Principal Nested Torii

Implementations based on the note *"Principal Nested Torii"* by James S. Marron, Andy Wood, and Huiling Le

Eduardo García-Portugués

2017-09-22

Objective

Given data $\mathbf{X}_1, \ldots, \mathbf{X}_n \in \mathbb{S}^1 \times \mathbb{S}^1 \subset \mathbb{R}^4$ we seek a rank-2 projection matrix $\mathbf{P}$ on $\mathbb{R}^4$ that optimally projects $\mathbf{X}_1, \ldots, \mathbf{X}_n$ as close as possible to $\mathbb{S}^1 \times \mathbb{S}^1$ *and* also explain as little variation as possible in the orthogonal subspace to $\mathbf{P}$. This is an *extrinsic* approach to principal components analysis on the torus: the projections $\mathbf{PX}_1, \ldots, \mathbf{PX}_n$ lay outside $\mathbb{S}^1 \times \mathbb{S}^1$ and are projected again in $\mathbb{S}^1 \times \mathbb{S}^1$.

Given a regularization parameter $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ that weights these two objectives, we look for $\mathbf{P}$ minimizing

$$\lambda_1 \sum_{i=1}^{n} \left(\frac{1}{\sqrt{2}} ||\mathbf{PX}_i|| - 1 \right)^2 + \lambda_2 \sum_{i=1}^{n} ||(\mathbf{I} - \mathbf{P})\mathbf{X}_i||^2.$$

Since $\mathbf{P}$ is a rank-2 projection matrix, it projects points in $\mathbb{R}^4$ into a 2-dimensional plane spanned by two linearly independent vectors. Then it can be parametrized as $\mathbf{P} = \mathbf{uu}' + \mathbf{vv}'$, with $\mathbf{u}, \mathbf{v} \in \mathbb{S}^3$ and $\mathbf{u} \perp \mathbf{v}$. The vectors are normalized since multiples of $\mathbf{u}$ and $\mathbf{v}$ yield the same plane. Note that this is a many-to-one parametrization[1].

Implementation

The vector $\mathbf{u} \in \mathbb{S}^3$ can be parametrized using 3-spherical coordinates as

$$\mathbf{u} = \begin{pmatrix} \cos(\theta_{u,1}) \\ \sin(\theta_{u,1})\cos(\theta_{u,2}) \\ \sin(\theta_{u,1})\sin(\theta_{u,2})\cos(\theta_{u,3}) \\ \sin(\theta_{u,1})\sin(\theta_{u,2})\sin(\theta_{u,3}) \end{pmatrix},$$

with $\theta_{u,1}, \theta_{u,2} \in [0, \pi)$ and $\theta_{u,3} \in [0, 2\pi)$. Parametrizing $\mathbf{v} \in \mathbb{S}^3$ such that $\mathbf{v} \perp \mathbf{u}$ is more challenging; in general there is no immediate form from the spherical coordinates above. But if $\mathbf{u} = \mathbf{e}_1$, then $\mathbf{v}$ is such that its first entry is zero, and can be parametrized as

$$\begin{pmatrix} 0 \\ \cos(\theta_{v,1}) \\ \sin(\theta_{v,1})\cos(\theta_{v,2}) \\ \sin(\theta_{v,1})\sin(\theta_{v,2}) \end{pmatrix},$$

with $\theta_{v,1} \in [0, \pi)$ and $\theta_{v,2} \in [0, 2\pi)$. The trick is then to rotate $\mathbf{u}$ to $\mathbf{e}_1$ using (Mardia and Jupp, 2000, page 193)

$$\mathbf{H}_{\mathbf{u}} := \frac{(\mathbf{u} + \mathbf{e}_1)(\mathbf{u} + \mathbf{e}_1)'}{1 + u_1} - \mathbf{I}.$$

Then, $\mathbf{v} = \mathbf{H}_{\mathbf{u},-1}(\cos(\theta_{v,1}), \sin(\theta_{v,1})\cos(\theta_{v,2}), \sin(\theta_{v,1})\sin(\theta_{v,2}))'$, where $\mathbf{H}_{\mathbf{u},-1}$ denotes the deletion of the first column of $\mathbf{H}_{\mathbf{u}}$. The final parametrization then reads as

[1] If we pick two orthonormal vectors in $\text{span}\{\mathbf{u}, \mathbf{v}\}$ we obtain the same plane – this corresponds to a rotation of the basis $\{\mathbf{u}, \mathbf{v}\}$ inside the plane.

$$
\begin{aligned}
\mathbf{u} &= (\cos(\theta_{u,1}), \sin(\theta_{u,1})\cos(\theta_{u,2}), \sin(\theta_{u,1})\sin(\theta_{u,2})\cos(\theta_{u,3}), \sin(\theta_{u,1})\sin(\theta_{u,2})\sin(\theta_{u,3}))', \\
\mathbf{v} &= \mathbf{H}_{\mathbf{u},-1}(\cos(\theta_{v,1}), \sin(\theta_{v,1})\cos(\theta_{v,2}), \sin(\theta_{v,1})\sin(\theta_{v,2}))', \\
\mathbf{P} &= \mathbf{u}\mathbf{u}' + \mathbf{v}\mathbf{v}',
\end{aligned}
$$

for $(\theta_{u,1}, \theta_{u,2}, \theta_{u,3}, \theta_{v,1}, \theta_{v,2}) \in [0, \pi) \times [0, \pi) \times [0, 2\pi) \times [0, \pi) \times [0, 2\pi)$ and where $\mathbf{H}_{\mathbf{u},-1}$ denotes the deletion of the first column of $\mathbf{H}_{\mathbf{u}}$.

Finally, note that this parametrization is completely scalable in the dimension ($\mathbf{H}_{\mathbf{u}}$ can be constructed for any $\mathbf{u} \in \mathbb{S}^{p-1}$).

Codes for implementation

Sphere and torus transformation functions

The following are functions to switch between angular and Cartesian coordinates of $\mathbb{S}^1 \times \overset{p}{\cdots} \times \mathbb{S}^1$ and $\mathbb{S}^{p-1}$.

```
#' @title Conversion between angular and Cartesian coordinates of the flat torus
#'
#' @description Transform the angles \eqn{(theta_1,\ldots,theta_p)} in
#' \eqn{[-\pi,\pi)^p} into the Cartesian coordinates
#' \deqn{(\cos(x_1), \sin(x_1),\ldots,\cos(x_p), \sin(x_p))}
#' of the flat torus, and viceversa.
#'
#' @param theta matrix of size \code{c(n, p)} with the angles.
#' @param x matrix of size \code{c(n, 2 * p)} with the Cartesian coordinates.
#' @return For \code{anglesToTorus}, the matrix \code{x}. For
#' \code{torusToAngles}, the matrix \code{theta}.
#' @examples
#' # Check changes of coordinates
#' torusToAngles(anglesToTorus(c(0, pi / 3, pi / 2)))
#' torusToAngles(anglesToTorus(rbind(c(0, pi / 3, pi / 2), c(0, 1, -2))))
#' anglesToTorus(torusToAngles(c(0, 1, 1, 0)))
#' anglesToTorus(torusToAngles(rbind(c(0, 1, 1, 0), c(0, 1, 0, 1))))
#' @author Eduardo Garcia-Portugues (\email{edgarcia@@est-econ.uc3m.es}).
#' @export
anglesToTorus <- function(theta) {

  # Convert to matrix
  if (!is.matrix(theta)) {

    theta <- matrix(theta, nrow = 1)

  }

  # Apply transformation
  p <- ncol(theta)
  ind <- c(rbind(1:p, (p + 1):(2 * p)))
  cbind(cos(theta), sin(theta))[, ind]

}

#' @rdname anglesToTorus
```

```r
#' @export
torusToAngles <- function(x) {

  # Convert to matrix
  if (!is.matrix(x)) {

    x <- matrix(x, nrow = 1)

  }

  # Check p
  if (ncol(x) %% 2) {

    stop("p is not even")

  }

  # Apply transformation
  n <- nrow(x)
  p2 <- ncol(x)
  indCos <- seq(1, p2, by = 2)
  indSin <- seq(2, p2, by = 2)
  matrix(atan2(y = x[, indSin], x = x[, indCos]), nrow = n,
         ncol = p2 / 2L)[, , drop = TRUE]

}

#' @title Conversion between angular and Cartesian coordinates of the
#' (hyper)sphere
#'
#' @description Transform the angles \eqn{(theta_1,\ldots,theta_p)} in
#' \eqn{[0,\pi)^{p-1}\times[-\pi,\pi)} into the Cartesian coordinates
#' \deqn{(\cos(x_1),\sin(x_1)\cos(x_2),\sin(x_1)\sin(x_2)\cos(x_3),\ldots,
#' \sin(x_1)\cdots\sin(x_{p-1})\cos(x_p),
#' \sin(x_1)\cdots\sin(x_{p-1})\sin(x_p))}
#' of the \eqn{p}-sphere in the \eqn{p+1}-Euclidean space, and viceversa.
#'
#' @param theta matrix of size \code{c(n, p)} with the angles.
#' @param x matrix of size \code{c(n, p + 1)} with the Cartesian coordinates.
#' @return For \code{anglesToSphere}, the matrix \code{x}. For
#' \code{sphereToAngles}, the matrix \code{theta}.
#' @examples
#' # Check changes of coordinates
#' sphereToAngles(anglesToSphere(c(pi / 2, 0, pi)))
#' sphereToAngles(anglesToSphere(rbind(c(pi / 2, 0, pi), c(pi, pi / 2, 0))))
#' anglesToSphere(sphereToAngles(c(0, sqrt(0.5), sqrt(0.1), sqrt(0.4))))
#' anglesToSphere(sphereToAngles(rbind(c(0, sqrt(0.5), sqrt(0.1), sqrt(0.4)),
#'                                     c(0, sqrt(0.5), sqrt(0.5), 0),
#'                                     c(0, 1, 0, 0),
#'                                     c(0, 0, 0, -1),
#'                                     c(0, 0, 1, 0))))
#' @author Eduardo Garcia-Portugues (\email{edgarcia@@est-econ.uc3m.es}).
#' @export
```

```r
anglesToSphere <- function(theta) {

  # Convert to matrix
  if (!is.matrix(theta)) {

    theta <- matrix(theta, nrow = 1)

  }

  # Apply transformation
  q <- ncol(theta)
  x <- matrix(0, nrow = nrow(theta), ncol = q + 1)
  cosTheta <- cos(theta)
  sinTheta <- sin(theta)
  prodSinTheta <- t(apply(sinTheta, 1, cumprod))
  cbind(cosTheta[, 1, drop = FALSE],
        prodSinTheta[, -q, drop = FALSE] * cosTheta[, -1, drop = FALSE],
        prodSinTheta[, q, drop = FALSE])[, , drop = TRUE]

}

#' @rdname anglesToSphere
#' @export
sphereToAngles <- function(x) {

  # Convert to matrix
  if (!is.matrix(x)) {

    x <- matrix(x, nrow = 1)

  }

  # Apply transformation
  q <- ncol(x) - 1
  iNorm <- t(apply(x^2, 1, function(x) rev(sqrt(cumsum(rev(x))))[, -(q + 1)]
  theta <- x[, -c(q + 1), drop = FALSE] / iNorm
  theta[is.nan(theta)] <- 1 # Avoid NaNs
  theta <- acos(theta)
  xq <- (x[, q + 1] < 0)
  theta[, q] <- (2 * pi) * xq + (1 - 2 * xq) * theta[, q]
  theta[, , drop = TRUE]

}
```

