Package ‘numDeriv’

April 11, 2006

Title Accurate Numerical Derivatives

Description Accurate Numerical Derivatives. See ?numDeriv.Intro for more details.

Depends R (>= 1.8.1)

Version 2006.4-1

LazyLoad yes

License GPL Version 2.

Author Paul Gilbert

Maintainer Paul Gilbert and Ravi Varadhan <pgilbert@bank-banque-canada.ca>

URL http://www.bank-banque-canada.ca/pgilbert

R topics documented:

`00.numDeriv.Intro` *Accurate Numerical Derivatives*

Description

Calculate (accurate) numerical approximations to derivatives.

Details

See `numDeriv-package` (in the help system use package?numDeriv or ?"numDeriv-package") for an overview.

genD *Generate Bates and Watts D Matrix*

Description

Generate a matrix of function derivative information.

Usage

```
genD(func, x, method="Richardson",
                   method.args=list(), ...)
## Default S3 method:
genD(func, x, method="Richardson",
  method.args=list(eps=1e-4, d=0.1, r=4, v=2), ...)
```

Arguments

`func`	a function for which the first (vector) argument is used as a parameter vector.
`x`	The parameter vector first argument to `func`.
`method`	one of `"Richardson"` or `"simple"` indicating the method to use for the aproximation.
`method.args`	arguments passed to method. (Arguments not specified remain with their default values.)
`...`	any additional arguments passed to `func`.

Details

The derivatives are calculated numerically using Richardson improvement. The method "simple" is not supported in this function.) The "Richardson" method calculates a numerical approximation of the first and second derivatives of `func` at the point `x`. For a scalar valued function these are the gradient vector and Hessian matrix. (See `grad` and `hessian`.) For a vector valued function the first derivative is the Jacobian matrix (see `jacobian`). See `grad` for more details on the Richardson's extrapolation parameters.

The the first order derivative with respect to x_i is

$$f'_i(x) = < f(x_1, \ldots, x_i + d, \ldots, x_n) - f(x_1, \ldots, x_i - d, \ldots, x_n) > /(2 * d)$$

The second order derivative with respect to x_i is

$$f''_i(x) = < f(x_1, \ldots, x_i + d, \ldots, x_n) - 2 * f(x_1, \ldots, x_n) + f(x_1, \ldots, x_i - d, \ldots, x_n) > /(2 * d)$$

The second order derivative with respect to x_i, x_j is

$$f''_{i,j}(x) = < f(x_1, \ldots, x_i + d, \ldots, x_j + d, \ldots, x_n) - 2 * f(x_1, \ldots, x_n) +$$

$$f(x_1, \ldots, x_i - d, \ldots, x_j - d, \ldots, x_n) > /(2 * d^2) - (f''_i(x) + f''_j(x))/2$$

Value

A list with elements as follows: `D` is a matrix of first and second order partial derivatives organized in the same manner as Bates and Watts, the number of rows is equal to the length of the result of `func`, the first p columns are the Jacobian, and the next p(p+1)/2 columns are the lower triangle of the second derivative (which is the Hessian for a scalar valued `func`). `p` is the length of `x` (dimension of the parameter space). `f0` is the function value at the point where the matrix `D` was calculated. The `genD` arguments `func`, `x`, `d`, `method`, and `method.args` also are returned in the list.

References

Linfield, G.R. and Penny, J.E.T. (1989) "Microcomputers in Numerical Analysis." Halsted Press.

Bates, D.M. & Watts, D. (1980), "Relative Curvature Measures of Nonlinearity." J. Royal Statistics Soc. series B, 42:1-25

Bates, D.M. and Watts, D. (1988) "Non-linear Regression Analysis and Its Applications." Wiley.

See Also

`hessian`, `grad`

Examples

```
func <- function(x){c(x[1], x[1], x[2]^2)}
z <- genD(func, c(2,2,5))
```

---

grad — *Numerical Gradient of a Function*

Description

Calculate the gradient of a function by numerical approximation.

Usage

```
 grad(func, x, method="Richardson", method.args=list(), ...)

## Default S3 method:
grad(func, x, method="Richardson",
  method.args=list(eps=1e-4, d=0.0001, r=4, v=2, show.details=FALSE), ...)
```

Arguments

`func`	a function with a scalar real result (see details).
`x`	a real scalar or vector argument to func, indicating the point(s) at which the gradient is to be calculated.
`method`	one of `"Richardson"` or `"simple"` indicating the method to use for the approximation.
`method.args`	arguments passed to method. (Arguments not specified remain with their default values.)
`...`	an additional arguments passed to `func`.

Details

The function `grad` calculates a numerical approximation of the first derivative of `func` at the point `x`. Any additional arguments in ... are also passed to `func`, but the gradient is not calculated with respect to these additional arguments. It is assumed `func` is a scalar value function. If a vector `x` produces a scalar result then `grad` returns the numerical approximation of the gradient at the point `x` (which has the same length as `x`). If a vector `x` produces a vector result then the result must have the same length as `x`, and it is assumed that this corresponds to applying the function to each of its arguments (for example, `sin(x)`). In this case `grad` returns the gradient at each of the points in `x` (which also has the same length as `x` - so be careful). An alternative for vector valued functions is provided by `jacobian`.

If method is "simple", the calculation is done using a simple epsilon difference. For this case, only the element `eps` of `methods.args` is used.

