Package ‘chebpol’

December 9, 2019

Version 2.1-2
Date 2019-12-09
Title Multivariate Interpolation
Copyright 2013-2019, Simen Gaure
Imports compiler, stats, geometry
Suggests lattice, knitr, cubature, plot3D
SystemRequirements fftw3 (>= 3.1.2), gsl
Description Contains methods for creating multivariate/multidimensional interpolations of functions on a hypercube. If available through fftw3, the DCT-II/FFT is used to compute coefficients for a Chebyshev interpolation.
Other interpolation methods for arbitrary Cartesian grids are also provided, a piecewise multilinear,
and the Floater-Hormann barycenter method. For scattered data polyharmonic splines with a linear term is provided. The time-critical parts are written in C for speed. All interpolants are parallelized if used to evaluate more than one point.
License Artistic-2.0
Classification/MSC 41A05, 41A10, 41A50, 41A63, 65D05, 65T40
Classification/ACM G.1.2
URL https://github.com/sgaure/chebpol
VignetteBuilder knitr
BugReports https://github.com/sgaure/chebpol/issues
Encoding UTF-8
RoxygenNote 6.1.1
NeedsCompilation yes
Author Simen Gaure [aut, cre] (<https://orcid.org/0000-0001-7251-8747>)
Maintainer Simen Gaure <Simen.Gaure@frisch.uio.no>
Repository CRAN
Date/Publication 2019-12-09 12:00:03 UTC
Description

The package contains methods for creating multivariate/multidimensional interpolations for real-valued functions on hypercubes. The methods include classical Chebyshev interpolation, multilinear and Floater-Hormann for arbitrary Cartesian product grids, and simplex linear and polyharmonic spline for scattered multi dimensional data.

Details

The primary method of the package is \texttt{ipol} which dispatches to some other method. All the generated \texttt{interpolants} accept as an argument a matrix of column vectors. The generated functions also accept an argument 

\begin{verbatim}
threads=getOption("chebpol.threads")
\end{verbatim}


\begin{verbatim}
to utilize more than one CPU if a matrix of column vectors is evaluated. The option \texttt{chebpol.threads} is initialized from the environment variable \texttt{CHEBPOL_THREADS} upon loading of the package. It defaults to 1.
\end{verbatim}

The interpolants are ordinary R-objects and can be saved with \texttt{save()} and loaded later with \texttt{load()} or serialized/unserialized with other tools, just like any R-object. However, they contain calls to functions in the package, and while the author will make efforts to ensure that generated interpolants are compatible with future versions of \texttt{chebpol}, I can issue no such absolute guarantee.

Chebyshev

If we are free to evaluate the function to interpolate in arbitrary points, we can use a Chebyshev interpolation. The classical one is available with \texttt{ipol(...,method='chebyshev')}.

Uniform grids

There are several options if your function must be evaluated in a uniform grid. There is the Floater-Hormann rational interpolation available with \texttt{ipol(...,method='fh')}.

There is a transformed Chebyshev variant \texttt{ipol(...,method='uniform')}.
Arbitrary grids

For grids which are not uniform, but still Cartesian products of one-dimensional grids, there is the Floater-Hormann interpolation `ipol(...,method='fh')`, and a transformed Chebyshev variant `ipol(...,method='general')`, as well as a multilinear `ipol(...,method='multilinear')`. These methods work on uniform grids as well. There is also a method='stalker' which is a shape preserving spline, and a variant method='hstalker'. Both are described in a vignette. These splines in one dimension attempts to honour monotonicity and local extrema, in particular the 'hstalker' is positivity-preserving in the sense that it will preserve maxima and minima.

Scattered data

For scattered data, not necessarily organised as a Cartesian product grid, there is a simplex linear interpolation available with `ipol(...,method='simplexlinear')`, and a polyharmonic spline with `ipol(...,method='polyharmonic')`.

Support functions

There are also functions for producing Chebyshev grids (`chebknots`) as well as a function for evaluating a function on a grid (`evalongrid`), and a function for finding the Chebyshev coefficients (`chebcoef`).

See Also

`ipol`, `interpolant`

Examples

```r
## make some function values on a 50x50x50 grid
f <- function(x) sqrt(1+x[1])*exp(x[2])*sin(5*x[3])^2
value <- evalongrid(f, dims)
##Fit a Chebyshev approximation to it. Note that the value-array contains the grid-structure.
ch <- ipol(value,method='cheb')
## To see the full grid, use the chebknots function and expand.grid
## Not run:
head(cbind(expand.grid(chebknots(dims)), value=as.numeric(value),
        appx=as.numeric(evalongrid(ch,dims))))

## Make a Floater-Hormann approximation on a uniform grid as well
fh <- ipol(f,grid=lapply(dims,function(n) seq(-1,1,length.out=n)),method='fh',k=5)
## evaluate in some random points in R3
m <- matrix(runif(15,-1,1),3)
rbind(true=apply(m,2,f), cheb=ch(m), fh=fh(m))
```
chebcoef

Compute Chebyshev-coefficients given values on a Chebyshev grid

Description
Compute the multivariate Chebyshev-coefficients, given values on a Chebyshev grid.