Parametrization of P

Functions for parametrizing $\mathbf{P}$ and computing $\mathbf{u}$ and $\mathbf{v}$.

```r
#' @title Rotation matrix that rotates a given unit vector to
#' \eqn{(1,0,\ldots,0)}
#'
#' @description Computes the rotation matrix that translates the vector
#' \eqn{u=(u_1,\ldots,u_{p+1})} of the \eqn{p}-sphere into
#' \eqn{e_1=(1,0,\ldots,0)}:
```

```
#' \deqn{H_u=(u+e_1)(u+e_1)' / (1 - u_1) - I.}
#'
#' @param u a unit vector of size \code{p + 1}.
#' @return Rotation matrix \eqn{H_u} of size \code{p + 1}.
#' @examples
#' # p = 2
#' u <- sqrt(c(0.1, 0.3, 0.6))
#' H <- Hu(u = u)
#' H %*% u
#' H - solve(H)
#'
#' # p = 100
#' u <- rnorm(n = 101); u <- u / sqrt(sum(u^2))
#' H <- Hu(u = u)
#' sum(abs(c(1, rep(0, 100)) - H %*% u))
#' sum(abs(H - solve(H)))
#' @author Eduardo Garcia-Portugues (\email{edgarcia@@est-econ.uc3m.es}).
#' @export
Hu <- function(u) {

  # Arbitrary dimension
  p <- length(u)
  I <- diag(1, nrow = p, ncol = p)

  # Check for u[1] == -1 for avoid dividing by 0
  if (u[1] != -1) {

    u[1] <- u[1] + 1
    tcrossprod(u) / u[1] - I

  } else {

    -I

  }

}

#' @title Parametrization of a rank-2 projection matrix
#'
#' @description Parametrization of a rank-2 projection matrix \eqn{P} of size
#' \code{p} by means of \eqn{P=uu'+vv'}, where \eqn{u} and \eqn{v} are unit
#' orthogonal vectors.
#'
#' @param theta a vector of size \code{2 * p - 3} parametrizing the vectors
#' \eqn{u} and \eqn{v}. The first \code{p - 1} entries parametrize \eqn{u},
#' and the last \code{p - 2} parametrize \eqn{v} (orthogonal to \eqn{u}).
#' All entries must be in \eqn{[0,\pi)}, except the \code{p - 1} and the
#' \code{2 * p - 3}, which must be in \eqn{[-\pi,\pi)} (or \eqn{[0,2*\pi)}).
#'
#' @return A rank-2 projection matrix of size \code{p}.
#' @examples
#' P <- PTheta(theta = c(pi / 3, pi / 3, -pi / 2, 0, -pi / 2))
```

```r
#' sum(abs(P - P %*% P))
#' eigen(P)
#' @author Eduardo Garcia-Portugues (\email{edgarcia@@est-econ.uc3m.es}).
#' @export
PTheta <- function(theta) {

  # Get p and compute u
  p <- (length(theta) + 3) / 2L
  u <- anglesToSphere(theta = theta[1:(p - 1)])
  uu <- tcrossprod(u)

  # Do not call Hu and reuse uu' for saving a matrix multiplication
  I <- diag(1, nrow = p, ncol = p)
  if (u[1] != -1) {

    ue1 <- uu
    ue1[1, ] <- ue1[1, ] + u
    ue1[, 1] <- ue1[, 1] + u
    ue1[1, 1] <- ue1[1, 1] + 1
    v <- (ue1 / (1 + u[1]) - I)[, -1] %*%
      anglesToSphere(theta = theta[p:(2 * p - 3)])

  } else {

    v <- c(0, -anglesToSphere(theta = theta[p:(2 * p - 3)]))

  }

  # Slower than the previous code chunk
  # v <- Hu(u = u)[, -1] %*% anglesToSphere(theta = theta[p:(2 * p - 3)])

  # P
  uu + tcrossprod(v)

}

#' @title Gram-Schmidt completion of a unit vector to an orthonormal basis
#'
#' @description Completes a unit vector \eqn{u} size \code{p + 1} to an
#' orthonormal basis \eqn{\{u, b_1, \ldots, b_p\}}.
#'
#' @param u a unit vector of size \code{p + 1}.
#'
#' @return A matrix of size \code{p + 1} with orthonormal columns
#' \eqn{\{u, b_1, \ldots, b_p\}}.
#' @examples
#' # p = 2
#' u <- sqrt(c(0.0, 0.4, 0.6))
#' H <- gramSchmidt(u = u)
#' u - H[, 1]
#' H %*% t(H)
#'
#' # p = 100
```

```r
#' u <- rnorm(n = 101); u <- u / sqrt(sum(u^2))
#' H <- gramSchmidt(u = u)
#' sum(abs(u - H[, 1]))
#' sum(abs(H %*% t(H)))
#' @author Eduardo Garcia-Portugues (\email{edgarcia@@est-econ.uc3m.es}).
#' @export
gramSchmidt <- function(u) {

  # The vectors in U are linearly independent if u[1] != 0
  p <- length(u)
  U <- cbind(u, diag(1, nrow = p, ncol = p)[, -1])
  normU <- rep(1, p)

  # Efficient Gram-Schmidt for U
  for (k in 2:p) {

    # Slower
    # U[, k] <- U[, k] -
    #   rowSums(sapply(1:(k - 1), function(j)
    #     sum(U[, k] * U[, j]) / normU[j] * U[, j]))

    # Normalize (implicit column recycling)
    U[, k] <- U[, k] -
      colSums(drop(U[, k] %*% U[, 1:(k - 1)]) /
                normU[1:(k - 1)] * t(U[, 1:(k - 1)]))
    normU[k] <- sum(U[, k] * U[, k])

  }

  # Normalize (implicit column recycling)
  U <- t(t(U) / sqrt(colSums(U^2)))

  # Patch if u[1] = 0
  if (u[1] == 0) {

    U[, p] <- c(1, rep(0, p - 1))

  }

  return(U)

}
```

Principal Nested Torii

Main function minimizing the objective function for $\boldsymbol{\lambda}$.

```r
# The sdetorus package is required - install it with
# install.packages("devtools")
# devtools::install_github("egarpor/sdetorus", dep = TRUE)

# Load sdetorus
library(sdetorus)
```

```r
#' @title Principal Nested Torii
#'
#' @description Computes and plots Principal Nested Torii in dimension two.
#'
#' @param X,theta data input. Either \code{X}, a matrix of size
#' \code{c(n, 2 * p)} with the Cartesian coordinates or \code{theta}, a matrix
#' of size \code{c(n, p)} with the angular coordinates.
#' @param lambda regularization vector. Defaults to \code{c(1, 1)}.
#' @param drawPlot flag to indicate whether a plot should be draw.
#' @param usePTheta flag to indicate whether the \code{PTheta} parametrization
#' for \eqn{P} should be used. Defaults to \code{TRUE} since it is the fastest
#' parametrization.
#' @param useGram flag to indicate whether a Gram-Schmidt-like parametrization
#' for \eqn{P} shoud be used.
#' @param nStart the number of random starting points for minimizing the
#' objective function.
#'
#' @return A list with the following elements:
#' \itemize{
#'   \item \code{u, v}: vectors \eqn{u} and \eqn{v}.
#'   \item \code{P}: projection matrix \eqn{P}.
#'   \item \code{PX, IPX}: projected data in \eqn{P} and \eqn{I -P},
#' respectively.
#'   \item \code{opt}: optimization output from \code{mleOptimWrapper}.
#' }
#' @examples
#' p <- 2
#' n <- 1e2
#' set.seed(345678)
#' X <- anglesToTorus(matrix(2 * pi * runif(p * n), nrow = n, ncol = p))
#' princNestTorii(X = X)
#' @author Eduardo Garcia-Portugues (\email{edgarcia@@est-econ.uc3m.es}).
#' @export
princNestTorii <- function(X, theta, lambda = c(1, 1), drawPlot = TRUE,
                           usePTheta = TRUE, useGram = FALSE, nStart = 20) {

  # Input arguments
  if (missing(X)) {

    if (missing(theta)) {

      stop("X or theta must be provided")

    } else {

      X <- anglesToTorus(theta = theta)

    }

  }

  # Identity
  p <- ncol(X) / 2L
```

```r
I <- diag(1, nrow = 2 * p, ncol = 2 * p)

# Function to minimize
p1 <- 2 * p - 1
p2 <- 4 * p - 3
objFunction <- function(th) {

  # Parametrize P as u, v in S^3 and u'v = 0
  if (!usePTheta) {

    if (useGram) {

      u <- anglesToSphere(theta = th[1:p1])
      v <- drop(gramSchmidt(u = u)[, -1] %*%
                  anglesToSphere(theta = th[(p1 + 1):p2]))

    } else {

      u <- anglesToSphere(theta = th[1:p1])
      v <- drop(Hu(u = u)[, -1] %*% anglesToSphere(theta = th[(p1 + 1):p2]))

    }
    P <- tcrossprod(u) + tcrossprod(v)

  } else {

    P <- PTheta(theta = th)

  }

  # Data projections
  PX <- X %*% P
  IPX <- X %*% (I - P)

  # Minimization of:
  # lambda1 * (projections distance to the torus) +
  # lambda2 * (variance explained by second component)
  lambda[1] * sum((sqrt(0.5 * rowSums(PX^2)) - 1)^2) +
    lambda[2] * sum(sqrt(rowSums(IPX^2)))

}

# Minimization using several starting values
set.seed(234567)
thRand <- cbind(runif(nStart, 0, pi), runif(nStart, 0, pi),
                runif(nStart, -pi, pi), runif(nStart, 0, pi),
                runif(nStart, -pi, pi))
res <- mleOptimWrapper(minusLogLik = objFunction, start = rbind(thRand),
                       lower = c(0, 0, -pi, 0, -pi), upper = rep(pi, 5),
                       checkCircular = TRUE, maxit = 500, verbose = 0,
                       selectSolution = "lowest")

# Result
```

```
u <- anglesToSphere(theta = res$par[1:p1])
if (useGram) {

  v <- drop(gramSchmidt(u = u)[, -1] %*%
              anglesToSphere(theta = res$par[(p1 + 1):p2]))

} else {

  v <- drop(Hu(u = u)[, -1] %*% anglesToSphere(theta = res$par[(p1 + 1):p2]))

}
P <- tcrossprod(u) + tcrossprod(v)
I <- diag(1, nrow = 2 * p, ncol = 2 * p)
PX <- X %*% P
IPX <- X %*% (I - P)

if (drawPlot & p == 2) {

  # Projections
  dat <- torusToAngles(X)
  dat1 <- torusToAngles(PX)
  dat2 <- torusToAngles(IPX)

  # Plot data
  plot(dat, xlim = c(-pi, pi), ylim = c(-pi, pi), axes = FALSE,
       xlab = expression(theta[1]), ylab = expression(theta[2]),
       main = substitute(lambda == "(" * l1 * ", " * l2 * "), " * "obj = " * o,
                         env = list(l1 = lambda[1], l2 = lambda[2],
                                    o = res$value)))
  axis(1, at = seq(-pi, pi, by = pi/2),
       labels = expression(-pi, -pi/2, 0, pi/2, pi))
  axis(2, at = seq(-pi, pi, by = pi/2),
       labels = expression(-pi, -pi/2, 0, pi/2, pi))
  abline(v = c(-pi, pi), col = "gray")
  abline(h = c(-pi, pi), col = "gray")
  box()

  # Add principal curves
  # First: spanned by u and v
  thGrid <- seq(-pi, pi, l = 200)
  xGrid <- cbind(cos(thGrid), sin(thGrid))
  linesTorus(x = atan2(xGrid %*% c(u[2], v[2]), xGrid %*% c(u[1], v[1])),
             y = atan2(xGrid %*% c(u[4], v[4]), xGrid %*% c(u[3], v[3])),
             col = 3)

  # Second: spanned by vectors ortogonal to u and v
  uv <- eigen(I - P)$vectors[, 1:2]
  linesTorus(x = atan2(xGrid %*% c(uv[2, 1], uv[2, 2]),
                       xGrid %*% c(uv[1, 1], uv[1, 2])),
             y = atan2(xGrid %*% c(uv[4, 1], uv[4, 2]),
                       xGrid %*% c(uv[3, 1], uv[3, 2])),
             col = 2)
```

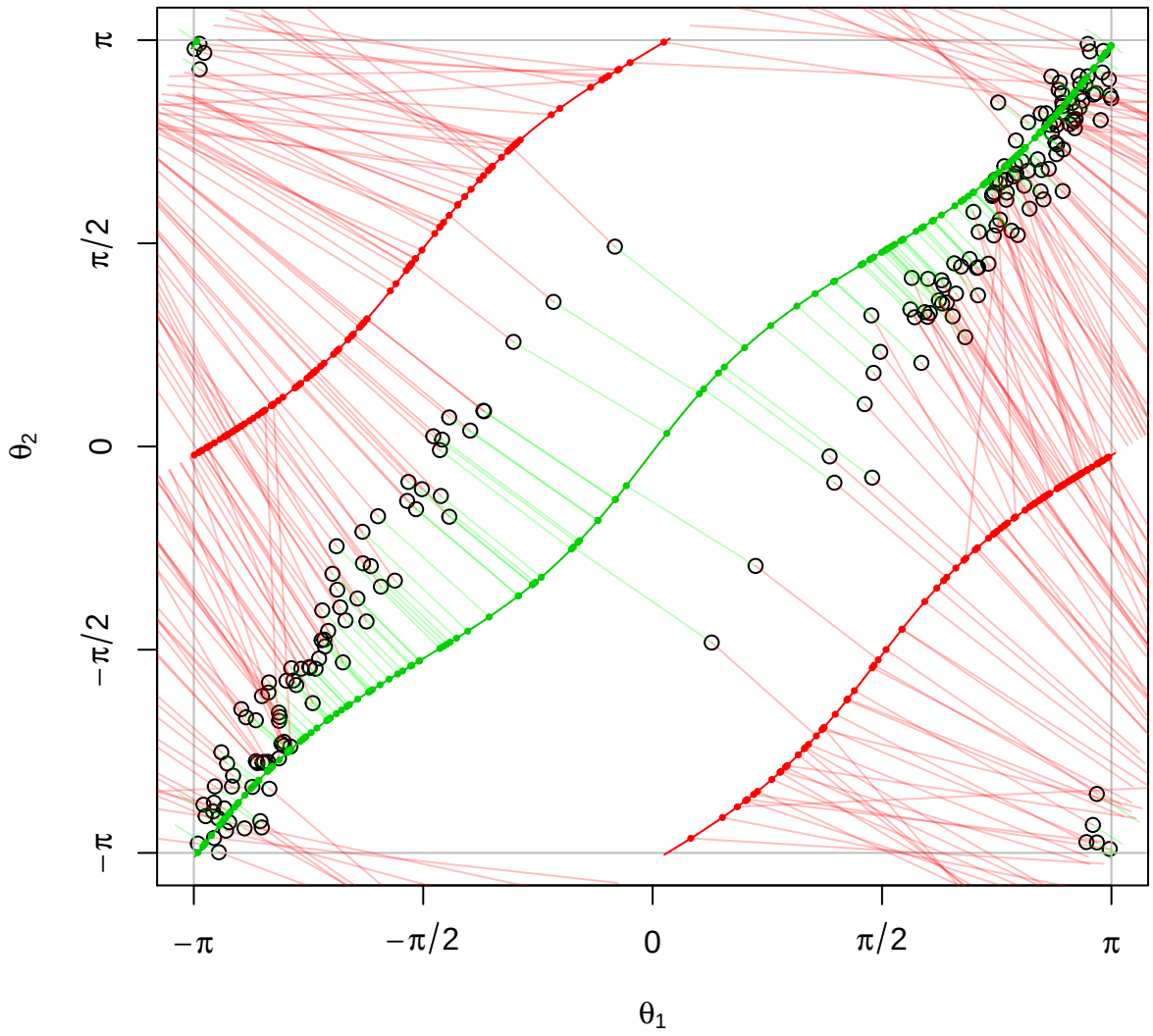
```
    # Plot 2d projections in principal curves
    points(dat1, col = 3, cex = 0.5, pch = 16)
    points(dat2, col = 2, cex = 0.5, pch = 16)
    for (i in 1:nrow(X)) {
      linesTorus(x = c(dat[i, 1], dat1[i, 1]), y = c(dat[i, 2], dat1[i, 2]),
                 col = rgb(0, 1, 0, alpha = 0.25))
      linesTorus(x = c(dat[i, 1], dat2[i, 1]), y = c(dat[i, 2], dat2[i, 2]),
                 col = rgb(1, 0, 0, alpha = 0.25))
    }

  }

  # Result
  return(list("u" = u, "v" = v, "P" = P,
              "PX" = PX, "IPX" = IPX, "opt" = res))

}
```

