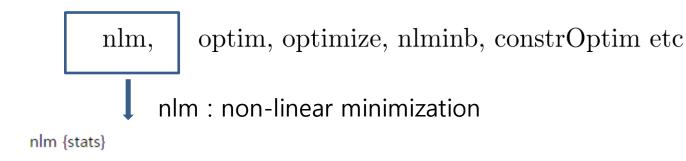
Numerical optimization II

Woojoo Lee

Some optimization functions in R



Non-Linear Minimizatio

Description

This function carries out a minimization of the function f using a Newton-type algorithm. See the references for detail

Usage

```
nlm(f, p, ..., hessian = FALSE, typsize = rep(1, length(p)),
   fscale = 1, print.level = 0, ndigit = 12, gradtol = 1e-6,
   stepmax = max(1000 * sqrt(sum((p/typsize)^2)), 1000),
   steptol = 1e-6, iterlim = 100, check.analyticals = TRUE)
```

Arguments

the function to be minimized, returning a single numeric value. This should be a function with first arg argument.

If the function value has an attribute called gradient or both gradient and hessian attributes, these will be used. deriv returns a function with suitable gradient attribute and optionally a hessian attribute.

starting parameter values for the minimization.

··· additional arguments to be passed to f.

R-example

nlm uses finite differences in its default mode. If you provide the derivatives, nlm works better!

```
f <- function(x, a) sum((x-a)^2)
nlm(f, c(10,10), a = c(3,5))
f <- function(x, a)
{
    res <- sum((x-a)^2)
    attr(res, "gradient") <- 2*(x-a)
    res
}
nlm(f, c(10,10), a = c(3,5))</pre>
```

Given x, find μ and σ maximizing

```
\ell(\mu, \sigma) = n \log(\sigma) - \sum_{i=1}^{n} \log(\sigma^2 + (x_i - \mu)^2)
ell <- function(theta,x) {
mu<-theta[1]
sigma<-theta[2]
n < -length(x)
res < -- n*log(sigma) + sum(log(sigma^2+(x-mu)^2))
denom < -(sigma^2 + (x-mu)^2)
deriv1<-sum(-2*(x-mu)/denom)
deriv2<--n/sigma+ sum(2*sigma/denom)
attr(res, "gradient") < -c(deriv1, deriv2)
return(res)
x < -rcauchy(100)
out < -nlm(ell, c(-0.5,1), x=x)
out
```

optim

General-purpose optimization based on Nelder–Mead, quasi-Newton and conjugate-gradient algorithms.

```
Usage
optim(par, fn, gr = NULL, ...,
    method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN",
      lower = -lnf, upper = lnf,
      control = list(), hessian = FALSE)
optimHess(par, fn, gr = NULL, ..., control = list())
Arguments
par
            Initial values for the parameters to be optimized over.
fn
            A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.
gr
            A function to return the gradient for the "BFGS", "CG" and "L-BFGS-B" methods. If it is NULL, a finite-difference approximation will be used.
            For the "SANN" method it specifies a function to generate a new candidate point. If it is NULL a default Gaussian Markov kernel is used.
            Further arguments to be passed to fn and gr.
method
            The method to be used. See 'Details', Can be abbreviated.
lower, upper
            Bounds on the variables for the "L-BFGS-B" method, or bounds in which to search for method "Brent".
control
            A list of control parameters. See 'Details'.
```

```
fr <- function(x) { ## Rosenbrock Banana function
   x1 < -x[1]
   x2 < -x[2]
   100 * (x2 - x1 * x1)^2 + (1 - x1)^2
grr <- function(x) { ## Gradient of 'fr'
   x1 < -x[1]
   x2 < -x[2]
   c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
     200 * (x2 - x1 * x1))
optim(c(-1.2,1), fr)
(res <- optim(c(-1.2,1), fr, grr, method = "BFGS"))
optimHess(res$par, fr, grr)
optim(c(-1.2,1), fr, NULL, method = "BFGS", hessian = TRUE)
## These do not converge in the default number of steps
optim(c(-1.2,1), fr, grr, method = "CG")
optim(c(-1.2,1), fr, grr, method = "CG", control = list(type = -1.2,1))
2))
```

Useful reference

Boyd, S. and Vandernberghe, L. (2004) Convex Optimization, Cambridge University Press