Usage
chebcoef(val, dct = FALSE)

Arguments
val
An array of function values on a Chebyshev grid. The dim-attribute must be appropriately set. If not set, it is assumed to be one-dimensional.
dct
Logical. Since the Chebyshev coefficients are closely related to the DCT-II transform of val, the non-normalized real-even DCT-II coefficients may be retrieved instead. I.e. those from FFTW_REDFT10 in each dimension. This is not used anywhere in the package, it is merely provided as a convenience for those who might need it.

Details
If val has no dim-attribute, it is assumed to be one-dimensional of length the length of val.
If chebpol was compiled without FFTW, running chebcoef on large grids may be slow and memory-demanding.

Value
An array of Chebyshev-coefficients for an interpolating Chebyshev-polynomial.

See Also
havefftw

Examples

## Coefficients for a 2x3x4 grid
a <- array(rnorm(24), dim=c(2,3,4))
chebcoef(a)
Evaluate a Chebyshev interpolation in a point

Description

Given Chebyshev coefficients, evaluate the interpolation in a point.

Usage

chebeval(x, coef, intervals = NULL, threads = getOption("chebpol.threads"))

Arguments

x | The point to evaluate.
coef | The Chebyshev coefficients. Typically from a call to chebcoef, possibly modified.
intervals | A list of minimum and maximum values. One for each dimension of the hypercube.
threads | And integer. In case x is a matrix of column vectors, use this number of threads in parallel to evaluate.

Value

A numeric. The interpolated value.

Examples

# make a function which is known to be unsuitable for Chebyshev approximation
f <- function(x) sign(x)
# make a standard Chebyshev interpolation
ch <- ipol(f, dims=50, method='chebyshev')
# then do a truncated interpolation
val <- evalongrid(f, 50)
coef <- chebcoef(val)
# truncate the high frequencies
coef[-(1:10)] <- 0
# make a truncated approximation
tch <- Vectorize(function(x) chebeval(x, coef))
# make a lower degree also
ch2 <- ipol(f, dims=10, method='chebyshev')
# plot the functions
## Not run:
s <- seq(-1, 1, length.out=400)
plot(s, ch(s), col='red', type='l')
lines(s, tch(s), col='blue')
lines(s, f(s))
chebknots

Create a Chebyshev-grid

Description

Create a Chebyshev grid on a hypercube.

Usage

chebknots(dims, intervals = NULL)

Arguments

dims

The number of grid-points in each dimension. For Chebyshev-polynomial of
degree dims-1.

intervals

A list of vectors of length 2. The lower and upper bounds of the hypercube.

Details

If intervals is not provided, it is assumed that the domain of the function in each dimension is
[-1,1]. Thus, standard Chebyshev knots are produced. If dims is of length 1, intervals may be a
vector of length 2 rather than a list with a vector of length 2.

Value

A array of dimension dims. The Chebyshev grid-points.

Examples

## Standard knots for degree 3
chebknots(4)
## Knots in the interval [2,3] for degree 3
chebknots(4, interval=c(2,3))
## Multivariate knots
chebknots(c(x=3, y=4, z=3))
## Multivariate grid
## Not run:
expand.grid(chebknots(c(x=3, y=4, z=5), list(c(1,3), c(4,6), c(800,900))))
## End(Not run)
Description

As of version 1.7, the functions for creating various interpolants have been deprecated. These are `chebappx`, `chebappxf`, `chebappxg`, `chebappxgf`, `fhappx`, `mlappx`, `slappx`, `ucappx`, `ucappxf`, `polyh`. The wrapper `ipol` should be used instead. The wrapper gives a uniform interface.

evalongrid

Evaluate a function on a grid

Description

Evaluate a function on a Chebyshev grid, or on a user-specified grid.

Usage

evalongrid(fun, dims, intervals = NULL, ..., grid = NULL)
evalongridV(fun, dims, intervals = NULL, ..., grid = NULL)

Arguments

- `fun` Multivariate real-valued function to be evaluated. Must be defined on the hyper-cube described by `intervals`.
- `dims` A vector of integers. The number of grid-points in each dimension.
- `intervals` A list. Each entry is a vector of length 2 with the lower and upper end of the interval in each dimension.
- `...` Further arguments to `fun`.
- `grid` Rather than specifying `dims` and `intervals` to get a Chebyshev grid, you may specify your own `grid` as a list of vectors whose Cartesian product will be the grid, as in `expand.grid(grid)`.

