Package ‘cheby’

October 18, 2015

Type Package

Title Computes Chebychev approximations to 1- and 2-dimensional functions

Version 1.0

Date 2013-12-23

Author Philip Barrett

Maintainer Philip Barrett <pobarrett@gmail.com>

Description Computes Chbychev approximation to arbitrary 1- and 2- dimensional functions. Calculates shape-preserving approximations to 1-dimensional functions.

Depends polynom, orthopolynom, nloptr, multipol, RUnit

License GPL-3

Suggests knitr

VignetteBuilder knitr

R topics documented:

cheby-package .......................................................... 2
d1.grid ................................................................. 3
d1.normalize .......................................................... 3
d1.poly ................................................................. 4
dn.poly ................................................................. 5
sp1.poly ............................................................... 6

Index 8
cheby-package

Computes polynomial approximations of arbitrary functions

Description

Package to implement Chebychev and shape-preserving Chebychev approximations in one and two dimensions.

Details

Package: cheby
Type: Package
Version: 1.0
Date: 2013-12-24
License: GPL-3

Generates polynomial approximations of arbitrary functions in one and two dimensions, whilst preserving slope, concavity and higher derivatives wherever specified.

d1.poly is the main routine for approximating one-dimensional functions. sp1.poly does the same but preserves the sign of an arbitrary number of derivatives. 2-Dimensional Chebychev polynomials are also available with dn.poly. Higher order Chebychev and shape-preserving approximations to follow later.

Comments and suggestions are gratefully received by the author.

Author(s)

Philip Barrett <pobarrett@uchicago.edu>

References


See Also
d1.poly, sp1.poly, dn.poly

Examples

## Compute basic approximations to natural logarithm
RR <- d1.poly( log, c(0,4), 6, 10 )
SS <- sp1.poly( log, c(0,4), 6, 10, n.shape=c(5,10),
                sign.deriv=c(1,-1), solver='NLOPT_LD_SLSQP' )
pp <- seq( 0, 4, length.out=100 )
plot( pp, sapply(pp, RR), lwd=2, col=2, type='l' )
lines( pp, sapply(pp, log), lwd=2, col=1 )
lines( pp, sapply(pp, SS), lwd=2, col=4 )
**d1.grid**

*Calculate grid of approximating points*

**Description**

Computes either a Uniform or Chebychev grid of collocation nodes

**Usage**

```r
d1.grid(vRange, iPts, stMethod = "Chebychev")
```

**Arguments**

- **vRange**: the vector of the range over which the nodes are computed
- **iPts**: number of colocations points
- **stMethod**: Either Chebychev or Uniform

---

**d1.normalize**

*Normalizes x in [a,b] to [-1,1]*

**Description**

Normalizes x in [a,b] to [-1,1].

**Usage**

```r
d1.normalize(x, range)
```

**Arguments**

- **x**: A number in range
- **range**: The range

**Value**

\[
2(x-a)/(b-a)-1
\]
Description

Standard Chebychev approximation of an arbitrary function.

Usage

d1.poly(fn, range, iOrder, iPts, fn.opts = NULL, fn.vals = NULL, grid = NULL, details = FALSE)

Arguments

- fn: a function \( f(x) \) or \( f(x, \beta) \) for \( \beta \) a list of function parameters. If the latter, must be coded with second argument a list names opts, i.e. \( fn <- \text{function}( x, \text{opts} ) \)
- range: the range of the approximation.
- iOrder: the order of the polynomial approximation.
- iPts: the number of points at which the approximation is computed. Must be at least as large as iOrder.
- fn.opts: (optional) options passed to fn
- fn.vals: the values of fn on grid. Useful if fn is very slow to evaluate.
- grid: (optional) the grid on which the function is to be approximated.
- details: If TRUE, returns extra details about the approximation.

Value

A function which approximates the input fn over the interval range. If details=TRUE, return is a list with entries fn, poly, fn.deriv, poly.deriv, residuals, which are, respectively, the approximating function, the polynomial description over [-1,1], the derivative of the approximation, the polynomial description of the derivative, and the approximation errors.

See Also

sp1.poly

Examples

cube <- function( x, opts ) opts$A * x^3
approx <- d1.poly( cube, c(-4,2), 4, 20, fn.opts=list(A=2) )
apply( c(-3, -2, 0, .5 ), function( x ) abs( approx(x) - 2 * x^3 ) )
**dn.poly**

**Mutli-dimensional Chebychev approximation**

**Description**

Standard Chebychev approximation of an arbitrary function. **Currently only works for two-dimensional approximation**

**Usage**

\[
dn.poly(fn, range, iOrder, iPts, fn.opts = NULL, fn.vals = NULL, grid = NULL, details = false)
\]

