Automatic Search Interval (R code)

Zheyuan Li (zheyuan.li@bath.edu) and Jiguo Cao (jiguo_cao@sfu.ca)

This vignette accompanies our paper "Automatic Search Interval of Smoothing Parameter in Penalized Splines". (Submitted to *Statistics and Computing*)

Package **gps** can be installed from GitHub (it will be on CRAN at a later date).

```
## you may need to first install "devtools" from CRAN
devtools::install_github("ZheyuanLi/gps")
```

Load the package.

```
library(gps)
```

Define two useful functions:

- `TestRhoLim` for testing automatic search intervals (the workhorse of our simulations);
- `PlotRhoLim` for illustrating automatic search intervals.

```
## demonstrate the automatic search interval of rho under different scenarios
## (uniform, deriv.pen) = (TRUE, TRUE), O-spline on equidistant knots
## (uniform, deriv.pen) = (FALSE, TRUE), O-spline on unevenly spaced knots
## (uniform, deriv.pen) = (TRUE, FALSE), standard P-spline on equidistant knots
## (uniform, deriv.pen) = (FALSE, FALSE), general P-spline on unevenly spaced knots
## in all cases, observations can be weighted
## d, B-spline order (degree + 1, i.e., 4 for cubic)
## m, penalty order (2 for penalizing 2nd derivative or difference)
## p, number of B-splines
## ng, number of grid points on the rho interval for grid search
TestRhoLim <- function (p = 10, d = 4, m = 2, deriv.pen = FALSE, uniform = FALSE,
                        weighted = FALSE, ng = 0) {
  ## number of interior knots
  k <- p - d
```

```r
  ## number of full knots
  K <- k + 2 * d
  ## simulate knots
  if (uniform) {
    xt <- 1:K
  } else {
    xt <- sort(rnorm(K, 1:K, sd = ceiling(K / 10)))
  }
  ## domain knots
  xd <- xt[d:(K - d + 1)]
  nd <- length(xd)
  ## simulate x-values by drawing 10 uniform values on each interval
  x <- sort(runif(10 * (nd - 1), rep(xd[-nd], 10), rep(xd[-1], 10)))
  ## simulate non-negative weights
  w <- NULL
  if (weighted) {
    ## feel free to use other shape parameters
    w <- rbeta(length(x), 3, 3)
    ## scale the weights so that they sum up to 1
    w <- w / sum(w)
  }
  ## set up a general P-spline for (x, w); the setup does not depend on y-values
  ## if ng > 0, go on to fit this general P-spline to (x, x, w)
  ## the fitted spline is nonsense but its computation time makes sense
  gps2 <- suppressWarnings(gps::gps2GS(x, x, w, xt, d, m, !deriv.pen, ng = ng))
  ## demo result on rho bounds
  TestOut <- suppressWarnings(gps::DemoRhoLim(gps2, plot = FALSE))
  ## get computation time
  approx.lambda <- TestOut$eigen$approx$secs
  exact.lambda <- TestOut$eigen$exact$secs
  pwls <- if (ng > 0) gps2$pwls$secs else NA
  TestOut$runtime <- c(approx.lambda = approx.lambda, exact.lambda = exact.lambda, pwls =
        pwls)
  ## keep knots and x-values for reconstruction
  TestOut$data <- list(knots = xt, x = x)
  ## return
  TestOut
}

## plot the result of TestRhoLim()
PlotRhoLim <- function (TestOut, show.heuristic = TRUE) {
  ## number of eigenvalues
  q <- length(TestOut$eigen$exact$values)
  ## extract rho bounds
  rho.min <- TestOut$limit$rho$exact[1]
  rho.max <- TestOut$limit$rho$exact[2]
  rho.min.star <- TestOut$limit$rho$wider[1]
  rho.max.star <- TestOut$limit$rho$wider[2]
  rho.max.hat <- TestOut$limit$rho$heuristic
  ## extract mapped redf
  redf.min <- TestOut$limit$redf$exact[1]
  redf.max <- TestOut$limit$redf$exact[2]
```

```
  redf.min.star <- TestOut$limit$redf$wider[1]
  redf.max.star <- TestOut$limit$redf$wider[2]
  redf.min.hat <- TestOut$limit$redf$heuristic[1]
  ## a grid of rho-values
  rho.grid <- TestOut$redf$rho
  ## exact redf on this grid, using exact eigenvalues
  exact.redf <- TestOut$redf$exact
  ## approximated redf on this grid, using approximated eigenvalues
  approx.redf <- TestOut$redf$approx
  ## plot exact redf against log smoothing parameters
  plot(rho.grid, exact.redf, type = "l", lwd = 2, ann = FALSE, bty = "n")
  ## show approximated redf
  if (show.heuristic) {
    lines(rho.grid, approx.redf, type = "l", lwd = 2, lty = 2)
  }
  ## show exact search interval
  segments(rho.min, q, rho.min, 0.75 * q, lty = 2, lwd = 2, col = 2)
  segments(rho.max, 0, rho.max, 0.25 * q, lty = 2, lwd = 2, col = 2)
  ## show a wider search interval
  points(rho.min.star, redf.max.star, pch = 19, cex = 1.5)
  points(rho.max.star, redf.min.star, pch = 19, cex = 1.5)
  ## show heuristic search interval
  if (show.heuristic) {
    points(rho.max.hat, redf.min.hat, pch = 15, cex = 1.5)
  }
  ## show mapping
  exact <- paste("exact:", TestOut$mapping$exact)
  wider <- paste("wider:", TestOut$mapping$wider)
  heuristic <- paste("heuristic:", TestOut$mapping$heuristic)
  legend.txt1 <- c(exact, wider, heuristic)[c(TRUE, TRUE, show.heuristic)]
  legend.txt2 <- c("exact.redf", "approx.redf")[c(TRUE, show.heuristic)]
  legend("topright", legend = legend.txt1, pch = c(NA, 19, 15), lty = c(2, NA, NA),
         col = c(2, 1, 1), text.col = c(2, 1, 1), lwd = 2, text.font = 2, bty = "n")
  legend("bottomleft", legend = legend.txt2, lwd = 2, lty = c(1, 2), text.font = 2,
         bty = "n")
}
```

