Package ‘optimise2’

April 14, 2015

Type Package

Title optim corrections.

Version 1.0

Date 2014-04-17

Author Yifan Yang

Maintainer Yifan Yang <yifan.yang@uky.edu>

Description The rule used here is gsl_min_fminimizer_quad_golden with more than one guess. Constrained (linear/equal region) MLE. Lambda Selection

License GPL

Depends Rcpp (>= 0.10.2), RcppGSL, quadprog, glmnet

NeedsCompilation yes

Archs i386

R topics documented:

`optimise2-package` *Optimization in one dimensional problem.*

Description

Minimizer: gsl min fminimizer quad golden This is a variant of Brentâ€™s algorithm which uses the safeguarded step-length algorithm of Gill and Murray.

Details

Package:	optimise2
Type:	Package
Version:	1.0
Date:	2014-04-17
License:	GPL2

The usage is exactly the same as optimise(stats) unless you want to enlarge the number of iterations.

Author(s)

Yifan Yang

Maintainer: Yifan Yang <yifan.yang@uky.edu>

References

Richard Brent, Algorithms for minimization without derivatives, Prentice-Hall (1973), republished by Dover in paperback (2002), ISBN 0-486-41998-3.

See Also

optimise

Examples

```
f <- function(x) sin(x^2) + x/10
optimise(f, c(1, 4), tol = 1e-06)
optimise(f, c(1.5, 11))  # WRONG!!!
optimise2(f, c(1.5, 11), tol = 1e-06)
```

---

lambda_Select *Select Lambda using Dr Zhou's approach*

Description

Select Lambda using Dr Zhou's approach

Usage

```
lambda_Select(Yvec, Xmat, lambdas = seq(from = 0.001, to = 1, length.out =
  50), nPsep = 20, perc = 0.2, ...)
```

Arguments

Yvec	Y vector(n)
Xmat	Design Matrix(n by p)
lambdas	Proposed lambda, default value is seq(from=.001,to=1,length.out=50)
nPsep	Number of independent simulation data.
perc	Stopping percentage

Value

result Result from glmnet

selection T/F of lambdas. F means

Examples

```
set.seed(65535)
Xmat = matrix(rnorm(100*80),ncol=80)
beta0 = rnorm(80,sd=2)
beta0[sample(1:80,70)] = 0.
epsilon = rnorm(100)
Yvec = Xmat%*%beta0 + epsilon
lambdas = seq(from=.001,to=1,length.out=50) # a vec
lambda_Select(Yvec,Xmat)
```

`lmcholsolve` *lmcholsolve*

Description

This function solves

$$H\beta = y$$

, where H>=0;

Usage

```
lmcholsolve(x, y)
```

Arguments

x	Must be a numeric matrix: integer matrix are NOT allowed.
y	Response variable

Details

Considering:

$$H\beta = y$$

, in each iteration newton's scalor is

$$\beta_{LSE}$$

. The fatest algorithm is 'sweeping method'/Cholesky Decomposition. The mian reason is that: H >= 0. If we tend not to check the condition, then

$$H = LL^T$$

, where L is low tri-angle matrix. Hence we derive

$$[H, y]^T[H, y] = \begin{pmatrix} L & 0 \\ l^T & d \end{pmatrix} \times \begin{pmatrix} L^T & l \\ 0 & d \end{pmatrix}$$

and

$$Ll = H^T y, L^T\beta = l, d^2 = \|y - hat(y)\|^2_{\ell_2}$$

The performance cound be found in inst/doc.

Value

`beta`	The estimations.
`Convergences`	Number of non-zero diag-elements.

Note

tol=1e-9 in Cholesky decomposition.

Author(s)

Yifan Yang

References

Lange, Kenneth. Numerical analysis for statisticians. chap 7. Springer, 2010.

Examples

```
x <- matrix(runif(100), 10)
y = rnorm(10) * 2
x <- t(x) %*% x
lmcholsolve(x, y)
solve(x, y)
```

`lmconst` *LSE with equal constraint*

Description

Y = X beta with x0 beta = y0 constraint.