Application to some datasets

Green stands for the **first principal component**. *Red* stands for the *second principal component*. We use $\boldsymbol{\lambda} = (1, 1)$.

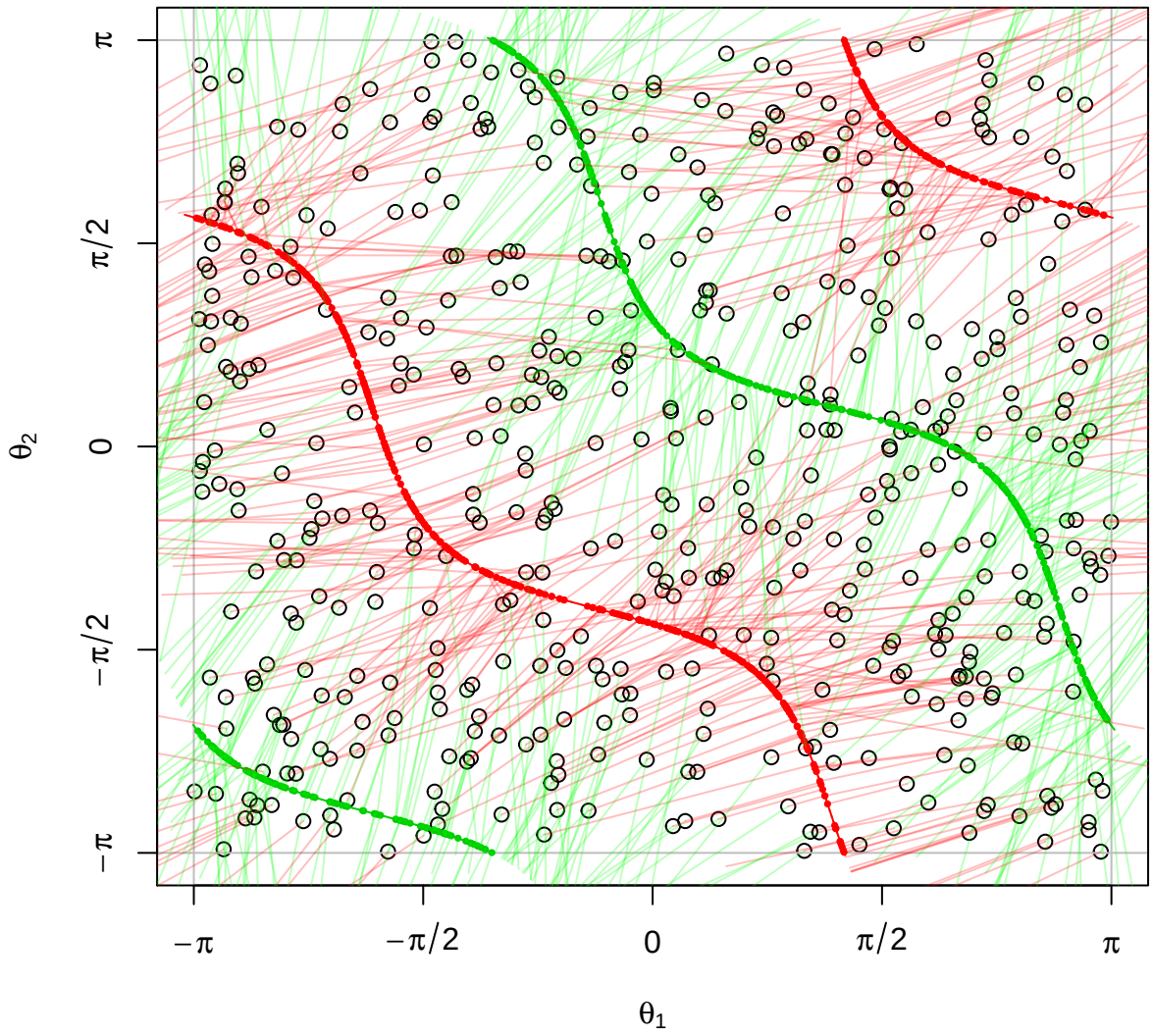
Steve's dataset

```
# X <- Steve's dataset
res <- princNestTorii(X = X)
```

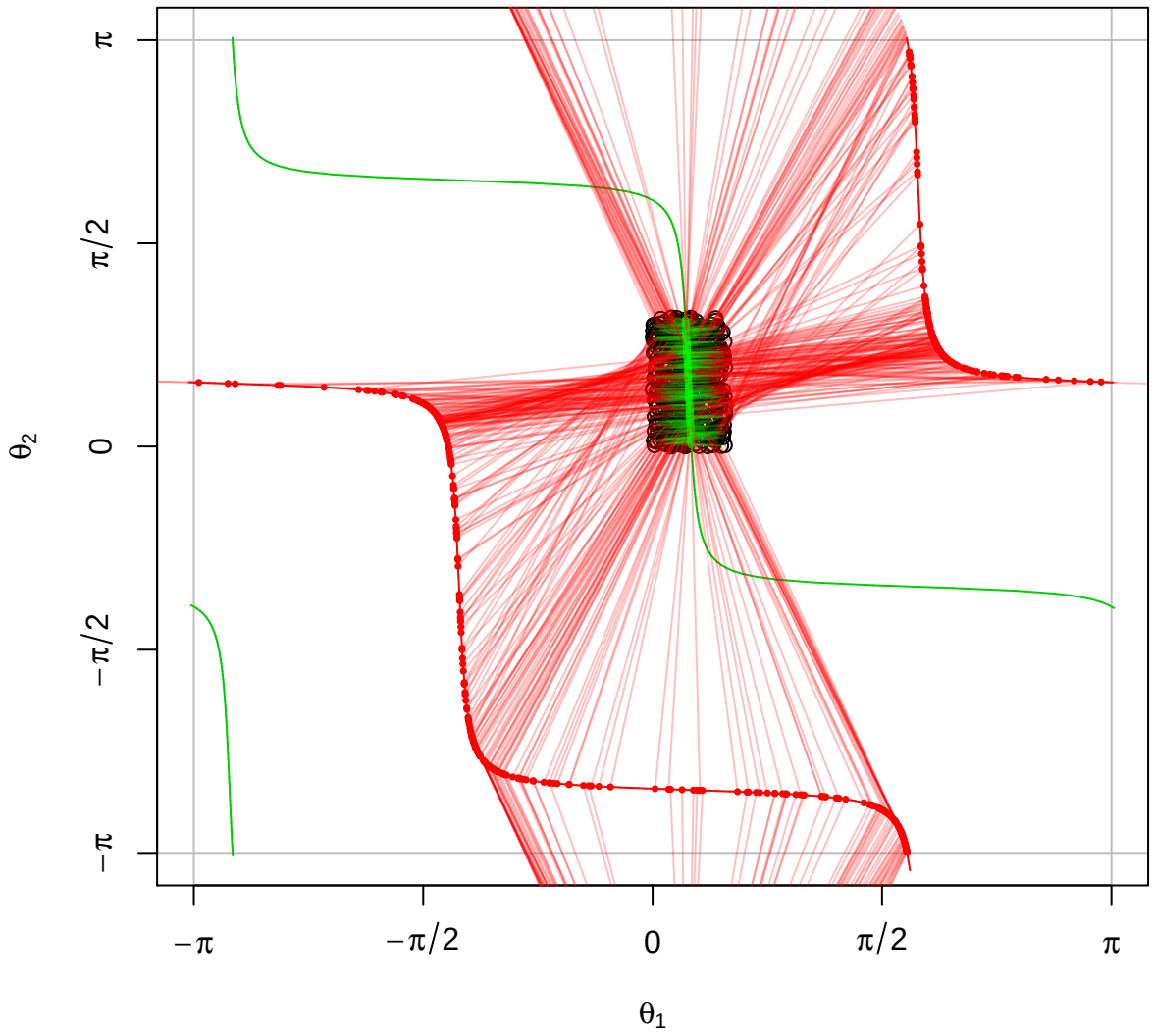


Uniform-like

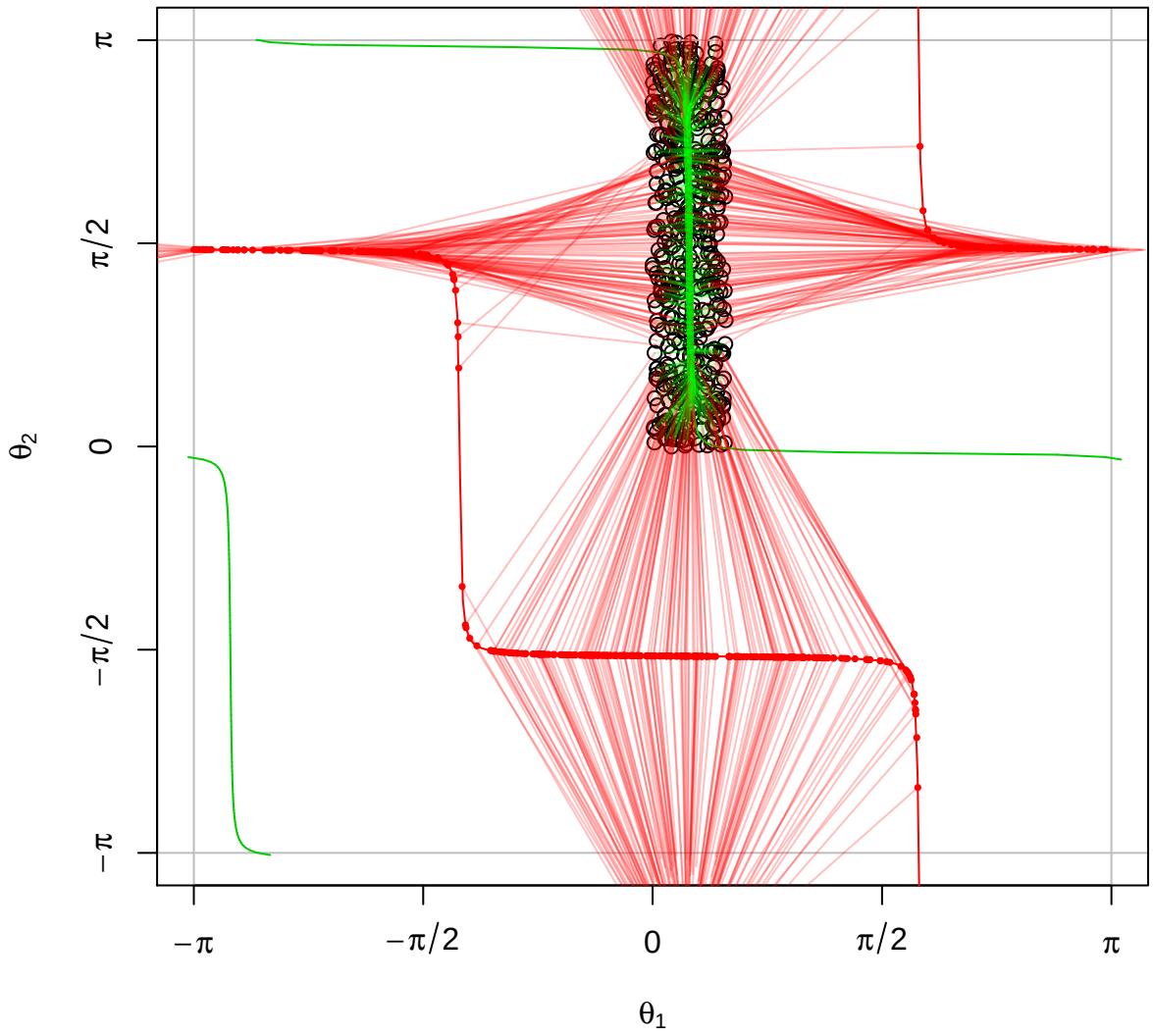
```
# Uniform in [-pi, pi) x [-pi, pi)
n <- 500
set.seed(123456)
X <- anglesToTorus(matrix(2 * pi * runif(2 * n), nrow = n, ncol = 2))
res <- princNestTorii(X = X)
```



