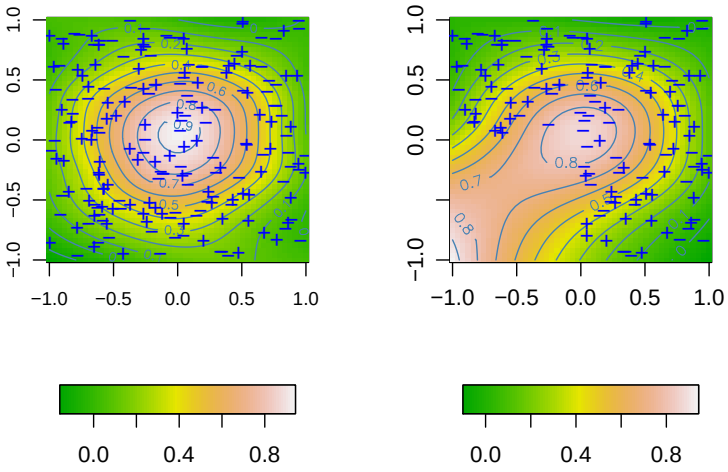
Tensor product P-spline fit and extrapolation (Ethanol data)



Fitting of scattered data with tensor product P-splines: complete data (left) and a data subset after removal of data in the lower left corner (right). The surface is extrapolated automatically, but it looks unrealistic. R code in `f-psp-corner.R`

```
# Tensor product P-spline fit and extrapolation (Ethanol data)
# A graph in the book 'Practical Smoothing. The Joys of P-splines'
# Paul Eilers and Brian Marx, 2019

library(SemiPar)
library(fields)
library(JOPS)

# simulate data
m = 200
set.seed(2017)
x = 2 * (runif(m) - 0.5)
y = 2 * (runif(m) - 0.5)
z = exp(-x^2 - y^2)^2 + rnorm(m) * 0.1
sel = x > -0 | y > -0
m = length(x)

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

# Set P-spline parameters, fit and compute surface
xseg <- 10
xdeg <- 3
xpars <- c(xlo, xhi, xseg, xdeg)
yseg <- 20
ydeg <- 3
ypars <- c(ylo, yhi, yseg, ydeg)

# Compute basis
Bx = bbase(x, xpars[1], xpars[2], xpars[3], xpars[4])
By = bbase(y, ypars[1], ypars[2], ypars[3], ypars[4])
```

```
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

# B = as.spam(B * (abs(B) > 1e-5))
n = ncol(B)
BtB = t(B) %*% B
Btz = t(B) %*% z

# Compute penalty matrices
Dx = diff(diag(nx), diff = 2)
Dy = diff(diag(ny), diff = 2)
delta = 1e-10
Px = kronecker(diag(ny), t(Dx) %*% Dx)
Py = kronecker(t(Dy) %*% Dy, diag(nx))
E = diag(n)

lambda1 = lambda2 = 1
a = solve(BtB + lambda1 * Px + lambda2 * Py, Btz)
zhat = B %*% a

# Compute grid for predicted surface
nu <- 50
nv <- 50
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)

par(mfcol = c(1, 2), mar = c(3, 3, 2, 1), mgp = c(1.6, 0.8, 0))
# Plot result and data
cols = c("blue", "red")[(z > zhat) + 1]
pchs = c("+", "-")[(z > zhat) + 1]
image.plot(u, v, Fit, col = terrain.colors(100), xlab = "", ylab = "",
    horizontal = T, legend.width = 0.7, cex = 0.8, legend.cex = 0.7,
    cex.axis = 0.8)
contour(u, v, Fit, add = T, col = "steelblue")
points(x, y, pch = pchs, col = "blue", cex = 1.1)

x = x[sel]
y = y[sel]
z = z[sel]
m = length(x)

# Compute basis
Bx = bbase(x, xpars[1], xpars[2], xpars[3], xpars[4])
By = bbase(y, ypars[1], ypars[2], ypars[3], ypars[4])
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

# B = as.spam(B * (abs(B) > 1e-5))
n = ncol(B)
BtB = t(B) %*% B
Btz = t(B) %*% z

lambda1 = lambda2 = 3
```

```
a = solve(BtB + lambda1 * Px + lambda2 * Py, Btz)
zhat = B %*% a
A = matrix(a, nx, ny)
Fit = Bgx %*% A %*% t(Bgy)

# Plot result and data
cols = c("blue", "red")[(z > zhat) + 1]
pchs = c("+", "-")[(z > zhat) + 1]
image.plot(u, v, Fit, col = terrain.colors(100), xlab = "", ylab = "",
    horizontal = T, legend.width = 0.7)
contour(u, v, Fit, add = T, col = "steelblue", labcex = 0.7)
points(x, y, pch = pchs, col = "blue", cex = 1.1)
```