

Appendix A

Splus Code

All of the programming was done using Splus. In retrospect, it would have been a good idea to, at the least, write code in C that could be called by the Splus functions to improve the speed at which they run. This is especially so because the functions involve nested loops that result in slow performance in Splus.

A.1 Miscellaneous Functions

The following Splus functions were used in variance estimation and elsewhere. The first two functions create data without missing values, the first, `vok` by removing missing values from vectors, and the second, `mvok` by taking a matrix and dropping rows with missing values (or acting like `vok` if `vm` is a vector).

```
vok <- function(x) x[!is.na(x)]

mvok <- function(vm) {
  if(!is.matrix(vm)) {
    vmiss <- is.na(vm)
```

```

        vm[!vmiss]
    }
    else {
        vmiss <- rep(FALSE, nrow(vm))
        for(i in 1:ncol(vm)) vmiss <- vmiss | is.na(vm[, i])
        vm[!vmiss, ]
    }
}

```

The next two functions are for variance estimation. The first, `var3`, uses all available pairs of data to estimate each element of a covariance matrix, leaving a value of 0 for the cases where no data are available. Note that this can create a matrix that is not nonnegative definite. (This corresponds to (3.29).)

```

var3 <- function(x) {
  if(sum(is.na(x)) == 0)
    V <- var(x)
  else {
    d <- ncol(x)
    V <- matrix(0, nrow = d, ncol = d)
    for(i in 1:d) {
      for(j in i:d) {
        x.ij <- mvok(x[, c(i, j)])
        if(is.matrix(x.ij)) {
          V[i, j] <- V[j, i] <- var(x.ij[, 1], x.ij[, 2])
        }
      }
    }
  }
  V
}

```

Since we want our variance estimates to be nonnegative definite, we use function `var33`, corresponding to (3.31) as discussed in Section 3.6,

```

var33 <- function(x) {
  vmat <- var3(x)
  vmat <- (vmat + t(vmat))/2
  # ensures symmetric (should already be true)
  veig <- eigen(vmat)
  vvec <- veig$vectors
  vval <- veig$values
  vval[vval < 0] <- 0
  varret <- vvec %*% diag(vval) %*% t(vvec)
  varret
}

```

Because we often have to invert poorly conditioned matrices, the default Splus matrix inversion function, `solve`, is not sufficient as it sometimes thinks the matrices are singular. To get around this problem, we use the following matrix inversion function, `solve.eig`, which is meant to be used only on symmetric matrices and uses the eigenvalue decomposition to perform the inversion.

```

solve.eig <- function(x) {
  # for use only with symmetric matrices
  xx <- (x + t(x)) / 2
  eig.xx <- eigen(xx)
  ee <- eig.xx$vectors
  ll <- eig.xx$values
  inv.xx <- ee %*% diag(1/ll) %*% t(ee)
  inv.xx
}

```

A.2 Simple Smoothing Functions

The following Splus function produces the vector of second derivatives of a smoothing spline given the value `g` of the spline at the knots and the matrices `Q` and `R` associated with the smoothing spline. (Note that in this code, we have used `g` rather than `f` to represent the spline functions.)

```
gam.s.spline <- function(g, Q, R) {
  x <- t(Q) %*% g
  gam <- solve(R, x)
  gam
}
```

The following Splus function produces the value of a smoothing spline at times x given its vector of values g at knots tim and second derivatives $gamm$ at the interior knots of tim .

```
pred2.s.spline <- function(x, g, gamm, tim) {
  n <- length(g)
  nx <- length(x)
  gamm2 <- c(0, gamm, 0)
  if (length(gamm2) != n) stop("g must be 2 elements longer than gamma")
  if (length(tim) != n) stop("g and tim must be of same length")
  fit <- x * 0
  h <- tim[2:n] - tim[1:(n-1)]
  fit[x < tim[1]] <- g[1] - (tim[1]-x[x < tim[1]]) *
    ((g[2]-g[1])/(tim[2]-tim[1]) - (tim[2]-tim[1]) * gamm2[2]/6)
  fit[x >= tim[n]] <- g[n] + (x[x >= tim[n]] - tim[n]) *
    ((g[n]-g[n-1])/(tim[n]-tim[n-1]) + (tim[n]-tim[n-1]) * gamm2[n-1]/6)
  for (i in 1:(n-1)) {
    ind.j <- rep(F, nx)
    for (j in 1:nx) if (tim[i] <= x[j] & x[j] < tim[i+1]) ind.j[j] <- T
    fit[ind.j] <- ((x[ind.j] - tim[i]) * g[i+1] +
      (tim[i+1] - x[ind.j]) * g[i]) / h[i] -
      (x[ind.j] - tim[i])*(tim[i+1] - x[ind.j]) *
      ( (1 + (x[ind.j] - tim[i])/h[i]) * gamm2[i+1] +
      (1 + (tim[i+1] - x[ind.j])/h[i]) * gamm2[i] ) /6
  }
  fit
}
```

The above are as in Section 2.3.

A.3 Smoothing Functions With Replications

A.3.1 One Parameter Versions

The first function, `step.lto.g1.e`, is for performing the leave out one time cross-validation with no variance regularization. The arguments called are as follows: `datamat` is a matrix which contains a row for each t_{ij} with a column for an identifier for individual denoted by an `idstring` and $\max_{i,j,k}\{n_{ijk}\}$ columns for the y_{ijk} denoted by `datstrings`; `Ndat` is the number of individuals; `ids` their identifier values; `Nis` a vector with the total number of observations for each individual; `nis` a vector with the total number of times for each individual; `nijs` a matrix whose (i, j) th element is n_{ij} and similarly `tijs` a matrix of t_{ij} ; `vijs` a matrix whose (i, j) th element is the empirical variance of the y_{ijk} ; `his` is a matrix with rows containing the values of h , the differences $t_{i,j+1} - t_{ij}$; `grid1` is the initial grid of λ on which to perform the cross-validation; `useV` is a Boolean value saying whether there is a variance estimate for the coefficients to be used; `Vhat` is that variance estimate (one can put anything, such as 0, if `useV` is false); `nless` is a matrix whose (i, j) th element is the number of observations before t_{ij} , 0 for $i = 1$ and $\sum_{j' < j} n_{ij'}$ for $j > 1$; `nbasisfns` is the number of B-spline basis functions; and `basisfns` is a function that evaluates the value of the basis functions at any time. The working of the function is discussed below.

```
step.lto.g1.e <- function(datamat, Ndat, ids, Nis, nis, nijs, tijs, vijs,
                        his, grid1, useV, Vhat, nless, nbasisfns = 21,
                        basisfns = B.spl.18.frdiab, idstring="NIH",
                        datstrings=c("GFR1","GFR2","GFR3","GFR4")) {
  lengr <- length(grid1)
  ss <- rep(0, lengr)
  for (i in 1:Ndat) {
    ni <- nis[i]
```

```

nim1 <- ni - 1
nim2 <- ni - 2
Ni <- Nis[i]
Dmati <- diag(rep(0, Ni))
Nmati <- matrix(0, nrow=Ni, ncol=ni)
datamati <- datamat[datamat[,idstring]==ids[i], ]
datai <- 0
for (j in 1:ni) {
  nij <- nijs[i, j]
  nlij <- nless[i, j]
  datai <- c(datai, vok(datamati[j, datstrings]))
  Dmati[nlij + 1:nij, nlij + 1:nij] <- diag(rep(vijs[i, j], nij))
  Nmati[nlij + 1:nij, j] <- 1
}
datai <- datai[-1]
if (useV==F) Vmati <- Dmati else {
  Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
  for (j in 1:ni) {
    bvecij <- basisfns(tijs[i, j])
    for (k in 1:nijs[i, j]) Xi[nless[i, j] + k, ] <- bvecij
  }
  Vmati <- Xi %*% Vhat %*% t(Xi) + Dmati
}
for (j in 1:ni) {
  k4 <- nless[i, j]
  nij <- nijs[i, j]
  Nmatimj <- Nmati[-(k4 + 1:nij), -j] # (Ni-nij)x(ni-1)
  # drop rows AND column corresponding to tij
  Vmatimjinv <- solve(Vmati[-(k4 + 1:nij), -(k4 + 1:nij)])
  # drop rows and columnS corresponding to tij (Ni-nij)x(Ni-nij)
  Mat1mj <- t(Nmatimj) %*% Vmatimjinv %*% Nmatimj # (ni-1)x(ni-1)
  Mat2mj <- t(Nmatimj) %*% Vmatimjinv # (ni-1)x(Ni-nij)
  multmj <- 1 / Dmati[k4 + 1, k4 + 1]

  tijsmj <- (tijs[i, 1:ni])[-j]
  if (j == 1) hismj <- his[i, 2:nim1] else {
    if (j == ni) hismj <- his[i, 1:nim2] else {
      hismj <- tijsmj[2:nim1] - tijsmj[1:nim2]
    }
  }
}
Qmatimj <- matrix(0, nrow=nim1, ncol = nim1) # drop 2 columns below
for (k in 2:nim2) {
  Qmatimj[k-1,k] <- 1 / hismj[k-1]
}