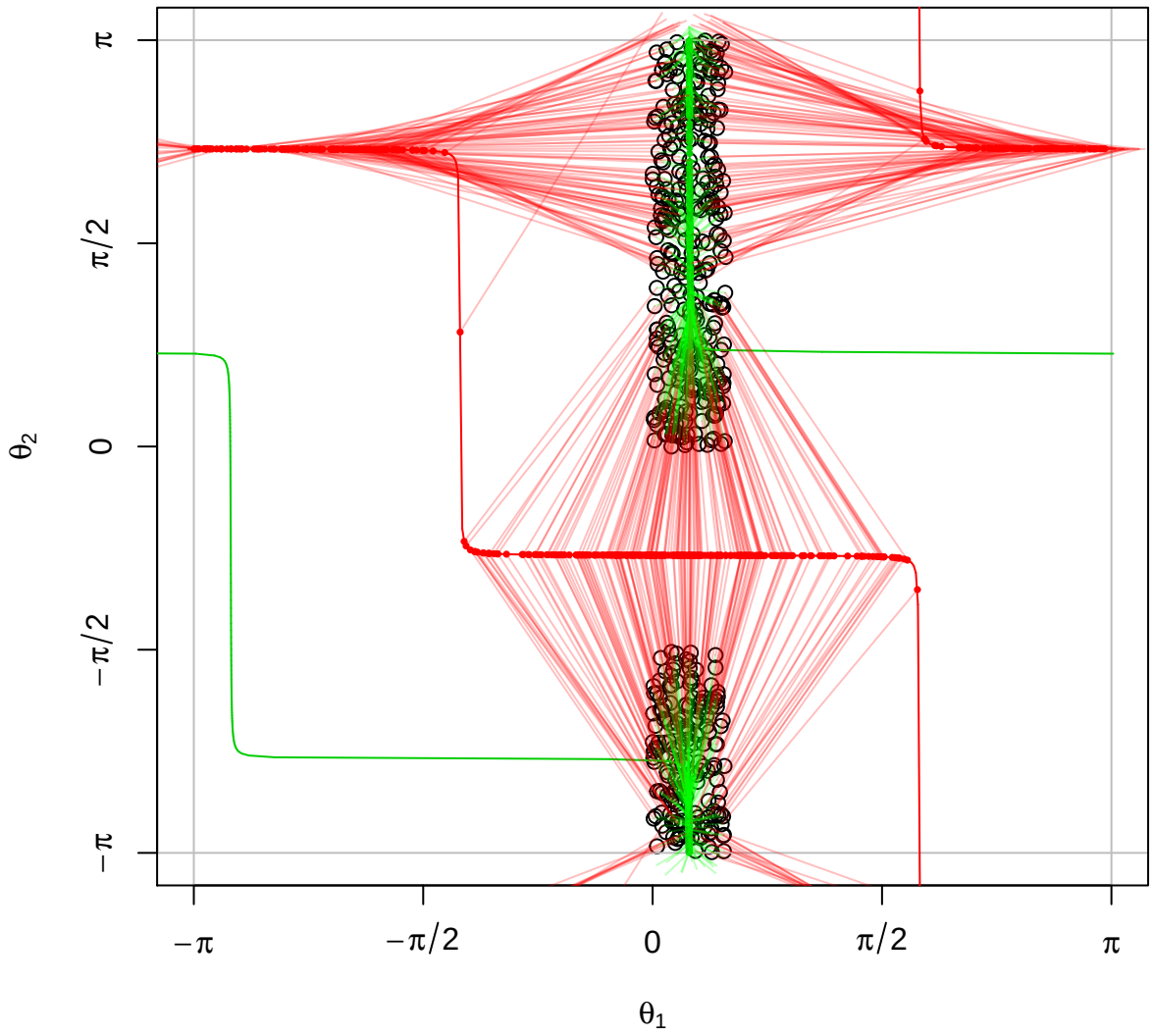

```
# Uniform vertical band of length 1
n <- 500
set.seed(123456)
X <- anglesToTorus(cbind(0.5 * runif(n), 1 * runif(n)))
res <- princNestTorii(X = X)
```




```
# Uniform vertical band of length pi
n <- 500
set.seed(123456)
X <- anglesToTorus(cbind(0.5 * runif(n), pi * runif(n)))
res <- princNestTorii(X = X)
```

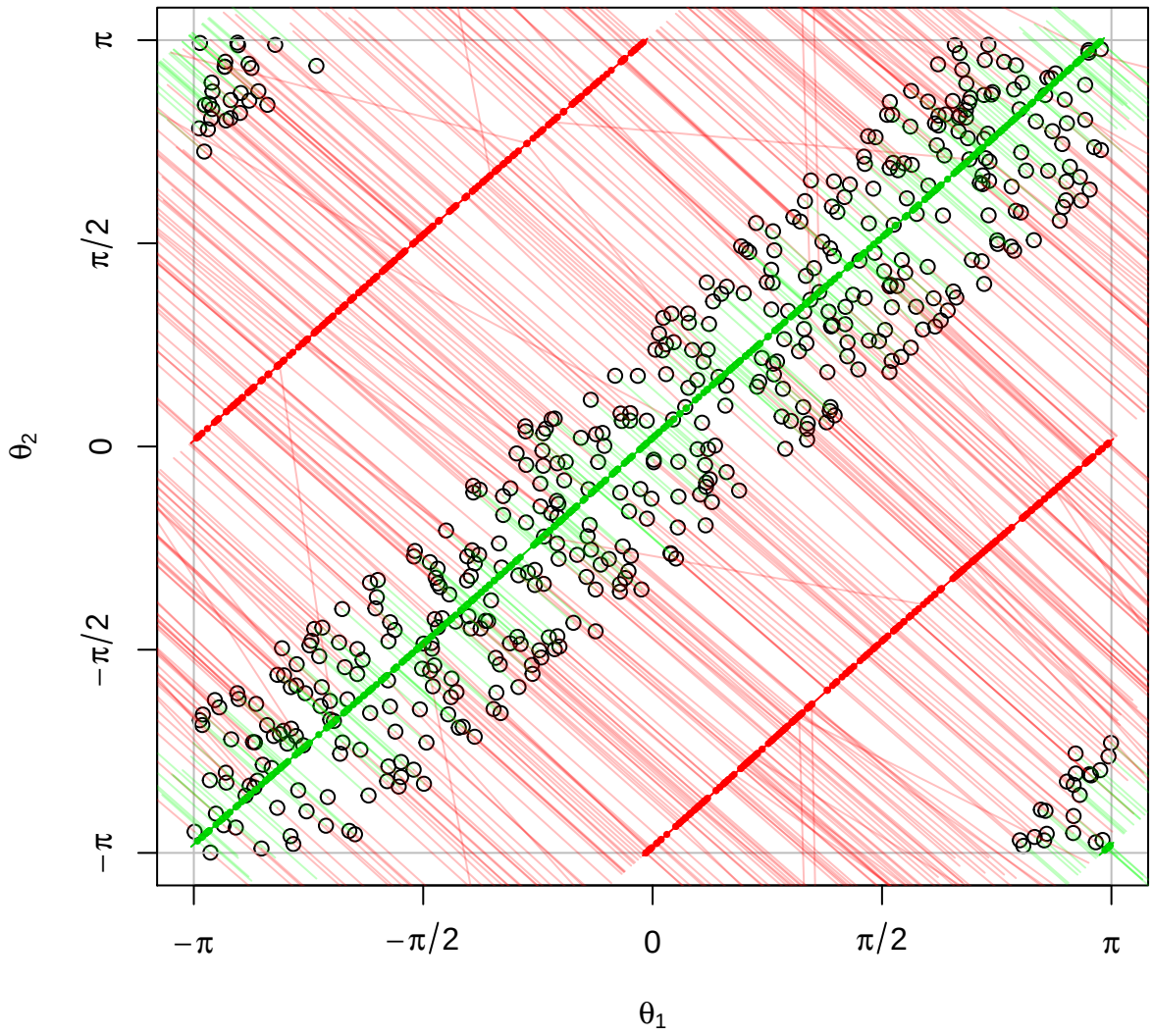



```
# Uniform vertical band of length 3 * pi / 2
n <- 500
set.seed(123456)
X <- anglesToTorus(cbind(0.5 * runif(n), 3 * pi / 2 * runif(n)))
res <- princNestTorii(X = X)
```

