Package 'bezier'

May 7, 2026

Date 2018-12-08

Title Toolkit for Bezier Curves and Splines

Description
The bezier package is a toolkit for working with Bezier curves and splines. The package provides functions for point generation, arc length estimation, degree elevation and curve fitting.

Version 1.1.2

Author Aaron Olsen

Maintainer Aaron Olsen <aarolsen@gmail.com>

Repository CRAN

License GPL (>= 2)

NeedsCompilation no

Date/Publication 2018-12-14 21:30:20 UTC

Contents

bezier-package *Bezier Curve and Spline Toolkit*

Description

The bezier package is a toolkit for working with Bezier curves and splines. The package provides functions for point generation, arc length estimation, degree elevation and curve fitting.

Details

Package:	bezier
Type:	Package
Version:	1.1.2
Date:	2018-12-08
License:	GPL-2

Author(s)

Aaron Olsen

bezier *Generates points along a Bezier curve or spline*

Description

This function generates points along a Bezier curve or spline (concatenated Bezier curves) at specified parametric values. The Bezier curve can be of any degree and any number of dimensions.

Usage

```
bezier(t, p, start = NULL, end = NULL, deg = NULL)
```

Arguments

t	a vector of parametric value(s), on the interval [0, 1] for a Bezier curve and on the interval [0, n] for a Bezier spline of n concatenated Bezier curves.
p	control points, input either as vector, matrix or list.
start	a vector giving an initial control point (must be of the same dimensionality as p). If provided, the first point of p is assumed to be the second control point.
end	a vector giving an final control point (must be of the same dimensionality as p). If provided, the last point of p is assumed to be the second to last control point.
deg	a numeric indicating the degree (or order) of a Bezier spline. For Bezier curves, the degree is computed based on the number of control points.

Details

This function uses the generalized formula for a Bezier curve (see http://en.wikipedia.org/wiki/Bezier_curve#Explicit_definition). If deg is NULL, p is assumed to be a Bezier curve and the degree (or order) is assumed to be the number of control points minus one. Thus, an input of two control points would return a linear Bezier curve, three control points would return a quadratic curve, four a cubic curve, etc.

For a Bezier curve, the parametric values, t, should be on the interval [0, 1]. Values greater than one are used to generate points along a Bezier spline, treating these as concatenated Bezier curves. For example, points would be generated along a Bezier spline consisting of a single Bezier curve using the interval [0, 1], for a spline consisting of two concatenated Bezier curves, the interval would be [0, 2], three curves would be [0, 3], etc. An interval of [1, 2] for a Bezier spline consisting of two concatenated Bezier curves would return points along the second Bezier curve in the spline.

Note that evenly spaced parametric values for t does not produced evenly spaced points along a Bezier curve (except for a linear Bezier curve). Point density increases with sharper curvature along a Bezier. To generate evenly spaced points along a Bezier curve use the function pointsOnBezier.

For p, the first and last values are the fixed, start and end points (through which the Bezier curve must pass) and the values in-between dictate the curvature of the Bezier between the start and end points. For a unidimensional Bezier curve, p is simply a vector in which length(p) - 1 specifies the degree. For multidimensional Bezier curves, p can either be a matrix or a list. If p is a matrix, each row is a control point where nrow(p) - 1 specifies the degree of the curve and ncol(p) specifies the dimensions. Thus, if p is a matrix of five rows and three columns, bezier would generate points along a four-degree, three-dimensional Bezier curve. If p is a list, each list element (p[[1]], p[[2]], etc.) is a dimension of the Bezier curve and the values of each list element (p[[1]][1], p[[1]][2], etc.) are the control points. The same control points can be input via either matrix or list (see MATRIX VS. LIST INPUT in Examples).

Since a Bezier spline is a series of concatenated Bezier curves, the control points alternate between end points (through which the Bezier must pass) and intermediate points (points to which the Bezier "reaches"). For a spline, the final end point of one Bezier curve is the starting end point for the next Bezier curve. Thus, for control point input the end point shared by two adjoining Bezier curves is listed just once. For example, a spline consisting of two Bezier curves with one intermediate point would require a total of five control points.

Since Bezier curves are parametric, the degree of each dimension need not be the same (i.e. each dimension can be specified by a different number of control points). This scenario is encountered when fitting Bezier curves to points in two or more dimensions if the Bezier curves are fit to each dimension separately (as with bezierCurveFit). Since the Bezier formula requires that the control points be of the same degree along each dimension, bezier elevates the degree of each dimension to the maximum degree using the function elevateBezierDegree (degree elevation does not change the shape of the Bezier curve). Inputs of this type (control points input as a list of non-uniform degrees along different dimensions) must be a single Bezier curve, not a Bezier spline.

Value

a vector (unidimensional Bezier) or matrix of bezier curve or spline points.

