Package 'RcppEigenAD'

September 11, 2025

Type Package

Title Generate Partial Derivatives using 'Rcpp', 'Eigen' and 'CppAD'

Version 1.1.0

Date 2025-09-03

Description Compiles 'C++' code using 'Rcpp' <doi:10.18637/jss.v040.i08>, 'Eigen' <doi:10.18637/jss.v052.i05> and 'CppAD' to produce first and second order partial derivatives. Also provides an implementation of Faa' di Bruno's formula to combine the partial derivatives of composed functions.

License GPL (>= 2)

Imports Rcpp (>= 0.12.12), methods, functional, memoise, readr, Rdpack

Suggests RcppEigen,BH

LinkingTo Rcpp,RcppEigen,BH

RdMacros Rdpack

NeedsCompilation yes

Author Daniel Grose [aut, cre],
Robert Crouchley [aut, ctb]

Maintainer Daniel Grose <dan.grose@lancaster.ac.uk>

Repository CRAN

Date/Publication 2025-09-11 05:50:07 UTC

Contents

H 2
J 3
sourceCppAD 5
%.% 6

Index **8**

H *Construct a function to calculate the Hessian of a function.*

Description

Constructs a function to calculate the Hessian of a function produced either using sourceCppAD or the composition operator %.% . The returned function has the same argument signature as f but returns a matrix representing the blocked Hessians of f evaluated at the functions arguments. The partial derivatives are formed with respect to the argument specified when f was created with sourceCppAD.

In what follows it is assumed that f has a single matrix argument (the one with which f is differentiated with respect to). When this is not the case, the other arguments will be considered constant at the point the Hessian is evaluated at. Consequently, the structure of the output of the function produced by H is unchanged by the additional arguments. The blocked Hessian $\mathbf{H}$ is organised as follows.

If $f : \mathbf{R}^n \to \mathbf{R}^m$ where $\mathbf{Y}_{n_Y \times m_Y} = f(\mathbf{X}_{n_X \times m_X})$ and $n = n_X m_X$, $m = n_Y m_Y$ then by numbering the elements of the matrices row-wise so that,

$$\mathbf{Y} = \begin{bmatrix} y_1 & \cdots & y_{m_Y} \\ y_{m_Y+1} & \cdots & y_{2m_Y} \\ \vdots & \ddots & \vdots \\ y_{(n_Y-1)m_Y+1} & \cdots & y_{n_Y m_Y} \end{bmatrix}$$

and

$$\mathbf{X} = \begin{bmatrix} x_1 & \cdots & x_{m_X} \\ x_{m_X+1} & \cdots & x_{2m_X} \\ \vdots & \ddots & \vdots \\ x_{(n_X-1)m_X+1} & \cdots & x_{n_X m_X} \end{bmatrix}$$

then the $n \times nm$ blocked Hessian matrix $\mathbf{H}$ is structured as

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 & \mathbf{H}_2 & \ldots & \mathbf{H}_m \end{bmatrix}$$

with $\mathbf{H}_k$ the Hessian matrix for y_k i.e.,

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial^2 y_k}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 y_k}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 y_k}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 y_k}{\partial x_n \partial x_n} \end{bmatrix}$$

Usage

```
H(f)
```

Arguments

f	A function created using either sourceCppAD or the composition operator %.%.

Value

A function which computes the Hessian of the function.

Examples

```
library(RcppEigenAD)
# define f as the eigen vectors of its argument x
# calculated using the Eigen library
f<-sourceCppAD('
ADmat f(const ADmat& X)
{
   Eigen::EigenSolver<ADmat> es(X);
   return es.pseudoEigenvectors();
}
')
Hf<-H(f)
X<-matrix(c(1,2,3,4),2,2)
Hmat<-Hf(X)
Hmat # the Hessian matrices of second derivatives stacked column wise
Hmat[3,9] # the second derivative of f(X)[2,1] with respect to X[2,1] and X[1,2]
```

J *Construct a function to calculate the Jacobian of a function.*

Description

Constructs a function to calculate the Jacobian of a function produced either using sourceCppAD or the composition operator %.%. The returned function has the same argument signature as f but returns a matrix representing the Jacobian of f evaluated at the functions arguments. The partial derivatives are formed with respect to the arguments specified when f was created with sourceCppAD.

In what follows it is assumed that f has a single matrix argument (the one with which f is differentiated with respect to). When this is not the case, the other arguments will be considered constant at the point the Jacobian is evaluated at. Consequently, the structure of the output of the function produced by J is unchanged by the additional arguments. The Jacobian matrix $\mathbf{J}$ is organised as follows.

If $f : \mathbf{R}^n \to \mathbf{R}^m$ where $\mathbf{Y}_{n_Y \times m_Y} = f(\mathbf{X}_{n_X \times m_X})$ and $n = n_X m_X$, $m = n_Y m_Y$ then by numbering the elements of the matrices row-wise so that,

$$\mathbf{Y} = \begin{bmatrix} y_1 & \cdots & y_{m_Y} \\ y_{m_Y+1} & \cdots & y_{2m_Y} \\ \vdots & \ddots & \vdots \\ y_{(n_Y-1)m_Y+1} & \cdots & y_{n_Y m_Y} \end{bmatrix}$$

and

$$\mathbf{X} = \begin{bmatrix} x_1 & \cdots & x_{m_X} \\ x_{m_X+1} & \cdots & x_{2m_X} \\ \vdots & \ddots & \vdots \\ x_{(n_X-1)m_X+1} & \cdots & x_{n_X m_X} \end{bmatrix}$$

then the $m \times n$ Jacobian matrix is given by

$$\mathbf{J} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Usage

```
J(f)
```

Arguments

f	A function created using either sourceCppAD or the composition operator %.%.