**Arguments**

- **fn**: a function \( f(x_1, ..., x_n) \) or \( f(x_1, ..., x_n, \beta) \) for \( \beta \) a list of function parameters. If the latter, must be coded with second argument a list names opts, i.e. \( fn \leftarrow \text{function( } x, \text{ opts )} \)
- **range**: the range of the approximation, given as a list of vectors. Eg for a 3-dimensional function approximated over \([1,2] \times [-1,2] \times [0,4] \), would be \( \text{range = list( c(1,2), c(-1,2), c(0,4) )} \)
- **iOrder**: the vector of orders of the polynomial approximation. Eg. to approximate using polynomials of order 3 and 5 in the 1st and 2nd dimensions respectively, would be \( iOrder=c(5,6) \)
- **iPts**: the vector of number of points at which the approximation is computed. Must be at least as large as \( iOrder \) (element-by-element).
- **fn.opts**: (optional) options passed to \( fn \) [NOT YET FUNCTIONAL]
- **fn.vals**: the values of \( fn \) on \( grid \). Useful if \( fn \) is very slow to evaluate.
- **grid**: (optional) the grid on which the function is to be approximated. Should be submitted as a list of vectors for the grids in each dimension.
- **details**: If TRUE, returns extra details about the approximation.

**Value**

A function which approximates the input \( fn \) over the box defined by \( range \). If \( details=\text{TRUE} \), also includes the polynomial description over \([-1,1] \), as well as the approximation errors

**See Also**

- **d1.poly**
Examples

```
test.fn <- function( x, y ) x^2*y^.5
ff <- dn.poly( test.fn, list( c(0,4), c(0,4) ), c( 6, 6 ), c(12,12) )
XX <- seq(0,4,.05)
plot( XX, mapply( test.fn, XX, 1 ), type='l' )
lines( XX, mapply( ff, XX, 1 ), col=2 )
plot( XX, mapply( test.fn, 1, XX ), type='l' )
lines( XX, mapply( ff, 1, XX ), col=2 )
```

---

sp1.poly

*Shape-preserving polynomial approximation*

Description

Approximates the function `fn` using shape-preserving polynomials evaluated at the points in `grid`.

Usage

```
sp1.poly(fn, range, iOrder, iPts, fn.opts = NULL, fn.vals = NULL,
grid = NULL, n.shape = 0, sign.deriv = NULL, x0 = NULL,
solver = "NLOPT_LD_SLSQP", tol = 1e-06, details = FALSE,
quiet = FALSE)
```

Arguments

- **fn**: a function $f(x)$ or $f(x, \beta)$ for $\beta$ a list of function parameters. If the latter, must be coded with second argument a list names `opts`, i.e. `fn <- function( x, opts )`
- **range**: the range of the approximation.
- **iOrder**: the order of the polynomial approximation.
- **iPts**: the number of points at which the approximation is computed. Must be at least as large as `iOrder`.
- **fn.opts**: (optional) options passed to `fn`
- **fn.vals**: the values of `fn` on `grid`. Useful if `fn` is very slow to evaluate.
- **grid**: (optional) the grid on which the function is to be approximated.
- **n.shape**: a vector of the number of shape-preserving points for each order of differentiation. For example, to specify the slope at 5 Chebychev points and the concavity at 10, use `n.shape=c(5, 10)`
- **sign.deriv**: a vector of signs +1, 0, -1 defining the sign of each derivative. For example, for a concave approximation with positive slope `sign.deriv=c(1,-1)`.
- **x0**: initial guess of the polynomial approximation. Default is the standard Chebychev approximation.
- **solver**: the `nlopt` solver to use in computing the best fit. Default is `NLOPT_LD_SLSQP`.
- **tol**: tolerance for solver convergence. Default is `1e-06`.
- **details**: If TRUE, returns extra details about the approximation.
- **quiet**: Supresses output about success of least-error fitting. Failure will always be reported.
sp1.poly

Value

A function which approximates fn. If details=TRUE, return is a list with entries fn, poly, fn.deriv, poly.deriv, residuals, which are, respectively, the approximating function, the polynomial description over [-1,1], the derivative of the approximation, the polynomial description of the derivative, and the approximation errors.

References

nloptr documentation

See Also
d1.poly

Examples

```r
base <- d1.poly( log, c(0,4), 6, 10, details=TRUE )
sp.compare <- sp1.poly( log, c(0,4), 6, 10, details=TRUE )
sp.flat.x0 <- sp1.poly( log, c(0,4), 6, 10, x0=c(1,1,1,1,1,1), details=TRUE )
print( base$poly - sp.compare$poly )
print( base$poly - sp.flat.x0$poly )
# Comparison without using shape-preserving methods
sp.concave <- sp1.poly( log, c(0,4), 6, 10, n.shape=c(5,10), sign.deriv=c(1,-1) )
pp <- seq( 0, 4, length.out=100 )
plot( pp, sapply(pp, base$fn), lwd=2, col=2, type='l' )
lines( pp, sapply(pp, log), lwd=2, col=1 )
lines( pp, sapply(pp, sp.concave), lwd=2, col=4 )
legend( 'bottomright', c( 'log', 'Order 6 polynomial approx',
                          'Order 6 shape-preserving polynomial approx' ), lwd=2,
                          col=c(1,2,4), bty='n' )
# Compare the Chebychev and shape-preserving approximations
```
Index

*Topic **package**
  cheby-package, 2

cheby (cheby-package), 2
cheby-package, 2

d1.grid, 3
d1.normalize, 3
d1.poly, 2, 4, 5, 7
dn.poly, 2, 5

sp1.poly, 2, 4, 6