```

```

    Qmatimj[k,k]  <- - 1 / hismj[k-1] - 1 / hismj[k]
    Qmatimj[k+1,k] <- 1 / hismj[k]
  }
  Qmatimj <- Qmatimj[, 2:nim2]           # (ni - 1) x (ni - 3)
  Rmatimj <- matrix(0, nrow=nim1, ncol = nim1) # will reduce below
  for (k in 2:(nim2)) Rmatimj[k, k] <- (hismj[k-1] + hismj[k]) / 3
  for (k in 2:(ni-3)) Rmatimj[k, k+1] <- Rmatimj[k+1, k] <- hismj[k]/6
  Rmatimj <- Rmatimj[2:(nim2), 2:(nim2)]   # (ni - 3) x (ni - 3)
  Kmatimj <- Qmatimj %*% solve(Rmatimj) %*% t(Qmatimj) # (ni-1)x(ni-1)
  Mat3mj <- Mat2mj %*% datai[-(k4 + 1:nij) ] # (ni - 1)x(1)

# THE ABOVE MATRICES DON'T CHANGE WITHIN ANY GROUP OF nij OBS.
for (m in 1:lengr) {
  fhatimjlammm <- solve.eig(Mat1mj + grid1[m] * Kmatimj) %*% Mat3mj
  gamimjlammm <- gam.s.spline(fhatimjlammm, Qmatimj, Rmatimj)
  # above are fit and second derivative at ni-1 times
  fhatijmjlamm <-
    pred2.s.spline(tijs[i, j], fhatimjlammm, gamimjlammm, tijsmj)
    # fit at time tij leaving out nij observations at tij
  ss[m] <- ss[m] + multmj *
    sum(square(datai[(k4 + 1:nij)] - fhatijmjlamm))
}
}
}
list(grid = grid1, cvss = ss)
}

```

The matrix D_{mati} is a diagonal matrix of the v_{ij} s; N_{mati} is the incidence matrix; datai is a vector containing all of the individual measurements for subject i ; V_{mati} is the estimated variance for the individual; N_{matimj} is the incidence matrix leaving out the j th time; V_{matimj} is the inverse of the variance matrix without time t_{ij} ; Q_{matimj} , R_{matimj} and K_{matimj} are the matrices \mathbf{Q} , \mathbf{R} and \mathbf{K} for subject i without time t_{ij} , fhatimjlammm is the vector of the unique values of the smoothing spline at times other than t_{ij} obtained by using $\lambda = \text{grid1}[m]$; gamimjlammm is the corresponding vector of second derivatives; fhatijmjlamm is the value of the preceding spline at t_{ij} ; and ss is a vector that contains the values of the cross-validation sum for each value of the

smoothing parameter.

Function `step.lto.g2.e` is basically the same as `step.lto.g1.e` with one minor addition: it takes three additional arguments and uses them to create a new grid of smoothing parameters for which to evaluate the cross-validation sum. The new arguments are: `grid1` and `ss1` are a grid and corresponding cross-validation from a running of `step.lto.g1.e` or `step.lto.g2.e` while `lengr2` sets the length of the finer grid of smoothing parameters on which to search. The first few lines of the function set up the new grid. The three smallest values of `ss1` are found along with their corresponding indices. Then the smaller of the smallest of these indices and one less than the index corresponding to the smallest value of `ss1` is taken and the new grid starts from the corresponding value of `grid1` (or the first value of `grid1` if it corresponds to the minimum of `ss1`). The upper end of the new grid is calculated similarly.

```
step.lto.g2.e <- function(grid1, ss1, datamat, Ndat, ids, Nis, nis,
  nijs, tijs, vijs, his, useV, Vhat, nless,
  nbasisfns=21, basisfns=B.spl.18.frdiab, lengr2=40,
  idstring="NIH", datstrings=c("GFR1","GFR2","GFR3","GFR4")) {
  minm <- order(ss1)[1]
  ind1m <- max(1, min(c(minm - 1, order(ss1)[1:3])))
  ind2m <- min(length(ss1), max(c(minm+1, order(ss1)[1:3])))
  grid2 <- seq(grid1[ind1m], grid1[ind2m], length = lengr2)
  ss2 <- rep(0, lengr2)
  for (i in 1:Ndat) {
    ni <- nis[i]
    nim1 <- ni - 1
    nim2 <- ni - 2
    Ni <- Nis[i]
    Dmati <- diag(rep(0, Ni))
    Nmati <- matrix(0, nrow=Ni, ncol=ni)
    datamati <- datamat[datamat[,idstring]==ids[i], ]
    datai <- 0
```

```

for (j in 1:ni) {
  nij <- nijs[i, j]
  nlij <- nless[i, j]
  datai <- c(datai, vok(datamati[j, datstrings]))
  Dmati[nlij + 1:nij, nlij + 1:nij] <- diag(rep(vijs[i, j], nij))
  Nmati[nlij + 1:nij, j] <- 1
}
datai <- datai[-1]
if (useV==F) Vmati <- Dmati else {
  Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
  for (j in 1:ni) {
    bvecij <- basisfns(tijs[i, j])
    for (k in 1:nijs[i, j]) Xi[nless[i, j] + k, ] <- bvecij
  }
  Vmati <- Xi %*% Vhat %*% t(Xi) + Dmati
}
for (j in 1:ni) {
  k4 <- nless[i, j]
  nij <- nijs[i, j]
  Nmatimj <- Nmati[-(k4 + 1:nij), -j] # (Ni-nij)x(ni-1)
  # drop rows AND column corresponding to tij
  Vmatimjinv <- solve(Vmati[-(k4 + 1:nij), -(k4 + 1:nij)])
  # drop rows and columnS corresponding to tij (Ni-nij)x(Ni-nij)
  Mat1mj <- t(Nmatimj) %*% Vmatimjinv %*% Nmatimj # (ni-1)x(ni-1)
  Mat2mj <- t(Nmatimj) %*% Vmatimjinv # (ni-1)x(Ni-nij)
  multmj <- 1 / Dmati[k4 + 1, k4 + 1]

  tijsmj <- (tijs[i, 1:ni])[-j]
  if (j == 1) hismj <- his[i, 2:nim1] else {
    if (j == ni) hismj <- his[i, 1:nim2] else {
      hismj <- tijsmj[2:nim1] - tijsmj[1:nim2]
    }
  }
  Qmatimj <- matrix(0, nrow=nim1, ncol = nim1) # drop 2 columns below
  for (k in 2:nim2) {
    Qmatimj[k-1,k] <- 1 / hismj[k-1]
    Qmatimj[k,k] <- - 1 / hismj[k-1] - 1 / hismj[k]
    Qmatimj[k+1,k] <- 1 / hismj[k]
  }
  Qmatimj <- Qmatimj[, 2:nim2] # (ni - 1) x (ni - 3)
  Rmatimj <- matrix(0, nrow=nim1, ncol = nim1) # will reduce below
  for (k in 2:(nim2)) Rmatimj[k, k] <- (hismj[k-1] + hismj[k]) / 3
  for (k in 2:(ni-3)) Rmatimj[k, k+1] <- Rmatimj[k+1, k] <- hismj[k]/6
}

```

```

Rmatimj <- Rmatimj[2:(nim2), 2:(nim2)]      # (ni - 3) x (ni - 3)
Kmatimj <- Qmatimj %%% solve(Rmatimj) %%% t(Qmatimj) # (ni-1)x(ni-1)
Mat3mj <- Mat2mj %%% datai[-(k4 + 1:nij) ] # (ni - 1)x(1)

# THE ABOVE MATRICES DON'T CHANGE WITHIN ANY GROUP OF nij OBSERVATIONS.
for (m in 1:lengr2) {
  fhatimjlamm <- solve.eig(Mat1mj + grid2[m] * Kmatimj) %%% Mat3mj
  gamimjlamm <- gam.s.spline(fhatimjlamm, Qmatimj, Rmatimj)
  # above are fit and second derivative at ni-1 times
  fhatijmjlamm <-
    pred2.s.spline(tijs[i, j], fhatimjlamm, gamimjlamm, tijsmj)
    # fit at time tij leaving out nij observations at tij
  ss2[m] <- ss2[m] + multmj *
    sum(square(datai[(k4 + 1:nij)] - fhatijmjlamm))
}
}
}
list(grid = grid2, cvss = ss2)
}

