Package 'linBilin'

September 18, 2015

Type Package

Title Fast 1-, 2-, and 3-dimensional linear interpolation

Version 1.0

Date 2015-09-08

Author Philip Barrett

Maintainer Philip Barrett <pobarrett@gmail.com>

Description Provides fast linear interpolation of grids in up to 3 dimensions using RcppArmadillo. Includes C++ header file for interface in C++.

License GPL 3

Imports Rcpp (>= 0.11.5)

LinkingTo Rcpp, RcppArmadillo

R topics documented:

linBilin-package *Fast linear interpolation of 1-, 2- and 3-dimensional functions*

Details

Package:	linBilin
Type:	Package
Version:	1.0
Date:	2015-09-08
License:	GPL 3

Given a function defined on one, two, or three variables, this package will provide very fast linear interpolation using the Armadillo C++ library. A C++ interface is also available in the `linBillin.hpp` file in the `src/` directory.

Author(s)

Philip Barrett

Maintainer: Philip Barrett <pobarrett@gmail.com>

See Also

`lin` `bilin` `trilin`

Examples

```
# See individual function documentation for examples
```

bilin *Fast 2D Linear interpolation*

Description

Provides very fast 2D linear interpolation.

Usage

```
bilin( pt, x, y, f, n_x, n_y)
```

Arguments

pt	An (x, y) vector at which to approximately evaluate a function
x	A vector of x-values
y	A vector of y-values
f	A vector of z-values ordered first by x and then by y. If $x = (x_1, x_2)$ and $y = (y_1, y_2)$ then $f = (f(x_1, y_1), f(x_1, y_2), f(x_2, y_1), f(x_2, y_2))$
n_x	An integer. The length of x
n_y	An integer. The length of y

Value

Returns the bilinear interpolation of (x, y, f) evaluated at pt

Examples

```
## 2D Interpolation
f <- function(pt) pt[1] * cos(pt[2]) + pt[2] ^ 2 * sin(pt[1])
    # An arbitrary function
n.coarse <- 7
    # Number of points at which the interpolating grid is defined
n.fine <- 201
    # Number of poiints to evaluate the interpolation
xy.coarse <- cbind( rep( 1:n.coarse, each=n.coarse), rep( 1:n.coarse, n.coarse ) )
    # The matrix of coarse (x,y) values at which to evaluate f
v.f.w <- apply( xy.coarse, 1, f )
m.f.w <- matrix( v.f.w, n.coarse, n.coarse )
    # The vector & matrix form of the z-values

xy.fine <- seq( 1, n.coarse, length.out=n.fine )
    # A fine grid on which to evaluate the interpolation
m.xy.fine <- cbind( rep( xy.fine, each=n.fine), rep( xy.fine, n.fine ) )
    # The matrix form
v.f.v <- apply( m.xy.fine, 1, bilin, x=1:n.coarse, y=1:n.coarse, f=v.f.w, n_x=n.coarse, n_y=n.coarse )
    # The linear interpolation
v.f.true <- apply( m.xy.fine, 1, f )
    # The true value
m.f.v <- matrix( v.f.v, n.fine, n.fine )
m.f.true <- matrix( v.f.true, n.fine, n.fine )
    # Convert to matrices for plotting

image( 1:n.coarse, 1:n.coarse, m.f.w )
    # f evaluated on the approximating grid
image( xy.fine, xy.fine, m.f.v )
contour( xy.fine, xy.fine, m.f.v, add=TRUE )
    # The linear interpolation based on the coarse grid
image( xy.fine, xy.fine, m.f.true )
contour( xy.fine, xy.fine, m.f.true, add=TRUE )
    # The true function evaluated on the fine grid
```

lin *Fast 1D Linear interpolation*

Description

Provides very fast 1D linear interpolation. Typically about 10 times faste than R's built-in approx function.