Author(s)

Aaron Olsen

References

http://en.wikipedia.org/wiki/Bezier_curve

See Also

elevateBezierDegree, bezierArcLength, bezierCurveFit, pointsOnBezier

Examples

```
## BEZIER CURVES ##
## SPECIFY PARAMETRIC VALUES FROM 0 TO 1 FOR SAMLPING A BEZIER CURVE
t <- seq(0, 1, length=100)

## BEZIER CONTROL POINTS
p <- matrix(c(0,0,0, 1,4,3, 2,2,0, 3,0,2, 5,5,0), nrow=5, ncol=3, byrow=TRUE)

## CREATE A 1D, 3-POINT BEZIER CURVE
bezier_points <- bezier(t=t, p=p[1:3, 1])

## CREATE THE SAME 1D, 3-POINT BEZIER CURVE, SPECIFYING THE START AND END POINTS SEPARATELY
bezier_points <- bezier(t=t, p=p[2, 1], start=p[1, 1], end=p[3, 1])

## CREATE A 2D, 3-POINT BEZIER CURVE
bezier_points <- bezier(t=t, p=p[1:3, 1:2])

## CREATE A 2D, 5-POINT BEZIER CURVE
bezier_points <- bezier(t=t, p=p[, 1:2])

## PLOT A BEZIER CURVE
## NOTE THAT POINTS ARE NOT EVENLY SPACED ALONG THE CURVE
plot(bezier(t=t, p=p[, 1:2]))

## CREATE A 3D, 3-POINT BEZIER CURVE
bezier_points <- bezier(t=t, p=p[1:3, ])

## CREATE A 3D, 5-POINT BEZIER CURVE
bezier_points <- bezier(t=t, p=p)


## MATRIX VS. LIST INPUT ##
## BEZIER CURVE WITH MATRIX INPUT
p <- matrix(c(0,0,0, 1,4,3, 2,2,0, 3,0,2, 5,5,0), nrow=5, ncol=3, byrow=TRUE)
bezier(t=seq(0, 1, length=10), p=p)

## THE SAME CONTROL POINTS INPUT AS LIST
p <- list(c(0, 1, 2, 3, 5), c(0, 4, 2, 0, 5), c(0, 3, 0, 2, 0))
bezier(t=seq(0, 1, length=10), p=p)
```

```
## BEZIER SPLINES ##
## SPECIFY PARAMETRIC VALUES FROM 0 TO 3 FOR SAMLPING A BEZIER SPLINE
t <- seq(0, 3, length=100)

## BEZIER CONTROL POINTS
p <- matrix(c(0,0,0, 1,4,3, 2,2,0, 3,0,2, 5,5,0, 8,0,4, 8,3,7), nrow=7, ncol=3, byrow=TRUE)

## CREATE A 2D BEZIER SPLINE WITH 3, 2-DEGREE BEZIER CURVES
bezier_points <- bezier(t=t, p=p[, 1:2], deg=2)

## PLOT BEZIER SPLINE
plot(bezier_points)

## PLOT FIXED POINTS ALONG SPLINE IN RED
points(rbind(p[1, ], p[3, ], p[5, ], p[7, ]), col="red", cex=0.75)

## CREATE A 3D BEZIER SPLINE WITH 3, 2-DEGREE BEZIER CURVES
bezier_points <- bezier(t=t, p=p, deg=2)


## BEZIER CURVE WITH DIFFERENT DEGREES FOR EACH DIMENSION ##
## LIST OF CONTROL POINTS FOR TWO DIMENSIONS
p_list <- list(c(0, 2, 1, 0), c(0, 4, 2, 0, 5, 0))

## CREATE 2D BEZIER CURVE WITH DIFFERENT NUMBERS OF CONTROL POINTS FOR EACH DIMENSION
bezier(t=seq(0, 1, length=100), p=p_list)
```

`bezierArcLength` *Approximates the arc length of a Bezier curve or spline*

Description

Approximates the arc length (the length along the curve) of a Bezier curve or spline over a specified parametric range. The Bezier curve can be of any degree and any number of dimensions. Either relative and/or absolute changes in arc length are used as criteria for convergence.

Usage

```
bezierArcLength(p, t1 = 0, t2 = NULL, deg = NULL, relative.min.slope = 1e-06,
                absolute.min.slope = 0, max.iter = 20, n = NULL)
```

Arguments

`p`	control points, input either as vector, matrix or list (see `bezier`).
`t1`	an initial parametric value for a Bezier curve or spline.
`t2`	a final parametric value for a Bezier curve or spline.
`deg`	a numeric indicating the degree (or order) of a Bezier spline. For Bezier curves, the degree is computed automatically based on the number of control points.

`relative.min.slope`
: a numeric indicating at which change in arc length relative to the instaneous length estimated length is considered sufficiently close to the actual length.

`absolute.min.slope`
: a numeric indicating at which absolute change in arc length estimated length is considered sufficiently close to the actual length.

`max.iter`
: the maximum number of iterations to reach the convergence criteria.

`n`
: a fixed number of points with which to calculate arc length.