Define functions for fitting (x, y) data.

```
## post-process the output of gps::gps2GS() to compute REML score
GetREML <- function (gps2fit) {
  ## estimating equation
  B <- gps2fit$eqn$B
  BWB <- gps2fit$eqn$BWB
  S <- gps2fit$eqn$S
  ## PWLS fit
  rho <- gps2fit$pwls$rho
  exp.rho <- exp(rho)
  b <- gps2fit$pwls$beta
  edf <- gps2fit$pwls$edf
  gcv <- gps2fit$pwls$gcv
```

```r
  ## n, p, m, q, N
  n <- nrow(B)
  p <- nrow(b)
  q <- ncol(gps2fit$E)
  m <- p - q
  N <- length(rho)
  ## estimated error variance
  sig2 <- gcv * (1 - edf / n)
  ## weighted residual sum of squares
  WRSS <- sig2 * (n - edf)
  ## b'Sb
  btSb <- colSums(b * as.matrix(S %*% b))
  ## positive eigenvalues of S
  evalues <- eigen(S, symmetric = TRUE, only.values = TRUE)$values[1:q]
  ## REML score: -2 * log(restricted.likelihood)
  term1 <- (n - m) * log(2 * pi * sig2)
  term2 <- (WRSS + exp.rho * btSb) / sig2
  term3 <- numeric(N)
  for (i in 1:N) {
    term3[i] <- 2 * sum(log(Matrix::diag(Matrix::chol(BWB + exp.rho[i] * S))))
  }
  term4 <- q * rho + sum(log(evalues))
  score <- term1 + term2 + term3 - term4
  ## log(restricted.likelihood)
  -0.5 * score
}

## smooth (x, y) data using general P-splines
FitXY <- function (x, y, nknots.per.obs = 0.25) {
  x <- as.numeric(x)
  y <- as.numeric(y)
  ## number of data
  n <- length(x)
  ## choose a suitable number of interior knots
  k <- round(n * nknots.per.obs)
  xt <- gps::PlaceKnots(x, d = 4, k = k)
  ## fit a general P-spline
  fit <- gps::gps2GS(x, y, xt = xt, ng = 60)
  ## get REML
  fit$pwls$REML <- GetREML(fit)
  ## fitted values
  fit$pwls$fitted <- as.matrix(fit$eqn$B %*% fit$pwls$beta)
  ## identify all local minima of the GCV curve
  opt.gcv <- which(diff(sign(diff(fit$pwls$gcv))) == 2) + 1
  if (length(opt.gcv) == 0) opt.gcv <- which.min(fit$pwls$gcv)
  ## identify all local maxima of the REML curve
  opt.REML <- which(diff(sign(diff(fit$pwls$REML))) == -2) + 1
  if (length(opt.REML) == 0) opt.REML <- which.min(fit$pwls$REML)
  ## return
  fit$opt <- list(gcv = opt.gcv, REML = opt.REML)
  fit
```

```
}
```

