```
# In order for the program to run properly, please make sure you have
installed
# the gss package version 1.1-5 (downloadable from the archive of the
package).
# NOTE: A newer version of gss may not be compatible with this program
library(gss,lib.loc="~/bin/gss_1.1-5/") # gss_1.1-5
#library(gss)

partK <- function(x1,x2)
{# x1 and x2 vectors of same length
 # return value = k2(x1)*k2(x2)-k4(abs(x1 - x2)).
  val <- ((x1-0.5)^2-1/12)/2*((x2-0.5)^2-1/12)/2
  wk <- abs(x1-x2)
  val <- val-((wk-0.5)^4-(wk-0.5)^2/2+7/240)/24
}

# generalized functional linear regression model through penalty approach.
# assume the domain of x to be [0,1].
# Inputs:
#  y - dim(n*2) matrix or length n vector of responses.
#  xqmat - nmesh*n matrix with xqmat[j,i]=x_i(p_j), p_j are quadrature
nodes.
#  family - regression family including "binomial" and "poisson". For
"binomial"
#           the response enters as two columns of counts.
#  quad - quadrature to be used for integration, MUST BE USED FOR ALL
INTEGRALS.
#  id.basis - index vector for selected "knots" (x's).
#  lams - grid for log10(lambda)
#  xdomain - default is [0,1].
#  alpha - alpha used in the GACV score for smoothing parameter selection
gflr <- function
(y,xqmat,family,quad,id.basis=NULL,lams=seq(-3,-1,len=101),xdomain=c(0,
1),alpha=NULL)
{
  if (is.null(alpha)) {
    alpha <- 1.4
    if (family == "binomial")
      alpha <- 1
  }
```

```
  nobs <- dim(xqmat)[2]
  qpt <- (quad$pt-xdomain[1])/(xdomain[2]-xdomain[1])
  Kmat <- outer(qpt,qpt,partK)
  Kxmat <- Kmat%*%(quad$wt*xqmat)
  s <- rep(1,nobs)
  s <-
cbind(s,apply(quad$wt*xqmat,2,sum),apply(quad$wt*quad$pt*xqmat,2,su
m))
  q <- r <- t(Kxmat)%*%(quad$wt*xqmat)
  wt <- offset <- NULL
  nu.wk <- list(NULL, FALSE)
  if (qr(s)$rank < dim(s)[2])
    stop("gss error in gssanova: fixed effects are linearly dependent")
  if (is.null(id.basis)) id.basis <- 1:nobs
  r <- r[,id.basis]
  q <- q[id.basis,id.basis]
  Kxmat <- Kxmat[,id.basis]
  z <- mysspngreg(family,s,r,q,y,wt,offset,alpha,nu.wk,Kxmat,lams)

c(list(family=family,id.basis=id.basis,alpha=alpha,xdomain=xdomain,xqma
t=xqmat,Kxmat=Kxmat,quad=quad),z)
}

# function for evaluating coefficient function estimate from generalized
# functional linear regressiong at given grid points.
estimate.gflr <- function(object,tgrid,se.fit = FALSE)
{
  nnull <- length(object$d)
  nt <- length(tgrid)
  quad <- object$quad
  xdomain <- object$xdomain
  qpt <- (quad$pt-xdomain[1])/(xdomain[2]-xdomain[1])
  twk <- (tgrid-xdomain[1])/(xdomain[2]-xdomain[1])
  Kmat <- outer(twk,qpt,partK)
  Kxmat <- Kmat%*%(quad$wt*object$xqmat)
  Kxmat <- Kxmat[,object$id.basis]
  btest <- object$d[2]*rep(1,nt)+object$d[3]*tgrid+
drop(Kxmat%*%(10^object$theta*object$c))
  if (se.fit) {
    b <- object$varht/10^object$nlambda
```

```
    ss <- rbind(rep(0,nt),rep(1,nt),tgrid)
    rwk <- 10^object$theta*Kxmat
    rr <- t(rwk %*% object$se.aux$vec)
    wk <- object$se.aux$hfac %*% rbind(ss, rr)
    pse <- sqrt(b * apply(wk^2, 2, sum))
    list(fit = btest, se.fit = pse)
  }
  else btest
}

# mean prediction function for generalized functional linear regression
# xqmatnew - nmesh*nnew matrix of new x functions evaluated at
quadrature
#           points, which must match with the quadrature used in object.
predict.gflr <- function(object,xqmatnew,se.fit=FALSE)
{
  quad <- object$quad
  Kxmat <- object$Kxmat
  nnew <- dim(xqmatnew)[2]
  s <- rep(1,nnew)
  s <-
cbind(s,apply(quad$wt*xqmatnew,2,sum),apply(quad$wt*quad$pt*xqmatn
ew,2,sum))
  r <- t(Kxmat)%*%(quad$wt*xqmatnew)
  pred <- drop(s%*%object$d+t(r)%*%(10^object$theta*object$c))
  if (object$family=="binomial") mpred <- 1/(1+exp(-pred))
  else mpred <- exp(pred)
  if (se.fit) {
    b <- object$varht/10^object$nlambda
    rwk <- 10^object$theta*t(r)
    rr <- t(rwk %*% object$se.aux$vec)
    wk <- object$se.aux$hfac %*% rbind(t(s), rr)
    pse <- sqrt(b * apply(wk^2, 2, sum))
    list(pred=pred,mpred=mpred,se=pse)
  }
  else list(pred=pred,mpred=mpred)
}

mysspngreg <- function (family, s, r, q, y, wt, offset, alpha, nu, Kxmat, lams)
{
```

```
  nobs <- nrow(r)
  nxi <- ncol(r)
  nnull <- ncol(s)
  nn <- nxi + nnull
  tmp <- sum(r^2)
  theta <- log10(sum(s^2)/nnull/tmp * nxi)/2
  #theta <- 0
  log.la0 <- log10(tmp/sum(diag(q))) + theta
  ran.scal <- NULL
  dc <- rep(0, nn)
  fit <- NULL
  la <- log.la0
  minscore <- Inf
  for (lambda in lams+theta) {
    la.wk <- lambda
    nu.wk <- nu
    q.wk <- 10^(la.wk + theta) * q
    alpha.wk <- max(0, log.la0 - la.wk[1] - 5) * (3 - alpha) + alpha
    alpha.wk <- min(alpha.wk, 3)
    z <- ngreg(dc, family, cbind(s, 10^theta * r), q.wk,
          y, wt, offset, nu.wk, alpha.wk)
    assign("dc", z$dc, inherit = TRUE)
    score <- z$score
    if (score < minscore) {
      minscore <- score
      assign("fit", z[c(1:3, 5:10)], inherit = TRUE)
      nlambda <- lambda
    }
  }
  se.aux <- regaux(sqrt(fit$w) * s, 10^theta * sqrt(fit$w) *
     r, 10^theta * q, nlambda, fit)
  c <- fit$dc[nnull + (1:nxi)]
  d <- fit$dc[1:nnull]
  c(list(theta = theta, c = c, d = d, nlambda = nlambda, se.aux=se.aux),
fit[-1])
}
```