Package 'trustOptim'

March 27, 2018

Type Package

Title Trust Region Optimization for Nonlinear Functions with Sparse Hessians

Version 0.8.6.2

Date 2018-03-26

Maintainer Michael Braun <braunm@smu.edu>

URL http://coxprofs.cox.smu.edu/braunm

Description Trust region algorithm for nonlinear optimization. Efficient when the Hessian of the objective function is sparse (i.e., relatively few nonzero cross-partial derivatives). See Braun, M. (2014) <doi:10.18637/jss.v060.i04>.

License MPL (>= 2.0)

Depends R (>= 3.4.0)

Suggests testthat, knitr

Imports Matrix (>= 1.2.12), Rcpp (>= 0.12.13), methods

LinkingTo Rcpp, RcppEigen (>= 0.3.3.3.0)

VignetteBuilder knitr

SystemRequirements C++11

RoxygenNote 6.0.1

NeedsCompilation yes

Author Michael Braun [aut, cre, cph]

Repository CRAN

Date/Publication 2018-03-27 20:46:40 UTC

R topics documented:

binary 2
binary-data 2
trust.optim 3
trustOptim 6

Index **8**

`binary` *Binary choice example*

Description

Functions for binary choice example in the vignette.

Usage

```
binary.f(P, data, priors, order.row = FALSE)

binary.grad(P, data, priors, order.row = FALSE)

binary.hess(P, data, priors, order.row = FALSE)
```

Arguments

`P`	Numeric vector of length (N+1)*k. First N*k elements are heterogeneous coefficients. The remaining k elements are population parameters.
`data`	List of data matrices Y and X, and choice count integer T
`priors`	List of named matrices inv.Omega and inv.Sigma
`order.row`	Determines order of heterogeneous coefficients in parameter vector. Affects sparsity pattern of Hessian. See vignette.

Details

Hessian is sparse, and returned as a dgcMatrix object

Value

Log posterior density, gradient and Hessian.

`binary-data` *Sample simulated data for binary choice model in vignette*

Description

Simulated data. See vignette. Generated from data-raw/binary.R

`trust.optim` *Nonlinear optimizers using trust regions.*

Description

Run nonlinear minimizer using trust region algorithm with conjugate gradient search directions and quasi-Hessian updates.

Usage

```
trust.optim(x, fn, gr, hs = NULL, method = c("SR1", "BFGS", "Sparse"),
  control = list(), ...)
```

Arguments

`x`	A numeric vector of starting values for the optimizer.
`fn`	An R function that takes `x` as its first argument. Returns the value of the objective function at `x`. Note that the optimizer will *minimize* `fn` (see function.scale.factor under control)
`gr`	An R function that takes x as its first argument. Returns a numeric vector that is the gradient of `fn` at `x`. The length of the gradient must be the same as the length of `x`. The user must supply this function. If an analytic gradient is not available, and the method is `SR1` or `BFGS`, the user should consider a numerical approximation using finite differencing (see the numDeriv package). Do not use a finite-differenced gradient with the `Sparse` method. That will cause a world of hurt.
`hs`	An R function that takes x as its first argument. Returns a Hessian matrix object of class `dgCMatrix` (see the **Matrix** package). This function is called only if the selected method is `Sparse`.
`method`	Valid arguments are `SR1`,`BFGS`,and `Sparse`.
`control`	A list containing control parameters for the optimizer. See details.
`...`	Additional arguments passed to `fn`, `gr` and `hs`. All arguments must be named.

Value

List containing the following items:

`fval`	Value of the objective function
`solution`	Parameter vector at the optimum
`gradient`	Gradient at the optimum
`hessian`	Estimate of the Hessian at the optimum (as class `symmetricMatrix`, returned only for `Sparse` method).
`iterations`	Number of iterations before stopping
`status`	A message describing the last state of the iterator

nnz For the Sparse method only, the number of nonzero elements in the lower triangle of the Hessian

.

Details

The following sections explain how to use the package as a whole.

Control parameters

The control list should include the following parameters.

start.trust.radius Initial radius of the trust region. Default is 5. If the algorithm returns non-finite values of the objective function early in the process, try a lower number.

stop.trust.radius Minimum radius of trust region. Algorithm will terminate if radius is below this value. This is because it may not be possible to get the norm of the gradient smaller than prec, and this is another way to get the algorithm to stop.

cg.tol tolerance for the conjugate gradient algorithm that is used for the trust region subproblem. Set it to something very small. Default is sqrt(.Machine$double.eps)

prec Precision for how close the norm of the gradient at the solution should be to zero, before the algorithm halts. It is possible that the algorithm will not get that far, so it will also stop when the radius of the trust region is smaller thanstop.trust.radius. If the trust region radius collapses, but the norm of the gradient really isn't close to zero, then something terrible has happened.

report.freq An integer. The frequency at which the algorithm will display the current iteration number or function value, among other things (see `report.level`). Defaults to 1.

report.level The amount of detail in each report. Defaults to 2.

report.precision The number of significant digits used in each report. Defaults to 5.

maxit Maximum number of iterations. Defaults to 100.

contract.factor When the algorithm decides to shrink the trust region, it will multiply the trust radius by this factor. Defaults to 0.5.

expand.factor When the algorithm decides to expand the trust region, it will multiply the algorithm by this factor. Defaults to 3.