Read in the COVID-19 data used in the paper (accessed from Our World in Data on 2022-03-02):

```
Finland <- list(date = structure(c(18506, 18507, 18510, 18512, 18515, 18517,
18520, 18522, 18528, 18529, 18531, 18537, 18541, 18542, 18543,
18546, 18547, 18550, 18552, 18555, 18558, 18559, 18561, 18563,
18566, 18573, 18574, 18575, 18576, 18578, 18579, 18580, 18581,
18582, 18583, 18584, 18585, 18586, 18587, 18588, 18589, 18590,
18591, 18592, 18593, 18594, 18595, 18596, 18597, 18598, 18599,
18600, 18601, 18602, 18603, 18604, 18605, 18606, 18607, 18608,
18609, 18610, 18611, 18612, 18613, 18614, 18615, 18616, 18617,
18618, 18619, 18620, 18621, 18622, 18624, 18625, 18626, 18627,
18628, 18629, 18630, 18631, 18632, 18633, 18634, 18635, 18636,
18637, 18638, 18639, 18640, 18641, 18642, 18643, 18644, 18646,
18647, 18648, 18649, 18650, 18651, 18652, 18653, 18654, 18656,
18657, 18658, 18660, 18661, 18662, 18663, 18664, 18666, 18667,
18669, 18670, 18671, 18672, 18673, 18674, 18675, 18676, 18677,
18678, 18679, 18680, 18681, 18682, 18683, 18684, 18685, 18686,
18687, 18688, 18689, 18690, 18691, 18692, 18693, 18694, 18695,
18696, 18697, 18698, 18699, 18700, 18702, 18703, 18704, 18705,
18706, 18707, 18709, 18710, 18711, 18712, 18713, 18714, 18715,
18716, 18717, 18718, 18719, 18720, 18721, 18722, 18724, 18725,
18726, 18727, 18728, 18730, 18731, 18732, 18733, 18734, 18735,
18737, 18738, 18739, 18741, 18742, 18744, 18745, 18746, 18747,
18748, 18750, 18753, 18754, 18755, 18758, 18759, 18760, 18761,
18762, 18763, 18764, 18766, 18767, 18770, 18771, 18774, 18778,
18780, 18783, 18786, 18787, 18789, 18794, 18800, 18802, 18814,
18815, 18820, 18821, 18822, 18830, 18832, 18833, 18834, 18835,
18836, 18837, 18838, 18840, 18842, 18843, 18844, 18845, 18846,
18847, 18849, 18850, 18851, 18852, 18853, 18855, 18857, 18858,
18859, 18860, 18862, 18863, 18864, 18865, 18866, 18867, 18869,
18870, 18871, 18872, 18873, 18874, 18877, 18878, 18879, 18880,
18882, 18883, 18884, 18885, 18886, 18887, 18888, 18890, 18891,
18892, 18893, 18894, 18895, 18896, 18897, 18898, 18899, 18900,
18901, 18902, 18903, 18904, 18905, 18906, 18907, 18908, 18909,
18910, 18911, 18912, 18913, 18914, 18915, 18916, 18917, 18918,
18919, 18920, 18921, 18922, 18923, 18924, 18925, 18926, 18927,
18928, 18929, 18930, 18931, 18932, 18933, 18934, 18935, 18936,
18937, 18938, 18939, 18940, 18941, 18942, 18943, 18944, 18945,
18946, 18947, 18948, 18949, 18950, 18951, 18952, 18953, 18954,
18955, 18956, 18957, 18958, 18959, 18960, 18961, 18962, 18963,
18964, 18965, 18966, 18967, 18968, 18969, 18970, 18971, 18972,
18973, 18974, 18975, 18976, 18977, 18978, 18979, 18980, 18981,
18982, 18983, 18984, 18985, 18986, 18987, 18988, 18989, 18990,
18991, 18992, 18993, 18994, 18995, 18996, 18997, 18998, 18999,
19000, 19001, 19002, 19003, 19004, 19005, 19006, 19007, 19008,
19009, 19010, 19011, 19013, 19014, 19019, 19020, 19023, 19025,
19026, 19027, 19030, 19031, 19032, 19033, 19034, 19037, 19038,
19039, 19040, 19041, 19044, 19046, 19047), class = "Date"), deaths = c(1,
1, 1, 1, 3, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 1, 3, 1, 1,
```

```
1, 1, 1, 2, 2, 7, 3, 1, 4, 7, 2, 2, 1, 1, 3, 3, 6, 10, 4, 5,
6, 8, 2, 6, 2, 5, 10, 2, 7, 7, 3, 6, 10, 10, 3, 5, 4, 8, 3, 5,
11, 7, 4, 1, 5, 3, 4, 4, 5, 3, 3, 2, 4, 5, 2, 4, 6, 3, 4, 3,
7, 4, 3, 3, 4, 2, 3, 8, 9, 5, 5, 1, 1, 1, 4, 12, 4, 2, 2, 2,
5, 3, 1, 3, 1, 6, 2, 3, 1, 2, 2, 5, 5, 2, 2, 4, 4, 1, 5, 2, 2,
3, 2, 1, 5, 9, 2, 7, 3, 1, 1, 7, 2, 1, 2, 3, 1, 2, 9, 4, 4, 4,
4, 4, 1, 1, 3, 4, 7, 4, 6, 5, 4, 2, 3, 4, 1, 3, 2, 1, 2, 2, 3,
1, 1, 4, 1, 2, 2, 1, 4, 1, 2, 5, 2, 3, 1, 2, 1, 1, 3, 2, 2, 1,
2, 2, 1, 2, 1, 1, 1, 2, 2, 5, 6, 2, 3, 6, 2, 2, 1, 2, 1, 1, 1,
1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 4, 1, 3, 3,
4, 1, 2, 4, 1, 1, 1, 1, 2, 2, 1, 3, 2, 1, 2, 1, 3, 2, 2, 2, 1,
6, 2, 1, 2, 1, 6, 3, 1, 2, 1, 5, 1, 2, 1, 1, 1, 5, 1, 1, 2, 4,
3, 1, 3, 3, 4, 4, 3, 5, 3, 5, 4, 7, 2, 3, 8, 5, 3, 5, 3, 3, 4,
2, 1, 4, 6, 1, 2, 2, 8, 4, 2, 13, 4, 5, 5, 2, 3, 5, 12, 3, 7,
9, 11, 9, 4, 12, 9, 4, 6, 5, 3, 6, 17, 9, 5, 4, 6, 8, 5, 10,
10, 9, 11, 8, 8, 3, 15, 7, 12, 5, 6, 7, 5, 3, 7, 9, 7, 6, 7,
5, 7, 6, 8, 8, 8, 2, 5, 8, 6, 8, 7, 6, 4, 9, 17, 12, 13, 16,
14, 11, 12, 13, 13, 13, 11, 11, 10, 8, 9, 6, 6, 7, 6, 7, 4, 12,
4, 2, 1, 25, 23, 48, 22, 29, 17, 37, 2, 29, 15, 29, 44, 1, 27,
23, 12, 52, 22, 13), location = "Finland")

Netherlands <- list(date = structure(c(18871, 18872, 18873, 18874, 18875, 18876,
18877, 18878, 18879, 18880, 18881, 18882, 18883, 18884, 18885,
18886, 18887, 18888, 18889, 18890, 18891, 18892, 18893, 18894,
18895, 18896, 18897, 18898, 18899, 18900, 18901, 18902, 18903,
18904, 18905, 18906, 18907, 18908, 18909, 18910, 18911, 18912,
18913, 18914, 18915, 18916, 18917, 18918, 18919, 18920, 18921,
18922, 18923, 18924, 18925, 18926, 18927, 18928, 18929, 18930,
18931, 18932, 18933, 18934, 18935, 18936, 18937, 18938, 18939,
18940, 18941, 18942, 18943, 18944, 18945, 18946, 18947, 18948,
18949, 18950, 18951, 18952, 18953, 18954, 18955, 18956, 18957,
18958, 18959, 18960, 18961, 18962, 18963, 18964, 18965, 18966,
18967, 18968, 18969, 18970, 18971, 18972, 18973, 18974, 18975,
18976, 18977, 18978, 18979, 18980, 18981, 18982, 18983, 18984,
18985, 18986, 18987, 18988, 18989, 18990, 18991, 18992, 18993,
18994, 18995, 18996, 18997, 18998, 18999, 19000, 19001, 19002,
19003, 19004, 19005, 19006, 19007, 19008, 19010, 19011, 19012,
19013, 19014, 19015, 19016, 19017, 19018, 19019, 19020, 19021,
19022, 19023, 19024, 19025, 19026, 19027, 19028, 19029, 19030,
19031, 19032, 19033, 19034, 19035, 19036, 19037, 19038, 19039,
19040, 19041, 19042, 19043, 19044, 19045, 19046, 19047, 19048,
19049, 19050, 19051, 19052, 19053), class = "Date"), cases = c(2800,
2569, 2713, 2374, 2115, 2481, 2820, 2395, 2713, 2198, 22, 3831,
2029, 2376, 2098, 2170, 2017, 1663, 1347, 1690, 1929, 1838, 1698,
1611, 1613, 1387, 1729, 1783, 1703, 1733, 1666, 1619, 1638, 1903,
1895, 2759, 2792, 2510, 2258, 2719, 2887, 3733, 3634, 3716, 3703,
3677, 3354, 3899, 4582, 5206, 5869, 5680, 6296, 5296, 5749, 7264,
7685, 7391, 7946, 8174, 7694, 7731, 9112, 10214, 10911, 11962,
11349, 11805, 11289, 12648, 16296, 16204, 13848, 12030, 19197,
20178, 20764, 23593, 21032, 21798, 20646, 23002, 22956, 23713,
22189, 21281, 22031, 22135, 21443, 22163, 18519, 23048, 21529,
```

```
22614, 23082, 20965, 17921, 18067, 19767, 17478, 17716, 16588,
13787, 12760, 13493, 16329, 15369, 14616, 13279, 12154, 9316,
13353, 13362, 12436, 12590, 11886, 11432, 9175, 15817, 14778,
16813, 15713, 17582, 14536, 18506, 24897, 24812, 34872, 28357,
32484, 28171, 29039, 32244, 33835, 36318, 31853, 36511, 42352,
69695, 40038, 57570, 46147, 65367, 64677, 54160, 58700, 64555,
74088, 39444, 75220, 112227, 105818, 66999, 82886, 68936, 40135,
94894, 77619, 380399, 86559, 86789, 80630, 70357, 60165, 55011,
53519, 63880, 56378, 49289, 33609, 36440, 35987, 34672, 32709,
41109, 37523, 32879, 27389, 28882, 92079, 36495, 48489), location = "Netherlands")
```