Usage

```
lmconst( y,x,x0,y0)
```

Arguments

`y`	length n vector: response Variablei
`x`	n by p matrix: corresponding Variables
`x0`	p by l matrix: constraint beta coef
`y0`	length l vector: value to constraints

Details

Use quadratic programming.

Value

`coef`	Constraint LSE/MLE(Normal assumption).

Note

The function will only output the coefficients.

Author(s)

Yifan Yang

References

Richard Brent, Algorithms for minimization without derivatives, Prentice-Hall (1973), republished by Dover in paperback (2002), ISBN 0-486-41998-3.

Examples

```
# Example 1: dim =1
x=seq( 0,10,0.1)
y = matrix( 6*x+7+runif( 101,min=-1,max=1) ,ncol=1)
x=matrix( x,ncol=1)
re = lmconst( y,x,2,19)
re
re[ 1] +2*re[ 2]

# Example 2: multi dim
# R CMD check will skipp %...
#x = matrix(runif(100),ncol=2)
#y = x %*% c(6,2) +1 + rnorm( 50)
#re = lmconst(y=y,x=x,x0=c(2,1),y0=15)
#re
#re[1] +2*re[2] +re[ 3]
```

`optimise2` *OPTIMISE2*

Description

One-dimensional optimization problem.

Usage

```
optimise2(f, interval, ..., lower = min(interval), upper = max(interval),
    maximum = FALSE, tol = .Machine$double.eps^0.25, initials = runif(1, lower,
        upper), trace = F, maxit = 100)
```

Arguments

`f`	target function
`interval`	Interval used to identify the optimization problem, can't be inf!
`...`	Other parameters used by f.
`lower`	Lower bound of the interval.
`upper`	Upper bound of the interval.
`maximum`	Max number of itterations performed.

`tol`	|a - b| < tol + epsrel min(|a|,|b|) . epsrel=0 is my case.
`initials`	Testing, no use right now.
`trace`	Testing, no use right now.
`maxit`	Max number of simulations used.

Details

Minimizer: gsl_min_fminimizer_goldensection The golden section algorithm is the simplest method of bracketing the minimum of a function. It is the slowest algorithm provided by the library, with linear convergence.

On each iteration, the algorithm first compares the subintervals from the endpoints to the current minimum. The larger subinterval is divided in a golden section (using the famous ratio (3-sqrt 5)/2 = 0.3189660...) and the value of the function at this new point is calculated. The new value is used with the constraint f(a') > f(x') < f(b') to a select new interval containing the minimum, by discarding the least useful point. This procedure can be continued indefinitely until the interval is sufficiently small. Choosing the golden section as the bisection ratio can be shown to provide the fastest convergence for this type of algorithm.

Minimizer: gsl_min_fminimizer_brent The Brent minimization algorithm combines a parabolic interpolation with the golden section algorithm. This produces a fast algorithm which is still robust.

The outline of the algorithm can be summarized as follows: on each iteration Brent's method approximates the function using an interpolating parabola through three existing points. The minimum of the parabola is taken as a guess for the minimum. If it lies within the bounds of the current interval then the interpolating point is accepted, and used to generate a smaller interval. If the interpolating point is not accepted then the algorithm falls back to an ordinary golden section step. The full details of Brent's method include some additional checks to improve convergence.

Minimizer: gsl_min_fminimizer_quad_golden This is a variant of Brent's algorithm which uses the safeguarded step-length algorithm of Gill and Murray.

Notice that, in original optimise(stats), the condition "f(a') > f(x') < f(b')" will not be checked.

Value

`maximum`	Optimization point.
`minimum`	Optimization point.
`objective`	Function value.

Note

The function will ignore the discontinouse points.

Author(s)

Yifan Yang

References

Richard Brent, Algorithms for minimization without derivatives, Prentice-Hall (1973), republished by Dover in paperback (2002), ISBN 0-486-41998-3.

Examples

```
f <- function(x) sin(x^2) + x/10
optimise(f, c(1, 4), tol = 1e-06)
optimise(f, c(1.5, 11))  #WRONG
optimise2(f, c(1.5, 11), tol = 1e-06)
```

`print.lambda_select` *Print lambda_select*

Description

Print lambda_select

Usage

```
## S3 method for class 'lambda_select'
print(x)
```

Arguments

`x`	print S3 method for class lambda_select