Package 'polarzonoid'

June 13, 2025

Version 0.1-2

Date 2025-06-10

Title Compute Maps and Properties of Polar Zonoids

Depends R (>= 4.0.0)

Imports logger

Suggests knitr, rmarkdown, gifski, flextable, equatags, rgl, zonohedra, microbenchmark, magick, pdftools

Description In each odd dimension is a convex body - the polar zonoid - whose generating functions are trigonometric polynomials. The polar zonoid is a straightforward generalization of the polar zonohedron in dimension 3. The package has some applications of the polar zonoid, including the properties of configuration spaces of arcs on the circle and 3x3 rotation matrices. There is also a root solver for trigonometric polynomials.

License GPL (>= 2)

LazyLoad no

NeedsCompilation no

Biarch no

Author Glenn Davis [aut, cre]

Maintainer Glenn Davis <gdavis@gluonics.com>

Repository CRAN

VignetteBuilder knitr

BuildVignettes yes

ByteCompile no

Date/Publication 2025-06-13 04:50:07 UTC

Contents

`polarzonoid-package` *The Polar Zonoid Z_n in* $\mathbb{R}\hat{}2n+1$

Description

In each odd dimension is a special convex body - the polar zonoid - which is generated by trigonometric polynomials. The package has some applications of the polar zonoid, including the properties of spaces of arcs on the circle and 3x3 rotation matrices.

Introduction

A *zonoid* is a special type of convex body, see *Bolker*. Among its many properties, a zonoid is centrally symmetric. A zonoid has many equivalent definitions, but in this package a zonoid Z in $\mathbb{R}^m$ is defined by m real-valued functions $f_1, f_2, ..., f_m$ on the circle $\mathbb{S}^1$. These functions are called the *generators* of Z. When the generators are piecewise constant, one obtains a *zonotope*. For the precise definition of a zonoid, see the **User Guide** vignette.

For an integer $n \geq 0$, we define the *polar zonoid* Z_n by taking the generators to be the $2n+1$ functions:

$$\cos(\theta), \sin(\theta), \cos(2\theta), \sin(2\theta), ..., \cos(n\theta), \sin(n\theta), 1 \qquad \theta \in [0, 2\pi]$$

These functions are the standard basis of the *trignometric polynomials*. Note that it is convenient for us to put the constant function 1 *last*, instead of the usual convention of putting it first. The polar zonoid is a straighforward generalization of the *polar zonohedron*, see *Chilton and Coxeter*. In this paper, it is shown that as the number of sides of the polar zonohedron goes to ∞, the zonohedron converges to $Z_1 \subseteq \mathbb{R}^3$.

Let A_n be the space of n *or fewer* disjoint arcs in the circle. From properties of trigonometric polynomials, it can be shown that there is a natural homeomorphism $A_n \rightleftarrows \partial Z_n$. For a proof of this, including the definition of the topology of A_n, see the **User Guide** vignette. Among those properties is the fact that a trigonometric polynomial of degree n has at most $2n$ roots. It is clear that $2n$ roots define a set of n disjoint arcs, in two different ways.

In the special case $n = 0$, we define A_0 to be the 2 improper arcs: the empty arc and the full circle. We have the inclusions:

$$A_0 \subseteq A_1 \subseteq A_2 \subseteq ... \subseteq A_n$$

Now the boundary ∂Z_n is trivially homemorphic to the sphere $\mathbb{S}^{2n}$. This is true for *any* convex body in $\mathbb{R}^{2n+1}$. Thus we have homeomorphisms:

$$A_n \rightleftarrows \partial Z_n \rightleftarrows \mathbb{S}^{2n}$$

where the symbol $\rightleftarrows$ denotes a homeomorphism. The bulk of the API for this package is the numerical calculation of these maps. The maps that go from left to right are straightforward and implemented for all n. The inverse maps are much more complicated and, in this version of the package, are only implemented for $n = 0, 1, 2, 3$. Since Z_n is determined by the single parameter n, there is no need to have an object for Z_n in the package API. The parameter n can be inferred from the dimension of vector and matrix function arguments, or in some cases can be given explicitly.

As a sanity check, note that n arcs have $2n$ endpoints, so we expect A_n to be a space of dimension $2n$. The fact that it is a simple manifold like $\mathbb{S}^{2n}$ is somewhat surprising. However, in the simple cases n = 0 and 1, it is easy to visualize; see the **User Guide** for details.