1 Introduction

Figures 1: Introducing COVID-19 smoothing example.

```
FinlandFit <- FitXY(Finland$date, Finland$deaths)
NetherlandsFit <- FitXY(Netherlands$date, Netherlands$cases)

IntroCOVID19 <- function (fit, rho.cutoff, dat) {
  ## a zoomed-in display of GCV curve
  ind <- fit$pwls$rho < rho.cutoff
  rho <- fit$pwls$rho[ind]
  gcv <- fit$pwls$gcv[ind]
  opt.gcv <- fit$opt$gcv[ind]
  plot(rho, gcv, type = "l", ann = FALSE)
  points(rho[opt.gcv], gcv[opt.gcv], pch = 19, col = c(2, 4), cex = 2)
  title("GCV ~ rho")
  mtext(paste(dat$location, "new", names(dat[2])), line = 1.5)
  ## fitted spline at a local minimum
  plot(dat[[1]], dat[[2]], ann = FALSE, type = "n")
  points(dat[[1]], dat[[2]], cex = 0.5, pch = 19)
  lines(dat$date, fit$pwls$fitted[, opt.gcv[1]], lwd = 3, col = 2)
  title("fitted spline at local minimum")
  ## fitted spline at the global minimum
  plot(dat[[1]], dat[[2]], ann = FALSE, type = "n")
  points(dat[[1]], dat[[2]], cex = 0.5, pch = 19)
  lines(dat$date, fit$pwls$fitted[, opt.gcv[2]], lwd = 3, col = 4)
  title("fitted spline at global minimum")
}

pdf("fig1.pdf", height = 9 * 0.9, width = 6 * 0.9)
par(mfcol = c(3, 2), mar = c(2, 2, 1.5, 0.5), oma = c(0, 0, 1.5, 0))
IntroCOVID19(FinlandFit, rho.cutoff = 5, dat = Finland)
IntroCOVID19(NetherlandsFit, rho.cutoff = 5, dat = Netherlands)
dev.off()
```

3 Automatic Search Interval

3.4 An Illustration of the Intervals

Figures 2: An redf-oriented demonstration of automatic search intervals for cubic spline with a 2nd order penalty.

```
set.seed(2022)
test50 <- TestRhoLim(p = 50)
test500 <- TestRhoLim(p = 500)

pdf("fig2.pdf", width = 5 * 0.95, height = 6 * 0.95)
par(mfrow = c(2, 1), mar = c(2, 2, 1.5, 0.5))
PlotRhoLim(test50, show.heuristic = FALSE)
title("p = 50")
PlotRhoLim(test500, show.heuristic = FALSE)
title("p = 500")
dev.off()
```