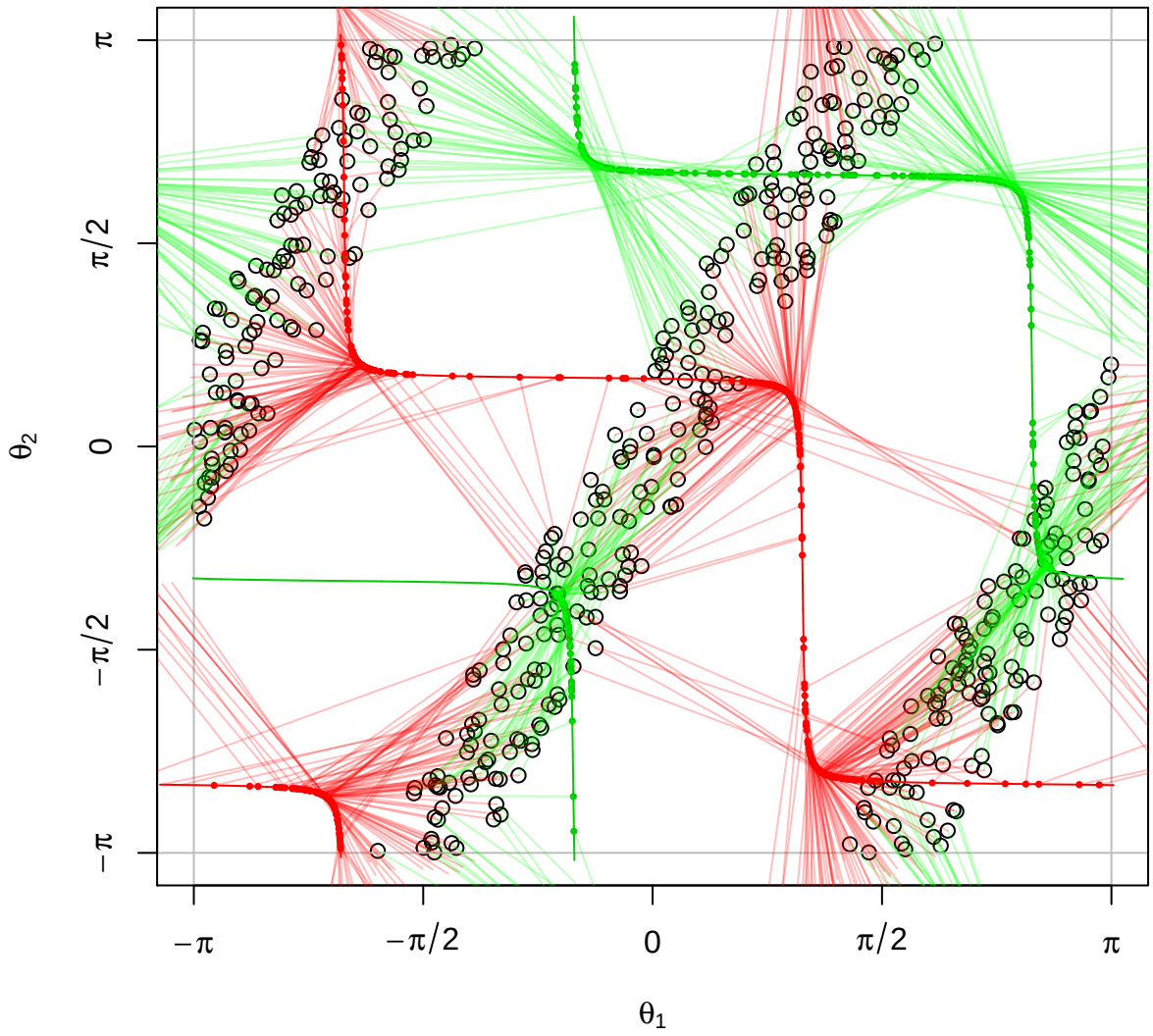


Diagonal-like

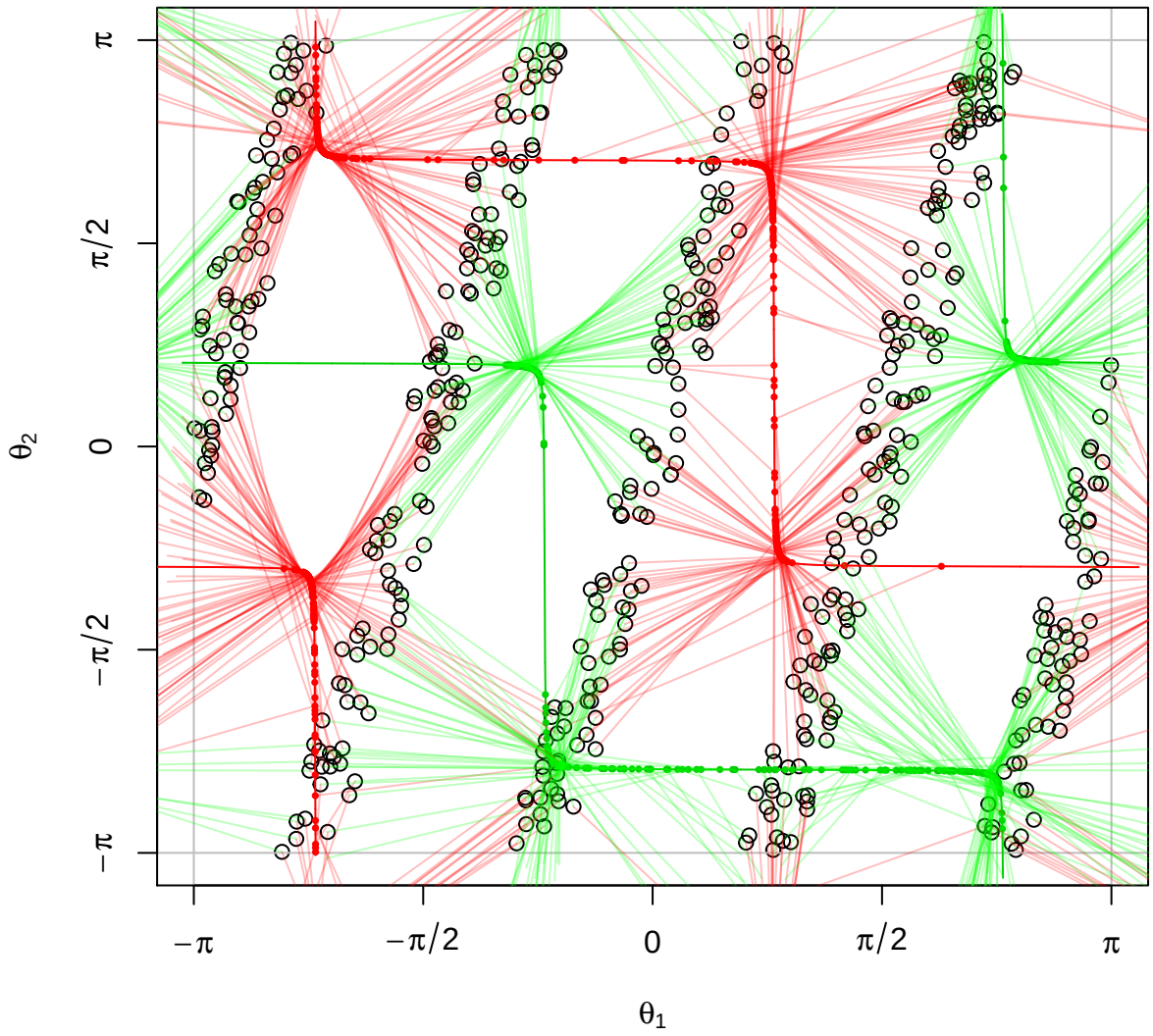
```
# Diagonal
n <- 500
set.seed(123456)
x <- 2 * pi * runif(n)
y <- x + runif(n, -pi / 3, pi / 3)
X <- anglesToTorus(cbind(x, y))
res <- princNestTorii(X = X)
```



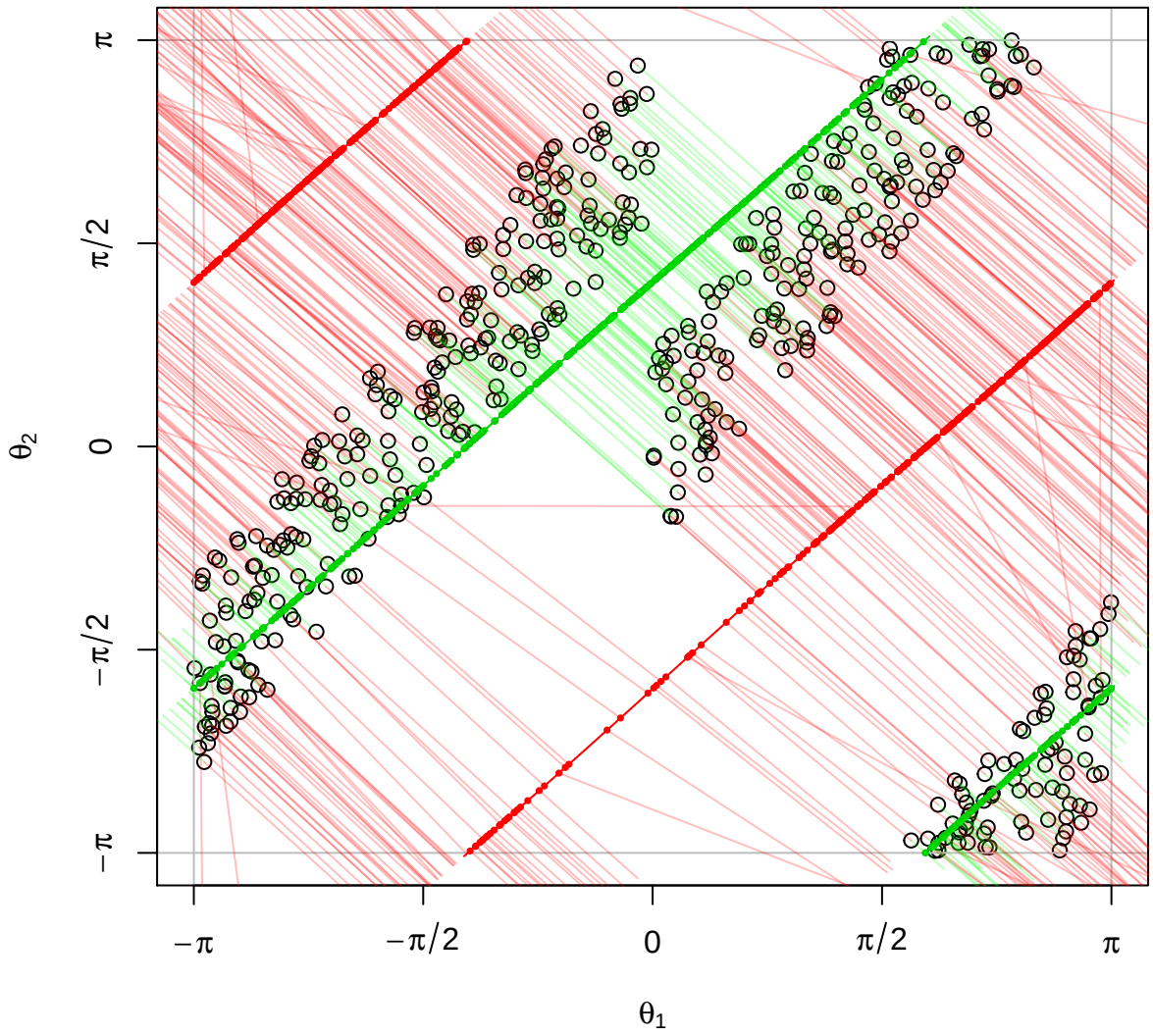

```
# Data along y = 2 * x
n <- 500
set.seed(123456)
x <- 2 * pi * runif(n)
y <- 2 * x + runif(n, -pi / 4, pi / 4)
X <- anglesToTorus(cbind(x, y))
res <- princNestTorii(X = X)
```




```
# Data along y = 4 * x
n <- 500
set.seed(123456)
x <- 2 * pi * runif(n)
y <- 4 * x + runif(n, -pi / 4, pi / 4)
X <- anglesToTorus(cbind(x, y))
res <- princNestTorii(X = X)
```

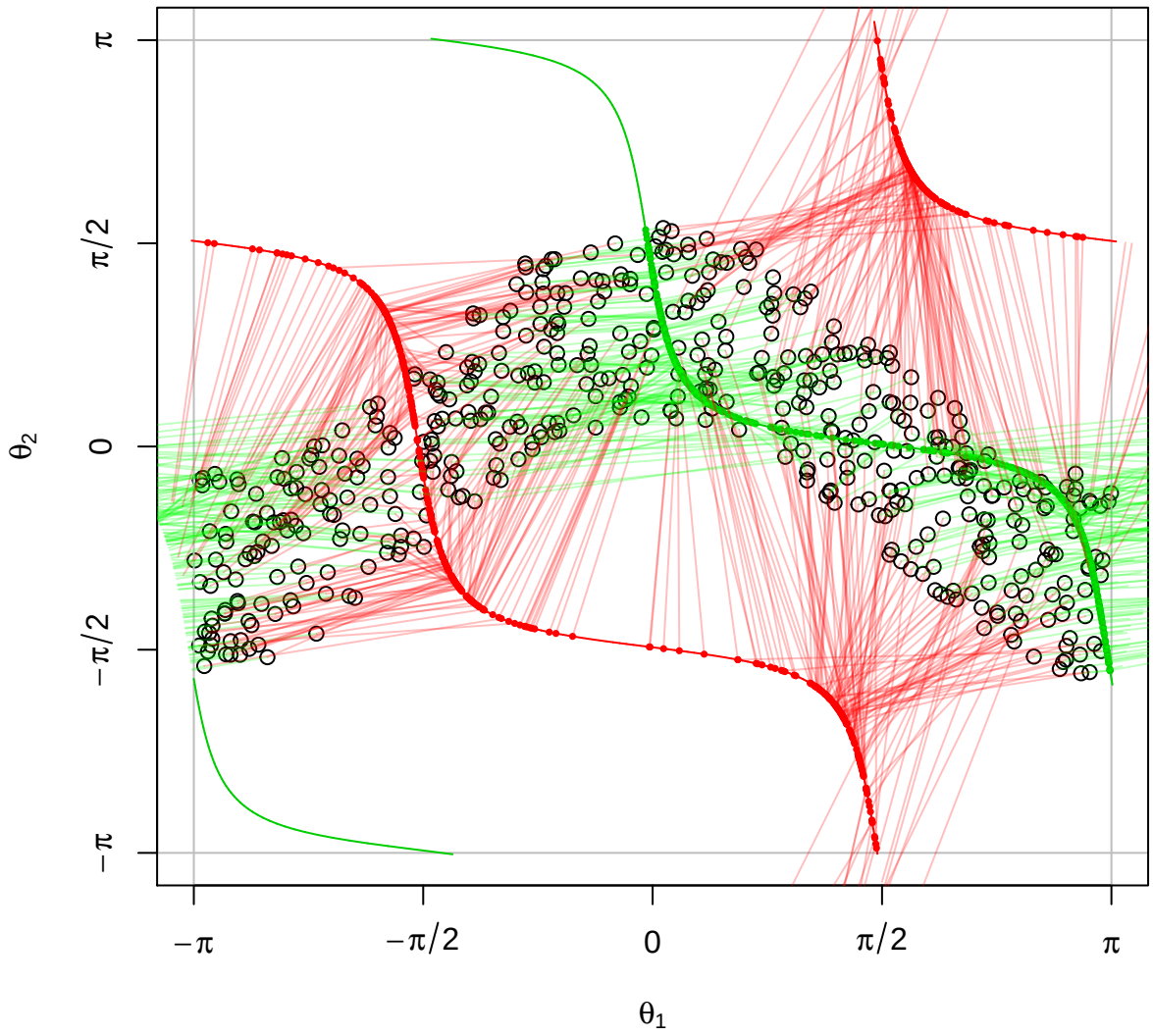



```
# Data along y = sqrt(2) * x
n <- 500
set.seed(123456)
x <- 2 * pi * runif(n)
y <- sqrt(2) * x + runif(n, -pi / 4, pi / 4)
X <- anglesToTorus(cbind(x, y))
res <- princNestTorii(X = X)
```