Details

The function `fun` should be a function(x,...), where `length(x)` equals `length(dims)` (or `length(grid)`).

If `grid` is provided, `fun` is evaluated on each point in the Cartesian product of the vectors in `grid`.

If `intervals` is not provided, it is assumed that the domain of the function is the hypercube [-1,1] x [-1,1] x ... x [-1,1]. Thus, the function is evaluated on a standard Chebyshev grid.

If `intervals` is provided, it should be a list with elements of length 2, providing minimum and maximum for each dimension.
The grid itself may be produced by `expand.grid(chebknots(dims, intervals))`, or `expand.grid(grid)`. This function does the same as `apply(expand.grid(grid), 1, fun)`, but it’s faster and more memory-efficient for large grids because it does not actually expand the grid.

The function `evalongridV` is for vectorized functions, i.e. those that can take a matrix of column vectors as argument. It’s equivalent to `fun(t(expand.grid(grid)))`.

**Value**

An array with the value of `fun` on each grid point. The `dim` attribute has been appropriately set for the grid. If `fun` returns a vector, this will be the first dimension of the returned array.

**Examples**

```r
f <- function(x) {a <- sum(x^2); ifelse(a == 0, 0, exp(-1/a))}
## Standard Chebyshev grid
evalongrid(f, dims=c(3,5))
## Then Chebyshev on [0,1] x [2,3]
evalongrid(f, dims=c(3,5), intervals=list(c(0,1), c(2,3)))
## And on my own grid
grid <- list(sort(rnorm(3)), sort(rnorm(5)))
evalongrid(f, grid=grid)
g <- ipol(f, grid=grid, method='fh')
evalongridV(g, grid=grid, threads=2)
## vector valued function
f <- function(x) c(prod(x), sum(x^2))
evalongrid(f, grid=grid)
```

---

`havealglib`  
*Check whether chebpol has the ALGLIB library*

**Description**

If ALGLIB was available at compile time, it can be used for compact support radial basis function interpolations on scattered data. This function checks whether ALGLIB is available.

**Usage**

`havealglib()`

**Value**

Returns `TRUE` if ALGLIB is available. Otherwise `FALSE`. 
havefftw  Check whether chebpol uses FFTW

Description

It is possible to compile chebpol without FFTW. If this is done, it will not be feasible with high-
degree Chebyshev polynomials. I.e. a 100x100x100 approximation will be possible, but not a
one-dimensional 1000000. This function checks whether chebpol uses FFTW.

Usage

havefftw()

Value

Returns TRUE if chebpol uses FFTW. Otherwise FALSE.

interpolant  Evaluate an interpolant in a point

Description

An interpolant is a function returned by ipol which has prespecified values in some points, and
which fills in between with some reasonable values.

Arguments

x  The argument of the function. A function of more than one variable takes a
vector. x can also be a matrix of column vectors.

threads  The number of threads to use for evaluation. All interpolants created by chebpol
are parallelized. If given a matrix argument x, the vectors can be evaluated in
parallel.

...  Other parameters. Currently used for simplex linear interpolants with the logical
argument epol which makes the interpolant extrapolate to points outside the do-
main. The stalker spline has the argument blend=c("linear","cubic","sigmoid")
where a blending function can be chosen as described in a vignette. The "multilinear"
interpolant also has such a blending function.

Value

A numeric. If more than one point was evaluated, a vector.
Examples

```r
grid <- list(x=seq(0,1,length.out=10), y=seq(0,1,length.out=10))
val <- runif(100)
dim(val) <- c(10,10)
ip <- ipol(val, grid=grid, method='fh')
ip(c(0.3, 0.8))
ip(matrix(runif(12),2), threads=2)
```

---

**ipol**

*Create interpolating function.*

---

**Description**

Create an interpolating function from given values. Several interpolation methods are supported.

**Usage**

```r
ipol(val, dims = NULL, intervals = NULL, grid = NULL, knots = NULL,
k = NULL, method = c("chebyshev", "multilinear", "fh", "uniform",
"general", "polyharmonic", "simplexlinear", "hstalker", "stalker",
"crbf", "oldstalker"), ...)
```

**Arguments**

- `val`: array or function. Function values on a grid, or the function itself. If it is the values, the "dim"-attribute must be appropriately set. If it is a function, it will be evaluated in the grid points.
- `dims`: integer vector. The number of grid points in each dimension. Not needed if `val` is an array or `grid` is used.
- `intervals`: list of length 2 numeric vectors. The lower and upper bound in each dimension. Not used if `grid` is specified.
- `grid`: list. Each element is a vector of ordered grid-points for a dimension. These need not be Chebyshev-knots, nor evenly spaced.
- `knots`: matrix. Each column is a point in an M-dimensional space.
- `k`: numeric. Additional value, used with some methods.
- `method`: character. The interpolation method to use.
- `...`: Further arguments to the function, if is.function(val). And some extra arguments for interpolant creation described in section Details.
Details

`ipol` is a wrapper around various interpolation methods in package `chebpol`. Which arguments to specify depends on the method. The interpolation methods are described in vignette("chebpol",package="chebpol").