Figure 3: A more conventional criterion-oriented demonstration of search intervals.

```
xt <- test50$data$knots
x <- test50$data$x
D <- gps::SparseD(xt, d = 4, m = 2)[[1]]
## simulate y values
set.seed(0)
b <- gps::PriorCoef(2, D)
b <- apply(b, 2, scale)
B <- splines::splineDesign(xt, x, sparse = TRUE)
f <- B %*% b
f1 <- f[, 1]; f2 <- f[, 2]
y1 <- rnorm(length(f1), f1, sd = 0.4 * sd(f1))
y2 <- rnorm(length(f2), f2, sd = 0.4 * sd(f2))
## fit (x, y)
fit1 <- gps::gps2GS(x, y1, xt = xt, d = 4, m = 2, ng = 50, gps = TRUE)
fit2 <- gps::gps2GS(x, y2, xt = xt, d = 4, m = 2, ng = 50, gps = TRUE)

PlotExample <- function (x, y, f, B, fit, label) {
  ## rho grid
  rho <- fit$pwls$rho
  ## decide optimal fit using GCV error
  gcv <- fit$pwls$gcv
  gcv.opt.id <- which.min(gcv)
  gcv.yp <- B %*% fit$pwls$beta[, gcv.opt.id]
  ## decide optimal fit using log restricted likelihood
  REML <- GetREML(fit)
  REML.opt.id <- which.max(REML)
  REML.yp <- B %*% fit$pwls$beta[, REML.opt.id]
  ## illustrate optimal fit
  plot(x, y, col = 8, ann = FALSE)
  lines(x, f, col = 2, lwd = 2)
  lines(x, gcv.yp, lwd = 2)
```

```
  lines(x, REML.yp, lwd = 2, lty = 2)
  legend("bottom", legend = c("GCV optimal", "REML optimal"), cex = 1.25, lwd = 2,
         lty = 1:2, bty = "n")
  title("simulated data and fitted spline")
  mtext(label, line = 1.5)
  ## GCV curve
  plot(rho, log(gcv), type = "l", ann = FALSE)
  points(rho[gcv.opt.id], log(gcv[gcv.opt.id]), pch = 19)
  title("log(GCV) ~ rho")
  ## REML curve
  plot(rho, REML, type = "l", ann = FALSE)
  points(rho[REML.opt.id], REML[REML.opt.id], pch = 19)
  title("log(REML) ~ rho")
}

## plot
pdf("fig3.pdf", width = 5, height = 7.5)
par(mfcol = c(3, 2), mar = c(2, 2, 1.5, 0.5), oma = c(0, 0, 1.5, 0))
PlotExample(x, y1, f1, B, fit1, label = "example 1")
PlotExample(x, y2, f2, B, fit2, label = "example 2")
dev.off()
```

3.5 Heuristic Improvement

Figure 4: Illustrate the decay of log Demmler-Reinsch eigenvalues, the asymptotic decay $z(t, 0)$ established by Xiao (2019) and our "S"-shaped curve $z(t, \gamma)$ for various γ values on [0, 1].

```
## map a vector onto [0, 1]
map01 <- function (x) (x - min(x)) / (max(x) - min(x))

## plot log(lambda) as well as z(t, gamma) against t for various 'gamma' values
PlotDecay <- function (lambda) {
  ## number of eigenvalues
  q <- length(lambda)
  ## percentage variable
  t <- (1:q) / (q + 1)
  ## gamma values to try
  gamma <- seq.int(0, 1, 0.25)
  ## z(t, gamma), mapped to [0, 1]
  z <- log(1 - t) - tcrossprod(log(t), gamma)
  z <- apply(z, 2, map01)
  colnames(z) <- gamma
  ## log eigenvalues, mapped to [0, 1]
  log.lambda <- map01(log(lambda))
  ## plot
  matplot(t, z, type = "l", lty = 2, lwd = 2, col = 2:6, ann = FALSE)
  lines(t, log.lambda, lwd = 2)
  legend.txt <- expression(log(lambda), z(list(t, "0.00")), z(list(t, "0.25")),
                           z(list(t, "0.50")), z(list(t, "0.75")), z(list(t, "1.00")))
  legend("bottomleft", legend = legend.txt, lwd = 2, bty = "n", lty = c(1, rep(2, 5)),
```

```
        col = 1:6, text.col = 1:6, cex = 1.1)
  ## return quantities
  list(t = t, z = z, log.lambda = log.lambda)
}

pdf("fig4.pdf", width = 6 * 1.1, height = 3.2 * 1.1)
par(mfrow = c(1, 2), mar = c(2, 2, 1.5, 0.5), oma = c(0, 0, 1.5, 0))
heuristic50 <- PlotDecay(test50$eigen$exact$values)
title("p = 50")
heuristic500 <- PlotDecay(test500$eigen$exact$values)
title("p = 500")
title(expression(list(log(lambda), z(t, gamma)) %~% t), outer = TRUE)
dev.off()
```