Author(s)

Glenn Davis <gdavis@gluonics.com>

References

Bolker, Ethan. **A Class of Convex Bodies**. Transactions of the American Mathematical Society. v. 145. Nov. 1969.

B. L. Chilton and H. S. M. Coxeter. **Polar Zonohedra**. The American Mathematical Monthly. Vol 70. No. 9. pp. 946-951. 1963.

`arc disjointness` *arc disjointness*

Description

Test whether a set of arcs are pairwise strictly disjoint.

Usage

```
disjointarcs( arcmat )
```

Arguments

`arcmat`	an Nx2 matrix with an arc definition in each row; so the total number of arcs is N. The 1st number in the row is the *center* of the arc, and the 2nd number is the *length* of the arc; both in radians. The length can be 0 or 2π, defining the empty arc and the full circle respectively. For these improper arcs, there must be only 1 row, and the center is ignored.

Value

`disjointarcs()` returns a logical - whether the given arcs are pairwise strictly disjoint. If 2 arcs overlap, or are abutting, the function returns `FALSE`.

See Also

`plotarcs()`, `complementaryarcs()`, `arcsintersection()`, `arcsunion()`, `arcssymmdiff()`

`arc operations` *Boolean arc operations, and also the distance between two collections of arcs.*

Description

Perform Boolean operations on collections of arcs in the circle. Arcs are always considered to be *closed*, i.e. to contain their endpoints.

Usage

```
complementaryarcs( arcmat )
arcsintersection( arcmat1, arcmat2 )
arcsunion( arcmat1, arcmat2 )
arcssymmdiff( arcmat1, arcmat2 )

arcsdistance( arcmat1, arcmat2 )
```

Arguments

`arcmat`	an Nx2 matrix with an arc definition in each row; so the total number of arcs is N. The 1st number in the row is the *center* of the arc, and the 2nd number is the *length* of the arc; both in radians. The length can be 0 or 2π, defining the empty arc and the full circle respectively. For these improper arcs, there must be only 1 row, and the center is ignored. The given arcs must be strictly disjoint.
`arcmat1`, `arcmat2`	2 matrices that define 2 collections of arcs, as in the previous argument. The number of arcs in each collection are not required to be equal.

Value

`complementaryarcs()` returns a matrix of the same size, which represents the closure of the complement of the union of the given arcs, as a subset of the circle. The given arcs must be strictly disjoint; if not the the function logs and error and returns `NULL`.

`arcsintersection()`, `arcsunion()`, and `arcssymmdiff()` return the intersection, union, and symmetric difference of the 2 given arc collections, respectively.

`arcsdistance()` returns the distance between two collections of arcs. This distance is the sum of the arc lengths of the symmetric difference.

See Also

`plotarcs()`, `disjointarcs()`

`arcs and boundary` *The Homeomorphism between the Space of n or fewer arcs, and the boundary of the Polar Zonoid*

Description

This section calculates the homeomorphism from the space of n or fewer arcs on the circle, which is denoted by A_n, to the boundary of the polar zonoid, and its inverse.

Usage

```
boundaryfromarcs( arcmat, n=NULL, gapmin=0 )
arcsfromboundary( p, tol=5.e-9 )
```

Arguments

`arcmat`	an Nx2 matrix with an arc definition in each row; so the total number of arcs is N. The 1st number in the row is the *center* of the arc, and the 2nd number is the *length* of the arc; both in radians. The length can be 0 or 2π, defining the empty arc and the full circle respectively. For these 2 improper arcs, there must be only 1 row, and the center is ignored.
`n`	the given set of arcs is taken to be in A_n. If `n` is `NULL`, then `n` is set to the number of rows in `arcmat`, unless `arcmat` defines an improper arc, when `n` is set to 0. If `n` is not `NULL`, then we must have `n` $\geq$ the default value. The returned vector is in $\mathbb{R}^{2n+1}$.
`gapmin`	the minimum gap between arcs in `arcmat` that is valid. The default `gapmin=0` allows abutting arcs, but not overlapping arcs. If one is sure that the arcs are not overlapping, then set `gapmin=-Inf` and this validation check is skipped, which saves a tiny bit of time.
`p`	a vector in $\mathbb{R}^{2N+1}$ or $\mathbb{R}^{2N}$. In the latter case (an even-dimensional space), a π is appended to it to make p odd-dimensional. The vector must be on the boundary of the polar zonoid in $\mathbb{R}^{2N+1}$, which usually means that it is computed by either `boundaryfromarcs()` or `boundaryfromsphere()`. In this version of the package, valid values for N are 0, 1, 2, and 3.
`tol`	if the approximate distance from `p` to a sub-stratum of the boundary is less than `tol`, then take `p` to actually *be* in this substratum. This step is necessary because the boundary is not smooth at the sub-strata.

