Package ‘chebpol’

October 28, 2015

Version 1.3-1789
Date 2015-10-28
Title Multivariate Chebyshev Interpolation
Author Simen Gaure, Ragnar Frisch Centre for Economic Research, Oslo, Norway
Maintainer Simen Gaure <Simen.Gaure@frisch.uio.no>
Copyright 2013, Simen Gaure
Imports compiler, stats
Suggests
Description Contains methods for creating multivariate Chebyshev approximation of functions on a hypercube. Some methods for non-Chebyshev grids are also provided.
License Artistic-2.0
Classification/MSC 41A05, 41A10, 41A50, 41A63, 65D05, 65T40
Classification/ACM G.1.2
NeedsCompilation yes
Repository CRAN
Date/Publication 2015-10-28 18:39:09

R topics documented:

chebpol-package ......................................................... 2
chebappx ............................................................... 3
chebappxg ............................................................. 4
chebcoef ............................................................... 6
chebeval ............................................................... 7
chebknots ............................................................. 8
evalongrid ............................................................ 9
havefftw ............................................................ 10
mlappx ............................................................... 10
polyh ................................................................. 11
ucappx ............................................................... 12
Description

The package contains methods for creating multivariate Chebyshev interpolations for real-valued functions on hypercubes. Some methods for non-Chebyshev grids are also provided.

Details

Given a real-valued function on a hypercube, or hyper-rectangle, it may be approximated by a (multivariate) Chebyshev polynomial. In the one-dimensional case, the Chebyshev approximation is constructed by evaluating the function in certain points, a Chebyshev grid, and fitting a polynomial to these values. Thus, one needs the function values on a set of prespecified points. The multivariate case is similar, the grid is the Cartesian product of one-dimensional grids. I.e. all combinations of grid-points.

The Chebyshev coefficients for the interpolating polynomial in the one-dimensional case is a simple linear transform of the function values. The Chebyshev-transform, or Discrete Cosine Transform, being a variant of the Fourier transform, factors over tensor products, thus the multivariate transform is just a tensor product of several one-dimensional transforms. If `FFTW` was available at compile time, `chebpol` uses it to generate the Chebyshev coefficients, otherwise a slower and more memory-demanding matrix method is used, and a warning message is issued when the package is attached. The Chebyshev-approximation is defined on the interval [-1,1], but it is straightforward to map any interval into [-1,1], thus making Chebyshev approximation on an interval of choice. Or, a hypercube of choice.

The primary method of the package is `chebappx` which takes as input the function values on the grid, possibly together with a hypercube specification in the form of a list of intervals. It produces a function which interpolates on the hypercube. There is also a wrapper called `chebappxf` which may be used if one has the function to be approximated rather than only its values in the grid-points. There is even an interpolation for uniform grids in `ucappx`, with a wrapper in `ucappxf` with interesting examples. And a more general for arbitrary Cartesian-product grids in `chebappxg` with a wrapper in `chebappxgf`. These are based on transforms of Chebyshev-polynomials. A multilinear interpolation is available in `mlappx`. And a polyharmonic spline interpolation in `polyh`.

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

Examples

```r
## make some function values on a 3x3x4 grid
dims <- c(x=3,y=3,z=4)
value <- array(runif(36),dims)
## fit a Chebyshev approximation to it. Note that the value-array contains the
## grid-structure.  (However, we don't really approximate the runif-function :)
ch <- chebappx(value)
```
---

**chebappx**  
*Chebyshev interpolation on a hypercube*

**Description**  
Given function, or function values on a Chebyshev grid, create an interpolation function defined in the whole hypercube.

**Usage**  
```r  
chebappx(val, intervals=NULL)  
chebappxf(fun, dims, intervals=NULL, ...)
```

**Arguments**  
- `val` The function values on the Chebyshev grid. `val` should be an array with appropriate dimension attribute.  
- `intervals` A list of minimum and maximum values. One for each dimension of the hypercube. If NULL, assume [-1,1] in each dimension.  
- `fun` The function to be approximated.  
- `dims` Integer. The number of Chebyshev points in each dimension.  
- `...` Further arguments to `fun`.  

