ntask=ntasks,
```

```
                        flag="Default",
                        arglist=c(slavename,workingdir,
                          outputdir))
Sys.sleep(10)

if (length(Chemicals) > 1) {
  tmp <- .PVM.spawn(task="slaveR.sh",ntask=1,flag="Host",
                    where="gandalf.localdomain",
                    arglist=c(slavename,workingdir,outputdir))
  children <- c(children,tmp)
}
Sys.sleep(10)

### Check for and delete any -1's (system errors)

if (any(children == -1)) {
  warning(paste(sum(children == -1),"failed starts out of",length(children),
                "potential"))
  children <- children[children != -1]
}

if ((Nchild <- length(children)) == 0) stop("No children started\n")

### Get the nodenames of the tasks

NodeNames <- character(length(children))
for (i in seq(along=NodeNames)) {
  NodeNames[i] <-
as.character(cfg$name[match(.PVM.tasks(where=children[i])$host,
                                  cfg$host.id)])
}
### Request notification of task exiting

.PVM.notify(msgtag=TASKFAIL,what="ExitTask",children)

### Start them

.PVM.initsend()
.PVM.pkintvec(1:3)
.PVM.mcast(children,GETSTARTED)

### Start looping

i <- 0
j <- 1
Nrunning <- Nchild
repeat {
  i <- (i %% Nchild) + 1
  while (buf <- .PVM.nrecv(-1, TASKFAIL) > 0) {
    tmp <- .PVM.upkint()
    kk <- match(tmp, children)
    if (!is.na(kk)) {
      cat(paste("\n>>> Task",tmp,"on",NodeNames[kk],"has exited\n"))
      Sys.sleep(5)
      children[kk] <- .PVM.spawn(task="slaveR.sh",ntask=1,
                                  flag="Host", where=NodeNames[kk],
                                  arglist=c(slavename,workingdir,
                                    outputdir))
      cat(paste("Replaced by task",children[kk],"\n\n"))
      Sys.sleep(3)
      .PVM.notify(msgtag=TASKFAIL,what="ExitTask",children[kk])
      next
    }
  }
  buf <- .PVM.nrecv(children[i], MOREWORK)
```

```
  if ( buf > 0) {
    tmp <- .PVM.upkstrvec()
    cat(paste("\n++",tmp,"completed at",date(),"\n"))
    if (j <= length(Chemicals)) {
      cat(paste("\n++ Sending",Chemicals[j],"to task",children[i],
                "on",NodeNames[i],"at",date(),"\n"))
      .PVM.initsend()
      .PVM.pkstrvec(Chemicals[j])
      .PVM.send(children[i],HERESWORK)
      j <- j + 1
    } else {
      Nrunning <- Nrunning - 1
    }
  }
  if (Nrunning <= 0) {
    break
  }
}
cat(paste("\n!!!!!! All Done!",date(),"\n"))
for (i in seq(along=children)) .PVM.kill(children[i])

rm(Chemicals) ## so we can use the one in opdata.rda
source("plotProfiles.R")

.PVM.exit()
### slave.R

invisible(options(echo=FALSE))
require(rpvm)
cat("\n=================================\n")
mytid <- .PVM.mytid()
cat(paste("I am task",mytid,"on",system("uname -n",intern=TRUE),"\n"))
cat("\n=================================\n\n")



attach("/home/setzer/tasks/CumRisk/post-Feb-SAP/Data/opdata.rda")
require(RBMDS)
require(nlme)
setwd("/home/setzer/tasks/CumRisk/post-Feb-SAP/ProfilesForSD")
savepath <- "FineFits"

### Walks diagonals of a grid, starting at the upper right hand corner.

MOREWORK <- 22
TASKFAIL <- 99
HERESWORK <- 33
GETSTARTED <- 1
myparent <- .PVM.parent()

chem <- "START"
.PVM.recv(myparent, msgtag=GETSTARTED)
tmp <- .PVM.upkintvec()

if (myparent <=  0) stop("PVM error")
repeat {
  .PVM.initsend()
  .PVM.pkstrvec(chem)
  .PVM.send(myparent,MOREWORK)
  buf <- .PVM.recv(myparent,HERESWORK)
  chem <- .PVM.upkstrvec()
  fname <- paste(chem,"rda",sep=".")
  cat(paste("\n-----------------",chem,"--------------------\n\n"))

  load(file.path("Skel2",fname))
```

```
### Get initial start value, the coefficients from xx$Fit[[indx]]
  start <- xx$Start