Details

There is not an exact solution for the arc length of a Bezier curve of any degree and dimension so a numerical estimation approach is needed. `bezierArcLength` estimates arc length by generating a number of points along a Bezier curve (using `bezier`) and summing the interpoint distances. Given a sufficient number of points on the Bezier, the sum of interpoint distances should approximate the actual length of the Bezier. In the case of Bezier splines, the arc length of each constituent Bezier curve is estimated separately and then summed. In this case, the return values are vectors in which each element corresponds to a separate call to `bezierArcLength` for each Bezier curve.

The function first generates five points along the curve and sums the interpoint distance. This is repeated ten times, increasing the number of points along the curve by one (to 15). In this way, the arc length is estimated for a Bezier curve along a ten point range. A linear regression (`lm`) is fit to these arc lengths in order find the slope of how arc length changes as a function of the number of points along the Bezier. This slope is tested against the convergence criteria and, if the arc length has not converged, the slope is also used to guess the next range of values over which arc length will be estimated. This is repeated, measuring the change in arc length over a ten point interval, until the change in arc length reaches the convergence criteria or the function exceeds the maximum number of iterations.

`t1` and `t2` control the range of parameter values over which `bezierArcLength` will estimate arc length. In this way, arc length can be estimated for a portion of the entire Bezier curve or spline. The `deg` specifies the degree (or order) of the Bezier curve or spline (see `bezier`).

The `relative.min.slope` and `absolute.min.slope` are two criteria used to evaluate whether the function has converged on the actual arc length. At each iteration, the change in arc length as a function of points along the curve is calculated both absolutely (in the same units as the control points) and relative to the maximum estimated arc length at that iteration. If the absolute change in arc length is less than `absolute.min.slope` or the relative change in arc length less than `relative.min.slope`, estimation is stopped and the current arc length returned. Either of the convergence criteria can be ignored by setting them to 0. The default for `absolute.min.slope` is set to zero since the desired value will depend on the units of the control points input by the user. If both `absolute.min.slope` and `relative.min.slope` are equal to 0 then the function will proceed until reaching the maximum number of iterations (`max.iter`).

A non-`NULL` input for `n` will simply return the sum of interpoint distances between `n` points along a Bezier curve or spline. No estimation is performed and the convergence criteria are ignored.

Value

a list of class `"bezierArcLength"` with the following elements:

`arc.length`
: the estimated arc length along a Bezier curve or spline.

slope.break	the change in arc length when the estimation is stopped.
n	the number of points along the Bezier used to estimate arc length.
break.cause	the reason arc length estimation stopped.
n.iter	the number of iterations used in estimation.

When the input arguments correspond to a Bezier spline, slope.break, n, break.cause and n.iter are vectors in which each element cooresponds to the output of bezierArcLength called for each constituent Bezier curve (see Details).

Author(s)

Aaron Olsen

See Also

bezier, pointsOnBezier

Examples

```
## BEZIER CURVE ARC LENGTH ##
## BEZIER CURVE CONTROL POINTS
p <- matrix(c(0,0, 1,4, 2,2), nrow=3, ncol=2, byrow=TRUE)

## FIND THE ARC LENGTH ALONG THE BEZIER CURVE
bezierArcLength(p=p, t1=0, t2=1)

## FIND THE ARC LENGTH ALONG THE BEZIER CURVE
## HERE WE FIND THE ARC LENGTH OVER A SUBSET OF A BEZIER CURVE
bezierArcLength(p=p, t1=0.3, t2=0.8)


## BEZIER SPLINE ARC LENGTH ##
## BEZIER SPLINE CONTROL POINTS
p <- matrix(c(0,0, 1,4, 2,2, 3,0, 4,4), nrow=5, ncol=2, byrow=TRUE)

## FIND THE ARC LENGTH ALONG THE BEZIER SPLINE
## HERE t2 = 1 SO ARC LENGTH IS ONLY CALCULATED FOR THE
##   FIRST BEZIER CURVE OF THE SPLINE
bezierArcLength(p=p, t1=0, t2=1, deg=2)

## HERE t2 = 2 SO ARC LENGTH IS CALCULATED FOR BOTH THE
##   THE FIRST AND SECOND BEZIER CURVES
## SINCE THE TWO CURVES IN THE SPLINE ARE THE SAME -
##   JUST IN DIFFERENT ORIENTATIONS, THE ARC LENGTH
##   IS EXACTLY DOUBLE THE PREVIOUS ARC LENGTH
bezierArcLength(p=p, t1=0, t2=2, deg=2)


## COMPARE CONVERGENCE ##
## BEZIER SPLINE CONTROL POINTS
p <- matrix(c(0,0, 1,4, 2,2), nrow=3, ncol=2, byrow=TRUE)
```

```
## FIND ARC LENGTH BY ESTIMATION
bconv <- bezierArcLength(p=p, t1=0, t2=1)

## FIND ARC LENGTH WITH DIFFERENT NUMBERS OF POINTS
b1000 <- bezierArcLength(p=p, t1=0, t2=1, n=1000)
b10000 <- bezierArcLength(p=p, t1=0, t2=1, n=10000)
b100000 <- bezierArcLength(p=p, t1=0, t2=1, n=100000)

## COMPARE RESULTS
## ESTIMATION DIFFERS FROM 1000 PT SUM BY 0.0001311936
b1000$arc.length - bconv$arc.length

## ESTIMATION DIFFERS FROM 10000 PT SUM BY 0.0001321184
b10000$arc.length - bconv$arc.length

## ESTIMATION DIFFERS FROM 100000 PT SUM BY 0.0001321277
b100000$arc.length - bconv$arc.length
```