The method "chebyshev" needs only the number of Chebyshev knots in each dimension. This is either the "dim" attribute of the array `val`, or the `d` argument if `val` is a function. Also the intervals can be specified if different from [-1, 1].

The method "uniform" is similar to the "chebyshev", but uniformly spaced knots are created. The argument intervals generally goes with `d` when something else than standard intervals [-1,1] are used.

The methods "multilinear", "fh" (Floater-Hormann), "stalker", "hstalker", and "general" needs the argument `grid`. These are the methods which can use arbitrary Cartesian grids. The stalker spline is described in vignette("stalker",package="chebpol"). The Floater-Hormann method ("fh") also needs the `k` argument, the degree of the blending polynomials. It defaults to 4.

The method "polyharmonic" needs the arguments `knots` and `k`. In addition it can take the logical argument `normalize` for normalizing the knots to the unit hypercube. The default is NA, which uses normalization if any of the knots are outside the unit hypercube. Also, a logical `nowarn` is accepted, it is used to suppress a warning in case the system can’t be solved exactly and a least squares fallback method is used.

The method "simplexlinear" needs the argument `knots`. It creates a Delaunay triangulation from the knots, and does linear interpolation in each simplex by weighting the vertex values with the barycentric coordinates.

If knots are required, but the grid argument is given, knots are constructed as `t(expand.grid(grid))`

The "crbf" is the multilayer compact radial basis function interpolation in ALGLIB ([http://www.alglib.net/interpolation/fastrbf.php](http://www.alglib.net/interpolation/fastrbf.php)). It is only available if ALGLIB was available at compile time. It takes the extra arguments "rbase", "layers", and "lambda". These are discussed in the ALGLIB documentation.

There are also some usage examples and more in vignette("chebpol") and vignette('chebusage').

Value

A function(`x,threads=getOption('chebpol.threads')`) defined on a hypercube, an interpolant for the given function. The argument `x` can be a matrix of column vectors which are evaluated in parallel in a number of threads. The function yields values for arguments outside the hypercube as well, though it will typically be a poor approximation. `threads` is an integer specifying the number of parallel threads which should be used when evaluating a matrix of column vectors.

Examples

```r
## evenly spaced grid-points
su <- seq(0,1,length.out=10)
## irregularly spaced grid-points
s <- su^3
## create approximation on the irregularly spaced grid
ml1 <- ipol(exp, grid=list(s), method='multilin')
fh1 <- ipol(exp, grid=list(s), method='fh')
## test it, since exp is convex, the linear approximation lies above
```
## the exp between the grid points
ml1(su) - \exp(su)
fh1(su) - \exp(su)

## multi dimensional approximation
f <- function(x) 10/(1+25*mean(x^2))
# a 3-dimensional $10 \times 10 \times 10$ grid, first and third coordinate are non-uniform
grid <- list(s, su, sort(1-s))

# make multilinear, Floater-Hormann, Chebyshev and polyharmonic spline.
ml2 <- ipol(f, grid=grid, method='multilin')
fh2 <- ipol(f, grid=grid, method='fh')
hst <- ipol(f, grid=grid, method='hstalker')
ch2 <- ipol(f, dims=c(10,10,10), intervals=list(0:1,0:1,0:1), method='cheb')
knots <- matrix(runif(3*1000),3)
ph2 <- ipol(f, knots=knots, k=2, method='poly')
sl2 <- ipol(f, knots=knots, method='simplexlinear')

# my alglib is a bit slow, so stick to 100 knots
if(havealglib()) crb <- ipol(f, knots=knots[,1:100], method='crbf',
    rbase=2, layers=5, lambda=0)

# make 7 points in R3 to test them on
m <- matrix(runif(3*7),3)
rbind(true=apply(m,2,f), ml=ml2(m), fh=fh2(m), cheb=ch2(m), poly=ph2(m), sl=sl2(m), hst=hst(m),
crb=if(havealglib()) crb(m) else NULL )
Index

* Topic Chebyshev
  chebpol-package, 2

* Topic DCT
  chebpol-package, 2

* Topic Floater-Hormann
  chebpol-package, 2

* Topic interpolation
  chebpol-package, 2

chebcoef, 3, 4, 5
chebeval, 5
chebknots, 3, 6, 8
chebpol (chebpol-package), 2
chebpol-deprecated, 7
chebpol-package, 2

evalongrid, 3, 7
evalongridV (evalongrid), 7

havealglib, 8
havefftw, 4, 9

interpolant, 2, 3, 9, 11
ipol, 2, 3, 9, 10