Figure 5: Motivate heuristic functions $Q_1(z, \alpha)$ and $Q_2(z, \alpha)$ for approximating the decay of log Demmler-Reinsch eigenvalues.

```
## compute z(t, gamma)
zfun <- function (t, gamma) {
  if (length(gamma) == 1) {
    z <- log(1 - t) - gamma * log(t)
  } else {
    z <- log(1 - t) - tcrossprod(log(t), gamma)
    z <- apply(z, 2, map01)
    colnames(z) <- gamma
  }
  map01(z)
}

## compute Q1(z, alpha)
Q1fun <- function (z, a = 0, b = 1, alpha) {
  if (length(alpha) == 1) {
    Qz <- a + (b - a) * z + alpha * (z ^ 2 - z)
  } else {
    Qz <- a + (b - a) * z + tcrossprod(z ^ 2 - z, alpha)
    colnames(Qz) <- alpha
  }
  Qz
}

## compute Q2(z, alpha)
Q2fun <- function (z, a = 0, b = 1, alpha) {
  C0 <- (1 - z) ^ 3
  C1 <- 3 * z * (1 - z) ^ 2
  C2 <- 3 * z ^ 2 * (1 - z)
  C3 <- z ^ 3
  if (length(alpha) == 1) {
    Qz <- (C0 + C2) * a + (C2 + C3) * b + alpha * (C1 - C2)
  } else {
    Qz <- (C0 + C2) * a + (C2 + C3) * b + tcrossprod(C1 - C2, alpha)
    colnames(Qz) <- alpha
```

```
  }
  Qz
}

PlotQfun <- function (t, log.lambda, gamma, num = 1) {
  a <- 0; b <- 1
  z <- zfun(t, gamma)
  if (num == 1) {
    alpha <- seq.int(0, b - a, length.out = 5)
    Qz <- Q1fun(z, alpha = alpha)
  } else if (num == 2) {
    alpha <- seq.int(a, (2 * a + b) / 3, length.out = 5)
    Qz <- Q2fun(z, alpha = alpha)
  } else {
    stop("unknown Q(z)!")
  }
  matplot(z, Qz, type = "l", lty = 2, col = 2:6, lwd = 2, ann = FALSE)
  lines(z, log.lambda, type = "l", lwd = 2)
  title.txt <- sprintf("list(log(lambda), Q[%d](list(z, alpha))) %%~%% z(t, '%.2f')",
                       num, gamma)
  title(parse(text = title.txt))
  legend.txt <- vector("expression", 6)
  legend.txt[1] <- expression(log(lambda))
  for (i in 1:5) {
    legend.txt[i + 1] <- parse(text = sprintf("Q[%d](list(z, '%.2f'))", num, alpha[i]))
  }
  legend("topleft", legend = legend.txt[1:3], lwd = 2, col = 1:3,
         text.col = 1:3, lty = c(1, 2, 2), bty = "n", cex = 1.1)
  legend("bottomright", legend = legend.txt[4:6], lwd = 2, col = 4:6,
         text.col = 4:6, lty = 2, bty = "n", cex = 1.1)
}

pdf("fig5.pdf", width = 6 * 0.95, height = 6 * 0.95)
par(mfcol = c(2, 2), mar = c(2, 2, 1.5, 0.5))
t <- heuristic50$t; log.lambda <- heuristic50$log.lambda
# t <- heuristic500$t; log.lambda <- heuristic500$log.lambda
## convex Q1(z, alpha)
PlotQfun(t, log.lambda, gamma = 0.10, num = 1)
PlotQfun(t, log.lambda, gamma = 0.45, num = 1)
## 'S'-shaped Q2(z, alpha)
PlotQfun(t, log.lambda, gamma = 0.10, num = 2)
PlotQfun(t, log.lambda, gamma = 0.45, num = 2)
dev.off()
```

Figure 6: Illustrate the approximated log Demmler-Reinsch eigenvalues.

```
pdf("fig6.pdf", width = 6, height = 3.2)
par(mfrow = c(1, 2), mar = c(2, 2, 1.5, 0.5), oma = c(0, 0, 1.5, 0))
## p = 50
plot(heuristic50$t, heuristic50$log.lambda, type = "l", lwd = 2, lty = 2, ann = FALSE)
lines(heuristic50$t, map01(log(test50$eigen$approx$values)), lwd = 2)
```

```
title("p = 50")
legend("topright", legend = expression(log(hat(lambda)), log(lambda)), lty = 1:2,
       lwd = 2, bty = "n", cex = 1.25)
## p = 500
plot(heuristic500$t, heuristic500$log.lambda, type = "l", lwd = 2, lty = 2, ann = FALSE)
lines(heuristic500$t, map01(log(test500$eigen$approx$values)), lwd = 2)
title("p = 500")
legend("topright", legend = expression(log(hat(lambda)), log(lambda)), lty = 1:2,
       lwd = 2, bty = "n", cex = 1.25)
title(expression(list(log(lambda), log(hat(lambda))) %~% t), outer = TRUE)
dev.off()
```

Figure 7 (continuing with Figure 2): A demonstration of automatic search intervals for cubic spline with a 2nd order penalty.

```
pdf("fig7.pdf", width = 5 * 0.95, height = 6 * 0.95)
par(mfrow = c(2, 1), mar = c(2, 2, 1.5, 0.5))
PlotRhoLim(test50, show.heuristic = TRUE)
title("p = 50")
PlotRhoLim(test500, show.heuristic = TRUE)
title("p = 500")
dev.off()
```

3.6 Computation Time

Computation time vary with CPUs, but they deliver the same message: exact interval is expensive to compute.

```
## measure computation time
Timing <- function (p, ng = 20) {
  runtime <- numeric(3)
  ## average over multiple runs to improve accuracy
  for (i in 1:10) {
    TestOut <- TestRhoLim(p = p, d = 4, m = 2, ng = ng)
    runtime <- runtime + TestOut$runtime
  }
  runtime / 10
}

p <- c(500, 1000, 1500, 2000)
runtime <- matrix(0, length(p), 3)
rownames(runtime) <- p
colnames(runtime) <- c("ApproxEigen", "ExactEigen", "PWLS")
for (i in 1:length(p)) runtime[i, ] <- Timing(p[i], ng = 20)
t(runtime)
```