`bezierCurveFit` *Fits a Bezier curve to a set of points*

Description

Fits a Bezier curve of any degree and dimension to a set of points. A range or particular number of control points can be specified. If a range of control points is input, `bezierCurveFit` will find the minimum number of control points required to reach a specified residual standard error threshold. `bezierCurveFit` is intended to fit a Bezier curve to a large number of sample points, at least double the number of expected Bezier control points, and therefore differs from Bezier curve interpolation, in which the number of sample points are approximately equal to the number of expected Bezier control points.

Usage

```
bezierCurveFit(m, min.control.points = 3, max.control.points = 20,
               fix.start.end = FALSE, max.rse = NULL,
               max.rse.percent.change = 0.01, na.fill = FALSE,
               maxiter = 50, minFactor = 1/1024)
```

Arguments

`m`	a vector or matrix of points to which the Bezier curve is to be fit.
`min.control.points`	the minimum number of control points to use in the curve fit.
`max.control.points`	the maximum number of control points to use in the curve fit.
`fix.start.end`	whether the curve fit should be constrained to start and end at the first and last points in `m`, respectively.

max.rse	the threshold for residual standard error at which curve fitting is stopped.
max.rse.percent.change	the threshold for percent change in residual standard error at which curve fitting is stopped.
na.fill	logical indicating whether missing points (value of NA) in m should be filled by linear interpolation between neighboring non-NA points. Start and end points cannot be NA.
maxiter	a positive integer specifying the maximum number of iterations allowed (to be passed to nls function).
minFactor	a positive numeric value specifying the minimum step-size factor allowed on any step in the iteration (to be passed to nls function).

Details

This function fits a Bezier curve to a vector or matrix of points. If m is a vector, the fitted curve is unidimensional. If m is a matrix, a multidimensional fitted curve is returned (where the number of dimensions is equal to ncol(m)). In either case, the curve fitting is performed on each dimension separately. This can produce different number of control points for each dimension; bezier resolves this through degree elevation (elevateBezierDegree).

min.control.points specifies the minimum number of control points used in the curve fitting while max.control.points specifies the maximum. The number of control points includes the start and end points. If min.control.points is not equal to max.control.points, bezierCurveFit will find the minimum number of control points needed to reach the specified residual standard error threshold. If min.control.points is equal to max.control.points, the number of control points is fixed and bezierCurveFit will perform a single fit using that number of control points. bezierCurveFit is intended to fit a Bezier curve to a large number of sample points, at least double the number of expected Bezier control points, and therefore differs from Bezier curve interpolation, in which the number of sample points are approximately equal to the number of expected Bezier control points.

The nls function is used to find the control point coordinates that minimize the residual standard error (RSE) between the fitted Bezier curve and the input points m. If the number of control points is not fixed, the RSE is found for increasing numbers of control points and used to test for convergence. If the input convergence criteria are met, bezierCurveFit will return the control points at the current iteration. Thus, the number of control points may be less than max.control.points. The two convergence criteria are max.rse and max.rse.percent.change. If the absolute RSE reaches max.rse, bezierCurveFit stops increasing the number of control points and returns the fit at the current iteration.

Once the number of control points exceeds three, regression is used to find the change in RSE as a function of the number of control points. A function is fit to RSE versus the number of control points (a linear function for 3-6 points and a three-parameter exponential function for 7 or more points) to find the rate of change in RSE (the slope). The slope at the current number of control points is divided by the current RSE to find the percent change in RSE. If the percent change in RSE reaches max.rse.percent.change, bezierCurveFit stops increasing the number of control points and returns the fit at the current iteration. If max.rse and max.rse.percent.change are both NULL, bezierCurveFit will continue fitting increasing numbers of control points until max.control.points is reached.

Value

a list of class 'bezierCurveFit' with the following elements:

p	a list of the control points for the fitted Bezier curve with one element per dimension. p can be input into bezier as the p parameter. See bezier for details on Bezier control point formats.
rse	a vector of the final residual standard error for each dimension.
fit.stopped.by	a vector of the reason curve fitting was stopped (see "Reasons iterations stop" under "Examples").