**Details**  
If `intervals` is not provided, it is assumed that the domain of the function is the Cartesian product [-1,1] x [-1,1] x ... x [-1,1]. Where the number of grid-points are given by `dim(val)`.  
For `chebappxf`, the function is provided instead, and the number of grid points in each dimension is in the vector `dims`. The function is evaluated on the Chebyshev grid.  
If `intervals` is provided, it should be a list with elements of length 2, providing minimum and maximum for each dimension. Arguments to the function will be transformed from these intervals into [-1,1] intervals.  
The approximation function may be evaluated outside the hypercube, but be aware that it may be highly erratic there, especially if of high degree.
Value

A function defined on the hypercube. A Chebyshev approximation to the function \texttt{fun}, or the values provided in \texttt{val}.

Examples

\begin{verbatim}
f <- function(x) exp(-sum(x^2))
## we want 3 dimensions, i.e. something like
## f(x,y,z) = exp(-(x^2 + y^2 + z^2))
## 8 points in each dimension
gridsize <- list(8,8,8)
# get the function values on the Chebyshev grid
values <- evalongrid(f,gridsize)
# make an approximation
ch <- chebappx(values)
## test it:
a <- runif(3,-1,1); ch(a)-f(a)

## then one with domain [0.1,0.3] x [-1,-0.5] x [0.5,2]
intervals <- list(c(0.1,0.3),c(-1,-0.5),c(0.5,2))
# evaluate on the grid
values <- evalongrid(f,gridsize,intervals)
# make an approximation
ch2 <- chebappx(values,intervals)
a <- c(0.25,-0.68,1.43); ch2(a)-f(a)
# outside of domain:
a <- runif(3); ch2(a); f(a)

# Make a function on [0,2] x [0,1]
f <- function(y) uniroot(function(x) x-y[[1]]*cos(pi*x^2),lower=0,upper=1)$root*sum(y^2)
# approximate it
ch <- chebappxf(f,c(12,12),intervals=list(c(0,2),c(0,1)))
# test it:
a <- c(runif(1,0,2),runif(1,0,1)); ch(a); f(a)
\end{verbatim}

chebappxg \hspace{1cm} Interpolation on a non-Chebyshev grid

Description

A poor-man’s approximation on non-Chebyshev grids. If you for some reason can’t evaluate your function on a Chebyshev-grid, but instead have some other grid which still is a Cartesian product of one-dimensional grids, you may use this function to create an interpolation.

Usage

\begin{verbatim}
chebappxg(val, grid=NULL, mapdim=NULL)
chebappxgf(fun, grid, ..., mapdim=NULL)
\end{verbatim}
chebappxg

Arguments

val
Array. Function values on a grid.

grid
A list. Each element is a sorted vector of grid-points for a dimension. These need not be Chebyshev-knots, nor evenly spaced.

fun
The function to be approximated.

... Further arguments to fun.

mapdim
Deprecated.

Details

A call fun <- chebappxg(val, grid) does the following. A Chebyshev interpolation ch for val is created, on the [-1,1] hypercube. For each dimension a grid-map function gm is created which maps the grid-points monotonically into Chebyshev knots. For this, the function splinefun with method = "monoH.FC" is used. When fun(x) is called, it translates to ch(gm(x)). For uniform grids, the function ucappx will produce a faster interpolation in that a closed form gm is used.

chebappxgf is used if the function, rather than its values, is available. The function will be evaluated on the grid.

Even though this approach works in simple cases it is not a panacea. The grid in each dimension should probably not be too irregularly spaced. I.e. short and long gaps interspersed is likely to cause problems.

Value

A function(x) defined on the hypercube, approximating the given function.