4 Simulation Studies

The 8 scenarios.

```
deriv.pen <- rep.int(c(FALSE, TRUE), 4)
uniform <- rep.int(rep.int(c(FALSE, TRUE), c(2, 2)), 2)
weighted <- rep.int(c(FALSE, TRUE), c(4, 4))
scenarios <- cbind(deriv.pen, uniform, weighted)
rownames(scenarios) <- 1:8
scenarios
#   deriv.pen uniform weighted
# 1     FALSE   FALSE    FALSE
# 2      TRUE   FALSE    FALSE
# 3     FALSE    TRUE    FALSE
# 4      TRUE    TRUE    FALSE
# 5     FALSE   FALSE     TRUE
# 6      TRUE   FALSE     TRUE
# 7     FALSE    TRUE     TRUE
# 8      TRUE    TRUE     TRUE
```

Figure 8: Simulation results on the heuristic upper bound. (Caution: The program is time-consuming!)

```
## simulation on the heuristic upper bounds for all scenarios
## d, B-spline order (degree + 1, i.e., 4 for cubic)
## m, penalty order (2 for penalizing 2nd derivative or difference)
SimHeuristic <- function (n.sim = 100, d = 4, m = 2) {
  ## 8 scenarios
  deriv.pen <- rep.int(c(FALSE, TRUE), 4)
  uniform <- rep.int(rep.int(c(FALSE, TRUE), c(2, 2)), 2)
  weighted <- rep.int(c(FALSE, TRUE), c(4, 4))
  ## for each scenario, try p = 40, 200 and 1000
  p <- c(40, 200, 1000)
  lst <- vector("list", length(p))
  names(lst) <- p
  ## loop through p
  for (j in 1:length(p)) {
    star <- hat <- matrix(0, n.sim, 8)
    colnames(star) <- colnames(hat) <- 1:8
    rownames(star) <- rownames(hat) <- 1:n.sim
    ## loop through scenarios
    for (i in 1:8) {
      ## simulations
      for (n in 1:n.sim) {
        TestOut <- TestRhoLim(p[j], d, m, deriv.pen[i], uniform[i], weighted[i])
        star[n, i] <- TestOut$limit$redf$wider[1]
        hat[n, i] <- TestOut$limit$redf$heuristic
      }
    }
    lst[[j]] <- list(star = star, hat = hat)
  }
  structure(lst, d = d, m = m)
}
```

```r
## cubic spline with a 2nd order penalty
SimOutCubic <- SimHeuristic(n.sim = 200, d = 4, m = 2)
## quadratic spline with a 1st order penalty
SimOutQuadratic <- SimHeuristic(n.sim = 200, d = 3, m = 1)

## plot the result of SimHeuristic()
PlotCoverage <- function (SimOut) {
  d <- attr(SimOut, "d")
  m <- attr(SimOut, "m")
  p <- as.integer(names(SimOut))
  n.sim <- nrow(SimOut[[1]])
  for (i in 1:length(p)) {
    ## q
    q <- p[i] - m
    ## the probability density of P(rho.max.star) degenerates to a verticle line
    P.star <- 1 -  SimOut[[i]]$star / q
    P.star.mean <- mean(P.star)
    P.star.sd <- sd(P.star)  ## too low
    P.star.upr <- P.star.mean + 3 * P.star.sd
    P.star.lwr <- P.star.mean - 3 * P.star.sd
    ## estimate probability density of P(rho.max.hat)
    den.x <- matrix(0, 256, 8)
    den.y <- matrix(0, 256, 8)
    for (j in 1:8) {
      P.hat <- 1 -  SimOut[[i]]$hat[, j] / q
      den <- density(na.omit(P.hat), n = 256)
      den.x[, j] <- den$x
      den.y[, j] <- den$y
    }
    ## plot density curves
    xlim <- range(den.x, P.star.upr, P.star.lwr)
    ylim <- range(den.y)
    plot(0, 0, type = "n", xlim = xlim, ylim = ylim, yaxt = "n", ann = FALSE)
    abline(v = 0.99, lwd = 4, col = 8)
    abline(v = c(P.star.lwr, P.star.mean, P.star.upr), lwd = 4, col = 7)
    color <- c(1:4, 1:4)
    lty <- rep(1:2, each = 4)
    for (j in 1:8) {
      lines(den.x[, j], den.y[, j], lty = lty[j], col = color[j], lwd = 2)
    }
    title(sprintf("d = %d, m = %d, p = %d", d, m, p[i]))
  }
}

pdf("fig8.pdf", width = 6 * 0.9, height = 9 * 0.9)
par(mfcol = c(3, 2), mar = c(2, 0.5, 1.5, 0.5))
PlotCoverage(SimOutCubic)
PlotCoverage(SimOutQuadratic)
legend(x = 0.996, y = 1700, legend = 1:8, lwd = 2, col = 1:4, lty = rep(1:2, each = 4),
       text.col = 1:4, bty = "n", cex = 1.5)
dev.off()
```