Author(s)

Aaron Olsen

See Also

bezier, pointsOnBezier, elevateBezierDegree

Examples

```
## RUN BEZIER CURVE FIT ON BEZIER CURVE ##
## BEZIER CONTROL POINTS
p <- matrix(c(0,0, 1,4, 2,2, 3,0, 5,5), nrow=5, ncol=2, byrow=TRUE)

## POINTS ON BEZIER
m <- bezier(t=seq(0, 1, length=300), p=p)

## RANDOM VARIATION (NOISE) AROUND POINTS
## SENDING EXACT POINTS WILL ISSUE WARNING IN NLM FUNCTION
mrnorm <- m + cbind(rnorm(nrow(m), 1, 0.1), rnorm(nrow(m), 1, 0.1))

## RESTORE POSITION OF POINTS
mrnorm <- mrnorm - cbind(rep(1, nrow(m)), rep(1, nrow(m)))

## RUN BEZIER CURVE FIT UNCONSTRAINED NUMBER OF CONTROL POINTS
## DEFAULT IS THAT CURVE FIT IS NOT CONSTRAINED TO START AND END POINTS
bfitu <- bezierCurveFit(mrnorm)

## PLOT ORIGINAL BEZIER
plot(m, type="l")

## PLOT POINTS USED IN FITTING
points(mrnorm, col="green", cex=0.25)

## PLOT FIT CURVE
lines(bezier(t=seq(0, 1, length=500), p=bfitu$p), col="red", cex=0.25)
```

compareBezierArcLength *Returns difference between input length and a Bezier arc length*

Description

This function calls bezierArcLength and returns the absolute difference between the Bezier curve or spline arc length and an input length. The primary use of this function is to supply optim with a single value for identifying a parametric value at a particular arc length in pointsOnBezier.

Usage

```
compareBezierArcLength(p, l, t1 = 0, t2 = NULL, deg = NULL,
                       relative.min.slope = 1e-6, absolute.min.slope = 0)
```

Arguments

p	control points, input either as vector, matrix or list (see bezier).
l	the length against which the arc length is compared.
t1	an initial parametric value for a Bezier curve or spline.
t2	a final parametric value for a Bezier curve or spline.
deg	a numeric indicating the degree (or order) of a Bezier spline. For Bezier curves, the degree is computed automatically based on the number of control points.
relative.min.slope	a numeric indicating at which change in arc length relative to the instaneous length estimated length is considered sufficiently close to the actual length.
absolute.min.slope	a numeric indicating at which absolute change in arc length estimated length is considered sufficiently close to the actual length.

Details

The performance of this function is identical to bezierArcLength except that fewer input parameters are available. See "Details" in bezierArcLength.

Value

the absolute difference between the input length l and the Bezier arc length.

Author(s)

Aaron Olsen

See Also

bezier, pointsOnBezier, bezierArcLength

Examples

```
## BEZIER CURVE ARC LENGTH COMPARISON ##
## BEZIER CURVE CONTROL POINTS
p <- matrix(c(0,0, 1,4, 2,2), nrow=3, ncol=2, byrow=TRUE)

## COMPARE THE BEZIER ARC LENGTH TO ZERO
## SIMPLY RETURNS ARC LENGTH
compareBezierArcLength(p=p, l=0)

## COMPARE THE BEZIER ARC LENGTH TO ONE
compareBezierArcLength(p=p, l=1)

## SPECIFYING DIFFERENT T PARAMETERS
compareBezierArcLength(p=p, l=1, t1=0.3, t2=0.8)


## BEZIER SPLINE ARC LENGTH COMPARISON ##
## BEZIER SPLINE CONTROL POINTS
p <- matrix(c(0,0, 1,4, 2,2, 3,0, 4,4), nrow=5, ncol=2, byrow=TRUE)

## COMPARE THE BEZIER ARC LENGTH TO ZERO
## SIMPLY RETURNS ARC LENGTH
compareBezierArcLength(p=p, l=0, deg=2)
```

---

elevateBezierDegree *Raises the degree of a Bezier curve*

Description

This function raises the degree (or order) of a Bezier curve to a specified degree. Degree elevation increases the number of control points describing the Bezier without changing its shape.

Usage

```
elevateBezierDegree(p, deg)
```

Arguments

p	a vector of unidimensional Bezier control points.
deg	the degree to which the Bezier curve is to be elevated.

Details

Degree elevation of a Bezier curve increases the number of control points without changing the curve shape. This is useful when the number of control points differs for different dimensions of the curve, such as when Bezier curves are fit separately to each dimension of a multidimensional point set (as in `bezierCurveFit`). In order to perform degree elevation on a matrix of control points (multidimensional control points), `elevateBezierDegree` can be called on each dimension individually (using `apply` as in the example below).

The degree of a Bezier curve is one less than the number of control points (including the start and end point). If the input deg is equal to the current degree of the Bezier, the input control points are returned unmodified.

Value

a vector of new Bezier control points of the specified degree.