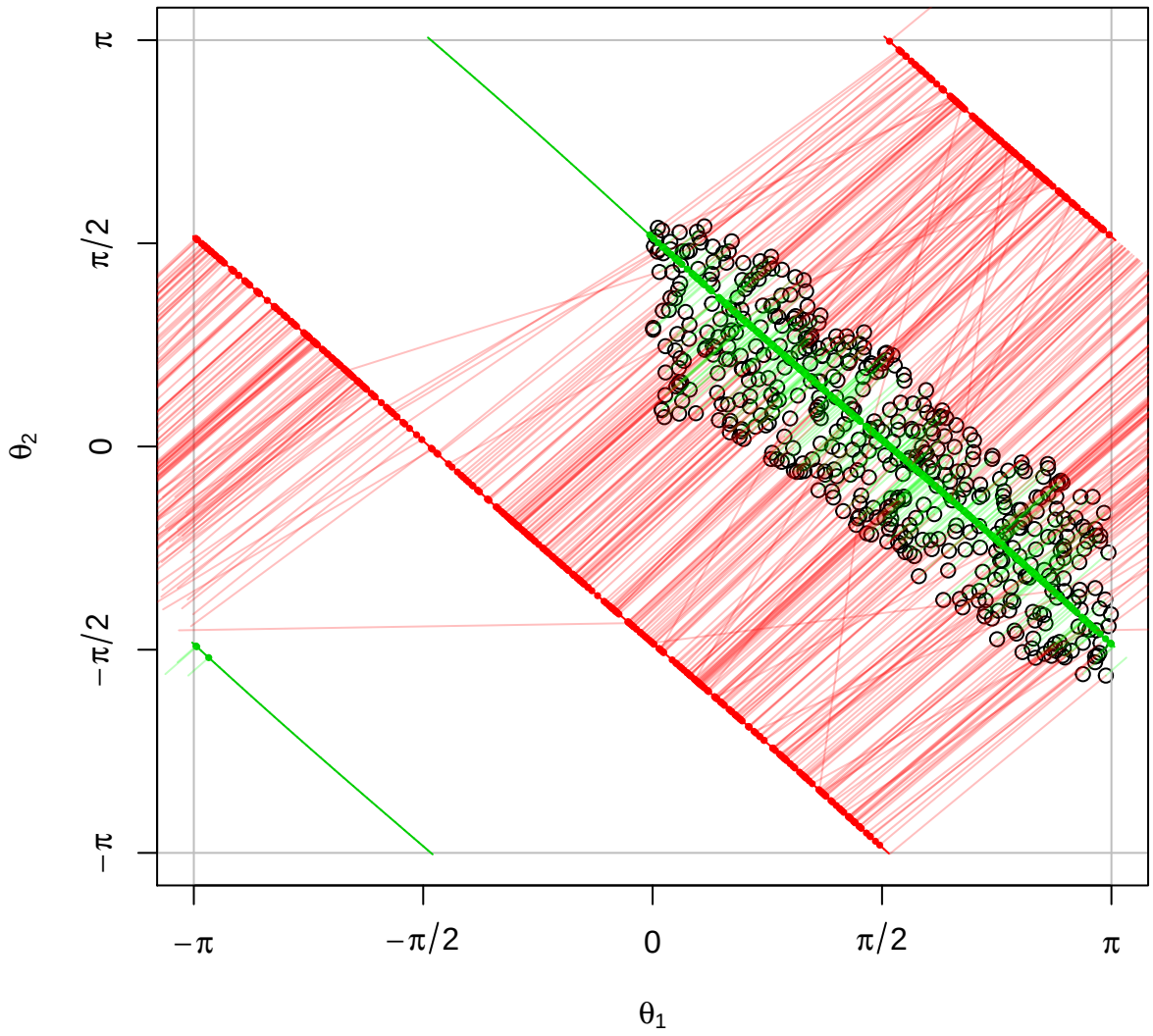


Trigonometric-like

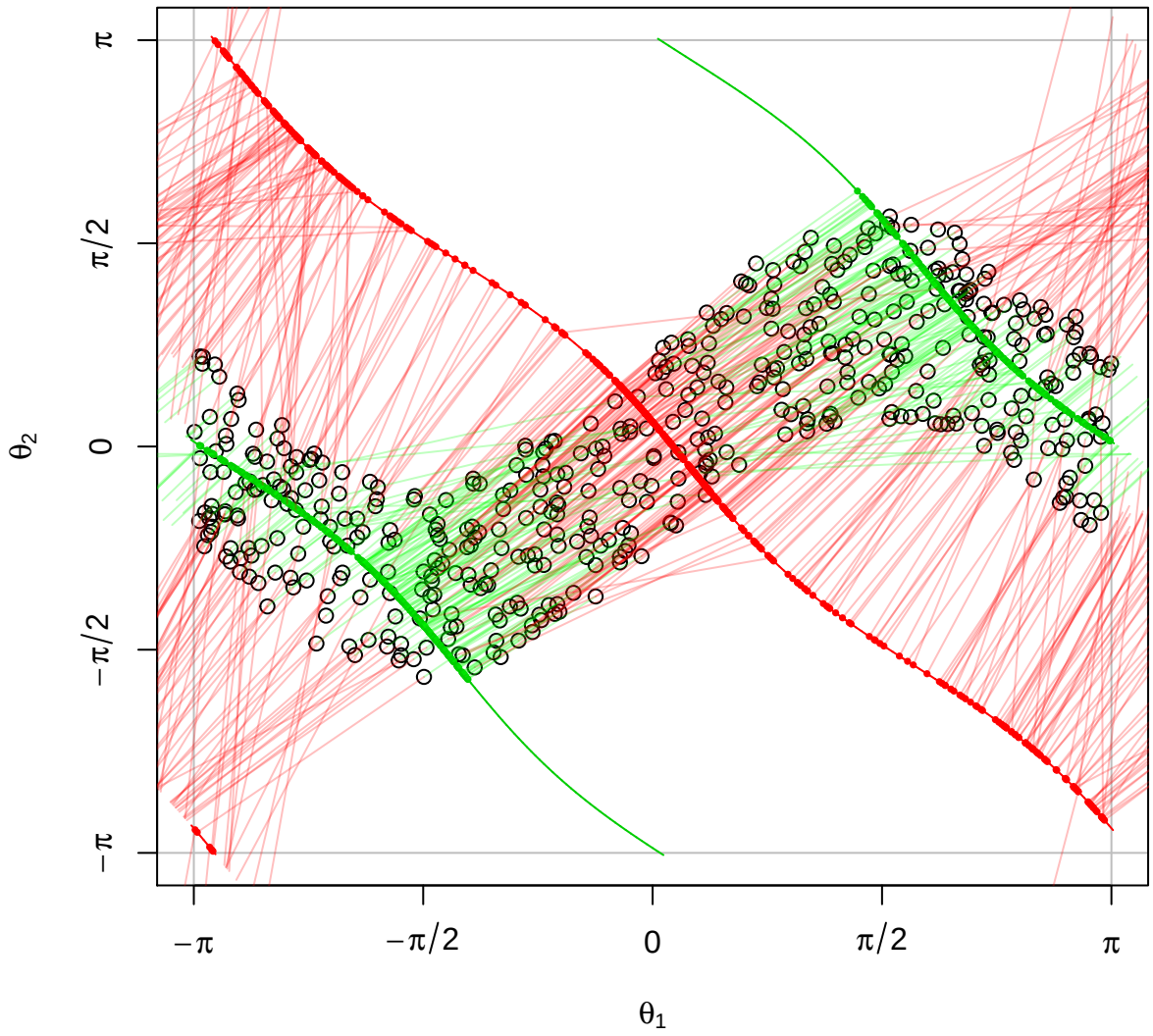
```
# Data along y = cos(x)
n <- 500
set.seed(123456)
x <- 2 * pi * runif(n)
y <- cos(x) + runif(n, -pi / 4, pi / 4)
X <- anglesToTorus(cbind(x, y))
res <- princNestTorii(X = X)
```



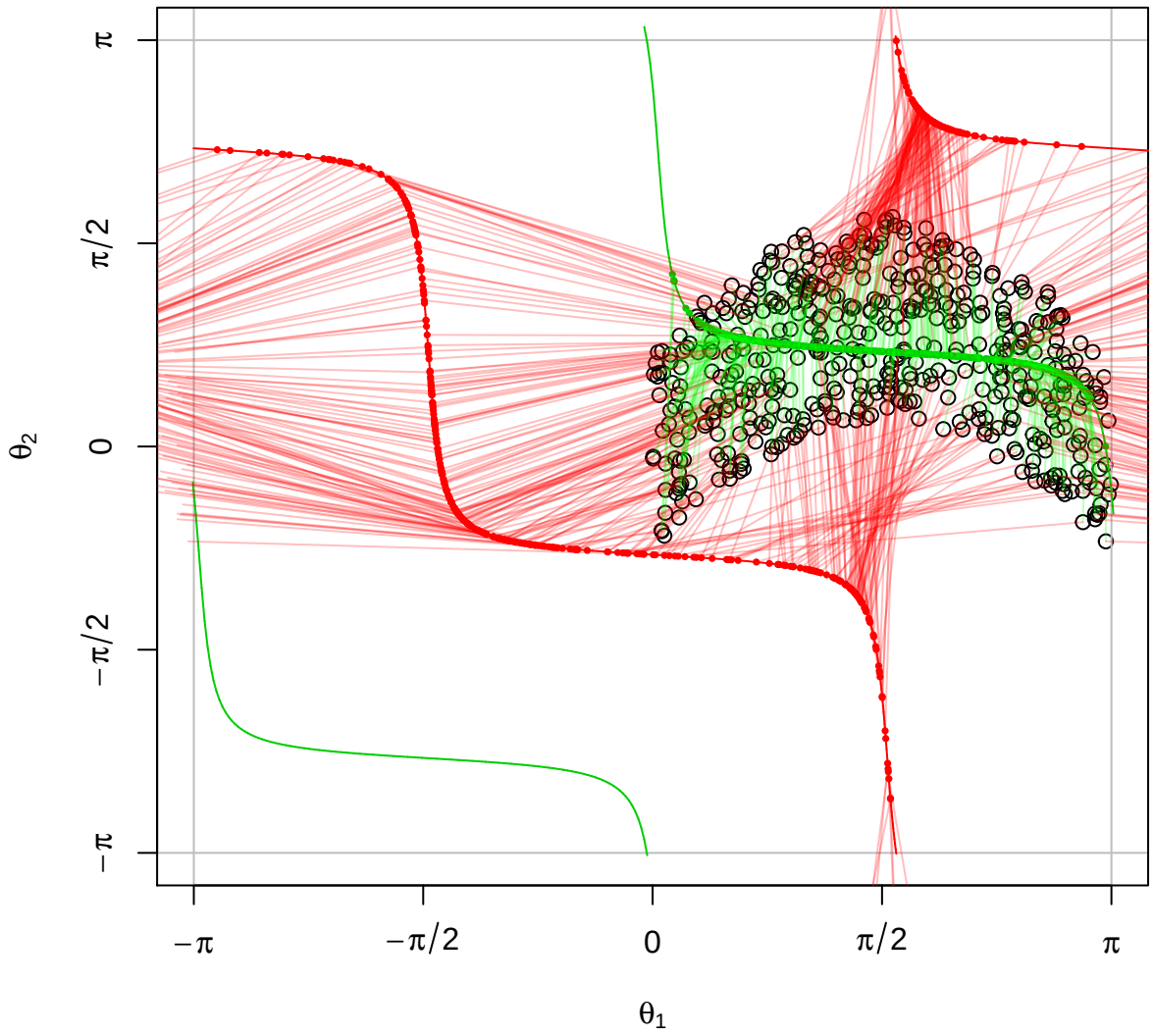

```
# Data along y = cos(x), shorter support
n <- 500
set.seed(123456)
x <- pi * runif(n)
y <- cos(x) + runif(n, -pi / 4, pi / 4)
X <- anglesToTorus(cbind(x, y))
res <- princNestTorii(X = X)
```




```
# Data along y = sin(x)
n <- 500
set.seed(123456)
x <- 2 * pi * runif(n)
y <- sin(x) + runif(n, -pi / 4, pi / 4)
X <- anglesToTorus(cbind(x, y))
res <- princNestTorii(X = X)
```