5 Application

Define a function to illustrate the result of smoothing.

```
PlotXYFit <- function (fit, dat, show.fitted.spline = TRUE) {
  rho <- fit$pwls$rho
  edf <- fit$pwls$edf
  gcv <- fit$pwls$gcv
  REML <- fit$pwls$REML
  ## global optimum
  opt.gcv <- which.min(gcv)
  opt.REML <- which.max(REML)
  rho.opt.gcv <- rho[opt.gcv]
  rho.opt.REML <- rho[opt.REML]
  edf.opt.gcv <- edf[opt.gcv]
  edf.opt.REML <- edf[opt.REML]
  ## local optimum
  local.opt.gcv <- setdiff(fit$opt$gcv, opt.gcv)
  local.opt.REML <- setdiff(fit$opt$REML, opt.REML)
  ## edf ~ rho
  plot(rho, edf, type = "l", ann = FALSE)
  points(rho.opt.gcv, edf.opt.gcv, pch = 19, cex = 1.5)
  text(rho.opt.gcv, edf.opt.gcv, sprintf("%.1f", edf.opt.gcv), pos = 4, cex = 1.5)
  points(rho.opt.REML, edf.opt.REML, pch = 15, cex = 1.5)
  text(rho.opt.REML, edf.opt.REML, sprintf("%.1f", edf.opt.REML), pos = 4, cex = 1.5)
  title("edf ~ rho")
  mtext(paste(dat$location, "new", names(dat[2])), line = 1.5)
  legend("topright", legend = c("GCV optimal", "REML optimal"), pch = c(19, 15),
         bty = "n", cex = 1.5)
  ## GCV ~ rho
  plot(rho, gcv, type = "l", ann = FALSE)
  points(rho.opt.gcv, gcv[opt.gcv], pch = 19, cex = 1.5)
  points(rho[local.opt.gcv], gcv[local.opt.gcv], pch = 4, cex = 1.5)
  title("GCV ~ rho")
  ## REML ~ rho
  plot(rho, REML, type = "l", ann = FALSE)
  points(rho.opt.REML, REML[opt.REML], pch = 15, cex = 1.5)
  points(rho[local.opt.REML], gcv[local.opt.REML], pch = 4, cex = 1.5)
  title("REML ~ rho")
  ## fitted spline
  if (show.fitted.spline) {
    ## GCV optimal
    plot(dat[[1]], dat[[2]], ann = FALSE, type = "n")
    points(dat[[1]], dat[[2]], cex = 0.5, pch = 19, col = 8)
    lines(dat$date, fit$pwls$fitted[, opt.gcv], lwd = 2)
    title("fitted spline (GCV optimal)")
    legend("top", sprintf("edf: %.1f", edf.opt.gcv), bty = "n", cex = 1.5)
    ## REML optimal
    plot(dat[[1]], dat[[2]], ann = FALSE, type = "n")
    points(dat[[1]], dat[[2]], cex = 0.5, pch = 19, col = 8)
```

```
    lines(dat$date, fit$pwls$fitted[, opt.REML], lwd = 2)
    title("fitted spline (REML optimal)")
    legend("top", sprintf("edf: %.1f", edf.opt.REML), bty = "n", cex = 1.5)
  }
}
```

Figure 9: When placing $n/4$ knots.

```
FinlandFit.25 <- FitXY(Finland$date, Finland$deaths, nknots.per.obs = 0.25)
NetherlandsFit.25 <- FitXY(Netherlands$date, Netherlands$cases, nknots.per.obs = 0.25)
round(unlist(FinlandFit.25$rho.lim), 2)
round(unlist(NetherlandsFit.25$rho.lim), 2)

pdf("fig9.pdf", width = 5, height = 12.5)
par(mfcol = c(5, 2), mar = c(2, 2, 1.5, 0.5), oma = c(0, 0, 1.5, 0))
PlotXYFit(FinlandFit.25, Finland, show.fitted.spline = TRUE)
PlotXYFit(NetherlandsFit.25, Netherlands, show.fitted.spline = TRUE)
dev.off()
```

Figure 10: When placing $n/8$ knots.

```
FinlandFit.125 <- FitXY(Finland$date, Finland$deaths, nknots.per.obs = 0.125)
NetherlandsFit.125 <- FitXY(Netherlands$date, Netherlands$cases, nknots.per.obs = 0.125)
round(unlist(FinlandFit.125$rho.lim), 2)
round(unlist(NetherlandsFit.125$rho.lim), 2)

pdf("fig10.pdf", width = 5, height = 7.5)
par(mfcol = c(3, 2), mar = c(2, 2, 1.5, 0.5), oma = c(0, 0, 1.5, 0))
PlotXYFit(FinlandFit.125, Finland, show.fitted.spline = FALSE)
PlotXYFit(NetherlandsFit.125, Netherlands, show.fitted.spline = FALSE)
dev.off()
```

Appendix A Example D_m

```
## equidistant knots
xt1 <- seq.int(0, 1, length.out = 10)
## unevenly spaced knots
xt2 <- c(0, 0, 0, 0, 1/3, 1/2, 1, 1, 1, 1)

## cubic B-splines
x <- seq.int(0, 1, length.out = 200)
B1 <- splines::splineDesign(xt1, x, outer.ok = TRUE)
B2 <- splines::splineDesign(xt2, x, outer.ok = TRUE)
B1[B1 == 0] <- NA
B2[B2 == 0] <- NA

pdf("figA.pdf", width = 6, height = 3)
par(mfrow = c(1, 2), mar = c(2, 2, 1.5, 0.5))
```

```r
matplot(x, B1, type = "l", lty = 1, lwd = 2, col = 1:6)
abline(h = 0, col = 8)
points(xt1, numeric(10), pch = 19)
title("equidistant knots")
matplot(x, B2, type = "l", lty = 1, lwd = 2, col = 1:6)
abline(h = 0, col = 8)
points(xt2, c(0.15, 0.1, 0.05, 0, 0, 0, 0, 0.05, 0.1, 0.15), pch = 19)
title("unevenly spaced knots")
dev.off()

## 2nd order D matrices for cubic B-splines on 'xt1'
ans1 <- gps::SparseD(xt1, d = 4, m = 2, gps = FALSE)
( D2a.os <- ans1[[1]] )
( D2a.gps <- attr(ans1, "sandwich")$D[[1]] )
## we can standardized the results, as if the spacing between knots is 1
h <- xt1[2] - xt1[1]
D2a.os * (h ^ 2)
D2a.gps * (h ^ 2)  ## D2a.sps

## 2nd order D matrices for cubic B-splines on 'xt1'
ans2 <- gps::SparseD(xt2, d = 4, m = 2, gps = FALSE)
( D2b.os <- ans2[[1]] )
( D2b.gps <- attr(ans2, "sandwich")$D[[1]] )
```