Usage

```
lin( x, y, pt, n_x)
```

Arguments

x	A vector of x-values
y	A vector of y-values
pt	An x-value
n_x	An integer. The length of x (and y).

Value

Returns the linear interpolation of (x,x) evaluated at x=pt. That is, if x= $(x_1, x_2, \ldots, x_n)$, y= $(y_1, y_2, \ldots, y_n)$, and pt= $x, x_i < x \leq x_{i+1}$, then the output is:

$$y = \left(\frac{x - x_i}{x_{i+1} - x_i} \right) y_i + \left(1 - \frac{x - x_i}{x_{i+1} - x_i} \right) y_{i+1}$$

Examples

```
## 1D Interpolation
f.1 <- function(x) x^2
n <- 5
x.grid <- seq( 0, 10, length.out = n )
f.1.grid <- sapply( x.grid, f.1 )
lin( x.grid, f.1.grid, 3, n ) == 3
    # Check interpolation
x.grid.fine <- seq( 0, 10, length.out = n^2 )
plot( x.grid.fine, sapply( x.grid.fine, f.1 ),
      main='Exact and linear interpoliation of x^2', type='l' )
lines( x.grid.fine, sapply( x.grid.fine, lin, x=x.grid, y=f.1.grid, n_x=n ),
        col='red' )
    # Compare the exact and approximate solution
library(microbenchmark)
microbenchmark(sapply( x.grid.fine, lin, x=x.grid, y=f.1.grid, n_x=n ))
microbenchmark(sapply( x.grid.fine, approx, x=x.grid, y=f.1.grid ))
    # lin is about 10x faster than R's built-in approx function
```

`trilin` *Fast 3D linear interpolation*

Description

Takes values of a function evaluated on a grid and computes an approximate value on the interior of the grid using linear interpolation.

Usage

```
trilin(pt, x, y, z, m_f, n_x, n_y, n_z)
```

Arguments

`pt`	An (x, y, z) vector at which to approximately evaluate a function
`x`	A vector of x-values
`y`	A vector of y-values
`z`	A vector of z-values
`m_f`	The matrix of function values on the grid $x \times y \times z$. Must be ordered vertically by `x` and then `y`, and horizontally by `z`. For example, if $x = (x_1, x_2)$, $y = (y_1, y_2)$, and $z = (z_1, z_2)$, then `m_f` is: $$R = \begin{bmatrix} f(x_1, y_1, z_1) & f(x_1, y_1, z_2) \\ f(x_1, y_2, z_1) & f(x_1, y_2, z_2) \\ f(x_2, y_1, z_1) & f(x_2, y_1, z_2) \\ f(x_2, y_2, z_1) & f(x_2, y_2, z_2) \end{bmatrix}$$
`n_x`	An integer. The length of `x`
`n_y`	An integer. The length of `y`
`n_z`	An integer. The length of `z`

Value

Returns the trilinear interpolation of (x, y, z, f) evaluated at `pt`

Examples

```
f <- function(pt) pt[1] + pt[2] * pt[3] - pt[3] * pt[1]
    # An arbitrary function
nn <- 7
    # Number of points at which the interpolating grid is defined
v.x <- v.y <- v.z <- 1:nn
    # The vector of x, y, and z values (need not all be the same)
m.xy <- cbind( rep( v.x, each=nn), rep( v.y, nn ) )
    # The matrix of (x,y) values at which to evaluate f
v.f <- t( apply( m.xy, 1, function( xy ) sapply( v.z, function(z) f( c( xy, z ) ) ) ) )
    # The matrix of the z-values
```

```
test.pt <- 1:3
print(f(test.pt))
print(trilin( test.pt, v.x, v.y, v.z, v.f, 7, 7, 7 ) )
    # Is exact at grid points

test.pt.2 <- 1:3 + .2
print(f(test.pt.2))
print(trilin( test.pt.2, v.x, v.y, v.z, v.f, 7, 7, 7 ) )
    # Approximation is still exact with quadratic functions
```

Index