Details

In `boundaryfromarcs()`, the calculation of the returned point is a straightforward integration of trigonometric functions over the arcs. The last component of the vector is simply the sum of the arc lengths.
Let h denote `boundaryfromarcs()`. If s denotes the full circle, then $h(s)$ = (0,...,0,2π). If ϕ denotes the empty arc, then $h(\phi)$ = (0,...,0,0).

`arcsfromboundary()` first calculates the outward pointing normal at `p`. In this version of the package, the implicitization of the boundary, and thus the normal, is only available when N is 0, 1, 2, or 3. This normal determines a trigonometric polynomial whose roots are calculated with `trigpolyroot()`. These roots are the endpoints of the arcs.

These two functions are inverses of each other.

Value

`boundaryfromarcs()` maps from A_n to $\mathbb{R}^{2n+1}$. It returns the computed point on the boundary of the zonoid. Names are assigned indicating the corresponding term in the trigonometric polynomial. In case of error, the function returns `NULL`.

If m is the length of `p`, then `arcsfromboundary()` maps from $\mathbb{R}^m$ to A_N. It returns an Nx2 matrix defining N arcs as above. Because `p` might be in a substratum of the boundary, N might be less than expected, which is $(m-1)/2$.
In case of error, the function returns `NULL`.

See Also

`spherefromarcs()`, `arcsfromsphere()`, `boundaryfromsphere()`, `spherefromboundary()`, `complementaryarcs()`, `trigpolyroot()`

Examples

```
# make two disjoint arcs
arcmat = matrix( c(pi/4,pi/2,  pi,pi/4), 2, 2, byrow=TRUE ) ; arcmat

##           [,1]      [,2]
## [1,] 0.7853982 1.5707963
## [2,] 3.1415927 0.7853982

plotarcs( arcmat )

#  map to boundary of the zonoid
b = boundaryfromarcs( arcmat ) ;  b

##        x1        y1        x2        y2        L
## 0.2346331 1.0000000 0.7071068 1.0000000 2.3561945

#  map b back to arcs, and compare with original
arcsdistance( arcmat, arcsfromboundary(b) )

## [1] 2.220446e-16

# so the round trip returns to original pair of arcs, up to numerical precision
```

`arcs and sphere` *The Homeomorphism between the Space of n or fewer arcs, and the sphere $\mathbb{S}\hat{}2n$*

Description

This section calculates the natural homeomorphism from the space of n or fewer arcs on the circle, denoted by A_n, to the sphere $\mathbb{S}^{2n}$, and its inverse.

Usage

```
spherefromarcs( arcmat, n=NULL, gapmin=0 )
spherefromarcs_plus( arcmat, n=NULL, gapmin=5e-10 )
arcsfromsphere( u )
```

Arguments

`arcmat`	an `n0 x 2` matrix with an arc definition in each row; so the total number of arcs is `n0`. The 1st number in the row is the *center* of the arc, and the 2nd number is the *length* of the arc; both in radians. The length can be 0 or 2π, defining the empty arc and the full circle respectively. For these improper arcs, there must be only 1 row, and the center is ignored.
`n`	the given set of arcs is taken to be in A_n. If `n` is `NULL`, then `n` is set to `n0 = nrow(arcmat)`, unless `arcmat` defines an improper arc, when `n` is set to 0. If `n` is not `NULL`, then we must have `n` $\geq$ `n0`. The returned vector is in $\mathbb{R}^{2n+1}$.
`gapmin`	the minimum gap between arcs in `arcmat` that is valid. The value `gapmin=0` allows abutting arcs, but not overlapping arcs. If one is sure that the arcs are not overlapping, then set `gapmin=-Inf` and this validation check is skipped, which saves a tiny bit of time. In `spherefromarcs_plus()` abutting arcs are invalid, so `gapmin` must be positive.
`u`	a unit vector in $\mathbb{R}^{2N+1}$ or $\mathbb{R}^{2N}$. In the latter case (an even-dimensional space), a 0 is appended to make `u` odd-dimensional.

Details

These first and last functions are inverses of each other.

