Package 'minic'

June 24, 2024

Type Package

Title Minimization Methods for Ill-Conditioned Problems

Version 1.0

Date 2024-05-27

Maintainer Bert van der Veen <bert_van_der_veen@hotmail.com>

Description Implementation of methods for minimizing ill-conditioned problems. Currently only includes regularized (quasi-)newton optimization (Kanzow and Steck et al. (2023), <doi:10.1007/s12532-023-00238-4>).

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.1

Imports Rcpp (>= 1.0.12)

LinkingTo Rcpp, RcppEigen

URL https://github.com/BertvanderVeen/minic

BugReports https://github.com/BertvanderVeen/minic/issues

NeedsCompilation yes

Author Bert van der Veen [aut, cre]

Repository CRAN

Date/Publication 2024-06-24 12:50:02 UTC

Contents

rnewton *Regularized quasi-Newton optimization*

Description

Performs regularized (quasi-)Newton optimisation with limited-memory BFGS, SR1, or PSB updates.

Usage

```
rnewton(x0, fn, gr,
  he = NULL,
  quasi = TRUE,
  method = "LBFGS",
  verbose = FALSE,
  return.hess = FALSE,
  control = list(maxit = 1000, m = 5, sigma1 = 0.5, sigma2 = 4, c1 = 0.001,
    c2 = 0.9, pmin = 0.001, tol.g = 1e-08, tol.gamma = 1e-05, tol.obj = 1e-08,
    tol.mu = 1e-04, tol.mu2 = 1e+15, tol.c = 1e-08, report.iter = 10,
    grad.reject = FALSE, max.reject = 50, mu0 = 5),
  ...
)
```

Arguments

x0	Initial values for the parameters.
fn	A function to be minimized.
gr	A function that returns the gradient.
he	A function that returns the hessian (only used when quasi = FALSE).
quasi	logical. Defaults to TRUE. If FALSE implements regularised Newton optimization.
method	The method to be used when quasi = TRUE. Defaults to "LBFGS", alternatives are "LPSB", "LSR1" and ther full-memory alternatives "BFGS", "SR1", "PSB". The latter three options should probably not be used in practice (see details).
verbose	logical. Defaults to FALSE. If TRUE prints reports on each iteration.
return.hess	logical. Defaults to FALSE. If TRUE returns (approximation of) the hessian at the final iteration.
control	a "list" of control options.

- *maxit*: The maximum number of iterations. Defaults to 1000.
- *m*: The number of gradients to remember from previous optimisation steps. Defaults to 5.
- *sigma1*: Step decrement factor. Defaults to 0.5. Must be smaller than 1 but larger than 0.
- *sigma2*: Step increment factor. Defaults to 4. Must be larger than 1.

- *c1*: First constant for determining step success. Defaults to 1e-3. See details.
- *c2*: Second constant for determining step success. Defaults to 0.9. See details.
- *pmin*: Third constant for determining (lack of) step success. Defaults to 1e-3.
- *tol.g*: Convergence tolerance for gradient. Defaults to 1e-8.
- *tol.gamma*: Threshold for gamma paramter. Defaults to 1e-5.
- *tol.obj*: Convergence tolerance for relative reduction in the objective, similar to "reltol" in "optim". Defaults to 1e-8.
- *tol.mu*: Minimum threshold for the regularisation parameter. Defaults to 1e-4.
- *tol.mu2*: Maximum threshold for the regularisation parameter. Defaults to 1e15.
- *tol.c*: Tolerance for cautious updating. Defaults to 1e-8.
- *report.iter*: If 'verbose = TRUE', how often should a report be printed? Defaults to every 10 iterations.
- *max.reject*: Maximum number of consecutive rejections before algorithm terminates.
- *grad.reject*: Logical. If TRUE the gradient is evaluated at every iteration and information of rejected steps is incorporated in limited-memory methods. Defaults to FALSE.
- *mu0.reject*: Initial value of the regularisation parameter. Defaults to 5.

... Not used.

Details

This function implements some of the regularised (quasi-)Newton optimisation methods presented in Kanzow and Steck (2023) with one modification; gradient information of rejected steps is incorporated by default. The full-memory options that are implemented rely on explicitly inverting the approximated Hessian and regularisation penalty, are thus slow, and should probably not be used in practice.

The function start with a single More-Thuente line search along the normalized negative gradient direction. The code for this was originally written in matlab by Burdakov et al. (2017), translated to python by Kanzow and Steck (2023), and separately translated to R code for this package.

A step is considered somewhat successful for $c_1 < \rho \leq c_2$, where ρ is the proportion of achieved and predicted reduction in the objective function. Note the requirement $c_1 \in (0, 1)$ and $c_2 \in (c_1, 1)$. A step is considered highly successful for $c_2 < \rho$, where rho is the proportion of achieved and predicted reduction in the objective function.

The σ_1 constant controls the decrement of the regularisation parameter μ on a highly succesful step. The σ_2 constant controls the increment of the regularisation parameter μ on a unsuccesful step. A step is defned as unsuccesful if 1) the predicted reduction less than pmin times the product of the l2 norm for step direction and gradient, or 2) if $\rho \leq c_1$.

Value

An object of class "rnewton" including the following components:

`objective:`	The value of fn corresponding to par.
`iterations:`	Number of completed iterations.
`evalg:`	Number of calls to gr.
`par:`	The best set of parameters found.
`info:`	Convergence code.
`maxgr:`	Maximum absolute gradient component.
`convergence:`	logical, TRUE indicating succesful convergence (reached tol.obj or tol.g).

Author(s)

Bert van der Veen

References

Burdakov, O., Gong, L., Zikrin, S., & Yuan, Y. X. (2017). On efficiently combining limited-memory and trust-region techniques. Mathematical Programming Computation, 9, 101-134.

Kanzow, C., & Steck, D. (2023). Regularization of limited memory quasi-Newton methods for large-scale nonconvex minimization. Mathematical Programming Computation, 15(3), 417-444.

Examples

```
# Powell's quartic function
fn <- function(x) {
  (x[1] + 10*x[2])^2 + 5 * (x[3] - x[4])^2 +
  (x[2] - 2*x[3])^4 + 10 * (x[1] - x[4])^4
}

# Gradient
gr <- function(x) {
  c(2 * (x[1] + 10*x[2]) + 40 * (x[1] - x[4])^3,    # dfdx1
    20 * (x[1] + 10*x[2]) + 4 * (x[2] - 2 * x[3])^3,# dfdx2
    10 * (x[3] - x[4]) - 8 * (x[2] - 2*x[3])^3,     # dfdx3
    -(10 * (x[3] - x[4]) + 40 * (x[1] - x[4])^3))   # dfdx4
}

# Lower tolerances from default
rnewton(c(1, 1, 1, 1), fn, gr, control = list(mu0 = 1, tol.g = 1e-10, tol.obj = 0))
```

Index