### and the estimates of PB to use
  indx <- which.max(xx$SDgrid$LL)
  Bests <- as.vector(eval(xx$Fit[[indx]]$call$model[[3]]$fixed$PB))

### Get the model for the random parameters used in the basic model
  RandomParms <- xx$Random


  Pseldata <- xx$Pdata


  Mrids <- unique(xx$Data$mrid)
  if (length(Mrids) > 1) {
    Power <- rep(0.5, length(Mrids))
    names(Power) <- as.character(Mrids)
    Weights <- varComb(varIdent(form=~1|factor(mrid)),
                       varPower(form=~fitted(.)|factor(mrid),
                                fixed=Power))
  } else {
    Weights <- varPower(form=~fitted(.),fixed=1)
  }

  for (i in 1:nrow(xx$SDgrid)) {
    if (!is.na(xx$SDgrid$LL[i])) next
    cat("\n-------------\n")
    cat(paste("i:",i,"D[i]:",xx$SDgrid$D[i],",
S[i]:",xx$SDgrid$S[i],date(),"\n"))
    cat("\n-------------\n")

    ## set up the model
    Model <- eval(substitute(AChE.scaled ~
CpkexpB2wDS(Dose.scaled,A,BMD,sex,

fixed=list(PB=c(F=xxxx,
                                                                  M=yyyy),
                                                        S=c(F=zzzz, M=zzzz),
                                                        D=c(F=wwww,
M=wwww))),
                             list(xxxx=Bests[1],yyyy=Bests[2],
                                  zzzz=log(xx$SDgrid$S[i]),
                                  wwww=log(xx$SDgrid$D[i]))))
    ## estimate it
      fitpk <- if (!is.null(RandomParms)) {
        try(eval(substitute(nlme(Model,data=Pseldata,
                                 fixed=list(A ~ s.M.t - 1, BMD ~ sex - 1),
                                 random=xxxx,
                                 start=zzzz,
                                 weights=Weights,
                                 method="ML"),
                            list(Model=Model,xxxx=RandomParms,
                                 zzzz=start,
                                 Weights=Weights))))
      } else {
        try(eval(substitute(gnls(Model, data=Pseldata,
                                 params=list(A ~ s.M.t - 1, BMD ~ sex - 1),
                                 start=zzzz,
                                 weights=Weights),
                            list(Model=Model,
                                 zzzz=start,
                                 Weights=Weights))))
      }
```

```
    if (!inherits(fitpk, "try-error")) {
      xx$SDgrid$LL[i] <- logLik(fitpk)
      cat(paste("\nLL:",xx$SDgrid$LL[i],"\n"))
    }
    xx$Fit[[i]] <- fitpk
    save(xx,file=file.path("FineFits",paste(chem,"rda",sep=".")))
  }
  cat(paste(chem,"finished",date(),"\n\n"))

}
### makeFine2Grid.R --
### Uses the information in the files in ./Fits to create a new set of
templates
### in Fine2Fits.  Each new template already contains SDgrid and Fits
###
### The best expanded fit for the following chemicals is essentially the
basic model,
### so they are excluded from further action.  The criteria were:
### 1) Pvalue for the difference in log likelihoods was greater than 0.05
AND
### 2) BOTH BMDs were no more than 10% different between the expanded and
basic models
###    with the expanded model being the point of comparison (in the
denominator).