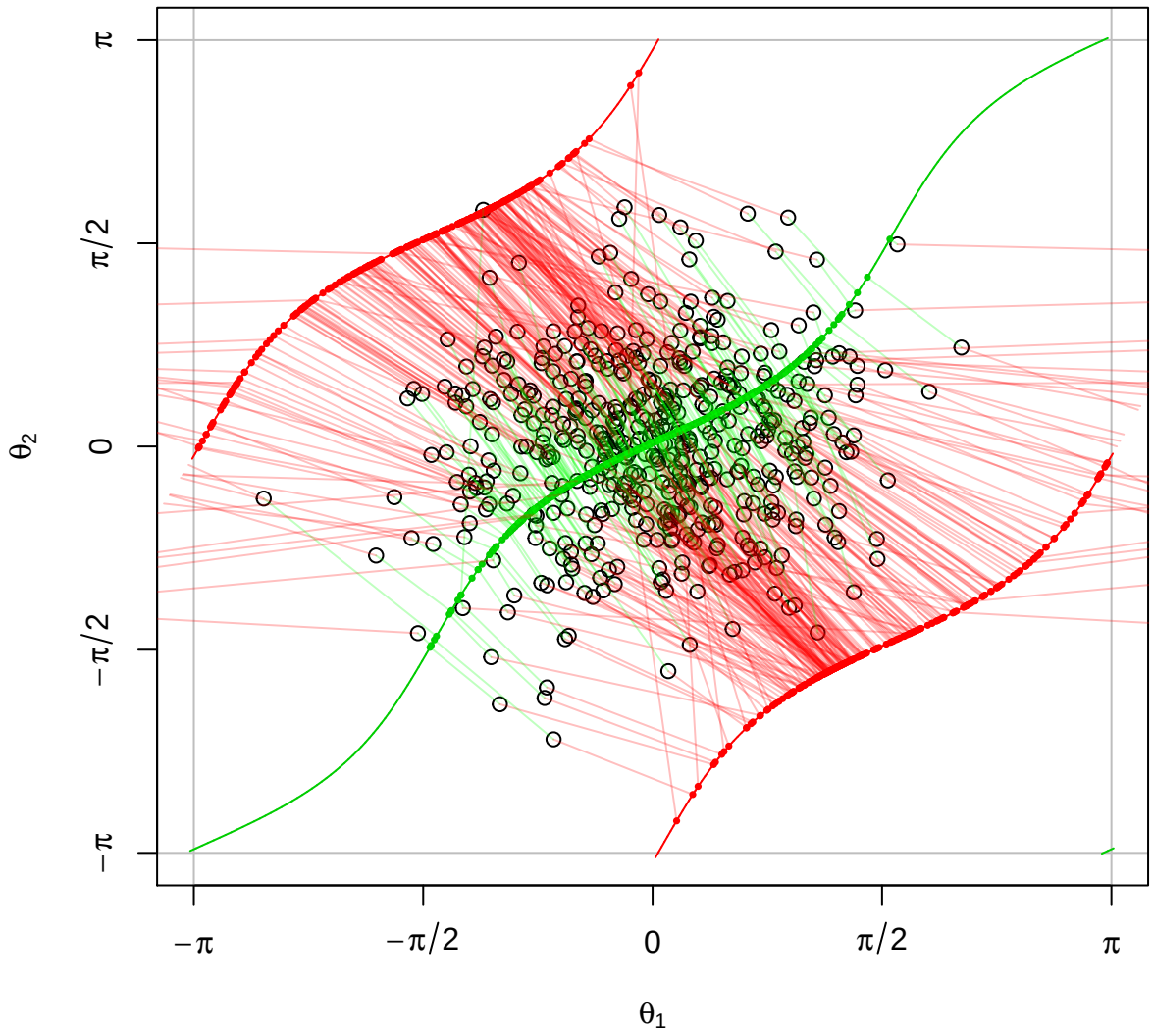



```
# Data along y = sin(x), shorter suppport
n <- 500
set.seed(123456)
x <- pi * runif(n)
y <- sin(x) + runif(n, -pi / 4, pi / 4)
X <- anglesToTorus(cbind(x, y))
res <- princNestTorii(X = X)
```

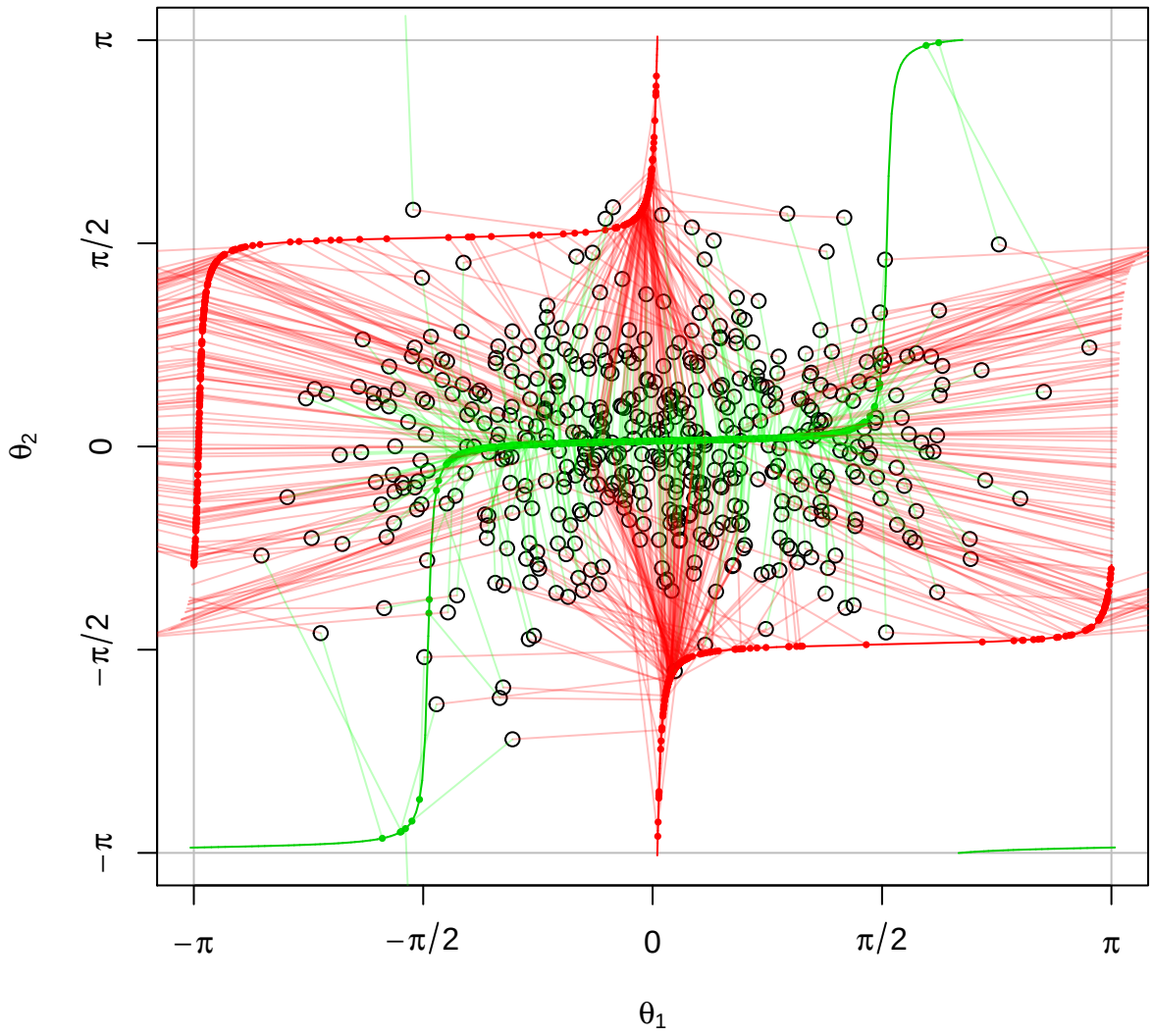


Gaussian-like

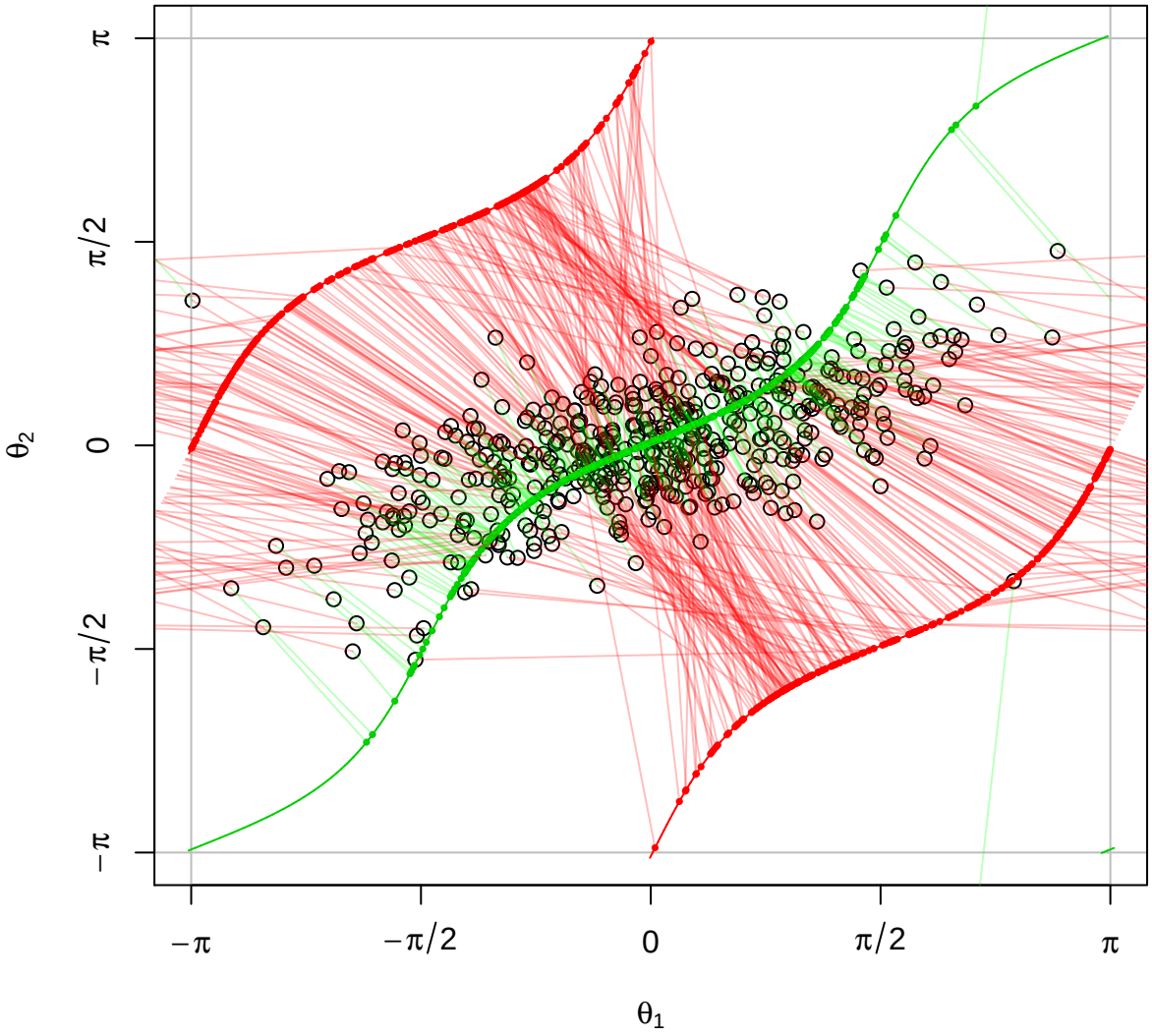
```
# Isotropic, high-concentration, Gaussian
n <- 500
set.seed(123456)
X <- anglesToTorus(rmvnorm(n = n, mean = c(0, 0), sigma = diag(c(0.5, 0.5))))
res <- princNestTorii(X = X)
```




```r
# Non-isotropic Gaussian
n <- 500
set.seed(123456)
X <- anglesToTorus(rmvnorm(n = n, mean = c(0, 0), sigma = diag(c(1, 0.5))))
res <- princNestTorii(X = X)
```




```r
# Rotated Gaussian
n <- 500
set.seed(123456)
X <- anglesToTorus(rmvnorm(n = n, mean = c(0, 0), sigma = cbind(c(1, 0.3), c(0.3, 0.25))))
res <- princNestTorii(X = X)
```

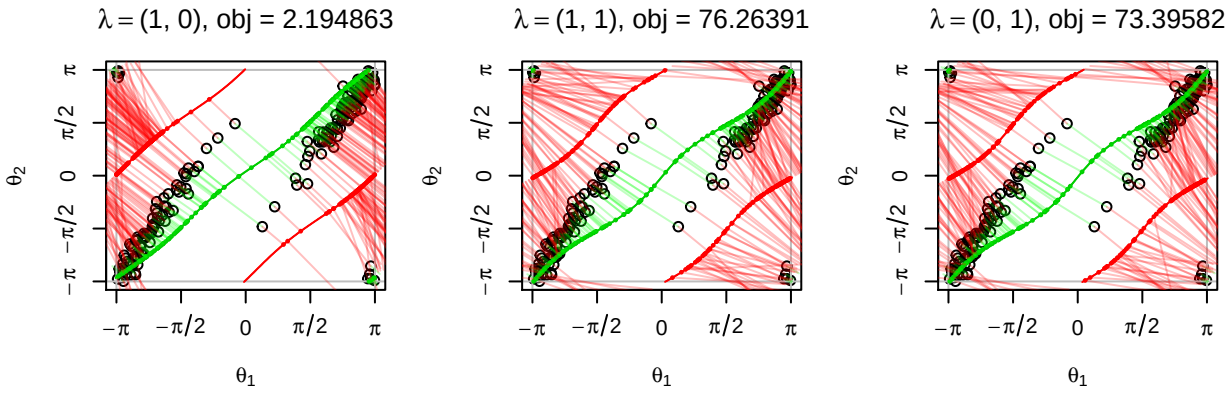


The effect of λ

λ has a **low impact on the principal components**.