```

The following function takes the results of a cross-validation grid search and finds the smoothing parameter and then calculates the estimated coefficients for each individual. It takes the following new arguments: `tlims` was intended to be a matrix with I and 2 columns corresponding to the smallest and largest observation time for each subject (but is actually not used below so anything would work); `clims` is a similar matrix telling which are the first and last B-spline basis functions that are not identically equal to zero within the range of `tlims` as defined previously; `ext.tlims` is an extended range of times for each individual on which the smoothing spline is evaluated and then approximated by using least squares regression on the extended range of B-splines `ext.clims`. Once this is done, only the coefficients corresponding to the range of the data are returned. In addition, `basismat` is a matrix containing the values of the B-spline basis on a fixed grid whose first value corresponds to time `tshift` and whose values are separated by `tdiv`. (Every element of `ext.tlims`

is a nonnegative multiple of `tdiv` above `tshift` and so the i th row of `basismat` corresponds to time `tshift + (i - 1) tdiv`.)

```

step.extbest <- function(grid2, ss2, datamat, Ndat, ids, Nis, nis,
                        nijs, tijs, vijs, his, useV, Vhat, nless,
                        tlims, clims, ext.tlims, ext.clims,
                        basismat = pima.B.spline.mat.18.frd,
                        tshift = 72, tdiv = 3, nbasisfns = 21,
                        basisfns = B.spl.18.frdiab, idstring = "NIH",
                        datstrings = c("GFR1","GFR2","GFR3","GFR4")) {
  coefmat <- matrix(NA, nrow = Ndat, ncol = nbasisfns)
  smoothpar <- grid2[order(ss2)[1]]
  for (i in 1:Ndat) {
    ni <- nis[i]
    nim1 <- ni - 1
    Ni <- Nis[i]
    Dmati <- diag(rep(0, Ni))
    Nmati <- matrix(0, nrow=Ni, ncol=ni)
    datamati <- datamat[datamat[,idstring]==ids[i], ]
    datai <- 0
    for (j in 1:ni) {
      datai <- c(datai, vok(datamati[j, datstrings]))
      Dmati[nless[i, j] + 1:nijs[i, j], nless[i, j] + 1:nijs[i, j]] <-
        diag(rep(vijs[i, j], nijs[i, j]))
      Nmati[nless[i, j] + 1:nijs[i, j], j] <- 1
    }
    datai <- datai[-1]
    if (useV==F) Vmati <- Dmati else {
      Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
      for (j in 1:ni) {
        bvecij <- basisfns(tijs[i, j])
        for (k in 1:nijs[i, j]) Xi[nless[i, j] + k, ] <- bvecij
      }
      Vmati <- Xi %*% Vhat %*% t(Xi) + Dmati
    }
    Qmati <- matrix(0, nrow=ni, ncol = ni) # will drop 2 columns below
    for (k in 2:nim1) {
      Qmati[k-1,k] <- 1 / his[i, k-1]
      Qmati[k,k] <- - 1 / his[i, k-1] - 1 / his[i, k]
      Qmati[k+1,k] <- 1 / his[i, k]
    }
  }
}

```

```

Qmati <- Qmati[, 2:nim1]                # ni x (ni - 2)
Rmati <- matrix(0, nrow=ni, ncol = ni) # will reduce below
for (k in 2:(nim1)) Rmati[k, k] <- (his[i, k-1] + his[i, k]) / 3
for (k in 2:(ni-2)) Rmati[k, k+1] <- Rmati[k+1, k] <- his[i, k]/6
Rmati <- Rmati[2:(nim1), 2:(nim1)]      # (ni - 2) x (ni - 2)
Kmati <- Qmati %%% solve(Rmati) %%% t(Qmati) # ni x ni

Shatlami <- solve(t(Nmati) %%% solve(Vmati) %%% Nmati +
                 smoothpar * Kmati) %%% t(Nmati) %%% solve(Vmati)
fveci <- Shatlami %%% datai

ext.minti <- ext.tlims[i, 1]
ext.maxti <- ext.tlims[i, 2]
ext.minci <- ext.clims[i, 1]
ext.maxci <- ext.clims[i, 2]
minci <- clims[i, 1]
maxci <- clims[i, 2]
lsfi <- lsfit(basismat[((ext.minti-tshift)/tdiv + 1):
                  ((ext.maxti-tshift)/tdiv + 1),
              ext.minci:ext.maxci],
              pred2.s.spline(seq(ext.minti, ext.maxti, by=tdiv), fveci,
                              gam.s.spline(fveci, Qmati, Rmati),
                              tijs[i, 1:ni]), intercept=F)
coefmat[i, minci:maxci] <- lsfi$coef[(minci-ext.minci+1):
                                       (maxci-ext.minci+1)]
}
list(coefs = coefmat, spar = smoothpar)
}

```

The variables are basically the same as in the previous two functions except that there is no dropping of time t_{ij} so that there is no m_j in the names. The function uses as smoothing parameter the value `smoothpar` that is the value of `grid2` corresponding to the minimum of the cross-validation sum `ss2`; `Shatlami` is the smoother matrix corresponding to `smoothpar`; and `fveci` is the value of the smoothing spline at the data times.

A.3.2 Two Parameter Versions

With two parameters, there is little change from the functions for one parameter. The function `step.V.lam.lto.g1.e` is like `step.lto.g1.e` with the following additions: there is a second component to the grid, `grid.theta` and so `Vhat.l` is the corresponding linear combination of `Vhat` and its median eigenvalue, `med.eig`, multiplied by the identity matrix.

```
step.V.lam.lto.g1.e <- function(datamat, Ndat, ids, Nis, nis, nijs, tijs,
                               vijs, his, grid1, useV, Vhat, nless,
                               grid.theta = seq(0, 1, by = .05),
                               nbasisfns = 21, basisfns = B.spl.18.frdiab,
                               idstring="NIH",
                               datstrings=c("GFR1","GFR2","GFR3","GFR4")) {
  lengr <- length(grid1)
  lengr.theta <- length(grid.theta)
  ss <- matrix(0, nrow = lengr, ncol = lengr.theta)

  med.eig <- median(Re(eigen(Vhat)$values))
  for (i in 1:Ndat) {
    ni <- nis[i]
    nim1 <- ni - 1
    nim2 <- ni - 2
    Ni <- Nis[i]
    Dmati <- diag(rep(0, Ni))
    Nmati <- matrix(0, nrow=Ni, ncol=ni)
    datamati <- datamat[datamat[,idstring]==ids[i], ]
    datai <- 0
    for (j in 1:ni) {
      nij <- nijs[i, j]
      nlij <- nless[i, j]
      datai <- c(datai, vok(datamati[j, datstrings]))
      Dmati[nlij + 1:nij, nlij + 1:nij] <- diag(rep(vijs[i, j], nij))
      Nmati[nlij + 1:nij, j] <- 1
    }
    datai <- datai[-1]
    Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
    for (j in 1:ni) {
```

```

    bvecij <- basisfns(tijs[i, j])
    for (k in 1:nijs[i, j]) Xi[nless[i, j] + k, ] <- bvecij
  }
for (j in 1:ni) {
  k4 <- nless[i, j]
  nij <- nijs[i, j]
  Nmatimj <- Nmati[-(k4 + 1:nij), -j]          # (Ni-nij)x(ni-1)
          # drop rows AND column corresponding to tij
  multmj <- 1 / Dmati[k4 + 1, k4 + 1]

  tijsmj <- (tijs[i, 1:ni])[-j]
  if (j == 1) hismj <- his[i, 2:nim1] else {
    if (j == ni) hismj <- his[i, 1:nim2] else {
      hismj <- tijsmj[2:nim1] - tijsmj[1:nim2]
    }
  }
}
Qmatimj <- matrix(0, nrow=nim1, ncol = nim1)
          # will drop 2 columns below
for (k in 2:nim2) {
  Qmatimj[k-1,k] <- 1 / hismj[k-1]
  Qmatimj[k,k]   <- - 1 / hismj[k-1] - 1 / hismj[k]
  Qmatimj[k+1,k] <- 1 / hismj[k]
}
Qmatimj <- Qmatimj[, 2:nim2]                # (ni - 1) x (ni - 3)
Rmatimj <- matrix(0, nrow=nim1, ncol = nim1) # will reduce below
for (k in 2:(nim2)) Rmatimj[k, k] <- (hismj[k-1] + hismj[k]) / 3
for (k in 2:(ni-3)) Rmatimj[k, k+1] <- Rmatimj[k+1, k] <- hismj[k]/6
Rmatimj <- Rmatimj[2:(nim2), 2:(nim2)]      # (ni - 3) x (ni - 3)
Kmatimj <- Qmatimj %%% solve(Rmatimj) %%% t(Qmatimj) # (ni-1) x (ni-1)

for (l in 1:lengr.theta) {
  Vhat.l <- med.eig * (grid.theta[l] * Vhat / med.eig +
                     (1 - grid.theta[l]) * diag(nrow(Vhat)))
  if (useV==F) Vmati <- Dmati else {
    Vmati <- Xi %%% Vhat.l %%% t(Xi) + Dmati
  }
  Vmatimjinv <- solve(Vmati[-(k4 + 1:nij), -(k4 + 1:nij)])
          # drop rows and columnS corresponding to tij (Ni-nij)x(Ni-nij)
  Mat1mj <- t(Nmatimj) %%% Vmatimjinv %%% Nmatimj   # (ni-1)x(ni-1)
  Mat2mj <- t(Nmatimj) %%% Vmatimjinv              # (ni-1)x(Ni-nij)
  Mat3mj <- Mat2mj %%% datai[-(k4 + 1:nij) ]      # (ni - 1)x(1)

  # THE ABOVE MATRICES DON'T CHANGE WITHIN ANY GROUP OF nij OBS.