DropChems <-
c("ACEPHATE","CHLORPYRIPHOSMETHYL","DICROTOPHOS","DIMETHOATE","ETHOPROP",

"FENTHION","METHAMIDOPHOS","METHIDATHION","NALED","OXYDEMETONMETHYL",
               "PIRIMIPHOSMETHYL","PROFENOFOS")

K2IJ <- function(k, Npoints) {
  c(Sindx = (i <- ((k-1)%%Npoints) + 1),
    Dindx = (k - i)/Npoints + 1)
}

IJ2K <- function(i, j, Npoints) {
  i + Npoints * (j - 1)
}

indxmin <- function(i, N) {
  if (i == 1) i else i - 1
}
indxmax <- function(i, N) {
  if (i == N) i else i + 1
}
Neighbors <- function(k, Npoints) {
  ij <- K2IJ(k, Npoints)
  i <- ij[1]
  j <- ij[2]
  iseq <- indxmin(i,Npoints):indxmax(i,Npoints)
  jseq <- indxmin(j, Npoints):indxmax(j, Npoints)

as.vector(apply(data.matrix(expand.grid(i=iseq,j=jseq)),1,function(x)IJ2K(x[
1],x[2],Npoints)))
}


require(RBMDS)
require(nlme)


attach("../Data/opdata.rda")
savedir <- "Skel2"
```

```
finegriddb <- read.csv("griddb.csv",row.names=1)

finegriddb[DropChems,"DoIt"] <- 0
finegriddb$Npoints[] <- 5

Dochem <- which(finegriddb$DoIt == 1)


for (chem in Chemicals) {
  fname <- paste(chem,"rda",sep=".")
  load(file.path("FineFits",fname))
  ##  'xx' is the new version
  if (chem %in% DropChems) {
    save(xx, file=file.path("Fine2Fits",fname))
    next
  } else {
    oldxx <- xx
    xx$Nsteps <- 5
    xx$Fit <- list()
    Slist <- sort(unique(oldxx$SDgrid$S))
    Dlist <- sort(unique(oldxx$SDgrid$D))

    ## Build the new SDgrid centered around the maximum LL on the old one
    indx <- which.max(oldxx$SDgrid$LL)
    SDindx <- K2IJ(indx,oldxx$Nsteps)
    Smin <- Slist[indxmin(SDindx[1],oldxx$Nsteps)]
    Smax <- Slist[indxmax(SDindx[1],oldxx$Nsteps)]
    Dmin <- Dlist[indxmin(SDindx[2],oldxx$Nsteps)]
    if (SDindx[2] < oldxx$Nsteps) {
      Dmax <- Dlist[indxmax(SDindx[2], oldxx$Nsteps)]
    } else {
      ## If the old D was on the upper border of the grid, expand it by one
step
      Delta <- (max(Dlist) - min(Dlist))/oldxx$Nsteps
      Dmax <- max(Dlist) + Delta
    }
    finegriddb[chem,"Smin"] <- Smin
    finegriddb[chem,"Smax"] <- Smax
    finegriddb[chem,"Dmin"] <- Dmin
    finegriddb[chem,"Dmax"] <- Dmax
    xx$SDgrid <- expand.grid(S=seq(Smin,Smax,length=xx$Nsteps),
                             D=seq(Dmin,Dmax,length=xx$Nsteps))
    xx$SDgrid$LL <- numeric(nrow(xx$SDgrid))
    xx$SDgrid$LL[] <- NA
    xx$Start <- if (inherits(oldxx$Fit[[indx]],"nlme")) {
      oldxx$Fit[[indx]]$coefficients$fixed
    } else {
      oldxx$Fit[[indx]]$coefficients
    }

    xx$Fit <- vector("list",nrow(xx$SDgrid))

    ## Finally, fill in the LLs and Fits we already know.
    ## What are all the indexes?
    nearby <- Neighbors(indx,oldxx$Nsteps)

    ## There must be a better way to do this, but I'm tired ... .
    for (i in seq(along=nearby)) {
      K <- nearby[i]
      a1 <- sapply(xx$SDgrid$S,
                   function(x)identical(all.equal(x,
oldxx$SDgrid$S[K]),TRUE))
      a2 <- sapply(xx$SDgrid$D,
                   function(x)identical(all.equal(x,
oldxx$SDgrid$D[K]),TRUE))
```

```
      j <- which(a1 & a2)
      if (length(j) == 1) {
        xx$SDgrid$LL[j] <- oldxx$SDgrid$LL[K]
        xx$Fit[[j]] <- oldxx$Fit[[K]]
      }
    }

    save(xx,file=file.path(savedir,fname))
  }
}

write.table(finegriddb,file="fine2griddb.csv",sep=",",col.names=NA)
```