Let a be a set of n strictly disjoint arcs in A_n, and denote the n complementary arcs by $\bar{a}$. Let $\alpha : \mathbb{S}^{2n} \to \mathbb{S}^{2n}$ denote the antipodal map. Let h denote `spherefromarcs()`. Then $h(\bar{a}) = \alpha(h(a))$. A fancier way to say this: the antipodal map α on $\mathbb{S}^{2n}$ and the complementary map $a \mapsto \bar{a}$ on A_n are *conjugate*.

If s is the full circle, then $h(s)$ = (0,...,0,1). If ϕ is the empty arc, then $h(\phi)$ = (0,...,0,-1).

Value

`spherefromarcs()` maps from A_n to $\mathbb{S}^{2n}$; it returns a unit vector in $\mathbb{R}^{2n+1}$. It is simply the composition of `boundaryfromarcs()` and `spherefromboundary()`, with a little error checking.
In case of error, it returns `NULL`.

`spherefromarcs_plus()` is the same as `spherefromarcs()`, except it returns additional data. It returns a list with these items:

`u`	the same unit vector returned by `spherefromarcs()`. Its length is `2*n+1`.
`tangent`	a `2*n+1 x 2*n0` matrix whose columns are tangent, at `u`, to the substratum $\mathbb{S}^{2n_0} \hookrightarrow \mathbb{S}^{2n}$. This is the jacobian of the map $A_{n_0} \to \mathbb{S}^{2n_0}$, when the space of arcs is parameterized by the endpoints of the arcs.
`normal`	a `2*n+1 x 2*(n-n0)` matrix whose columns are normal, at `u`, to the substratum $\mathbb{S}^{2n_0} \hookrightarrow \mathbb{S}^{2n}$. These columns are an orthonormal basis for the subspace. If `n==n0` there are 0 columns.

The matrix `cbind(u,tangent,normal)` is square. When `n>n0`, the last normal vector is flipped if necessary, so that the determinant of the square matrix is positive.

If m is the length of `u`, then `arcsfromsphere()` maps from $\mathbb{S}^{m-1}$ to A_N. It returns an Nx2 matrix defining N arcs as above. Because the space of arcs is stratified, N might be less than expected, which is $(m-1)/2$. It is simply the composition of `boundaryfromsphere()` and `arcsfromboundary()`, with a little error checking. In this version of the package, valid values for m are 1,3,5, and 7.
In case of error, it returns `NULL`.

See Also

`boundaryfromarcs()`, `spherefromboundary()`, `boundaryfromsphere()`, `arcsfromboundary()`, `complementaryarcs()`

`boundary and sphere` *The Homeomorphism between the Boundary of the Polar Zonoid in* $\mathbb{R}\hat{}2n+1$ *and the sphere* $\mathbb{S}\hat{}2n$

Description

This section calculates the homeomorphism from the space of n or fewer arcs on the circle to the boundary of the polar zonoid, and its inverse. In this version of the package, n must be 0, 1, 2, or 3.

Usage

```
spherefromboundary( p )
boundaryfromsphere( x )
```

Arguments

p	an vector of length $2n+1$ on the boundary of the polar zonoid in that dimension. p can also be even-dimensional, and in that case π is appended to make p odd-dimensional.
x	a non-zero vector of length $2n+1$, which is then unitized to put it on the sphere. It can also be even-dimensional, and in that case 0 is appended to make x odd-dimensional.

Details

`spherefromboundary()` is simply central projection of the given point onto the unit sphere centered at (0,0,...,0,0,π), which is the center of the zonoid. In this direction there is no restriction on n.

`boundaryfromsphere()` is much harder. One must find the intersection of two objects: 1) the ray based at the center of the zonoid in the direction `x`, and 2) the boundary of the zonoid. To do this, an implicit formula for the boundary has been programmed, but only when n is 0, 1, 2, or 3.

These two functions are inverses of each other.

Value

`spherefromboundary()` returns a unit vector in $\mathbb{S}^{2n}$. In case of error, the function returns `NULL`.

`boundaryfromsphere()` returns the computed point on the boundary of the zonoid. Names are assigned indicating the corresponding term in the trigonometric polynomial. In case of error, the function returns `NULL`.

See Also

`spherefromarcs()`, `arcsfromsphere()`, `boundaryfromarcs()`, `arcsfromboundary()`

---

plotarcs *plot a collection of arcs*

Description

The function `plotarcs()` plots a circle in the plane, with the given arcs overlayed with a thicker line.