Author(s)

Aaron Olsen

References

http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/bezier-elev.html

See Also

bezier, bezierCurveFit

Examples

```
## 4 DEGREE BEZIER CONTROL POINTS
p4 <- matrix(c(0,0, 1,4, 2,2, 3,0, 5,5), nrow=5, ncol=2, byrow=TRUE)

## GENERATE BEZIER CURVE FOR 4TH DEGREE BEZIER
b4 <- bezier(t=seq(0, 1, length=100), p=p4)

## ELEVATE BEZIER DEGREE
p5 <- apply(p4, 2, elevateBezierDegree, deg=5)

## GENERATE BEZIER CURVE FOR 5TH DEGREE BEZIER
b5 <- bezier(t=seq(0, 1, length=100), p=p5)

## ELEVATE BEZIER DEGREE
p6 <- apply(p4, 2, elevateBezierDegree, deg=6)

## GENERATE BEZIER CURVE FOR 6TH DEGREE BEZIER
b6 <- bezier(t=seq(0, 1, length=100), p=p6)

## PLOT ORIGINAL 4TH DEGREE BEZIER POINTS
plot(b4)

## PLOT 5TH DEGREE BEZIER POINTS WITHIN 4TH DEGREE POINTS
points(b5, col="red", cex=0.75)

## PLOT 6TH DEGREE BEZIER POINTS WITHIN 4TH DEGREE POINTS
## NOTE THAT POINTS COINCIDE EXACTLY WITH LOWER DEGREES
## THE CURVE IS UNCHANGED BY DEGREE ELEVATION
points(b6, col="green", cex=0.5)
```

`pointsOnBezier` *Generates points along a Bezier curve or spline*

Description

This function provides three different functionalities for generating points along a Bezier curve or spline. The first generates approximately evenly spaced points along a Bezier, optimizing point position according to specified convergence criteria. The second functionality places points along a Bezier such that the distance between consecutive points does not exceed a specified Euclidean distance. This second functionality does not generate evenly spaced points along the curve, instead providing a more rapid routine for generating a large number of points on a Bezier more evenly spaced than with parametric point generation. The last functionality generates adjoining points along a Bezier as a series of integers and is intended for use with pixel coordinates.

Usage

```
pointsOnBezier(p, n = NULL, method = 'evenly_spaced', t1 = 0, t2 = NULL,
               deg = NULL, max.dist = NULL, max.dist.factor = 0.1,
               relative.min.slope = 1e-7, absolute.min.slope = 0,
               sub.relative.min.slope = 1e-4, sub.absolute.min.slope = 0,
               print.progress = FALSE)
```

Arguments

`p`	control points, input either as vector, matrix or list (see `bezier`).
`n`	the number of points to generate along the Bezier. Ignored if `method` is `'max_dist'` or `'adjoining'`.
`method`	the method to be used in generating the points. Either `'evenly_spaced'`, `'max_dist'` or `'adjoining'`. Does not need to be specified if `max.dist` or `n` are non-`NULL`.
`t1`	a parametric value for a Bezier curve or spline at which the points will start.
`t2`	a parametric value for a Bezier curve or spline at which the points will end. Default is the end of the Bezier curve or spline.
`deg`	a numeric indicating the degree (or order) of a Bezier spline. For Bezier curves, the degree is computed based on the number of control points.
`max.dist`	the maximum Euclidean distance (not distance along the curve) between consecutive points for the more rapid routine.
`max.dist.factor`	a factor used to approximate point position based on maximum distance criteria (see Details). Ignored if `max.dist` is `NULL`.
`relative.min.slope`	parameter passed to `bezierArcLength` for estimating total arc length. Ignored if `max.dist` is non-`NULL`.
`absolute.min.slope`	parameter passed to `bezierArcLength` for estimating total arc length. Ignored if `max.dist` is non-`NULL`.

sub.relative.min.slope
: parameter passed to compareBezierArcLength for estimating total arc length (see compareBezierArcLength). Ignored if max.dist is non-NULL.

sub.absolute.min.slope
: parameter passed to compareBezierArcLength for estimating total arc length (see compareBezierArcLength). Ignored if max.dist is non-NULL.

print.progress
: logical indicating whether iterations should be printed for tracking function progress.

Details

Points can easily be generated along a Bezier curve or spline using parametric values (provided by the function bezier), however these points are not evenly spaced along the curve. Points generated by parametric values will be closer together in regions with the highest curvature and furthest apart in regions that approach a straight line. This function provides three different functionalities for generating points along a Bezier curve or spline that are more evenly spaced than those generated using parametric values. The 'evenly_spaced' method generates n approximately evenly spaced points along a Bezier, optimizing point position according to specified convergence criteria. The 'max_dist' method places points along a Bezier such that the distance between consecutive points does not exceed a specified Euclidean distance (max.dist). And the 'adjoining' method generates points along a Bezier as a series of integers and is intended for use with pixel coordinates.

The input of the control points p is identical to bezier and can be a vector, matrix or list (see Details in bezier). As with bezier, when control points are input as a list and the number of control points differs for different dimensions, the degree will be elevated so that all dimensions are of uniform degree (see elevateBezierDegree). t1 and t2 are parametric values along the Bezier curve or spline between which points will be generated. The default values for t1 and t2 are the start and end points of the Bezier curve or spline, respectively. For a Bezier spline, if t2 is not specified, it is calculated based on the number of control points and the degree (deg). When using pointsOnBezier for Bezier splines, deg must be specified or else the points will be treated as a single Bezier curve.