If method is "Richardson", the calculation is done by Richardson's extrapolation (see e.g. Linfield and Penny, 1989, or Fornberg and Sloan, 1994.) This method should be used if accuracy, as opposed to speed, is important. For this case, `methods.args=list(eps=1e-4, d=0.01, r=4, show.details=FALSE)` are used. `d` gives the fraction of `x` to use for the initial numerical approximation. The default means the initial approximation uses `0.0001 * x`. `eps` is used instead of `d` for elements of `x` which are zero. `r` gives the number of Richardson improvement iterations (repetitions with successly smaller `d`. The default `4` general provides good results, but this can be increased to `6` for improved accuracy at the cost of more evaluations. `v` gives the reduction factor. `show.details` is a logical indicating if detailed calculations should be shown.

The general approach in the Richardson method is to iterate for `r` iterations from initial values for interval value `d`, using reduced factor `v`. The the first order approximation to the derivative with respect to x_i is

$$f'_i(x) = < f(x_1, \ldots, x_i + d, \ldots, x_n) - f(x_1, \ldots, x_i - d, \ldots, x_n) > /(2 * d)$$

This is repeated `r` times with successively smaller `d` and then Richardson extraplolation. is applied.

Value

A real scalar or vector of the approximated gradient(s).

References

Linfield, G. R. and Penny, J. E. T. (1989) *Microcomputers in Numerical Analysis*. New York: Halsted Press.

Fornberg and Sloan (Acta Numerica, 1994, p. 203-267)

See Also

`jacobian`, `hessian`, `numericalDeriv`

Examples

```
grad(sin, pi)
grad(sin, (0:10)*2*pi/10)
func0 <- function(x){ sum(sin(x))  }
grad(func0 , (0:10)*2*pi/10)

func1 <- function(x){ sin(10*x) - exp(-x) }
```

```
curve(func1,from=0,to=5)

x <- 2.04
numd1 <- grad(func1, x)
exact <- 10*cos(10*x) + exp(-x)
c(numd1, exact, (numd1 - exact)/exact)

x <- c(1:10)
numd1 <- grad(func1, x)
exact <- 10*cos(10*x) + exp(-x)
cbind(numd1, exact, (numd1 - exact)/exact)
```

---

hessian — *Calculate Hessian Matrix*

Description

Calculate a numerical approximation to the Hessian matrix of a function at a parameter value.

Usage

```
hessian(func, x, method="Richardson", method.args=list(), ...)

## Default S3 method:
hessian(func, x, method="Richardson",
    method.args=list(eps=1e-4, d=0.1, r=4, v=2), ...)
```

Arguments

`func`	a function for which the first (vector) argument is used as a parameter vector.
`x`	the parameter vector first argument to func.
`method`	one of `"Richardson"` or `"simple"` indicating the method to use for the aproximation.
`method.args`	arguments passed to method. (Arguments not specified remain with their default values.)
`...`	an additional arguments passed to `func`.

Details

The function `hessian` calculates an numerical aproximation to the n x n second derivative of a scalar real valued function with n-vector argument. It uses `genD` and extracts the second derivative.

Value

An n by n matrix of the Hessian of the function calculated at the point `x`.

See Also

`jacobian`, `grad`, `genD`

jacobian *Gradient of a Vector Valued Function*

Description

Calculate the m by n numerical approximation of the gradient of a real m-vector valued function with n-vector argument.

Usage

```
jacobian(func, x, method="Richardson", method.args=list(), ...)

## Default S3 method:
jacobian(func, x, method="Richardson",
   method.args=list(eps=1e-4, d=0.0001, r=4, v=2, show.details=FALSE), ...)
```

Arguments

`func`	a function with a real (vector) result.
`x`	a real or real vector argument to func, indicating the point at which the gradient is to be calculated.
`method`	one of `"Richardson"` or `"simple"` indicating the method to use for the aproximation.
`method.args`	arguments passed to method. (Arguments not specified remain with their default values.)
`...`	any additional arguments passed to `func`.

Details

For $f : R^n -> R^m$ calculate the mxn Jacobian dy/dx. The function `jacobian` calculates a numerical approximation of the first derivative of `func` at the point `x`. Any additional arguments in ...are also passed to `func`, but the gradient is not calculated with respect to these additional arguments.

If method is "simple", the calculation is done using a simple epsilon difference. For this case, only the `methods.args` element `eps` is used. If method is "Richardson", the calculation is done by Richardson's extrapolation. See `link{grad}` for more details.

Value

A real m by n matrix.

See Also

`grad`, `hessian`, `numericalDeriv`

Examples

```
func2 <- function(x) c(sin(x), cos(x))
x <- (0:1)*2*pi
jacobian(func2, x)
```

`numDeriv-package` *Accurate Numerical Derivatives*

Description

Calculate (accurate) numerical approximations to derivatives.

Details

Package:	numDeriv
Depends:	R (>= 1.8.1)
License:	GPL Version 2. (See LICENSE file.)

The main functions are

`grad` to calculate the gradient (first derivative) of a scalar real valued function (possibly applied to all elements of a real vector argument).

`jacobian` to calculate the gradient of a real m-vector valued function with real n-vector argument.

`hessian` to calculate the Hessian (second derivative) of a scalar real valued function with real n-vector argument.

`genD` to calculate the gradient and second derivative of a real m-vector valued function with real n-vector argument.

Maintainer: Paul Gilbert <pgilbert@bank-banque-canada.ca> and Ravi Varadhan <rvaradhan@jhmi.edu>

Author(s)

Paul Gilbert, based on work by Xingqiao Liu

References

Linfield, G. R. and Penny, J. E. T. (1989) *Microcomputers in Numerical Analysis*. New York: Halsted Press.

Fornberg and Sloan, (1994) *Acta Numerica*, p. 203-267; Table 1, page 213)

Index