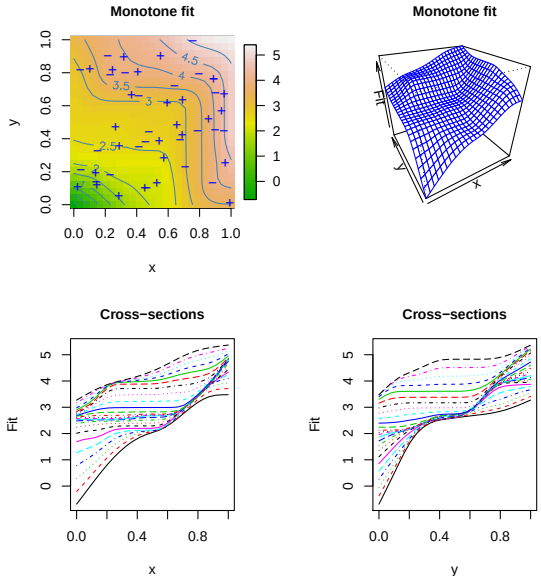
Monotone 2D smoothing with mixed model, updating λ (Simulated data)



Double monotone smoothing of simulated data with tensor product P-splines and automatic tuning of the (isotropic) roughness penalty with the HFS algorithm. Upper-left panel: positions of the data points and the signs of their residuals, indicated with − and + symbols. Contour lines of the fitted surface are shown in blue. Upper-right panel: perspective view of the fitted surface. Bottom panels: cross-sections of the fitted surface in two directions, emphasizing the monotonic behaviour. R code in `f-monotone-2d-mix.R`

```
# Monotone 2D smoothing with mixed model, updating lambda (simulated data)
# A graph in the book 'Practical Smoothing. The Joys of P-splines'
# Paul Eilers and Brian Marx, 2019

library(fields)
library(JOPS)
library(spam)

# Simulate the data
n <- 50
set.seed(123)
x <- runif(n)
y <- runif(n)
z0 <- 4 * (x + y - x * y)
sig = 1
z <- z0 + rnorm(n) * sig

t0 = Sys.time()

# Set parameters for domain
xlo <- 0
xhi <- 1
ylo <- 0
yhi <- 1

# Set P-spline parameters, fit and compute surface
xseg <- 20
xdeg <- 3
xpars <- c(xlo, xhi, xseg, xdeg)
yseg <- 20
ydeg <- 3
```

```
ypars <- c(ylo, yhi, yseg, ydeg)

# Compute one-dimensional bases
eps = 1e-6
Bx = bbase(x, xpars[1], xpars[2], xpars[3], xpars[4])
By = bbase(y, ypars[1], ypars[2], ypars[3], ypars[4])
Bx = as.spam(Bx, eps)
By = as.spam(By, eps)
nx = ncol(Bx)
ny = ncol(By)

# Compute tensor products
B1 <- kronecker(t(rep(1, ny)), Bx)
B2 <- kronecker(By, t(rep(1, nx)))
B <- B1 * B2
n = ncol(B)

# Compute penalty matrices
Ex = diag.spam(nx)
Ey = diag.spam(ny)
Dx = diff(Ex, diff = 2)
Dy = diff(Ey, diff = 2)
D1x = diff(Ex)
D1y = diff(Ey)
Cx = kronecker(Ey, D1x)
Cy = kronecker(D1y, Ex)
delta = 1e-10
Px = kronecker(Ey, t(Dx) %*% Dx)
Py = kronecker(t(Dy) %*% Dy, Ex)

lambda =1
lambdax = lambday = lambda
kappax = 1e5
kappay = 1e5
vx = rep(0, (nx - 1) * ny)
vy = rep(0, (ny - 1) * nx)
BtB = t(B) %*% B
Btz = t(B) %*% z

for (it in 1:30) {
  # Constraint matrices
  Qx = t(Cx) %*% diag.spam(c(vx)) %*% Cx
  Qy = t(Cy) %*% diag.spam(c(vy)) %*% Cy

  # Fit the model
  Pen = lambdax * Px + lambday * Py
  Pen = Pen  + kappax * Qx  + kappay * Qy
  a = solve(BtB + Pen, Btz)
  vynew = Cy %*% a < 0
  vxnew = Cx %*% a < 0
  dvx = sum(vx != vxnew)
  dvy = sum(vy != vynew)
  vx = vxnew
  vy = vynew
  cat(it, dvx, dvy, '\n')

  H = solve(BtB + Pen, BtB)
  ed = sum(diag(H))
  zhat = B %*% a
  v1 = sum((z - zhat) ^ 2 /(n - ed))
  v2 = sum((Pen %*% a) ^ 2) / ed
  lambda_new = v1 / v2
  cat(it, ed, lambda, lambda_new, '\n')
  dla = abs(lambda - lambda_new) / lambda
  lambda = lambda_new
  if (dvx + dvy == 0 & dla < 1e-4) break
}
```

```
zhat = B %*% a
r = z - zhat
# cat("SD of residuals:", sd(r), "\n")
cat('Time used: ', Sys.time() - t0, '\n')

# Compute grid for predicted surface
nu <- nv <- 25
u <- seq(xlo, xhi, length = nu)
v <- seq(ylo, yhi, length = nv)
Bgx = bbase(u, xpars[1], xpars[2], xpars[3], xpars[4])
Bgy = bbase(v, ypars[1], ypars[2], ypars[3], ypars[4])
A = matrix(a, nx, ny)
Fit = Bgx %*% A %*% t(Bgy)

# Plot result and data
par(mfrow = c(2,2), mar = c(3, 3, 3, 1))
pchs = c("+", "-")[(z > zhat) + 1]
image.plot(u, v, Fit, col = terrain.colors(100), xlab = "x",
    ylab = "y")
contour(u, v, Fit, add = T, col = "steelblue", labcex = 0.7)
points(x, y, pch = pchs, col = "blue", cex = 1.1)
title('Monotone fit', cex.main = 1)

persp(u, v, Fit, phi = 30, theta = -30, border = 'blue', xlab = 'x', ylab = 'y')
title('Monotone fit', cex.main = 1)

matplot(u, Fit, type = 'l', xlab = 'x', ylab = "Fit")
title('Cross-sections', cex.main = 1)

matplot(v, t(Fit), type = 'l', xlab = 'y', ylab = 'Fit')
title('Cross-sections', cex.main = 1)
```