If n is non-NULL, pointsOnBezier generates n evenly spaced points along a Bezier curve or spline. This requires accurate approximation of Bezier arc length. An initial estimation of the total arc length between t1 and t2 is made (using bezierArcLength) to determine the interval at which points should be placed to equally subdivide the curve. optim is used to find the optimal position of each point, calling bezierArcLength via compareBezierArcLength, such that the arc length between points is nearly equal to this interval. When positioning each point, the arc length is estimated from t1 (rather than from the previous point) so that errors are not compounded. As a consequence of repeated calls to optim and bezierArcLength, this functionality can be rather slow.

The parameters ending in min.slope are convergence criteria passed to bezierArcLength. The parameters relative.min.slope and absolute.min.slope are the criteria used in the initial arc length estimation, while sub.relative.min.slope and sub.absolute.min.slope are the criteria used to estimate arc length in placing each point along the curve. Larger convergence criteria values will cause pointsOnBezier to run faster but at lower accuracy. For a complete description of the convergence criteria, see Details in bezierArcLength.

pointsOnBezier runs an alternative routine when max.dist is non-NULL. In this case, n and the convergence criteria are ignored, bezierArcLength is not called and pointsOnBezier generates

points along a Bezier such that the distance between consecutive points does not exceed the specified Euclidean distance `max.dist`. The parameter `max.dist.factor` is a factor that is used to iteratively increase the parametric value to reach the next point without exceeding `max.dist`. The lower `max.dist.factor` is, the closer the interpoint Euclidean distance will be to `max.dist` but the longer `pointsOnBezier` will take to run (see Examples). If `max.dist` does not evenly divide the total arc length between `t1` and `t2`, the interval between the second-to-last point and the end point may not be close to `max.dist`. If `max.dist` evenly divides the arc length, if `max.dist.factor` is low and if `max.dist` is small, the points will be more evenly spaced than with parametric point generation.

When `method` is `'adjoining'`, `pointsOnBezier` will generate points as integers at adjoining positions along the Bezier curve or spline. The arc length is first measured (very roughly) to approximate the first parametric interval at which to find adjoining points. The function adds this initial interval to `t1` and finds the position of the next Bezier point, rounded to the nearest integer. The interval is decreased or increased depending on whether the point is too distant or too near until the next point adjoins the previous point. Adjoining is defined as two points whose positions (as integers) differ by one in either or both coordinates. Thus, 2D points adjoining on the diagonal are considered adjoining. For instance, the points [3,5] would be adjoining with [4,5], [3,4] and [4,6] but not [1,3]. The function continues to iterate through the parametric values up to `t2`, generating points adjoining to the previous point. This method is intended for use with pixel coordinates, such as when Bezier control points are used to trace Bezier curves and splines on an image. Unlike the previous two methods, most of the generated points will not fall exactly on the Bezier curve since they are rounded to the nearest integer. This method currently only works with curves or splines in two dimensions.

In the case of Bezier splines, note that borders between spline segments are not respected and arc lengths are calculated across spline segments. In order to generate points within spline segments, `pointsOnBezier` should be called separately for each segment.

Value

a list with the following elements:

`points`	evenly spaced or nearly evenly spaced points along a Bezier curve or spline.
`error`	an vector of the error for each point along the curve or spline. If the `method` is `'evenly_spaced'`, this is the `value` output from `optim` for each point estimation. If `method` is `'max_dist'`, this is `max.dist` minus the actual Euclidean distance between consecutive points. If `method` is `'adjoining'`, this is the difference between the actual position on the Bezier minus the position to the nearest integer.
`t`	the parametric values corresponding to each point in `points`.

Author(s)