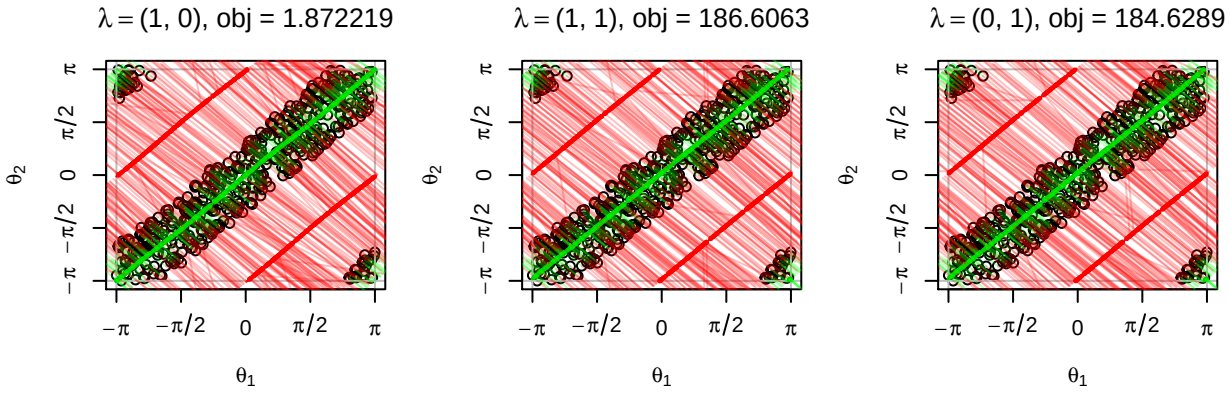
Steve's dataset

```
# X <- Steve's dataset
par(mfrow = c(1, 3), mar = c(4, 4, 3, 1) + 0.2, oma = rep(0, 4))
res <- princNestTorii(X = X, lambda = c(1, 0))
res <- princNestTorii(X = X, lambda = c(1, 1))
res <- princNestTorii(X = X, lambda = c(0, 1))
```

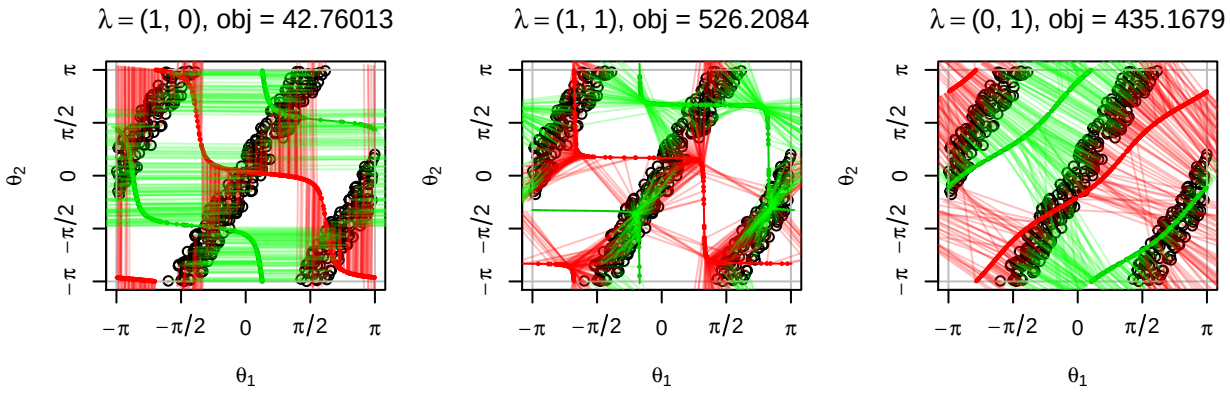


Diagonal-like

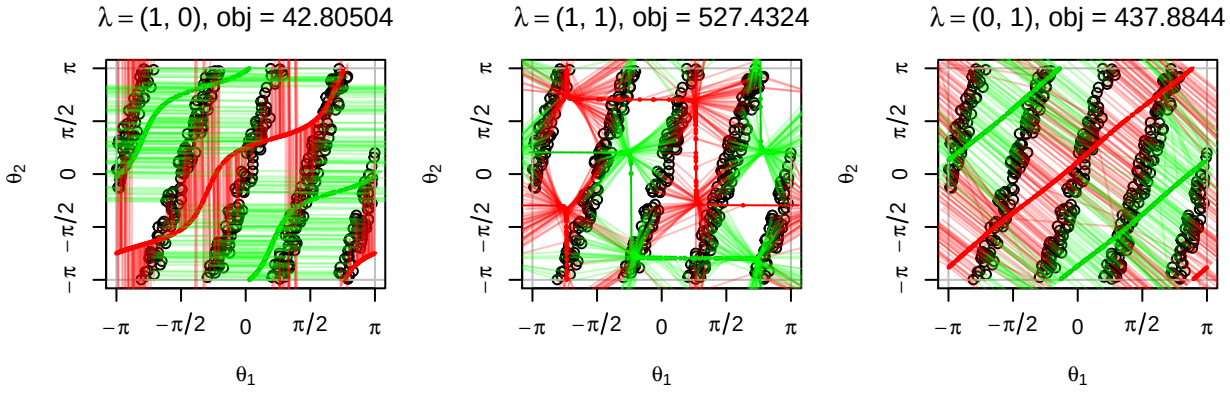
```
# Diagonal
n <- 500
set.seed(123456)
x <- 2 * pi * runif(n)
y <- x + runif(n, -pi / 3, pi / 3)
X <- anglesToTorus(cbind(x, y))
par(mfrow = c(1, 3), mar = c(4, 4, 3, 1) + 0.2, oma = rep(0, 4))
res <- princNestTorii(X = X, lambda = c(1, 0))
res <- princNestTorii(X = X, lambda = c(1, 1))
res <- princNestTorii(X = X, lambda = c(0, 1))
```



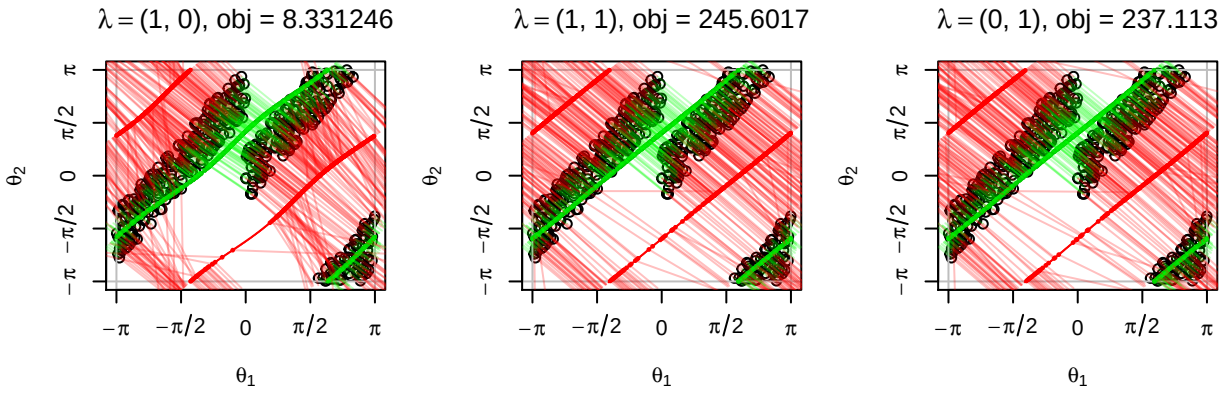

```
# Data along y = 2 * x
n <- 500
set.seed(123456)
x <- 2 * pi * runif(n)
y <- 2 * x + runif(n, -pi / 4, pi / 4)
X <- anglesToTorus(cbind(x, y))
par(mfrow = c(1, 3), mar = c(4, 4, 3, 1) + 0.2, oma = rep(0, 4))
res <- princNestTorii(X = X, lambda = c(1, 0))
res <- princNestTorii(X = X, lambda = c(1, 1))
res <- princNestTorii(X = X, lambda = c(0, 1))
```




```
# Data along y = 4 * x
n <- 500
set.seed(123456)
x <- 2 * pi * runif(n)
y <- 4 * x + runif(n, -pi / 4, pi / 4)
X <- anglesToTorus(cbind(x, y))
par(mfrow = c(1, 3), mar = c(4, 4, 3, 1) + 0.2, oma = rep(0, 4))
res <- princNestTorii(X = X, lambda = c(1, 0))
res <- princNestTorii(X = X, lambda = c(1, 1))
res <- princNestTorii(X = X, lambda = c(0, 1))
```

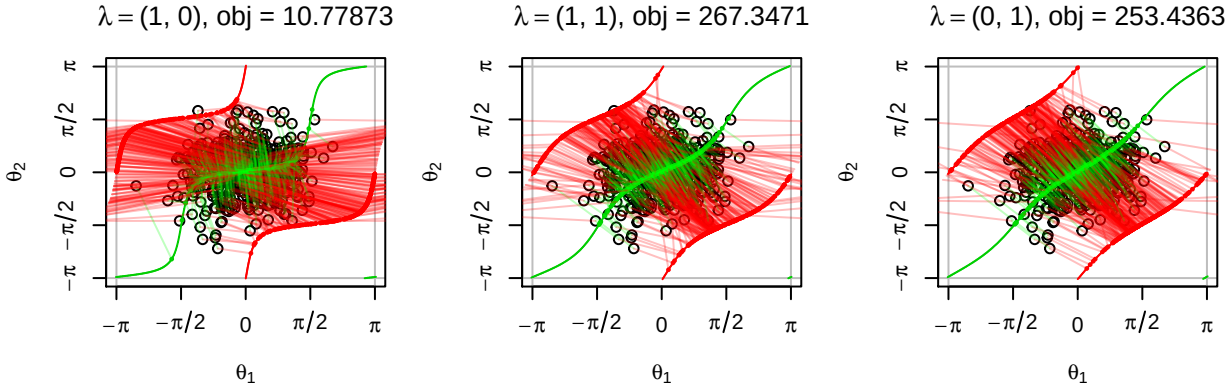



```
# Data along y = sqrt(2) * x
n <- 500
set.seed(123456)
x <- 2 * pi * runif(n)
y <- sqrt(2) * x + runif(n, -pi / 4, pi / 4)
X <- anglesToTorus(cbind(x, y))
par(mfrow = c(1, 3), mar = c(4, 4, 3, 1) + 0.2, oma = rep(0, 4))
res <- princNestTorii(X = X, lambda = c(1, 0))
res <- princNestTorii(X = X, lambda = c(1, 1))
res <- princNestTorii(X = X, lambda = c(0, 1))
```

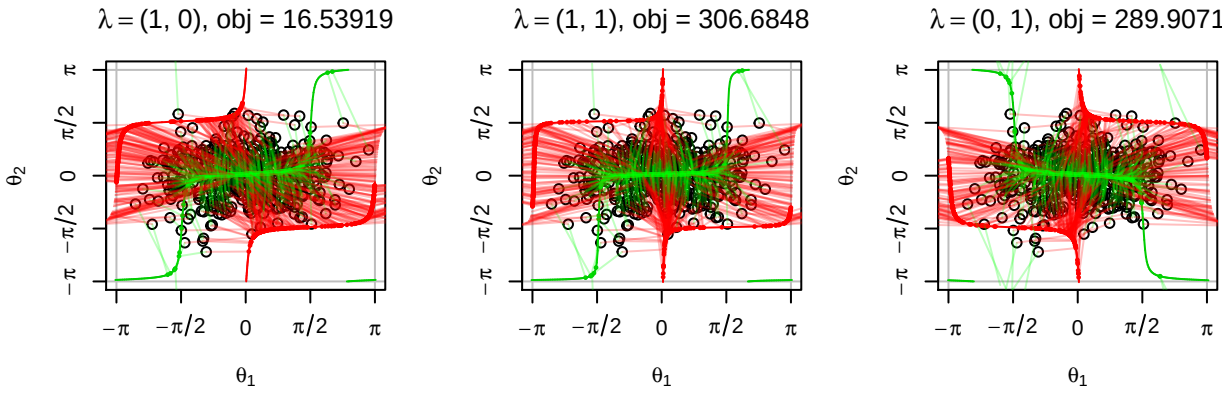


Gaussian-like

```r
# Isotropic, high-concentration, Gaussian
n <- 500
set.seed(123456)
X <- anglesToTorus(rmvnorm(n = n, mean = c(0, 0), sigma = diag(c(0.5, 0.5))))
par(mfrow = c(1, 3), mar = c(4, 4, 3, 1) + 0.2, oma = rep(0, 4))
res <- princNestTorii(X = X, lambda = c(1, 0))
res <- princNestTorii(X = X, lambda = c(1, 1))
res <- princNestTorii(X = X, lambda = c(0, 1))
```




```r
# Non-isotropic Gaussian
n <- 500
set.seed(123456)
X <- anglesToTorus(rmvnorm(n = n, mean = c(0, 0), sigma = diag(c(1, 0.5))))
par(mfrow = c(1, 3), mar = c(4, 4, 3, 1) + 0.2, oma = rep(0, 4))
res <- princNestTorii(X = X, lambda = c(1, 0))
res <- princNestTorii(X = X, lambda = c(1, 1))
res <- princNestTorii(X = X, lambda = c(0, 1))
```




```r
# Rotated Gaussian
n <- 500
set.seed(123456)
X <- anglesToTorus(rmvnorm(n = n, mean = c(0, 0), sigma = cbind(c(1, 0.3), c(0.3, 0.25))))
par(mfrow = c(1, 3), mar = c(4, 4, 3, 1) + 0.2, oma = rep(0, 4))
res <- princNestTorii(X = X, lambda = c(1, 0))
res <- princNestTorii(X = X, lambda = c(1, 1))
res <- princNestTorii(X = X, lambda = c(0, 1))
```