Usage

```
plotarcs( arcmat, labels=FALSE, main=NULL, margintext=NA, rad=1,
                        lwd=3, pch=20, cex=1.5, add=FALSE, ...  )
```

Arguments

arcmat	an Nx2 matrix with an arc definition in each row; so the total number of arcs is N. The 1st number in the row is the *center* of the arc, and the 2nd number is the *length* of the arc; both in radians. The length can be 0 or 2π, defining the empty arc and the full circle respectively. For these improper arcs, there must be only 1 row, and the center is ignored. For improper arcs the endpoints are not drawn.
labels	if TRUE then add a text label near the center of each arc
main	Text to put above the plot, passed to `title()`. If `main=NULL` then a suitable descriptive title is constructed.
margintext	if not NA then add this text under the x-axis
rad	radius of the arcs, which are centered at (0,0). A smaller radius can be used with `add=TRUE` to avoid overlap with the first set of arcs.
lwd	line width of the arcs. The default `lwd=3` makes the arcs bolder to distinguish them from the underlying circle, which is drawn with `lwd=1`. If`lwd=NA` or `lwd<0` then drawing is skipped and only the endpoints are drawn.
pch	symbol for the endpoints, see `points()`
cex	expansion factor for the endpoints, see `points()`
add	when `add=TRUE` the arcs are added to an existing plot
...	extra arguments for the both the arcs and their endpoints, passed to both `lines()` and `points()`. For example, the color `col`.

Value

`plotarcs()` returns TRUE or FALSE.

---

rotation *The homeomorphism between balanced configurations and rotation matrices*

Description

In the **Real Projective Spaces and 3x3 Rotation Matrices** vignette, it is shown that there is a homeomorphism between P_4^π, the space of balanced configurations of 4 or 2 points on the circle, and SO(3), the space of 3x3 rotation matrices. These functions implement the homeomorphim and its inverse.

Usage

```
rotationfromarcs( arcmat, tol=5.e-9 )
arcsfromrotation( rotation, tol=1.e-7 )
rotationaroundaxis( axis, theta )
```

Arguments

`arcmat`	a `2x2` or `1x2` matrix with an arc definition in each row; so the number of arcs is 2 or 1. The 1st number in the row is the *center* of the arc, and the 2nd number is the *length* of the arc; both in radians. The total length must be π, and this is tested. If there is only one arc, it must be a semicircle. Only the endpoints of `arcmat` matter; the complement of the arcs maps to the same rotation.
`tol`	In `rotationfromarcs()` `tol` is the tolerance for the test that the total arc length is π. In `arcsfromrotation()` `tol` is the tolerance for testing that the mapped point in ∂Z_4 is close to the substratum ∂Z_2. This is to avoid generating 2 arcs, one of which is very tiny.
`rotation`	a 3x3 rotation matrix. The numeric 3x3 matrix property is verified, but, in the interest of speed, orthogonality and orientation-preservation is not verified.
`axis`	a non-zero 3-vector defining an oriented axis in $\mathbb{R}^3$. `axis` is not required to be a unit vector.
`theta`	an angle in radians. The computed rotation matrix rotates by this angle.

Details

As currently implemented, a 2-point configuration in P_2^π, whose points must be antipodal, maps to a rotation around the x-axis. The configuration of the 2 points $(0, \pm 1)$ maps to the 3x3 identity matrix. See the **Real Projective Spaces and 3x3 Rotation Matrices** vignette, for an animation of this.

Value

`rotationfromarcs()` maps from P_4^π to SO(3); it returns a 3x3 rotation matrix.

`arcsfromrotation()` maps from SO(3) to P_4^π; the returned set of arcs (either 1 or 2 of them) is only defined up to complementation. Since all we want is the endpoints, the ambiguity does not matter. The total length of the arcs is π.

`rotationaroundaxis()` returns a 3x3 rotation matrix which rotates `theta` radians around the given `axis`.

In case of error, all these functions return `NULL`.

See Also

`boundaryfromarcs()`, `spherefromboundary()`, `boundaryfromsphere()`, `arcsfromboundary()`, `complementaryarcs()`

slerp *Spherical Linear Interpolation (slerp) on the Unit Sphere*

Description

Let u_0 and u_1 be two unit vectors on the unit sphere $\mathbb{S}^{m-1}$, that are not antipodal.

The primary purpose of `slerp()` is to compute uniformly spaced points along the great circle arc from u_0 to u_1. The angle step $\Delta\theta$ is computed to be as large as possible and also satisfy $\Delta\theta \leq \theta_{max}$, where θ_{max} is a positive user-given parameter. If the distance between u_0 and u_1 is θ, then $\Delta\theta = \theta/\text{ceiling}(\theta/\theta_{max})$.