Aaron Olsen

See Also

`bezier`, `bezierArcLength`, `compareBezierArcLength`, `elevateBezierDegree`

Examples

```
## EVENLY_SPACED METHOD ##
## BEZIER CURVE CONTROL POINTS
p <- matrix(c(3,2, 3,0, 5,5), nrow=3, ncol=2, byrow=TRUE)

## GET PARAMETRIC BEZIER POINTS
bp <- bezier(t=seq(0, 1, length=100), p=p)

## GET EVENLY SPACED POINTS ALONG CURVE
pob <- pointsOnBezier(p=p, n=10, method="evenly_spaced", print.progress=TRUE)

## FUNCTION WILL RUN FASTER BY INCREASING CONVERGENCE CRITERIA
pob_faster <- pointsOnBezier(p=p, n=10, method="evenly_spaced", sub.relative.min.slope=1e-2,
print.progress=TRUE)

## PLOT PARAMETRIC BEZIER POINTS
## NOTE THAT THEY ARE NOT EVENLY SPACED ALONG THE CURVE
plot(bp, cex=0.5, asp=1)

## ADD POINTS TO PLOT
## NOTE THAT THESE POINTS ARE EVENLY SPACED ALONG CURVE
points(pob$points, col="red")

## WITH FASTER RUN, SOME DEVIATION BUT POINTS ARE NEARLY IDENTICAL
points(pob_faster$points, col="blue", cex=1.5)


## MAX_DIST METHOD ##
## BEZIER CURVE CONTROL POINTS
p <- matrix(c(3,2, 3,0, 5,5), nrow=3, ncol=2, byrow=TRUE)

## GET PARAMETRIC BEZIER POINTS
bp <- bezier(t=seq(0, 1, length=100), p=p)

## GET POINTS ALONG CURVE WITH INTERPOINT DISTANCE LESS THAN 0.1
pob <- pointsOnBezier(p=p, max.dist=0.1, method="max_dist", print.progress=TRUE)

## PLOT PARAMETRIC BEZIER POINTS
plot(bp, cex=0.5, asp=1)

## ADD POINTS TO PLOT
points(pob$points, col="red")


## ADJOINING METHOD ##
## BEZIER CURVE CONTROL POINTS
p <- matrix(c(300,200, 300,0, 500,500), nrow=3, ncol=2, byrow=TRUE)

## GET PARAMETRIC BEZIER POINTS
bp <- bezier(t=seq(0, 1, length=100), p=p)

## GET POINTS ALONG CURVE WITH ROUNDED POSITIONS AT "PIXEL" SPACING
```

```
pob <- pointsOnBezier(p=p, method="adjoining", print.progress=TRUE)

## PLOT PARAMETRIC BEZIER POINTS
plot(bp, cex=0.5, asp=1)

## ADD POINTS TO PLOT
points(pob$points, col="red", cex=0.2, pch=16)
```

---

summary.bezierArcLength

Summary of a Bezier arc length estimation

Description

Prints a summary of the output of bezierArcLength (a list of class "bezierArcLength"). This includes the change in arc length once arc length estimation is stopped, the number of iterations until convergence and the number of points along the Bezier used to estimate arc length. If the input is a Bezier spline then the results are printed separately for each curve in the spline. See bezierArcLength for details.

Usage

```
## S3 method for class 'bezierArcLength'
summary(object, ...)
```

Arguments

object	a list of class "bezierArcLength" (the output of bezierArcLength).
...	further arguments passed to or from other methods.

Value

a NULL value.

Author(s)

Aaron Olsen

See Also

bezierArcLength, bezier

Examples

```
## BEZIER CURVE ARC LENGTH ESTIMATION ##
## BEZIER CURVE CONTROL POINTS
p <- matrix(c(0,0, 1,4, 2,2), nrow=3, ncol=2, byrow=TRUE)

## FIND THE ARC LENGTH ALONG THE BEZIER CURVE
bcurve <- bezierArcLength(p=p, t1=0, t2=1)

## PRINT SUMMARY
print(summary(bcurve))


## BEZIER SPLINE ARC LENGTH ##
## BEZIER SPLINE CONTROL POINTS
p <- matrix(c(0,0, 1,4, 2,2, 3,0, 4,4), nrow=5, ncol=2, byrow=TRUE)

## FIND THE ARC LENGTH ALONG THE BEZIER SPLINE
## HERE t2 = 1 SO ARC LENGTH IS ONLY CALCULATED FOR THE
##   FIRST BEZIER CURVE OF THE SPLINE
bspline <- bezierArcLength(p=p, t1=0, t2=2, deg=2)

## PRINT SUMMARY
print(summary(bspline))
```

`summary.bezierCurveFit` *Summary of a Bezier curve fit*

Description

Prints a summary of the output of `bezierCurveFit` (a list of class `"bezierCurveFit"`). For each dimension of the point set being fitted, this includes the number of parameters used in the fit, the residual standard error and the reason the fit iterations were stopped. See `bezierCurveFit` for details.

Usage

```
## S3 method for class 'bezierCurveFit'
summary(object, ...)
```

Arguments

`object`	a list of class `"bezierCurveFit"` (the output of `bezierCurveFit`).
`...`	further arguments passed to or from other methods.

Value

a `NULL` value.

Author(s)

Aaron Olsen

See Also

bezierCurveFit, bezier

Examples

```
## RUN BEZIER CURVE FIT ON BEZIER CURVE ##
## BEZIER CONTROL POINTS
p <- matrix(c(0,0, 1,4, 2,2, 3,0, 5,5), nrow=5, ncol=2, byrow=TRUE)

## POINTS ON BEZIER
m <- bezier(t=seq(0, 1, length=300), p=p)

## RANDOM VARIATION (NOISE) AROUND POINTS
## SENDING EXACT POINTS WILL ISSUE WARNING IN NLM FUNCTION
mrnorm <- m + cbind(rnorm(nrow(m), 1, 0.1), rnorm(nrow(m), 1, 0.1))

## RESTORE POSITION OF POINTS
mrnorm <- mrnorm - cbind(rep(1, nrow(m)), rep(1, nrow(m)))

## RUN BEZIER CURVE FIT UNCONSTRAINED NUMBER OF CONTROL POINTS
bfit <- bezierCurveFit(mrnorm)

## GET CURVE FIT SUMMARY
print(summary(bfit))
```

Index