Value

A function which computes the Jacobian of the function

Examples

```
# define f as the eigen vectors of its argument X
# calculated using the Eigen library
library(RcppEigenAD)
f<-sourceCppAD('
ADmat f(const ADmat& X)
{
   Eigen::EigenSolver<ADmat> es(X);
   return es.pseudoEigenvectors();
}
')
Jf<-J(f)
X<-matrix(c(1,2,3,4),2,2)
Jmat<-Jf(X)
Jmat # the Jacobian matrix of first derivatives
Jmat[2,3] # the derivative of f(X)[1,2] with respect to X[2,1]
```

sourceCppAD *Construct a function to calculate the Jacobian of a function.*

Description

Constructs an R function that wraps a call to a C++ function and compiles the C++ code used to define the function. The code must define a function having arguments of the type const ADmat& and a return type of ADmat. The ADmat type is a type definition for a dynamically sized matrix from the Eigen linear algebra library. (Guennebaud G, Jacob B and others 2010). The function can employ methods from the Eigen library, such as all of the standard operators from linear algebra, and a wide range of functions for calculating inverses, decompositions and so on. See Guennebaud G, Jacob B and others (2010) for further details.

The returned function can then be used as an argument to `J` or `H` which provide functions that apply algorithmic differentiation to calculate the Jacobian or Hessian matrices. The functions produced by `J` or `H` evaluate the partial derivatives with respect to the elements of the argument located at the position in the functions argument list that is specified by the `wrt` argument of sourceCppAD.

Usage

```
sourceCppAD(code=NULL,file=NULL,wrt=1,output="method")
```

Arguments

`code`	A character vector containing the C++ code to compile.
`file`	The filename of a file containing the C++ code to compile.
`wrt`	The position of the argument that `J` and `H` are differentiated with respect to.
`output`	Specifies what is generated by sourceCppAD. If output="method" (the default) then a function which invokes the compiled code is returned. If output="code" then the source code that wraps the users function for use with the algorithmic differentiation libraries is returned.

Value

A function which invokes the compiled code or the c++ code that wraps the users function for use with the algorithmic differentiation libraries.

References

Guennebaud G, Jacob B, others (2010). “Eigen v3.” http://eigen.tuxfamily.org.

Examples

```
library(RcppEigenAD)
# define a function to calculate sin(cos(x)) where x is a matrix
f<-sourceCppAD('
ADmat f(const ADmat& X)
{
```

```
  return X.array().cos().sin();
}
')
x<-matrix(c(1,2,3,4),2,2)
# call it
f(x)
```

`%.%` *Compose two functions created using either sourceCppAD or* `%.%` *itself*

Description

Returns a function with a matrix `X` as input which computes the value of $(f \circ g)(x) = f(g(x))$. Note that the order of the composition is such that `g` is applied to `X` first. `f` is then applied to the result of `g`. The returned function is compatible with both J and H, and if J and H are applied to a function produced by composition, the resulting Jacobian or Hessian matrices are constructed from the Jacobians and Hessians of `f` and `g` using a combinatorical form of Faa di Bruno's formula (Hardy 2006). The functions `f` and `g` must both be functions of a single argument. Note that a function of multiple arguments can be Curried into a function with a single argument. The R package functional provides the method `Curry` which is convenient for this purpose.

Usage

```
f %.% g
```

Arguments

`f`	Function to be composed with `g`
`g`	Function to be composed with `f`

Value

A function which computes the composition `f` and `g`

References

Hardy M (2006). "Combinatorics of Partial Derivatives." *Electr. J. Comb.*, **13**(1). https://dblp.uni-trier.de/db/journals/combinatorics/combinatorics13.html#Hardy06.

Ma T (2009). "Higher Chain Formula proved by Combinatorics." *Electr. J. Comb.*, **16**.

Examples

```
library(RcppEigenAD)
# define a function to calculate the eigen vectors of a matrix
f<-sourceCppAD('
ADmat f(const ADmat& X)
{
  Eigen::EigenSolver<ADmat> es(X);
  return es.pseudoEigenvectors();

}
')
# define function to calculate the inverse of a matrix
g<-sourceCppAD('
ADmat g(const ADmat& X)
{
  return X.inverse();
}
')
# compose f and g to produce a functions to calculate the eigenvectors of the inverse of a matrix
h<-f%.%g  #  h = f o g
x<-matrix(c(1,2,3,4),2,2)
x<-x%*%t(x) # positive definite matrix
J(h)(x) # Jacobian of h = f o g
H(h)(x) # Stacked Hessians of h = f o g
# redefine h as a function to directly calculate the eigenvectors of the inverse of a matrix
h<-sourceCppAD('
ADmat h(const ADmat& X)
{
  Eigen::EigenSolver<ADmat> es(X.inverse());
  return es.pseudoEigenvectors();
}
')
# calculate the Jacobian and Hessian of h to compare with previous result
J(h)(x) # Jacobian of h = f o g
H(h)(x) # Stacked Hessians of h = f o g
```

Index

%.%, *2–4*, 6

Curry, *6*

H, 2, *5*, *6*

J, 3, *5*, *6*

sourceCppAD, *2–4*, 5, *6*