```

```

for (m in 1:lengr) {
  fhatimjlamm <- solve.eig(Mat1mj + grid1[m] * Kmatimj) %% Mat3mj
  gamimjlamm <- gam.s.spline(fhatimjlamm, Qmatimj, Rmatimj)
  # above are fit and second derivative at ni-1 times
  fhatijmjlamm <-
    pred2.s.spline(tijs[i, j], fhatimjlamm, gamimjlamm, tijsmj)
    # fit at time tij leaving out nij observations at tij
  ss[m, l] <- ss[m, l] + multmj *
    sum(square(datai[(k4 + 1:nij)] - fhatijmjlamm))
}
}
}
}
list(grid = grid1, grid.theta = grid.theta, cvss = ss)
}

```

Note that in the above, m indexes values of λ , as before, while l indexes values of θ , and that we use `ss[m, l]`.

Function `step.V.lam.lto.g2.e` takes input as from above and is otherwise like function `step.V.lam.lto.g1.e` except that it finds the minimum value of the cross-validation sum and then sets up a new grid of size `lengr2` by `lengr.theta2` about the corresponding values of `grid1` and `grid.theta1`, making sure to stay within the bounds of the previous grid.

```

step.V.lam.lto.g2.e <- function(grid1, grid.theta1, ss1, datamat, Ndat,
                                ids, Nis, nis, nijs, tijs, vijs, his,
                                useV, Vhat, nless, nbasisfns = 21,
                                basisfns = B.spl.18.frdiab, lengr2 = 40,
                                lengr.theta2 = 11, idstring="NIH",
                                datstrings=c("GFR1","GFR2","GFR3","GFR4")) {
  minss1 <- min(ss1)
  lengr1 <- length(grid1)
  indmin.lam <- 0
  for (m1 in 1:lengr1) {
    if (min(ss1[m1, ]) == minss1) indmin.lam <- m1
  }
}

```

```

lengr.theta1 <- length(grid.theta1)
indmin.theta <- 0
for (l1 in 1:lengr.theta1) {
  if (min(ss1[, l1]) == minss1) indmin.theta <- l1
}
ind1m <- max(1, indmin.lam - 1)
ind2m <- min(lengr1, indmin.lam + 1)
grid2 <- seq(grid1[ind1m], grid1[ind2m], length = lengr2)
ind1l <- max(1, indmin.theta - 1)
ind2l <- min(lengr.theta1, indmin.theta + 1)
grid.theta2 <- seq(grid.theta1[ind1l], grid.theta1[ind2l],
                  length = lengr.theta2)
ss2 <- matrix(0, nrow = lengr2, ncol = lengr.theta2)

med.eig <- median(Re(eigen(Vhat)$values))
for (i in 1:Ndat) {
  ni <- nis[i]
  nim1 <- ni - 1
  nim2 <- ni - 2
  Ni <- Nis[i]
  Dmati <- diag(rep(0, Ni))
  Nmati <- matrix(0, nrow=Ni, ncol=ni)
  datamati <- datamat[datamat[,idstring]==ids[i], ]
  datai <- 0
  for (j in 1:ni) {
    nij <- nijs[i, j]
    nlij <- nless[i, j]
    datai <- c(datai, vok(datamati[j, datstrings]))
    Dmati[nlij + 1:nij, nlij + 1:nij] <- diag(rep(vijs[i, j], nij))
    Nmati[nlij + 1:nij, j] <- 1
  }
  datai <- datai[-1]
  Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
  for (j in 1:ni) {
    bvecij <- basisfns(tijs[i, j])
    for (k in 1:nijs[i, j]) Xi[nless[i, j] + k, ] <- bvecij
  }
  for (j in 1:ni) {
    k4 <- nless[i, j]
    nij <- nijs[i, j]
    Nmatimj <- Nmati[-(k4 + 1:nij), -j] # (Ni-nij)x(ni-1)
    # drop rows AND column corresponding to tij
    multmj <- 1 / Dmati[k4 + 1, k4 + 1]
  }
}

```

```

tjismj <- (tjjs[i, 1:ni])[-j]
if (j == 1) hismj <- his[i, 2:nim1] else {
  if (j == ni) hismj <- his[i, 1:nim2] else {
    hismj <- tjismj[2:nim1] - tjismj[1:nim2]
  }
}
}
Qmatimj <- matrix(0, nrow=nim1, ncol = nim1)
# will drop 2 columns below
for (k in 2:nim2) {
  Qmatimj[k-1,k] <- 1 / hismj[k-1]
  Qmatimj[k,k] <- - 1 / hismj[k-1] - 1 / hismj[k]
  Qmatimj[k+1,k] <- 1 / hismj[k]
}
}
Qmatimj <- Qmatimj[, 2:nim2] # (ni - 1) x (ni - 3)
Rmatimj <- matrix(0, nrow=nim1, ncol = nim1) # will reduce below
for (k in 2:(nim2)) Rmatimj[k, k] <- (hismj[k-1] + hismj[k]) / 3
for (k in 2:(ni-3)) Rmatimj[k, k+1] <- Rmatimj[k+1, k] <- hismj[k]/6
Rmatimj <- Rmatimj[2:(nim2), 2:(nim2)] # (ni - 3) x (ni - 3)
Kmatimj <- Qmatimj %*% solve(Rmatimj) %*% t(Qmatimj) # (ni-1) x (ni-1)

for (l in 1:lengr.theta2) {
  Vhat.l <- med.eig * (grid.theta2[l] * Vhat / med.eig +
    (1 - grid.theta2[l]) * diag(nrow(Vhat)))
  if (useV==F) Vmati <- Dmati else {
    Vmati <- Xi %*% Vhat.l %*% t(Xi) + Dmati
  }
}
Vmatimjinv <- solve(Vmati[-(k4 + 1:nij), -(k4 + 1:nij)])
# drop rows and columns corresponding to tij (Ni-nij)x(Ni-nij)
Mat1mj <- t(Nmatimj) %*% Vmatimjinv %*% Nmatimj # (ni-1)x(ni-1)
Mat2mj <- t(Nmatimj) %*% Vmatimjinv # (ni-1)x(Ni-nij)
Mat3mj <- Mat2mj %*% datai[-(k4 + 1:nij) ] # (ni - 1)x(1)

# THE ABOVE MATRICES DON'T CHANGE WITHIN ANY GROUP OF nij OBS.
for (m in 1:lengr2) {
  fhatimjlamm <- solve.eig(Mat1mj + grid2[m] * Kmatimj) %*% Mat3mj
  gamimjlamm <- gam.s.spline(fhatimjlamm, Qmatimj, Rmatimj)
  # above are fit and second derivative at ni-1 times
  fhatijmjlamm <-
    pred2.s.spline(tjjs[i, j], fhatimjlamm, gamimjlamm, tjismj)
  # fit at time tij leaving out nij observations at tij
  ss2[m, l] <- ss2[m, l] + multmj *
    sum(square(datai[(k4 + 1:nij)] - fhatijmjlamm))
}

```

```

    }
  }
}
list(grid = grid2, grid.theta = grid.theta2, cvss = ss2)
}

```

Function `step.V.lam.extbest` takes output from above and is otherwise like function `step.extbest`.

```

step.V.lam.extbest <- function(grid2, grid.theta2, ss2, datamat, Ndat,
                              ids, Nis, nis, nijs, tijs, vijs, his,
                              useV, Vhat, nless, tlims, clims,
                              ext.tlims, ext.clims,
                              basismat = pima.B.spline.mat.18.frd,
                              tshift = 72, tdiv = 3, nbasisfns = 21,
                              basisfns = B.spl.18.frdiab, idstring = "NIH",
                              datstrings=c("GFR1","GFR2","GFR3","GFR4")) {
  coefmat <- matrix(NA, nrow = Ndat, ncol = nbasisfns)

  minss2 <- min(ss2)
  lengr2 <- length(grid2)
  indmin.lam <- 0
  for (m2 in 1:lengr2) {
    if (min(ss2[m2, ]) == minss2) indmin.lam <- m2
  }
  lengr.theta2 <- length(grid.theta2)
  indmin.theta <- 0
  for (l2 in 1:lengr.theta2) {
    if (min(ss2[, l2]) == minss2) indmin.theta <- l2
  }
  spar <- grid2[indmin.lam]
  theta <- grid.theta2[indmin.theta]

  med.eig <- median(Re(eigen(Vhat)$values))
  Vhat.l <- med.eig * (theta * Vhat / med.eig +
                     (1 - theta) * diag(nrow(Vhat)))
  for (i in 1:Ndat) {
    ni <- nis[i]
    nim1 <- ni - 1
  }
}