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
ch <- Vectorize(chebappxg(exp(s),list(s)))
## test it:
ch(su) - exp(su)
# try one with three variables
f <- function(x) exp(-sum(x^2))
grid <- list(s,su,su^2)
ch2 <- chebappxg(evalongrid(f,grid=grid),grid)
# test it at 10 random points
replicate(10,{a<-runif(3); ch2(a)-f(a)})

# Try Runge's function on a uniformly spaced grid.
# Ordinary polynomial fitting of high degree of Runge's function on a uniform grid
# creates large oscillations near the end of the interval. Not so with chebappxgf
f <- function(x) 1/(1+25*x^2)
chg <- Vectorize(chebappxgf(f,seq(-1,1,length.out=15)))
# also compare with Chebyshev interpolation
ch <- Vectorize(chebappxf(f,15))
```
## Not run:

```r
# plot it
s <- seq(-1, 1, length.out=200)
plot(s, f(s), type='l', col='black')
lines(s, chg(s), col='blue')
lines(s, ch(s), col='red')
legend('topright',
      legend=c('Runge function', 'chebappxg on uniform grid', 'Chebyshev'),
      col=c('black', 'blue', 'red'), lty=1)
```

## End(Not run)

---

**chebcoef**

*Compute Chebyshev-coefficients given values on a Chebyshev grid*

### Description

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

### Usage

```r
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`.

- **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

```r
## Coefficients for a 2x3x4 grid
a <- array(rnorm(24),dim=c(2,3,4))
chebcoef(a)
```

Description

Given Chebyshev coefficients, evaluate the interpolation in a point.

Usage

```r
tchebeval(x, coef, intervals=NULL)
```

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 hyper-cube.

Value

A numeric. The interpolated value.

Examples

```r
# make a function which is known to be unsuitable for Chebyshev approximation
f <- function(x) sign(x)
# make a standard Chebyshev interpolation
ch <- Vectorize(chebappxf(f,50))
# 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 <- Vectorize(chebappxf(f,10))
# 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')
```
chebknets
lines(s,f(s))
lines(s,ch2(s),col='green')

## End(Not run)

---

chebknets

*Create a Chebyshev-grid*

**Description**

Create a Chebyshev grid on a hypercube.

**Usage**

```
chebknets(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**

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

## End(Not run)
**Description**

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

**Usage**

`evalongrid(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 $[-1,1] \times [-1,1] \times ... \times [-1,1]$. 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(chebknobs(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.

**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
def <- 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(rnorm(3), rnorm(5))
evalongrid(f, grid=grid)
# Vector valued function
f <- function(x) c(prod(x), sum(x^2))
evalongrid(f, grid=grid)
```

havefftw  

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

```r
havefftw()
```

Value

Returns TRUE if chebpol uses FFTW. Otherwise FALSE.

mlappx  

Multilinear interpolation on a grid

Description

Multilinear interpolation on an arbitrary Cartesian product.

Usage

```r
mlappx(val, grid, ...)
```

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.
- **grid**: A list. Each element is a vector of grid-points for a dimension. These need not be Chebyshev-knots, nor evenly spaced.
- **...**: Further arguments to the function, if `is.function(val)`. 

**Details**

A call `fun <- mappx(val,grid)` creates a multilinear interpolation on the grid. The value on the grid points will be exact, the value between the grid points is a convex combination of the values in the corners of the hypercube surrounding it.

If `val` is a function it will be evaluated on the grid.

**Value**

A function of `x` defined on the hypercube, approximating the given function. The function yields values for arguments outside the hypercube as well, as a linear extension.

