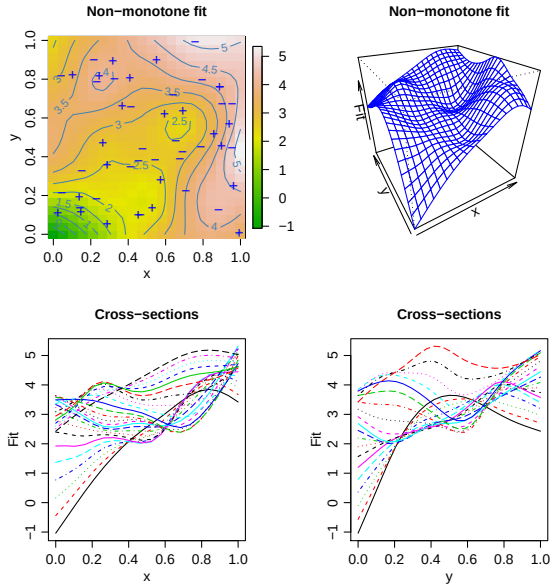
Non-monotone 2D smoothing (simulated data)



Non-monotone smoothing of simulated data with tensor product P-splines. 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 non-monotonic behaviour. R code in `f-non-monotone-2d.R`

```
# Non-monotone 2D smoothing (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

# 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, xlo, xhi, xseg)
By = bbase(y, ylo, yhi, yseg)
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)

lambdax = 1
lambday = 1
kappa = 0e8

kappax = kappa
kappay = kappa
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')
  if (dvx + dvy == 0) break
}
zhat = B %*% a
r = z - zhat

# Compute grid for predicted surface
ngx = 20
ngy = 25
xg <- seq(xlo, xhi, length = ngx)
yg <- seq(ylo, yhi, length = ngy)
Bgx = bbase(xg, xlo, xhi, xseg)
Bgy = bbase(yg, ylo, yhi, yseg)
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(xg, yg, Fit, col = terrain.colors(100), xlab = "x",
    ylab = "y")
contour(xg, yg, Fit, add = T, col = "steelblue", labcex = 0.7)
points(x, y, pch = pchs, col = "blue", cex = 1.1, )
title('Non-monotone fit', cex.main = 1)

persp(xg, yg, Fit, phi = 30, theta = -30, border = 'blue',
      xlab = "x", ylab = "y")
title('Non-monotone fit', cex.main = 1)

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

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