```

```

Ni <- Nis[i]
Dmati <- diag(rep(0, Ni))
Nmati <- matrix(0, nrow = Ni, ncol = ni)
datamati <- datamat[datamat[,idstring]==ids[i], ]
datai <- 0
for (j in 1:ni) {
  datai <- c(datai, vok(datamati[j, datstrings]))
  Dmati[nless[i, j] + 1:nijs[i, j], nless[i, j] + 1:nijs[i, j]] <-
    diag(rep(vijs[i, j], nijs[i, j]))
  Nmati[nless[i, j] + 1:nijs[i, j], j] <- 1
}
datai <- datai[-1]
if (useV==F) Vmati <- Dmati else {
  Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
  for (j in 1:ni) {
    bvecij <- basisfns(tijs[i, j])
    for (k in 1:nijs[i, j]) Xi[nless[i, j] + k, ] <- bvecij
  }
  Vmati <- Xi %*% Vhat.l %*% t(Xi) + Dmati
}
Qmati <- matrix(0, nrow=ni, ncol = ni) # drop 2 columns below
for (k in 2:nim1) {
  Qmati[k-1,k] <- 1 / his[i, k-1]
  Qmati[k,k] <- - 1 / his[i, k-1] - 1 / his[i, k]
  Qmati[k+1,k] <- 1 / his[i, k]
}
Qmati <- Qmati[, 2:nim1] # ni x (ni - 2)
Rmati <- matrix(0, nrow=ni, ncol = ni) # will reduce below
for (k in 2:(nim1)) Rmati[k, k] <- (his[i, k-1] + his[i, k]) / 3
for (k in 2:(ni-2)) Rmati[k, k+1] <- Rmati[k+1, k] <- his[i, k]/6
Rmati <- Rmati[2:(nim1), 2:(nim1)] # (ni - 2) x (ni - 2)
Kmati <- Qmati %*% solve(Rmati) %*% t(Qmati) # ni x ni

Shatlami <- solve(t(Nmati) %*% solve(Vmati) %*% Nmati +
  spar * Kmati) %*% t(Nmati) %*% solve(Vmati)
fveci <- Shatlami %*% datai

ext.minti <- ext.tlims[i, 1]
ext.maxti <- ext.tlims[i, 2]
ext.minci <- ext.clims[i, 1]
ext.maxci <- ext.clims[i, 2]
minci <- clims[i, 1]
maxci <- clims[i, 2]

```

```

lsfi <- lsfit(basismat[((ext.minti-tshift)/tdiv + 1):
                  ((ext.maxti-tshift)/tdiv + 1),
             ext.minci:ext.maxci],
             pred2.s.spline(seq(ext.minti, ext.maxti, by=tdiv), fveci,
                             gam.s.spline(fveci, Qmati, Rmati),
                             tijos[i, 1:ni]), intercept=F)
coefmat[i, minci:maxci] <- lsfi$coef[(minci-ext.minci+1):
                                     (maxci-ext.minci+1)]
}
list(coefs = coefmat, spar = spar, theta = theta)
}

```

Note that if we want to do the shrinkage using the mean eigenvalue rather than the median eigenvalue in the above, we may simply replace `med.eig` by `mean.eig` where `mean.eig <- mean(diag(Vhat))`.

A.4 Smoothing Functions Without Replications

A.4.1 One Parameter Versions

These functions are very similar to those in Section A.3.1. The differences are as follows: since $n_{ij} = 1$ for all i, j , we can use `Nis` alone; there is no incidence matrix; instead of a matrix of `vijos` we have a vector of `vis`. Here, there is no input for the vectors \mathbf{h}_i so `hismj` is calculated (with the j th time left out).

```

step.norep.g1.e <- function(datamat, Ndat, ids, Nis, tijos, vis,
                           grid1, useV, Vhat, nbasisfns = 7,
                           basisfns = B.spl.18.fourtypes,
                           idstring= "ID", datstring= "GFR") {
  lengr <- length(grid1)
  ss <- rep(0, lengr)
  for (i in 1:Ndat) {
    Ni <- Nis[i]
    Nim1 <- Ni - 1

```

```

Nim2 <- Ni - 2
Dmati <- diag(rep(vis[i], Ni))
datamati <- datamat[datamat[,idstring]==ids[i], ]
datai <- datamati[, datstring]
if (useV==F) Vmati <- Dmati else {
  Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
  for (j in 1:Ni) {
    Xi[j, ] <- basisfns(tijs[i, j])
  }
  Vmati <- Xi %*% Vhat %*% t(Xi) + Dmati
}
multi <- 1 / vis[i]

for (j in 1:Ni) {
  Vmatimjinv <- solve(Vmati[-j, -j])           # (Ni-1) x (Ni-1)
  tijsmj <- (tijs[i, 1:Ni])[-j]              # (Ni-1)-vector
  hismj <- tijsmj[2:Nim1] - tijsmj[1:Nim2]    # (Ni-2)-vector

  Qmatimj <- matrix(0, nrow = Nim1, ncol = Nim1)
  for (k in 2:Nim2) {
    Qmatimj[k-1,k] <- 1 / hismj[k-1]
    Qmatimj[k,k] <- - 1 / hismj[k-1] - 1 / hismj[k]
    Qmatimj[k+1,k] <- 1 / hismj[k]
  }
  Qmatimj <- Qmatimj[, -c(1, Nim1)]           # (Ni-1) x (Ni-3)

  Rmatimj <- matrix(0, nrow = Nim1, ncol = Nim1)
  for (k in 2:(Nim2)) Rmatimj[k, k] <- (hismj[k-1] + hismj[k]) / 3
  for (k in 2:(Ni-3)) Rmatimj[k, k+1] <- Rmatimj[k+1, k] <- hismj[k]/6
  Rmatimj <- Rmatimj[-c(1, Nim1), -c(1, Nim1)] # (Ni-3) x (Ni-3)

  Kmatimj <- Qmatimj %*% solve(Rmatimj) %*% t(Qmatimj)
  # (Ni-1) x (Ni-1)

  for (m in 1:lengr) {
    Mat3mj <- Vmatimjinv %*% datai[-j]
    fhatimjlamm <- solve.eig(Vmatimjinv + grid1[m] * Kmatimj) %*% Mat3mj
    gamimjlamm <- gam.s.spline(fhatimjlamm, Qmatimj, Rmatimj)
    fhatijmjlamm <- pred2.s.spline(tijs[i, j], fhatimjlamm,
                                   gamimjlamm, tijsmj)
    ss[m] <- ss[m] + multi * square(datai[j] - fhatijmjlamm)
  }
}

```

```

}
list(grid = grid1, cvss = ss)
}

```

The function `step.norep.g2.e` takes results of a grid search and refines them.

```

step.norep.g2.e <- function(grid1, ss1, datamat, Ndat, ids, Nis, tijs, vis,
                           useV, Vhat, nbasisfns = 7, basisfns = B.spl.18.fourtypes,
                           lengr2 = 40, idstring = "ID", datstring = "GFR") {
  minm <- order(ss1)[1]
  ind1m <- max(1, min(c(minm - 1, order(ss1)[1:3])))
  ind2m <- min(length(ss1), max(c(minm+1, order(ss1)[1:3])))
  grid2 <- seq(grid1[ind1m], grid1[ind2m], length = lengr2)
  ss2 <- rep(0, lengr2)
  for (i in 1:Ndat) {
    Ni <- Nis[i]
    Nim1 <- Ni - 1
    Nim2 <- Ni - 2
    Dmati <- diag(rep(vis[i], Ni))
    datamati <- datamat[datamat[,idstring]==ids[i], ]
    datai <- datamati[, datstring]
    if (useV==F) Vmati <- Dmati else {
      Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
      for (j in 1:Ni) {
        Xi[j, ] <- basisfns(tijs[i, j])
      }
      Vmati <- Xi %*% Vhat %*% t(Xi) + Dmati
    }
    multi <- 1 / vis[i]

    for (j in 1:Ni) {
      Vmatimjinv <- solve(Vmati[-j, -j]) # (Ni-1) x (Ni-1)
      tijsmj <- (tijs[i, 1:Ni])[-j] # (Ni-1)-vector
      hismj <- tijsmj[2:Nim1] - tijsmj[1:Nim2] # (Ni-2)-vector

      Qmatimj <- matrix(0, nrow = Nim1, ncol = Nim1) # drop 2 columns below
      for (k in 2:Nim2) {
        Qmatimj[k-1,k] <- 1 / hismj[k-1]
        Qmatimj[k,k] <- - 1 / hismj[k-1] - 1 / hismj[k]
        Qmatimj[k+1,k] <- 1 / hismj[k]
      }
    }
  }
}