**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 <- Vectorize(mappx(exp,list(s)))
test it, since exp is convex, the linear approximation lies above
the exp between the grid points
ml1(su) - exp(su)

multi linear approx
f <- function(x) exp(sum(x^2))
gird <- list(s,su)

ml2 <- mappx(evalongrid(f,grid=gird),grid)
# an equivalent would be ml2 <- mappx(f,grid)

a <- runif(2); ml2(a); f(a)
# we also get an approximation outside of the domain, of disputable quality
ml2(c(1,2)); f(c(1,2))
```

---

**polyh**  
*Polyharmonic splines on scattered data*

**Description**

Polyharmonic splines on scattered data.

**Usage**

`polyh(val, knots, k=2, ...)`
Arguments

- `val`: array or function. Function values on scattered data, or the function itself.
- `knots`: matrix. Each column is a point in a multidimensional space.
- `k`: positive integer or negative numeric. The degree of the polyharmonic spline.
- `...`: Further arguments to the function, if `is.function(val)`.

Details

`polyh` fits a polyharmonic spline with radial basis function $x^k$ for odd $k$, and $x^k \log(x)$ for even $k$. If $k < 0$, the basis $\exp(k \ x^2)$ is used. There are more details in a vignette.

If `val` is a function it will be evaluated on the knots.

Value

A function $f(x)$ defined on the multidimensional space, approximating the given function.

Examples

```r
# a function on a 20-dimensional space
f <- function(x) 10/(10+sum(sqrt(x)))
knots <- matrix(runif(16000), 20)
phs <- polyh(f, knots, 3)
# test it in a random point
a <- runif(20)
f(a); phs(a)
```

ucappx | Interpolation on a uniform grid

Description

A poor-man's approximation on uniform grids. If you for some reason can't evaluate your function on a Chebyshev-grid, but instead have a uniform grid, you may use this function to create an interpolation.

Usage

```r
ucappx(val, intervals=NULL)
ucappxf(fun, dims, intervals=NULL, ...)
```

Arguments

- `val`: Array. Function values on a grid.
- `intervals`: List of vectors of length two. Specifying the hypercube extent in each dimension
- `fun`: Function to be interpolated.
- `dims`: Integer. Number of grid points in each dimension.
- `...`: Further arguments to `fun`. 
Details

This does about the same as `chebappxg` for uniform grids, though no grid map function is constructed, as a fixed such function is used.

A Chebyshev-interpolation \( ch \) is made for \( val \) with `chebappx`. Upon evaluation the uniform grid in each dimension is mapped differentiably to the Chebyshev-knots so that \( ch \) is evaluated in 
\[
sin\left(\frac{\pi(1-\frac{n}{2})}{n}\right)
\]
where \( n \) is the number of knots in the dimension, possibly after \( x \) has been remapped from the hypercube interval to \([-1,1]\).

Thus, the interpolation is not a polynomial.

For `ucappx` the function values are provided, the number of grid points in each dimension is to be found in `dim(val)`. For `ucappxf` the function to be interpolated is `fun`, and the number of grid points is passed in `dims`.

As the example shows, this approximation is better than the Chebyshev approximation for some functions.

Value

A function \( x \) defined on the hypercube, approximating the given function.

Examples

```r
# Runge function
f <- function(x) 1/(1+25*x^2)
grid <- seq(-1,1,length.out=15)
val <- f(grid)
uc <- Vectorize(ucappx(val))
# and the Chebyshev
ch <- Vectorize(chebappx(f,15))
# test it at 10 random points
t(replicate(10,a<-runif(1,-1,1); c(arg=a, uc=uc(a), true=f(a), cheb=ch(a)))))
```
Index

*Topic **Chebyshev approximation**
  chebpol-package, 2

*Topic **DCT**
  chebpol-package, 2

*Topic **Discrete Cosine Transform**
  chebpol-package, 2

chebappx, 2, 3, 13
chebappxf, 2
chebappxf (chebappx), 3
chebappxg, 2, 4, 13
chebappxgf, 2
chebappxgf (chebappxg), 4
chebcoef, 2, 6, 7
chebeval, 7
chebknots, 2, 8, 9
chebpol (chebpol-package), 2
chebpol-package, 2

evalongrid, 2, 9

havefftw, 6, 10

mlappx, 2, 10
mlappxf (mlappx), 10

polyh, 2, 11

splinefun, 5

ucappx, 2, 5, 12
ucappxf, 2
ucappxf (ucappx), 12