The secondary purpose is to compute points on the great circle arc from user supplied interpolation parameters τ = a numeric vector with values typically in [0,1]. The value 0 generates u_0 and 1 generates u_1. For example to generate `n` uniformly space points, set τ = `(0:(n-1))/(n-1)` .

Usage

```
slerp( u0, u1, thetamax=pi/36, tau=NULL )
```

Arguments

`u0`, `u1`	numeric vectors with same length (verified) and unit length (not verified). If they are antipodal, the great circle arc is undefined and the function returns `NULL`.
`thetamax`	the maximum angular step between successive points on the arc. This value must be positive. The default corresponds to 5°.
`tau`	a numeric vector of interpolation parameters. The default is `NULL` which means to ignore it and use `thetamax`. If `tau` is not `NULL`, then `thetamax` is ignored.

Details

If **both** `thetamax` and `tau` are not `NULL`, `tau` has priority and `thetamax` is ignored.

The interpolation formula uses the `sin()` function. For details see **Shoemake**.

Value

`slerp()` returns an `n` x `m` matrix with computed points in the rows. `n` is the number of points, and `m` is the length of `u0` and `u1`.

When using `thetamax`, `n = theta / ceiling(theta/thetamax) + 1`, where `theta` is the angle between `u0` and `u1`. Buf if `u0` and `u1` are equal, then `n=1`.

When using `tau`, `n=length(tau)`. But if `tau` is empty, the function returns `NULL`.

In case of error, e.g. `u0` and `u1` are antipodal, the function returns `NULL`.

References

Shoemake, Ken. *Animating Rotation with Quaternion Curves*. pp. 245-254. SIGGRAPH 1985.

support *The Support Function for the Polar Zonoid in* $\mathbb{R}\hat{}2n+1$

Description

This section calculates the classical support function for convex bodies, for the specific case of the polar zonoid.

Usage

```
support( direction )
```

Arguments

`direction`	an `Rx(2n+1)` matrix with non-zero direction vectors in the `R` rows. If the number of columns is even, a column of 0s is appended to make it odd. `direction` can also be a vector of length `M`, which is then converted to a `1xM` matrix.

Details

Each direction is converted to a non-zero trignometric polynomial. The roots are computed using `trigpolyroot()` and the the roots are converted to arcs, and thus `arcs`. The function `boundaryfromarcs()` is then used to compute `argmax`, and then `value`.

Value

`support()` returns a data frame with `R` rows and these columns:

`direction`	the given matrix of directions
`value`	the value of the support function of the zonoid, in the given direction
`argmax`	the point on the boundary of the zonoid where the maximum is taken
`arcs`	the number of arcs for `argmax`. This number is always $\leq$ `n`.

In case of error, the function returns `NULL`.

See Also

`boundaryfromarcs()`, `trigpolyroot()`

`trigpolyroot` *compute roots of a trigonometric polynomial*

Description

Given a trigonometric polynomial of degree n, defined on the real interval $[0, 2\pi]$. The function transforms it to a polynomial of degree $2n$ in the standard way, and then solves that polynomial. Only the real roots of the original trigonometric polynomial are returned.

Usage

```
trigpolyroot( ablist, tol=.Machine$double.eps^0.5 )
```

Arguments

`ablist`	a list with 3 items
	`a0` the constant term of the trigonometric polynomial
	`a` real coefficients of $cos(jt),\ j = 1..n$
	`b` real coefficients of $sin(jt),\ j = 1..n$
`tol`	tolerance for the imaginary part; if the absolute value of the imaginary part of the root is less than or equal to `tol`, then the root is considered to be real, and is returned

Details

The given trigonometric polynomial is converted, using a change of variable and Euler's formula, to a standard polynomial of degree $2n$, see **Wikipedia**. This polynomial is solved using `base::polyroot()`. After the inverse change of variable, the imaginary roots are removed using `tol`.

Value

`trigpolyroot()` returns the real roots in the interval $[0, 2\pi)$ and in increasing order (as a numeric vector). There may be duplicates, which correspond to multiple roots. It is up to the user to handle multiple roots.

The number of roots returned is always even, and in the set $\{0, 2, 4, .., 2n\}$.

If n is 0, or all the coefficients are 0, it is an error. In case of error, the function returns `NULL`.

Source

Wikipedia. **Trigonometric polynomial**. https://en.wikipedia.org/wiki/Trigonometric_polynomial

See Also

`polyroot()`

Index