```

```

Qmatimj <- Qmatimj[, -c(1, Nim1)]           # (Ni-1) x (Ni-3)

Rmatimj <- matrix(0, nrow = Nim1, ncol = Nim1) # will reduce below
for (k in 2:(Nim2)) Rmatimj[k, k] <- (hismj[k-1] + hismj[k]) / 3
for (k in 2:(Ni-3)) Rmatimj[k, k+1] <- Rmatimj[k+1, k] <- hismj[k]/6
Rmatimj <- Rmatimj[-c(1, Nim1), -c(1, Nim1)] # (Ni-3) x (Ni-3)

Kmatimj <- Qmatimj %%% solve(Rmatimj) %%% t(Qmatimj) # (Ni-1)x(Ni-1)

for (m in 1:lengr2) {
  Mat3mj <- Vmatimjinv %%% datai[-j]
  fhatimjlamm <- solve.eig(Vmatimjinv + grid2[m] * Kmatimj) %%% Mat3mj
  gamimjlamm <- gam.s.spline(fhatimjlamm, Qmatimj, Rmatimj)
  fhatijmjlamm <- pred2.s.spline(tijs[i, j], fhatimjlamm,
                                gamimjlamm, tijsmj)
  ss2[m] <- ss2[m] + multi * square(datai[j] - fhatijmjlamm)
}
}
}
list(grid = grid2, cvss = ss2)
}

```

```

step.norep.extbest <- function(grid2, ss2, datamat, Ndat, ids, Nis, tijs,
                               vis, his, useV, Vhat, tlims, clims, ext.tlims,
                               ext.clims, basismat = fourtypes.B.spline.mat.18,
                               tshift=0, tdiv = 3,
                               nbasisfns = 7, basisfns = B.spl.18.fourtypes,
                               idstring = "ID", datstring = "GFR") {
coefmat <- matrix(NA, nrow = Ndat, ncol = nbasisfns)
smoothpar <- grid2[order(ss2)[1]]
for (i in 1:Ndat) {
  Ni <- Nis[i]
  Nim1 <- Ni - 1
  Dmati <- diag(rep(vis[i], Ni))
  datamati <- datamat[datamat[, idstring] == ids[i], ]
  datai <- datamati[, datstring]
  if (useV==F) Vmati <- Dmati else {
    Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
    for (j in 1:Ni) {
      Xi[j, ] <- basisfns(tijs[i, j])
    }
  }
}

```

```

    Vmati <- Xi %*% Vhat %*% t(Xi) + Dmati
  }
  multi <- 1 / vis[i]

  Qmati <- matrix(0, nrow = Ni, ncol = Ni) # will drop 2 columns below
  for (k in 2:Nim1) {
    Qmati[k-1,k] <- 1 / his[i, k-1]
    Qmati[k,k] <- - 1 / his[i, k-1] - 1 / his[i, k]
    Qmati[k+1,k] <- 1 / his[i, k]
  }
  Qmati <- Qmati[, 2:Nim1] # Ni x (Ni - 2)
  Rmati <- matrix(0, nrow = Ni, ncol = Ni) # will reduce below
  for (k in 2:(Nim1)) Rmati[k, k] <- (his[i, k-1] + his[i, k]) / 3
  for (k in 2:(Ni-2)) Rmati[k, k+1] <- Rmati[k+1, k] <- his[i, k]/6
  Rmati <- Rmati[2:(Nim1), 2:(Nim1)] # (Ni - 2) x (Ni - 2)
  Kmati <- Qmati %*% solve(Rmati) %*% t(Qmati) # Ni x Ni

  Vmatiinv <- solve(Vmati)
  Shatlami <- solve(Vmatiinv + smoothpar * Kmati) %*% Vmatiinv
  fveci <- Shatlami %*% datai

  ext.minti <- ext.tlims[i, 1]
  ext.maxti <- ext.tlims[i, 2]
  ext.minci <- ext.clims[i, 1]
  ext.maxci <- ext.clims[i, 2]
  minci <- clims[i, 1]
  maxci <- clims[i, 2]
  lsfi <- lsfit(basismat[((ext.minti-tshift)/tdiv + 1):
                ((ext.maxti-tshift)/tdiv + 1),
                ext.minci:ext.maxci],
                pred2.s.spline(seq(ext.minti, ext.maxti, by=tdiv), fveci,
                                gam.s.spline(fveci, Qmati, Rmati),
                                tijs[i, 1:Ni]), intercept=F)
  coefmat[i, minci:maxci] <- lsfi$coef[(minci-ext.minci+1):
                                       (maxci-ext.minci+1)]
}
list(coefs = coefmat, spar = smoothpar)
}

```

A.4.2 Two Parameter Versions

The functions here bear the same relationship to the one parameter versions as do the corresponding functions in the case with replicated data. Function `step.V.lam.norep.g1.e` evaluates the cross-validation sum on grids `grid1` and `grid.theta`.

```
step.V.lam.norep.g1.e <- function(datamat, Ndat, ids, Nis, tijs, vis, grid1,
                                useV, Vhat, grid.theta = seq(0, 1, by = .05),
                                nbasisfns = 7, basisfns = B.spl.18.fourtypes,
                                idstring= "ID", datstring= "GFR") {
  lengr <- length(grid1)
  lengr.theta <- length(grid.theta)
  ss <- matrix(0, nrow = lengr, ncol = lengr.theta)

  med.eig <- median(Re(eigen(Vhat)$values))
  for (i in 1:Ndat) {
    Ni <- Nis[i]
    Nim1 <- Ni - 1
    Nim2 <- Ni - 2
    Dmati <- diag(rep(vis[i], Ni))
    datamati <- datamat[datamat[,idstring]==ids[i], ]
    datai <- datamati[, datstring]
    Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
    for (j in 1:Ni) {
      Xi[j, ] <- basisfns(tijs[i, j])
    }
    multi <- 1 / vis[i]

    for (j in 1:Ni) {
      tijsmj <- (tijs[i, 1:Ni])[-j]           # (Ni-1)-vector
      hismj <- tijsmj[2:Nim1] - tijsmj[1:Nim2] # (Ni-2)-vector

      Qmatimj <- matrix(0, nrow = Nim1, ncol = Nim1)
      for (k in 2:Nim2) {
        Qmatimj[k-1,k] <- 1 / hismj[k-1]
        Qmatimj[k,k] <- - 1 / hismj[k-1] - 1 / hismj[k]
        Qmatimj[k+1,k] <- 1 / hismj[k]
      }
    }
  }
}
```

```

Qmatimj <- Qmatimj[, -c(1, Nim1)]           # (Ni-1) x (Ni-3)

Rmatimj <- matrix(0, nrow = Nim1, ncol = Nim1)
for (k in 2:(Nim2)) Rmatimj[k, k] <- (hismj[k-1] + hismj[k]) / 3
for (k in 2:(Ni-3)) Rmatimj[k, k+1] <- Rmatimj[k+1, k] <- hismj[k]/6
Rmatimj <- Rmatimj[-c(1, Nim1), -c(1, Nim1)] # (Ni-3) x (Ni-3)

Kmatimj <- Qmatimj %%% solve(Rmatimj) %%% t(Qmatimj)
# (Ni-1) x (Ni-1)

for (l in 1:lengr.theta) {
  Vhat.l <- med.eig * (grid.theta[l] * Vhat / med.eig +
                      (1 - grid.theta[l]) * diag(nrow(Vhat)))
  if (useV==F) Vmati <- Dmati else {
    Vmati <- Xi %%% Vhat.l %%% t(Xi) + Dmati
  }
  Vmatimjinv <- solve(Vmati[-j, -j])       # (Ni-1) x (Ni-1)
  Mat3mj <- Vmatimjinv %%% datai[-j]

  for (m in 1:lengr) {
    fhatimjlamm <- solve.eig(Vmatimjinv + grid1[m] * Kmatimj) %%% Mat3mj
    gamimjlamm <- gam.s.spline(fhatimjlamm, Qmatimj, Rmatimj)
    fhatijmjlamm <- pred2.s.spline(tijs[i, j], fhatimjlamm,
                                  gamimjlamm, tijsmj)
    ss[m, l] <- ss[m, l] + multi * square(datai[j] - fhatijmjlamm)
  }
}
}
}
list(grid = grid1, grid.theta = grid.theta, cvss = ss)
}

```

Function `step.V.lam.norep.g2.e` takes a previous set of grid values and cross-validation sums and sets up a finer grid on which it evaluates the new cross-validation sums.

```

step.V.lam.norep.g2.e <- function(grid1, grid.theta1, ss1, datamat, Ndat,
                                ids, Nis, tijs, vis, useV, Vhat,
                                nbasisfns = 7, basisfns = B.spl.18.fourtypes,