contract.threshold The algorithm with accept a proposed move if the ratio of the actual improvement in the objective function, to the predicted improvement from the trust region subproblem, is greater than this amount. Otherwise, the trust region will contract. Default is 0.25.

expand.threshold.ap First criterion to determine if the trust region should expand. If the ratio of the actual and proposed improvements in the objective function is less than this factor, the algorithm will consider expanding the trust region. See `expand.threshold.radius`. Default is 0.8.

expand.threshold.radius If the ratio of the actual and proposed improvement in the objective function is less than `expand.threshold.ap`, then, if the normed distance of the proposed move is greater than `expand.threshold.radius`, times the current trust region radius, the trust region will expand. Default is 0.8.

function.scale.factor The algorithm will minimize `fn` times this factor. If you want to maximize `fn`, this value should be negative (usually -1). Default is 1.

precond.refresh.freq Frequency at which the preconditioner for the conjugate gradiate estimation of the trust region subproblem is reestimated. Preconditioners can help the convergence properties of the algorithm. Default is 1.

preconditioner ID for choice of preconditioner. 0 is the identity matrix (default), For the `Sparse` method, 1 is a modified Cholesky preconditioner. For the `BFGS` method, 1 is the full Cholesky decomposition. If you select 1 for the `SR1` method, the algorithm will use the identity preconditioner instead.

trust.iter Maximum number of conjugate gradient iterations to run when solving the trust region subproblem. A higher number will lead to more accurate solutions to the subproblem, but may also lead to longer run times. Defaults to 2000.

Report levels

The `report.level` control parameter determines how much information is displayed each time the algorithm reports the current state. Possible values are

<=0 No information (a quiet run)

1 Current iteration number, and current value of the objective function.

2 Information from level 1, plus the current norm of the gradient and a status message.

3 Information from levels 1 and 2, plus the current normed radius of the trust region.

4 Information from levels 1, 2, and 3, plus information from each estimate of the trust region subproblem (number of conjugate gradient iterations and how/why the CG algorithm terminated).

Default level is 2. Levels 3 and 4 are available primarily for debugging purposes.

Stopping criteria

The algorithm will stop when one of the following three conditions are met:

- The norm of the gradient, divided by the square root of the number of parameters, is less than `prec`.
- The trust region collapse to a radius smaller than machine precision
- The algorithm proposes zero or negative improvement in the objective function (should never happen)
- The number of iterations reaches the control parameter `maxit`

If the algorithm appears to have stopped prematurely (i.e., the norm of the gradient is still too large), then one might just restart the algorithm. For the quasi-Newton algorithms (`SR1` and `BFGS`), this will refresh the Hessian, and might allow more progress to be made.

Estimating a sparse Hessian

Sometimes estimating the Hessian is easy (e.g., you have an analytic representation, or you are using some kind of algorithmic differentiation software). If you do not know the Hessian, but you do know the sparsity structure, try the **sparseHessianFD** package. The routines in **sparseHessianFD** compute the Hessian using finite differencing, but in a way that exploits the sparsity structure. In many cases, this can be faster than constructing an analytic Hessian for a large problem (e.g., when the Hessian has a block-arrow structure with a large number of blocks).

To use the **sparseHessianFD** package, you need to provide the row and column indices of the non-zero elements of the lower triangle of the Hessian. This structure cannot change during the course of the trust.optim routine. Also, you really should provide an analytic gradient. **sparseHessianFD** computes finite differences of the gradient, so if the gradient itself is finite-differenced, so much error is propagated through that the Hessians are nearly worthless close to the optimum.

Of course, **sparseHessianFD** is useful only for the `Sparse` method. That said, one may still get decent performance using these routines even if the Hessian is sparse, if the problem is not too large. Just treat the Hessian as if it were sparse.

Examples

```
## Not run:
data(binary)
N <- length(binary$Y)
k <- NROW(binary$X)
start <- rep(0,(N+1)*k)
priors <- list(inv.Sigma = diag(k), inv.Omega = diag(k))
opt <- trust.optim(start, fn=binary.f,
                   gr = binary.grad,
                   hs = binary.hess,
                   method = "Sparse",
                   control = list(
                       report.precision=1L,
                       function.scale.factor=-1
                   ),
                   data=binary, priors=priors
                   )

## End(Not run)
```

---

trustOptim *Trust-region optimization*

Description

Nonlinear optimizers using trust regions, with methods optimized for sparse Hessians.

Details

Trust region algorithm for nonlinear optimization. In addition to being more stable and robust than optim, this package includes methods that are scalable and efficient (in terms of both speed and memory usage) when the Hessian is sparse.

References

Braun, Michael. 2014. trustOptim: An R Package for Trust Region Optimization with Sparse Hessians. Journal of Statistical Software 60(4), 1-16. www.jstatsoft.org/v60/i04/.

Nocedal, Jorge, and Stephen J Wright. 2006. Numerical Optimization. Second edition. Springer.

Steihaug, Trond. 1983. The Conjugate Gradient Method and Trust Regions in Large Scale Optimization. SIAM Journal on Numerical Analysis 20(3), 626-637.

Index

binary, 2
binary-data, 2

trust.optim, 3
trustOptim, 6
trustOptim-package *(*trustOptim*)*, 6