```

```

                                lengr2 = 40, lengr.theta2 = 11,
                                idstring = "ID", datstring = "GFR") {
minss1 <- min(ss1)
lengr1 <- length(grid1)
indmin.lam <- 0
for (m1 in 1:lengr1) {
  if (min(ss1[m1, ]) == minss1) indmin.lam <- m1
}
lengr.theta1 <- length(grid.theta1)
indmin.theta <- 0
for (l1 in 1:lengr.theta1) {
  if (min(ss1[, l1]) == minss1) indmin.theta <- l1
}
ind1m <- max(1, indmin.lam - 1)
ind2m <- min(lengr1, indmin.lam + 1)
grid2 <- seq(grid1[ind1m], grid1[ind2m], length = lengr2)
ind1l <- max(1, indmin.theta - 1)
ind2l <- min(lengr.theta1, indmin.theta + 1)
grid.theta2 <- seq(grid.theta1[ind1l], grid.theta1[ind2l],
                  length = lengr.theta2)
ss2 <- matrix(0, nrow = lengr2, ncol = lengr.theta2)

med.eig <- median(Re(eigen(Vhat)$values))
for (i in 1:Ndat) {
  Ni <- Nis[i]
  Nim1 <- Ni - 1
  Nim2 <- Ni - 2
  Dmati <- diag(rep(vis[i], Ni))
  datamati <- datamat[datamat[,idstring]==ids[i], ]
  datai <- datamati[, datstring]
  Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
  for (j in 1:Ni) {
    Xi[j, ] <- basisfns(tijs[i, j])
  }
  multi <- 1 / vis[i]

  for (j in 1:Ni) {
    tijsmj <- (tijs[i, 1:Ni])[-j] # (Ni-1)-vector
    hismj <- tijsmj[2:Nim1] - tijsmj[1:Nim2] # (Ni-2)-vector

    Qmatimj <- matrix(0, nrow = Nim1, ncol = Nim1)
    for (k in 2:Nim2) {
      Qmatimj[k-1,k] <- 1 / hismj[k-1]
    }
  }
}

```

```

    Qmatimj[k,k] <- - 1 / hismj[k-1] - 1 / hismj[k]
    Qmatimj[k+1,k] <- 1 / hismj[k]
  }
  Qmatimj <- Qmatimj[, -c(1, Nim1)] # (Ni-1) x (Ni-3)

  Rmatimj <- matrix(0, nrow = Nim1, ncol = Nim1)
  for (k in 2:(Nim2)) Rmatimj[k, k] <- (hismj[k-1] + hismj[k]) / 3
  for (k in 2:(Ni-3)) Rmatimj[k, k+1] <- Rmatimj[k+1, k] <- hismj[k]/6
  Rmatimj <- Rmatimj[-c(1, Nim1), -c(1, Nim1)] # (Ni-3) x (Ni-3)

  Kmatimj <- Qmatimj %%% solve(Rmatimj) %%% t(Qmatimj)
  # (Ni-1) x (Ni-1)

  for (l in 1:lengr.theta2) {
    Vhat.l <- med.eig * (grid.theta2[l] * Vhat / med.eig +
                        (1 - grid.theta2[l]) * diag(nrow(Vhat)))
    if (useV==F) Vmati <- Dmati else {
      Vmati <- Xi %%% Vhat.l %%% t(Xi) + Dmati
    }
    Vmatimjinv <- solve(Vmati[-j, -j]) # (Ni-1) x (Ni-1)
    Mat3mj <- Vmatimjinv %%% datai[-j]

    for (m in 1:lengr2) {
      fhatimjlamm <- solve.eig(Vmatimjinv + grid2[m] * Kmatimj) %%% Mat3mj
      gamimjlamm <- gam.s.spline(fhatimjlamm, Qmatimj, Rmatimj)
      fhatijmjlamm <- pred2.s.spline(tijs[i, j], fhatimjlamm,
                                   gamimjlamm, tijsmj)
      ss2[m, l] <- ss2[m, l] + multi * square(datai[j] - fhatijmjlamm)
    }
  }
}
list(grid = grid2, grid.theta = grid.theta2, cvss = ss2)
}

```

Function `step.V.lam.norep.extbest.e` takes input of `grid` and `sum` from one of the previous two functions and finds the parameters corresponding to the minimum sum. It then estimates the coefficients for that pair of parameters.

```
step.V.lam.norep.extbest.e <- function(grid2, grid.theta2, ss2,
```

```

                                datamat, Ndat, ids, Nis, tijs,
                                vis, his, useV, Vhat,
                                tlims, clims, ext.tlims, ext.clims,
                                basismat = fourtypes.B.spline.mat.18,
                                tshift = 0, tdiv = 3, nbasisfns = 7,
                                basisfns = B.spl.18.fourtypes,
                                idstring = "ID", datstring = "GFR") {
coefmat <- matrix(NA, nrow = Ndat, ncol = nbasisfns)

minss2 <- min(ss2)
lengr2 <- length(grid2)
indmin.lam <- 0
for (m2 in 1:lengr2) {
  if (min(ss2[m2, ]) == minss2) indmin.lam <- m2
}
lengr.theta2 <- length(grid.theta2)
indmin.theta <- 0
for (l2 in 1:lengr.theta2) {
  if (min(ss2[, l2]) == minss2) indmin.theta <- l2
}
spar <- grid2[indmin.lam]
theta <- grid.theta2[indmin.theta]

med.eig <- median(Re(eigen(Vhat)$values))
Vhat.l <- med.eig * (theta * Vhat / med.eig +
                    (1 - theta) * diag(nrow(Vhat)))

for (i in 1:Ndat) {
  Ni <- Nis[i]
  Nim1 <- Ni - 1
  Dmati <- diag(rep(vis[i], Ni))
  datamati <- datamat[datamat[, idstring] == ids[i], ]
  datai <- datamati[, datstring]
  if (useV==F) Vmati <- Dmati else {
    Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
    for (j in 1:Ni) {
      Xi[j, ] <- basisfns(tijs[i, j])
    }
    Vmati <- Xi %*% Vhat.l %*% t(Xi) + Dmati
  }
}

Qmati <- matrix(0, nrow = Ni, ncol = Ni)
for (k in 2:Nim1) {

```

```

    Qmati[k-1,k] <- 1 / his[i, k-1]
    Qmati[k,k]    <- - 1 / his[i, k-1] - 1 / his[i, k]
    Qmati[k+1,k] <- 1 / his[i, k]
  }
  Qmati <- Qmati[, 2:Nim1]          # Ni x (Ni - 2)
  Rmati <- matrix(0, nrow = Ni, ncol = Ni)
  for (k in 2:(Nim1)) Rmati[k, k] <- (his[i, k-1] + his[i, k]) / 3
  for (k in 2:(Ni-2)) Rmati[k, k+1] <- Rmati[k+1, k] <- his[i, k]/6
  Rmati <- Rmati[2:(Nim1), 2:(Nim1)] # (Ni - 2) x (Ni - 2)
  Kmati <- Qmati %*% solve(Rmati) %*% t(Qmati) # Ni x Ni

  Vmatiinv <- solve(Vmati)
  Shatlami <- solve.eig(Vmatiinv + spar * Kmati) %*% Vmatiinv
  fveci <- Shatlami %*% datai

  ext.minti <- ext.tlims[i, 1]
  ext.maxti <- ext.tlims[i, 2]
  ext.minci <- ext.clims[i, 1]
  ext.maxci <- ext.clims[i, 2]
  minci <- clims[i, 1]
  maxci <- clims[i, 2]
  lsfi <- lsfit(basismat[((ext.minti-tshift)/tdiv + 1):
                  ((ext.maxti-tshift)/tdiv + 1),
                ext.minci:ext.maxci],
                pred2.s.spline(seq(ext.minti, ext.maxti, by=tdiv), fveci,
                                gam.s.spline(fveci, Qmati, Rmati),
                                tijs[i, 1:Ni]), intercept=F)
  coefmat[i, minci:maxci] <- lsfi$coef[(minci-ext.minci+1):
                                       (maxci-ext.minci+1)]
}
list(coefs = coefmat, spar = spar, theta = theta)
}

```

As before, it would be simple to change the code to use the mean eigenvalue rather than the median eigenvalue.

A.5 Supplemental Code

The following code corresponds to functions that are discussed as options in the main body of this dissertation but that are not demonstrated there.

A.5.1 Leaving Out One Observation

Here, for the case with replications, we leave out one observation at a time for the cross-validation, rather than leaving out all of the observations at one time. In this case, a number of the matrices do not change as individual observations at one time are left out in turn. (In particular, \mathbf{h} , \mathbf{Q} , \mathbf{R} and \mathbf{K} do not change.) The functions and arguments are otherwise similar to those when leaving out one time instead of one observation. We only include the code for the one parameter versions. First is code for first cross-validated grid search for smoothing parameter, with replications, over fixed grid:

```
step.g1 <- function(datamat, Ndat, ids, Nis, nis, nijs, tijs, vijs,
                    his, grid1, useV, Vhat, nless, nbasisfns = 21,
                    basisfns = B.spl.18.frdiab, idstring = "NIH",
                    datstrings = c("GFR1","GFR2","GFR3","GFR4")) {
  lengr <- length(grid1)
  ss <- rep(0, lengr)
  for (i in 1:Ndat) {
    ni <- nis[i]
    nim1 <- ni - 1
    Ni <- Nis[i]
    Dmati <- diag(rep(0, Ni))
    Nmati <- matrix(0, nrow=Ni, ncol=ni)
    datamati <- datamat[datamat[,idstring]==ids[i], ]
    datai <- 0
    for (j in 1:ni) {
      datai <- c(datai, vok(datamati[j, datstrings]))
      Dmati[nless[i, j] + 1:nijs[i, j], nless[i, j] + 1:nijs[i, j]] <-
```

```

        diag(rep(vijs[i, j], nijs[i, j]))
    Nmati[nless[i, j] + 1:nijs[i, j], j] <- 1
}
datai <- datai[-1]
if (useV==F) Vmati <- Dmati else {
    Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
    for (j in 1:ni) {
        bvecij <- basisfns(tijs[i, j])
        for (k in 1:nijs[i, j]) Xi[nless[i, j] + k, ] <- bvecij
    }
    Vmati <- Xi %%% Vhat %%% t(Xi) + Dmati
}
Qmati <- matrix(0, nrow=ni, ncol = ni) # will drop 2 columns below
for (k in 2:nim1) {
    Qmati[k-1,k] <- 1 / his[i, k-1]
    Qmati[k,k] <- - 1 / his[i, k-1] - 1 / his[i, k]
    Qmati[k+1,k] <- 1 / his[i, k]
}
Qmati <- Qmati[, 2:nim1] # ni x (ni - 2)
Rmati <- matrix(0, nrow=ni, ncol = ni) # will reduce below
for (k in 2:(nim1)) Rmati[k, k] <- (his[i, k-1] + his[i, k]) / 3
for (k in 2:(ni-2)) Rmati[k, k+1] <- Rmati[k+1, k] <- his[i, k]/6
Rmati <- Rmati[2:(nim1), 2:(nim1)] # (ni - 2) x (ni - 2)
Kmati <- Qmati %%% solve(Rmati) %%% t(Qmati) # ni x ni

for (j in 1:ni) {
    k4 <- nless[i, j]
    Nmatimk <- Nmati[-(k4 + 1), ] # (Ni-1)x(ni)
    Vmatimkinv <- solve(Vmati[-(k4 + 1), -(k4 + 1)]) # (Ni-1)x(Ni-1)
    Mat1mk <- t(Nmatimk) %%% Vmatimkinv %%% Nmatimk # (ni)x(ni)
    Mat2mk <- t(Nmatimk) %%% Vmatimkinv # (ni)x(Ni-1)
    multmk <- 1 / Dmati[k4 + 1, k4 + 1]
    # THE ABOVE MATRICES DON'T CHANGE WITHIN ANY GROUP OF nij OBSERVATIONS.
    for (kp in 1:nijs[i, j]) {
        Mat3mk <- Mat2mk %%% datai[-(k4 + kp)] # (ni)x(1)
        for (m in 1:lengr) {
            ss[m] <- ss[m] + multmk *
                square( datai[(k4 + kp)] -
                    (solve(Mat1mk + grid1[m] * Kmati) %%% Mat3mk)[j, 1] )
        }
    }
}
}
}
}

```

```

  list(grid = grid1, cvss = ss)
}

```

We have similar code for refining the grid and for finding the minimum and estimating the coefficients.

A.5.2 Weighted Gap Leaving Out One Observation

The following is a modification of the above in which there are additional arguments `gap`, a Boolean variable, and `gaploc` for the modification of the cross-validation where, if `gap=T`, in addition to leaving out one observation at a time one fits the spline using all of the observations for an individual and then adds to the cross-validation sum the square of the difference between this fit and a pseudo-observation located `gaploc` units from the start for the individual.

```

step.gap.g1 <- function(datamat, Ndat, ids, Nis, nis, nijs, tijs, vijs,
                        his, grid1, useV, Vhat, nless, nbasisfns=21,
                        basisfns=B.spl.18.frdiab, idstring="NIH",
                        datstrings=c("GFR1","GFR2","GFR3","GFR4"),
                        gap=F, gaploc=54) {
  lengr <- length(grid1)
  ss <- rep(0, lengr)
  for (i in 1:Ndat) {
    ni <- nis[i]
    nim1 <- ni - 1
    Ni <- Nis[i]
    Dmati <- diag(rep(0, Ni))
    Nmati <- matrix(0, nrow=Ni, ncol=ni)
    datamati <- datamat[datamat[,idstring]==ids[i], ]
    datai <- 0
    for (j in 1:ni) {
      datai <- c(datai, vok(datamati[j, datstrings]))
      Dmati[nless[i, j] + 1:nijs[i, j], nless[i, j] + 1:nijs[i, j]] <-
        diag(rep(vijs[i, j], nijs[i, j]))
    }
  }
}

```

```

    Nmati[nless[i, j] + 1:nijs[i, j], j] <- 1
  }
  datai <- datai[-1]
  if (useV==F) Vmati <- Dmati else {
    Xi <- matrix(0, nrow = Ni, ncol = nbasisfns)
    for (j in 1:ni) {
      bvecij <- basisfns(tijs[i, j])
      for (k in 1:nijs[i, j]) Xi[nless[i, j] + k, ] <- bvecij
    }
    Vmati <- Xi %*% Vhat %*% t(Xi) + Dmati
  }
  Qmati <- matrix(0, nrow=ni, ncol = ni) # will drop 2 columns below
  for (k in 2:nim1) {
    Qmati[k-1,k] <- 1 / his[i, k-1]
    Qmati[k,k] <- - 1 / his[i, k-1] - 1 / his[i, k]
    Qmati[k+1,k] <- 1 / his[i, k]
  }
  Qmati <- Qmati[, 2:nim1] # ni x (ni - 2)
  Rmati <- matrix(0, nrow=ni, ncol = ni) # will reduce below
  for (k in 2:(nim1)) Rmati[k, k] <- (his[i, k-1] + his[i, k]) / 3
  for (k in 2:(ni-2)) Rmati[k, k+1] <- Rmati[k+1, k] <- his[i, k]/6
  Rmati <- Rmati[2:(nim1), 2:(nim1)] # (ni - 2) x (ni - 2)
  Kmati <- Qmati %*% solve(Rmati) %*% t(Qmati) # ni x ni

  for (j in 1:ni) {
    k4 <- nless[i, j]
    Nmatimk <- Nmati[-(k4 + 1), ] # (Ni-1)x(ni)
    Vmatimkinv <- solve(Vmati[-(k4 + 1), -(k4 + 1)]) # (Ni-1)x(Ni-1)
    Mat2mk <- t(Nmatimk) %*% Vmatimkinv # (ni)x(Ni-1)
    Mat1mk <- Mat2mk %*% Nmatimk # (ni)x(ni)
    multmk <- 1 / Dmati[k4 + 1, k4 + 1]
    # THE ABOVE MATRICES DON'T CHANGE WITHIN ANY GROUP OF nij OBSERVATIONS.
    for (kp in 1:nijs[i, j]) {
      Mat3mk <- Mat2mk %*% datai[-(k4 + kp)] # (ni)x(1)
      for (m in 1:lengr) {
        ss[m] <- ss[m] + multmk *
          square( datai[(k4 + kp)] -
            (solve(Mat1mk + grid1[m] * Kmati) %*% Mat3mk)[j, 1] )
      }
    }
  }
}
if (gap==T && (max(tijs[i,1:ni]) - min(tijs[i,1:ni]) > gaploc)) {
  Mat2 <- t(Nmati) %*% solve(Vmati)
}

```

```

Mat1 <- Mat2 %*% Nmati
Ddiv <- median(vijs[i, 1:ni])
lastprev <- sum(tijs[i, 1:ni] - tijs[i, 1] < gaploc)
midgap <- mean(tijs[i, lastprev:(lastprev+1)])
premean <- mean(vok(datamati[lastprev, datstrings]))
postmean <- mean(vok(datamati[lastprev + 1, datstrings]))
midmean <- (premean + postmean) / 2
for (m in 1:lengr) {
  Shatlamim <- solve(Mat1 + grid1[m] * Kmati) %*% Mat2
  fvecim <- Shatlamim %*% datai
  gamim <- gam.s.spline(fvecim, Qmati, Rmati)
  midpredm <- pred2.s.spline(midgap, fvecim, gamim, tijs[i, 1:ni])
  ss[m] <- ss[m] + square(midmean - midpredm) / Ddiv
}
}
}
list(grid = grid1, cvss = ss)
}

```

The last section of the code deals with the pseudo-observation. If `gap==T` and there is data for the individual beyond `gaploc` units from the start of the data, then the psuedo-value is obtained by averaging the mean of the last observations before the gap and the first observations after the gap.

There is a similar function for refining the grid. There is no need for an additional function for the final fit since the output of `grid` and cross-validation sum can simply be input into the version without the weighted gap.